UNIX System Services z/OS Version 1 Release 7 Implementation

z/OS UNIX overview

z/OS UNIX setup

z/OS UNIX usage

Paul Rogers
Theodore Antoff
Patrick Bruinsma
Paul-Robert Hering
Lutz Kühner
Neil O'Connor
Lívio Sousa

Redbooks

ibm.com/redbooks

International Technical Support Organization

UNIX System Services z/OS Version 1 Release 7 Implementation

March 2006

SG24-7035-01

Note: Before using this information and the product it supports, read the information in "Notices" on page xiii.

Second Edition (March 2006)

This edition applies to Version 1 Release 7 of z/OS (5637-A01), and Version 1, Release 7 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Notices xiii
Trademarks xiv

Preface xv
The team that wrote this redbook. xv
Become a published author xvi
Comments welcome. xvii

Chapter 1. UNIX overview 1
1.1 UNIX fundamentals 2
1.1.1 UNIX objectives 2
1.1.2 What people like about UNIX 2
1.1.3 What people don't like about UNIX 3
1.1.4 UNIX operating system 3
1.1.5 UNIX file system 4
1.1.6 Parameter files 6
1.1.7 Daemons. 6
1.1.8 Accessing UNIX 6
1.1.9 UNIX standards. 7
1.1.10 MVS and UNIX functional comparison 8
1.2 z/OS UNIX System Services fundamentals 9
1.2.1 Dub and undub 10
1.2.2 z/OS UNIX support 10
1.2.3 Interaction with elements and features of z/OS 11
1.2.4 Hardware considerations 14
1.2.5 Configuration parameters 15
1.2.6 z/OS UNIX file system. 15
1.2.7 Address spaces 22
1.2.8 Accessing z/OS UNIX 22
1.2.9 What people like about z/OS UNIX. 23
1.2.10 What people don't like about z/OS UNIX 23
1.3 z/OS UNIX System Services release history. 23
1.3.1 MVS/ESA V4R3 - 1994 24
1.3.2 MVS/ESA V5R1 - 1994 24
1.3.3 MVS/ESA V5R2M2 - 1995 25
1.3.4 OS/390 V1R1 - 1996. 25
1.3.5 OS/390 V1R2 - 1996. 25
1.3.6 OS/390 V1R3 - 1997. 25
1.3.7 OS/390 V2R4 - 1997. 25
1.3.8 OS/390 V2R5 - 1998. 26
1.3.9 OS/390 V2R6 - 1998. 26
1.3.10 OS/390 V2R7 - 1999. 26
1.3.11 OS/390 V2R8 - 1999. 27
1.3.12 OS/390 V2R9 - 2000. 28
1.3.13 OS/390 V2R10 - 2000. 32
1.3.14 OS/390 V2R10 - 2000 Software Refresh 34
1.3.15 z/OS V1R1 - 2001 35
1.3.16 z/OS V1R2 - 2001 35
1.3.17 z/OS V1R3 - 2002 37

1.3.18 z/OS V1R4 - 2002 40
1.3.19 z/OS V1R5 - 2004 44
1.3.20 z/OS V1R6 - 2004 44
1.3.21 z/OS V1R7 - 2005 44
1.4 IBM exploitation of z/OS UNIX System Services 45
1.5 Additional material for this Redbook 46

Chapter 2. Installation 47
2.1 Introduction 48
2.2 Activating z/OS UNIX in minimum mode 49
2.2.1 Step 1 - Create the OMVS procedure 49
2.2.2 Step 2 - Create the BPXOINIT procedure 49
2.2.3 Step 3 - Establish security 50
2.2.4 Step 4 - Customize IEASYSxx 51
2.2.5 Step 5 - IPL 52
2.3 Activating z/OS UNIX in full function mode 53
2.3.1 Step 1 - Create the OMVS procedure 54
2.3.2 Step 2 - Create the BPXOINIT procedure 54
2.3.3 Step 3 - Create the BPXAS procedure 55
2.3.4 Step 4 - Establish security 55
2.3.5 Step 5 - Create HFS data sets 57
2.3.6 Step 6 - Customize BPXPRMxx 58
2.3.7 Step 7 - Customize ALLOCxx 67
2.3.8 Step 8 - Customize COFVLFxx 67
2.3.9 Step 9 - Customize CTnBPXxx 68
2.3.10 Step 10 - Customize IEADMR00 68
2.3.11 Step 11 - Customize SMFPRMxx 68
2.3.12 Step 12 - Customize IEASYSxx 69
2.3.13 Step 13 - IPL 69
2.3.14 Step 14 - Customize /etc/init.options 72
2.3.15 Step 15 - Customize /etc/rc 76
2.3.16 Step 16 - Customize /etc/profile 78

Chapter 3. Establish security for z/OS UNIX 81
3.1 Superuser authority 82
3.1.1 Defining superusers with appropriate privileges 82
3.1.2 Using the UNIXPRIV class profiles 83
3.1.3 Using the BPX.SUPERUSER profile 84
3.1.4 Switch to superuser mode 84
3.1.5 Assigning a UID of 0 87
3.1.6 Managing UIDs 88
3.2 Creating a RACF environment for products and subsystems 89
3.2.1 RACF group structure 90
3.2.2 Creating user IDs 91
3.2.3 System data set profiles 92
3.2.4 Ownership 92
3.3 The RACF database and z/OS UNIX 92
3.4 Identity mapping with VLF and UNIXMAP 93
3.4.1 VLF PARMLIB definitions 94
3.4.2 Populating and activating the UNIXMAP class 94
3.5 Application identity mapping 96
3.5.1 RACF IRRIRA00 utility 96
3.5.2 AIM conversion considerations 98

3.5.3 Recovering from errors with AIM. 100
3.6 RACF utilities and IRRIRA00 101
3.7 Defining and managing z/OS UNIX users and groups 102
3.7.1 Superuser authority. 103
3.7.2 Authentication and authorization of users to z/OS UNIX 104
3.7.3 Defining users and groups 105
3.7.4 Group access considerations 108
3.7.5 Defining protected user IDs for STCs 108
3.8 User access to the z/OS UNIX shell 109
3.8.1 Define a user's file system 109
3.8.2 Entering the shell from TSO/E 110
3.8.3 Entering the shell from rlogin 111
3.8.4 Entering the shell from telnet 111
3.8.5 Setting z/OS UNIX resource limits for users 112
3.8.6 Support for lowercase user IDs. 113
3.8.7 Setting up field access level for OMVS segment 114
3.9 UNIXPRIV class enhancements 114
3.10 Shared UIDs and GIDs 115
3.10.1 Automatic UID and GID assignment 115
3.10.2 Shared UID and GID prevention 119
3.10.3 SHARED keyword to allow duplicate UID and GID. 120
3.11 Protecting files in the file systems 120
3.11.1 File and directory access 121
3.12 Creating and managing files and directories. 123
3.12.1 Setting the permission bits 123
3.12.2 Setting the UID and GID 125
3.12.3 Change of file ownership 127
3.12.4 Changing permission bits 128
3.12.5 Setuid, setgid, and sticky bits 129
3.13 File and directory access checking 130
3.13.1 Controlling access to files for administrators. 130
3.13.2 Controlling access to files with RACF 131
3.14 Access control lists (ACLs) 134
3.14.1 ACL entries 134
3.14.2 z/OS UNIX V1R3 ACL overview 134
3.14.3 FSP and access control lists. 135
3.14.4 ACL mapping 135
3.14.5 ACL inheritance 136
3.15 Creating and accessing ACLs. 136
3.15.1 Authority to create ACLs 137
3.15.2 Controlling access to files having ACLs for administrators 137
3.15.3 RACF authorization checking flow with ACLs. 138
3.16 Defining ACLs from the z/OS UNIX shell 139
3.16.1 Define all three ACL types 140
3.16.2 Example of ACL inheritance 141
3.17 Defining ACLs from the ISHELL 143
3.17.1 ISHELL panels to display and define ACLs 144
3.17.2 The IRRHFSU utility and ACLs. 149
3.17.3 Modified commands with ACL support 149
3.17.4 Using ACLs in a sysplex 151
3.18 Daemons and security 151
3.18.1 Security environment for daemons 153
3.18.2 UNIX-level security 153

3.19 z/OS UNIX level security for daemons 154
3.19.1 BPX.DAEMON FACILITY class profile 154
3.19.2 RACF program control protection 155
3.19.3 Enhanced program security mode z/OS V1R4 158
3.19.4 z/OS UNIX highest level of security example 161
3.19.5 Defining daemon security 162
3.20 File security packet extattr bits 162
3.20.1 External attribute bits (extattr bits) 163
3.21 Using sanction lists 165
3.21.1 Creating a sanction list 165
3.22 Security for servers 166
3.22.1 The pthread_security_np() callable service 167
3.22.2 Establishing the correct level of security for servers 167
3.22.3 Two levels of z/OS UNIX security for servers 168
3.23 Checking authority to use protected resources 169
3.23.1 Limitations of RACF client ACEE support 170
3.23.2 Defining servers to use thread-level security 170
3.23.3 Defining servers to process users without passwords 171
3.24 Security for operations in z/OS UNIX 172
3.24.1 BPX.SAFFASTPATH 173
3.24.2 BPX.JOBNAME 174
3.24.3 BPX.STOR.SWAP 174
3.24.4 BPX.WLMSERVER 174
3.24.5 Security for ServerPac and CBPDO install 175
3.25 Auditing for z/OS UNIX 177
3.25.1 Setting up audit controls 179
3.25.2 Auditing access to files and directories 180
3.25.3 Specifying file audit options 181
3.25.4 Commands to activate auditing 182
3.25.5 Using the chaudit command 182
3.25.6 Auditing for superuser authority in the UNIXPRIV class 183

Chapter 4. Overview and customization of TCP/IP for z/OS UNIX 185
4.1 Overview of TCP/IP 186
4.2 Customizing and starting TCP/IP 189
4.2.1 Using the z/OS TCP/IP configuration wizard on the Web 190
4.2.2 TCP/IP data sets and configuration files 190
4.2.3 Implementing the sample system 198
4.3 Customizing for inetd and rlogind daemons 201
4.3.1 Customize inetd 202
4.3.2 Customizing the rlogind daemon 204
4.4 Define TCP/IP daemons 206
4.4.1 Syslogd daemon 207
4.4.2 Otelnetd daemon 208
4.4.3 REXECD and RSHD servers 211
4.4.4 FTPD daemon 212
4.4.5 Start daemons 214
4.5 SMTP server 215
4.6 Sending e-mail using SMTP commands 216
4.7 Customizing and starting NFS 217
4.7.1 Configuring the z/OS NFS client 217
4.7.2 Configuring the z/OS NFS server 219
4.7.3 Security settings for the z/OS NFS environment 223

Chapter 5. z/OS Distributed File Service zSeries File System (zFS) 229
5.1 zFS introduction 230
5.2 Application programming interfaces (APIs) 230
5.3 zFS physical file system 231
5.4 zFS colony address space 232
5.5 zFS supports z/OS UNIX ACLs 232
5.6 zFS file system aggregates 233
5.6.1 Compatibility mode aggregates 233
5.6.2 Multi-file system aggregates 234
5.7 Metadata cache 235
5.8 zFS file system clones 235
5.8.1 Backup file system 236
5.9 zFS log files 236
5.10 zFS recovery 236
5.11 Additional information 236

Chapter 6. USS sysplex sharing 237
6.1 USS sysplex sharing design 238
6.2 USS file system structures 239
6.2.1 Symbolic links 239
6.2.2 Single system image OS/390 V2R9 or later 240
6.2.3 USS enhancements in case of sysplex sharing 241
6.2.4 USS file system structures in a sysplex 242
6.2.5 New or changed BPXPRMxx statements 242
6.2.6 The UNIX sysplex sharing structures 243
6.3 USS file system sharing 251
6.3.1 Logical and physical file system relationship 252
6.3.2 Shared USS couple data set 254
6.3.3 USS file system recovery 255
6.4 Automove system list 256
6.4.1 Automove system list specification 256
6.4.2 Changing an automove system list 259
6.5 Showing all USS file sharing structures for a system 260
6.6 USS file system sharing implementation 260
6.6.1 Creating and defining the USS couple data sets 261
6.6.2 Creating the USS sysplex root file system 262
6.6.3 Creating the USS system-specific file system 263
6.7 Additional notes and comments 263
6.7.1 Using TFS filesytype in a colony address space 263
6.8 Effects of USS sysplex sharing 264
6.8.1 How USS sysplex sharing affects mount times 265
6.9 Shared HFS unmount option 265
6.9.1 UNMOUNT option 266
6.10 Mount table limit monitoring 268
6.11 Shared HFS support for the confighfs command 272
6.12 Byte-range locking in a shared HFS environment 272
6.13 Deciding whether to keep or to unmount a version root 274
6.14 Replacing the sysplex root without IPLing any system 275
6.15 USS file systems for licensed program products 278
6.15.1 Using a directory mount point in /usr/lpp 278
6.15.2 Solution 1, using symbolic links in /usr/lpp 282
6.15.3 Solution 2, using a new path structure without referencing /usr/lpp 283
6.15.4 Solution 3, using “Symlink Symbolics” introduced with z/OS V1R5 283

6.16 System-specific data under the version root structure 285
6.17 Replacing a version structure dynamically 285
6.18 File system remount function for USS sysplex sharing 286

Chapter 7. Defining users with z/OS UNIX 291
7.1 Setting up a general user 292
7.1.1 Defining an OMVS segment 292
7.1.2 Creating user file systems. 294
7.1.3 Creating zFS file systems 294
7.1.4 Allocate and format the VSAM linear data set 294
7.1.5 Attach the aggregate to the zFS address space 295
7.1.6 Define a zFS file system inside the aggregate 296
7.1.7 Creating an HFS file system 297
7.1.8 Mounting a file system 298
7.2 Mounting a file system using direct mount 299
7.2.1 Create a user file system for direct mounts. 300
7.3 Mounting file systems with the automount facility 302
7.3.1 Creating the automount facility 302
7.3.2 Display the current automount policy 304
7.3.3 Add to an existing policy 305
7.3.4 Support “#” as comment delimiter in the map file 305
7.3.5 Dynamic HFS allocation in automount policy 305
7.3.6 Generic match on lowercase names. 307
7.3.7 Support of system symbols in the map file 308
7.3.8 Using the automount facility for general users 309
7.4 Setting up started tasks. 310
7.5 Environment variables. 311
7.6 Code page tables 311
7.6.1 Specifying a code page. 312
7.7 Setting the time zone 312
7.7.1 User-defined variables 313

Chapter 8. Exploitation 315
8.1 BookManager BookServer 316
8.1.1 Publish on the Web. 316
8.1.2 Read BookManager books on the Web 316
8.2 DFS SMB 316
8.2.1 SMB support features 317
8.2.2 SMB processes. 317
8.2.3 Shared directories. 317
8.2.4 Shared printers 318
8.3 HTTP Server 318
8.3.1 Additional information 318
8.4 Infoprint Server 318
8.4.1 Printing from UNIX System Services 319
8.4.2 UNIX commands with Infoprint Server 320
8.5 Java support on z/OS 323
8.5.1 What is Java? 323
8.5.2 SDK installation and setup 324
8.5.3 Considerations when using Java 326
8.5.4 Garbage collection 328
8.5.5 Tuning Java and LE runtime options 330
8.5.6 Enhanced z/OS linkage and heap pools. 333

8.5.7 Reusable JVM. 335
8.6 NFS 337
8.6.1 Using z/OS UNIX files 338
8.6.2 Using conventional z/OS data sets 339
8.6.3 Supported servers for the z/OS NFS client 340
8.6.4 WebNFS support 341
8.6.5 Native ASCII support 341
8.7 Text Search 341
8.7.1 The Text Search Engine environment. 342
8.7.2 Client/server communication. 343
8.7.3 Text Search Engine concepts 343
8.8 Tivoli Storage Manager 344

Chapter 9. Interacting with z/OS UNIX. 347
9.1 Commands to monitor z/OS UNIX 348
9.1.1 Interprocess communication signals 348
9.1.2 Kill a process. 349
9.2 z/OS UNIX interactive interfaces. 351
9.3 Using the ISHELL 352
9.3.1 ISHELL enhancements 353
9.3.2 Using the cursor on the Directory List panel 355
9.3.3 Displaying colors on the Directory List panel 360
9.4 Invoking the z/OS UNIX shell 361
9.4.1 Using z/OS UNIX shell commands 362
9.4.2 History file 362
9.4.3 Customizing $HOME/.profile. 363
9.4.4 Command prompt 367
9.4.5 Built-in shell variables 368
9.4.6 Subcommand mode 369
9.4.7 REXX, CLISTs, and shell scripts. 369
9.4.8 Help facilities 371
9.5 Direct login to the z/OS UNIX shell 374
9.6 BPXBATCH. 376
9.6.1 BPXBATSL 379

Chapter 10. Tools, functions, and programming interfaces 383
10.1 Useful tools for z/OS UNIX 384
10.1.1 Skulker 384
10.1.2 Copytree 385
10.1.3 OpenSSH 387
10.1.4 The ssh daemon 387
10.1.5 USSTools 390
10.2 REXX functions and interfaces 393
10.3 Programming examples for REXX 395
10.4 Programming example for C 402
10.5 Using BPX.JOBNAME for USS processes 405
10.5.1 Using _BPX_JOBNAME in /etc/rc. 405
10.6 C Shell. 406
10.7 Disabling use of SMF exit IEFUSI for UNIX processes. 407
10.8 USS hard links used with SMP/E 408
10.9 OMVS syntax checker. 410
10.10 Storage limits for UNIX processes 411
10.11 Using BPXCOPY to load files into the UNIX file structure. 416

10.12 Using the magic number 417
10.13 Enhanced ASCII functionality 418
10.13.1 ASCII support overview 419
10.14 Automatic conversion 420
10.14.1 Autoconversion 420
10.14.2 Scope of autoconversion 421
10.15 File tagging 422
10.15.1 File tag metadata 423
10.15.2 How to tag files 423
10.15.3 Shell commands for tags 424
10.15.4 Accessing data by programs. 426
10.15.5 Other ways to tag files 427
10.15.6 C/C++ 429
10.15.7 Language environment run-time 429

Chapter 11. Administration 431
11.1 Shutting down z/OS UNIX without re-IPLing 432
11.1.1 Registration support 432
11.1.2 Shutting down z/OS UNIX. 433
11.1.3 Restarting z/OS UNIX 436
11.2 z/OS UNIX file systems 437
11.2.1 How to start colony address spaces 438
11.2.2 Start colony address spaces outside of JES. 438
11.2.3 Running a temporary file system in a colony address space 439
11.2.4 TFS in shared file system mode 440
11.3 Managing HFS data sets. 441
11.3.1 DFSMSdss dump and restore. 441
11.3.2 Increasing the size of an HFS data set 447
11.3.3 Logical backup and restoring of file systems using TSM 450
11.3.4 Physical copying of file systems 454
11.4 Monitoring z/OS UNIX. 454
11.4.1 Resource Measurement Facility (RMF) 454
11.4.2 SDSF process panel 456

Chapter 12. Tuning and performance 459
12.1 HFS and zFS file system comparison 460
12.1.1 zFS cache sizes 460
12.1.2 Comparison of results 463
12.2 Domino and zFS performance 464
12.3 The Domino server environment 464
12.3.1 Tasks performed by the Domino server 466
12.3.2 Test results 469
12.3.3 Client-driven workloads. 472
12.3.4 Domino performance conclusions. 481
12.4 Additional information about zFS 481

Chapter 13. Maintenance of z/OS UNIX. 483
13.1 HFS data set backup and recovery. 484
13.1.1 Backing up and restoring HFS data sets using DFSMShsm. 484
13.1.2 Backing up and restoring HFS data sets using DFSMSdss 484
13.2 Increasing the size of an existing HFS data set 487
13.2.1 Increasing the size of the root HFS data set. 487
13.2.2 Increasing the size of other HFS data sets 488
13.3 Installing service using SMP/E 489

13.3.1 Applying service to an active root 490
13.3.2 Installing service to products in the HFS. 493
13.3.3 Prepare for SMP/E APPLY 494
13.4 Post-installation tasks 497

Chapter 14. Problem determination. 499
14.1 Failures and messages in the z/OS UNIX environment 500
14.1.1 z/OS USS messages and codes. 500
14.1.2 Messages from failing z/OS UNIX functions 500
14.1.3 z/OS UNIX latches 501
14.1.4 Getting a console dump for a hang. 503
14.2 Slip trap settings and OMVS component trace 504
14.2.1 Setting a slip for SVCDUMP based on a UNIX reason code 504
14.2.2 Slip for SVDUMP on FSUM shell and utilities message 505
14.2.3 OMVS component trace 505
14.2.4 General message slip trap 506
14.3 USS sysplex sharing diagnosis. 506
14.3.1 Shared USS diagnostic and repair functions 506
14.3.2 USS sysplex sharing diagnostic procedures. 507

Appendix A. Managing z/OS UNIX user IDs and groups 509
A.1 Managing RACF user and group profiles 510
A.1.1 Listing GIDs 510
A.2 JCL example to define a user OMVS segment. 512
A.3 Methods to list UIDs 513
A.4 The ICETOOL utility 516
A.4.1 Unload the RACF database 516
A.4.2 Run a UIDs report using ICETOOL 517
A.4.3 GID reports using ICETOOL. 519
A.4.4 Backing up the primary RACF database 522
A.4.5 Statistics on the UNIXMAP class 523
A.4.6 Backing up the RACF database using the IRRIRA00 utility 523
A.4.7 IRRIRA00 utility - stage 0 to stage 1. 524
A.4.8 Inactivate UNIXMAP class 525
A.4.9 COFVLF00 parmlib member. 525
A.4.10 Rerun the IRRUT200 utility. 525
A.4.11 Replace the RACF backup database 526
A.4.12 Activate the backup database 526

Appendix B. Installation files 529
SYS1.PROCLIB(BPXAS). 530
SYS1.PROCLIB(BPXOINIT) 530
SYS1.PROCLIB(OMVS) 531
SYS1.SAMPLIB(BPXPRMXX) 531
/samples/inetd.conf 551
/samples/init.options. 552
/samples/profile 553
/samples/rc. 557

Appendix C. Access control list (ACL) support considerations. 559
C.1 Examples of the setfacl and getfacl commands 560
C.1.1 Change access level for user/group in an extended ACL 560
C.1.2 Set an entire ACL (base and extended). 561
C.1.3 Pipe output from getfacl to setfacl 562

C.2 Working with default ACLs 563
C.2.1 Use output of find as input to setfacl 564
C.3 Callable services for ACL support 565
C.4 z/OS UNIX REXX support for ACLs 566
C.4.1 Other interface changes 567
C.4.2 LE Callable Services support for ACLs 568

Related publications 571
IBM Redbooks 571
Other publications 571
Online resources 572
How to get IBM Redbooks 573
Help from IBM 573

Index 575

Notices

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®	Domino®	MVS™
@server®	DB2 OLAP Server™	MVS/ESA™
Redbooks (logo) ™	DB2 Universal Database™	Net.Data®
iNotes™	DB2®	NetView®
iSeries™	DFS™	Notes®
z/OS®	DFSMS/MVS®	OS/390®
zSeries®	DFSMSdss™	Redbooks™
AnyNet®	DFSMShsm™	RACF®
AD/Cycle®	DFSORT™	RAMAC®
AFP™	Electronic Service Agent™	RMF™
AIX®	Infoprint®	S/390®
AS/400®	Intelligent Miner™	Tivoli®
BookManager®	IBM®	VisualAge®
C/370™	IMS™	VM/ESA®
Common User Access®	Language Environment®	VTAM®
CICS®	Lotus Notes®	WebSphere®
CUA®	Lotus®	
Database 2™	MQSeries®	

The following terms are trademarks of other companies:

Java, JVM, Solaris, Sun, Sun Microsystems, WebNFS, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

bookshelf, Microsoft, Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbook presents the information you need to plan for and run an IBM z/OS® system with support for z/OS UNIX® System Services (z/OS UNIX) and z/OS.e. It provides information to facilitate the installation and use of z/OS Version 1 Release 7 UNIX System Services, and step-by-step instructions on how to install, customize, and use the z/OS UNIX System Services product set.

This redbook is written for MVS™ systems programmers who install and customize the z/OS UNIX System Services product set.

Practical examples are presented to demonstrate the installation and customization of UNIX System Services. This includes examples of the customization of DFSMS, RACF®, TCP/IP, and NFS required to set up a z/OS UNIX System Services environment.

Some knowledge of UNIX System Services is assumed.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

This edition was produced by the following team.

Paul Rogers is a Consulting IT Specialist at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on various aspects of z/OS JES3, Infoprint® Server, and z/OS UNIX. Before joining the ITSO 18 years ago, Paul worked in the IBM Installation Support Center (ISC) in Greenford, England, providing OS/390® and JES support for IBM EMEA and the Washington Systems Center in Gaithersburg, Maryland.

Paul-Robert Hering is an IT Specialist at the ITS Technical Support Center, Mainz, Germany. He advises customers on z/OS and UNIX System Services-related questions and problems. He has participated in several ITSO residencies since 1988, writing about UNIX-related topics. Before providing support on OS/390 and z/OS, he worked with VM and all its different flavors (VM/370, VM/HPO, VM/XA, and VM/ESA®) for many years.

Patrick Bruinsma is an IT Specialist working for IBM Global Services in the Netherlands. He has five years of general expertise on z/OS, DB2®, MQSeries®, Websphere MQ Workflow, Blaze Advisor, CICS®, and UNIX System Services.

These three authors also produced the previous version, along with the following additional team members:

Theodore Antoff is a Senior RACF Architect in Australia, director of his own companies in Australia (AG Glomar Pty Ltd) and USA (antoff IT, Inc), consulting on all aspects of the IBM Security Server components. He has 15 years of experience in RACF and MVS systems programming. His projects include SDSF, CICS and DB2 conversions to RACF, CA-Top Secret to RACF migrations, merging RACF databases, and Security Technical Reviews. In his previous career as a physicist, he was involved in the research and development of the technology of non-volatile memories based on MIS structures. He has worked for IBM in Australia for three years.

Neil O’Connor is a z/OS Technical Consultant working for IBM Global Services in Australia. He has 30 years of experience in the mainframe operating systems field. His areas of expertise include systems programming, automated operations, and the deployment of standardized z/OS platforms, tools and processes, throughout IBMGS SDCs worldwide. He has participated in authoring other ITSO projects including the previous USS redbook and USS Training Camp. As part of the Global Service Delivery Technical Architecture Committee, Neil is a frequent visitor to Poughkeepsie.

Lutz Kühner is a z/OS systems programmer working for IBM business services in Germany. He has 16 years of experience in the mainframe operating systems field. His areas of expertise include systems programming, tools, and processes.

Lívio Sousa is a System Engineer and member of the zSeries® Technical Sales Support team in Latin America. He has three years of experience in the operational systems and networking fields. He is a student of Computer Science at FASP, São Paulo. He has been working since 2002 at IBM, responsible for planning and implementation of new workload projects on zSeries.

The team, from left to right: Neil O’Connor, Lutz Kühner, Theodore Antoff, Lívio Sousa, Paul-Robert Hering, Paul Rogers, Patrick Bruinsma

Thanks to the following people for their contributions to this project:

Rich Conway
International Technical Support Organization, Poughkeepsie Center

Alfred Schwab, Alison Chandler
Editors, International Technical Support Organization, Poughkeepsie Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

UNIX overview

This chapter introduces UNIX.

We describe UNIX concepts for the benefit of heritage MVS users. References are made to equivalent MVS functions, wherever it is appropriate.

The terms *z/OS UNIX System Services* and *z/OS UNIX* both refer to the IBM UNIX implementation in the z/OS operating system.

1.1 UNIX fundamentals

UNIX is an interactive, multi-user, multi-tasking operating system, designed to be independent of the hardware platform it runs on. The first version of UNIX was created in 1971.

The term “UNIX” is not itself an acronym, but it was derived from the acronym of an earlier operating system called UNiplexed Information and Computing Service (UNICS). These days, “UNIX” is a registered trademark licensed exclusively through The Open Group.

Operating systems may only use the UNIX trademark if they have been certified to do so by The Open Group. UNIX-compatible operating systems that are not certified by The Open Group are typically referred to as “UNIX-like”. For instance, Linux® is a UNIX-like operating system.

1.1.1 UNIX objectives

Some of the objectives of the design of UNIX include:

- Make each program perform a single function well, and reuse that program wherever that function is required.
- Write each program expecting its output to become input to another. This means many simple programs can be combined to perform complex tasks.
- Develop programs incrementally. Start small, then test and modify incrementally until the program is completed.
- Use terse commands and messages to reduce typing and screen output.

1.1.2 What people like about UNIX

Some of the reasons why UNIX is popular include:

- Operating system hardware independence. Operating system code is written in the C language (rather than a specific assembly language) so it can easily be moved from one hardware platform to another.
- Applications are portable. Moving an application (or porting) from one hardware platform to another is generally as simple as transferring the source, then recompiling it.
- Full multitasking with protected memory. Multiple users can run multiple programs concurrently without interfering with each other.
- Very efficient virtual memory. Many programs can execute with only a small amount of physical memory available.
- Access controls and security. All users must be authenticated by a valid account and password to use the system. All files are owned by particular accounts. The owner can decide whether others have read or write access to the owner’s files.
- Productive development environment. For programmers, UNIX offers rich tooling and command language. Commands and utilities can be strung together in unlimited ways to accomplish complex tasks.
- Unified file system. Everything is a *file*: data, programs, and physical devices. The entire file system appears as a single large *tree* of nested directories.
- Distributed processing.

1.1.3 What people don’t like about UNIX

Some of the reasons why UNIX is *not* popular include:

- The traditional command line shell interface is “user unfriendly”. It is more designed for the programmer than the casual user.
- Commands typically have cryptic names and do not offer much feedback to the user. With heavy use of special keyboard characters, small typing errors can produce unexpected results.
- To use UNIX well, the user needs to understand some of the main design features. The power of UNIX comes from knowing how to make commands and programs interact with each other, not just from treating each as a fixed “black box”.
- UNIX has a huge number of commands and utilities that a user may find overwhelming. And the available documentation has few examples or tutorials to help users understand how to use these commands and utilities.

1.1.4 UNIX operating system

UNIX is functionally organized at three levels: kernel, shell and utilities. Technically, only the kernel and the shell form the operating system, while the utilities have evolved over time to make the operating system more immediately useful to the user.

Kernel

The *kernel* is the core of the UNIX operating system. It consists of a small collection of software that makes it possible for the operating system to provide other services. The kernel provides four basic types of services:

- Creation and management of processes
- A file system
- Communications
- A means to start the system

Kernel functions are of two broad types: autonomous and responsive. Kernel functions, such as allocation of memory and CPU, are performed without being explicitly requested by user processes. Other functions of the kernel, such as resource allocation, and process creation and management, are initiated by requests from processes.

UNIX users do not need to know anything about the kernel, just as TSO users do not need to know anything about MVS.

Processes

A *process* is the execution of a program. Some operating systems (such as MVS) call the basic unit of execution a “job” or “task”. In UNIX it's called a process. In the UNIX kernel, anything that's done, other than autonomous operations, is done by a process issuing system calls. Processes often spawn other processes (using the *fork()* system call) that run in parallel with them, accomplish subtasks and, when they are finished, terminate themselves.

All processes have “owners”. Typically the human owner of a process is the owner of the account whose login process spawned the process in question. When a process creates or spawns another process, the original process is known as the parent process while the process it creates is called a child process. The child process inherits the file access and execution privileges belonging to the parent.

Signals

One way that processes communicate with each other and with the kernel is through signals. Signals are used to inform processes of unexpected external events such as a time-out or forced termination of a process. A signal consists of a prescribed message with a default action embedded in it. Each signal has a unique number associated with it.

Virtual memory

UNIX utilizes paging and swapping techniques similar to MVS.

Shell

The *shell* is the interactive environment UNIX users encounter when they log in, similar to what MVS users encounter when they log on to TSO. The shell's prompt is usually visible at the cursor's position on the screen, similar to line-mode in a TSO session. To perform work, commands are entered at the prompt.

The shell is a command interpreter, that is, it takes each command entered and passes it to the operating system kernel to be acted upon. The results of this operation are displayed on the screen. Several shells might be available on a UNIX system for a user to choose from, each with its own strengths and weaknesses. A user may decide to use the default shell or override it. Some of the more common shells are:

- Bourne shell (sh)
- C shell (csh)
- Korn shell (ksh)
- TC shell (tcsh)
- Bourne Again shell (bash)

Each shell also includes its own programming language. Command files, called “shell scripts”, are used to accomplish a series of tasks.

There is a GUI shell available for UNIX systems, called “X-Windows” or simply “X”. This GUI has all the features found on a personal computer. In fact, the version used most commonly on modern UNIX systems (CDE), is made to look very similar to Microsoft® Windows®.

Utilities

UNIX includes many utility programs (often referred to as commands) to perform functions such as:

- Editing
- File maintenance
- Printing
- Sorting
- Programming support
- Online information

1.1.5 UNIX file system

A UNIX file system is a data structure or a collection of files. A file system has both a logical (hierarchical directory tree) and physical (arrangement of files on disk partitions) dimension.

Logical file system

The logical file system refers to the hierarchy of connected directories made of all the files (or disk partitions) that are accessible to the user. The UNIX file system is arranged in a tree or inverted pyramid, where all files are logically contained within the root directory. See Figure 1-1, where the shaded boxes represent directories, while the unshaded boxes represent files.

This is similar to the Microsoft Windows hierarchical file system, except that the directory separator is a forward slash (/), compared to the Windows backslash (\). There are slight differences in the arrangement of directories between variants of UNIX, however, the overall structure is basically the same. Note that UNIX is a case-sensitive operating system, so a file called “ABC” is different from a file called “abc”.

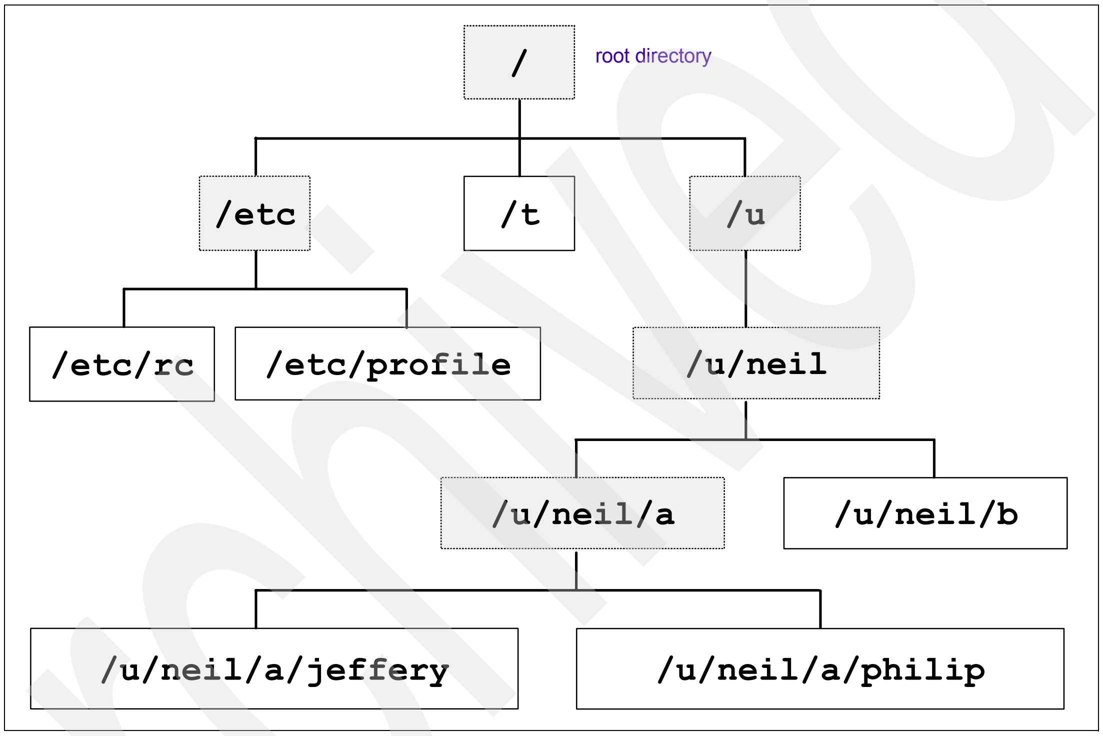


Figure 1-1 Hierarchical file system

Physical file system

The physical file system is divided first by disk partitions. Partition size determines the number of blocks that the file system uses. Each file system has a superblock, inodes, and data blocks. The superblock holds the control information for the system. Inodes contain similar information for individual files. The data blocks hold data, the information in the files.

File and directory permissions

Every file or directory in a UNIX file system has three types of permissions (or protections) that define whether certain actions can be carried out. The permissions are:

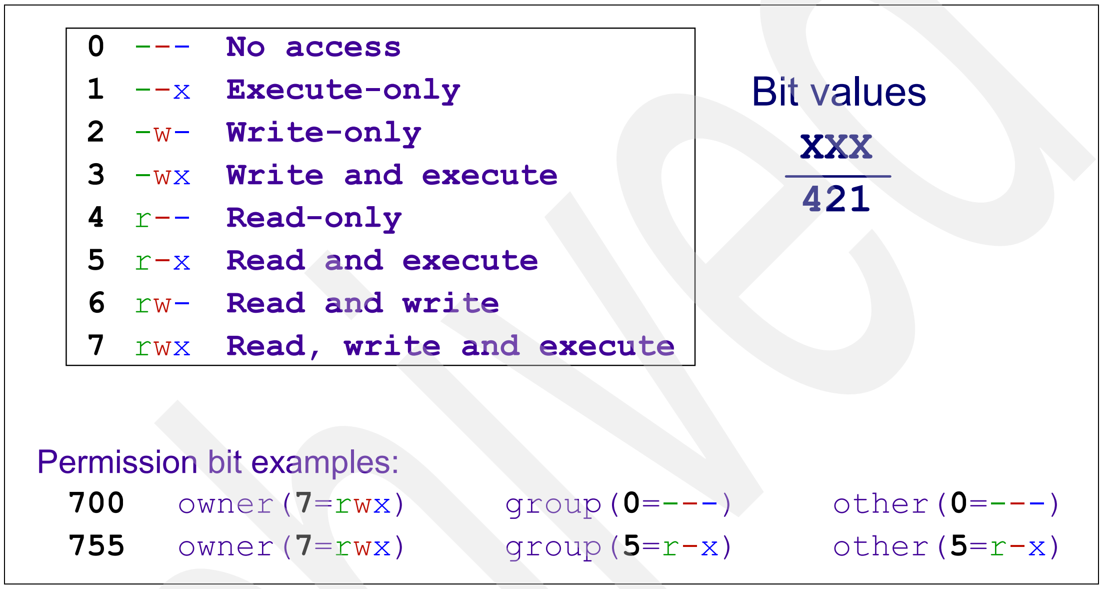
read [r] A user who has read permission for a file may look at its contents or make a copy of it. For a directory, read permission enables a user to find out what files are in that directory.

write [w] A user who has write permission for a file can alter or remove the contents of that file. For a directory, the user can create and delete files in that directory.

execute [x] A user who has execute permission for a file can cause the contents of that file to be executed (provided that it is executable). For a directory, execute permission allows a user to change to that directory.

These permissions are applied and tested at three levels: the owner's user ID; the owner's group, and other users.

Figure 1-2 shows how permission bits are often referred to by their octal representation. For example, if a file is to be updated only by its owner, while others are allowed to read/execute it, then the correct octal permission setting is 755 (owner = rwx = 4+2+1 = **7**; group = r-x = 4+0+1 = **5**; other = r-x = 4+0+1 = **5**).



0	---	No access
1	--x	Execute-only
2	-w-	Write-only
3	-wx	Write and execute
4	r--	Read-only
5	r-x	Read and execute
6	rw-	Read and write
7	rwx	Read, write and execute

Bit values

xxx
421

Permission bit examples:

```
700   owner(7=rwx)     group(0=---)     other(0=---)
755   owner(7=rwx)     group(5=r-x)     other(5=r-x)
```

Figure 1-2 Octal representation of permissions

1.1.6 Parameter files

Parameter files are typically stored in the /etc directory. This is similar to SYS1.PARMLIB on an MVS system.

1.1.7 Daemons

A daemon is a program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive. The daemon program forwards the requests to other programs (or processes) as appropriate. Daemons are like Started Tasks (STCs) in MVS.

1.1.8 Accessing UNIX

To access UNIX interactively, the user has to *log in* to their *user account* using the *rlogin* (remote login) or *telnet* interface. rlogin and telnet are similar except rlogin supports access from trusted hosts without requiring a password (hence security people will like this less than telnet).

Most platforms (including Microsoft Windows) include a telnet command/interface. When logging in, remember that UNIX is case-sensitive, so uppercase characters used in the userid or password are not the same as lowercase characters.

UNIX also has a console interface (similar to an MVS console) but that is normally only used by system administrators or computer operators.

UIDs

The *user account* of a UNIX user is represented in two ways: username and UID. Username is an easy-to-remember word, while UID is a number. This information might be stored in the file /etc/passwd. UID is typically a number between 0 and 65,535, where 0 thru 99 might be reserved. UID=0 has special meaning as the superuser.

Superuser (root)

Superuser is a privileged user (UID=0) who has unrestricted access to the whole system, that is, all commands and all files regardless of their permissions. By convention, the user name for the superuser account is root. Don't confuse the term *root* here with the *root subdirectory* in the file system—they are unrelated.

The root account is necessary because many system administration files and programs need to be kept separate from the executables available to non-privileged users.

Also, UNIX allows users to set permissions on the files they own. A system administrator (root) may need to override those permissions.

GIDs

Each UNIX user is also associated with a grouping so that people in the same workgroup can share data. This grouping is represented in two ways: group name and GID. Group name is an easy-to-remember word, while GID is a number. This information might be stored in the file /etc/group. GID is typically a number between 0 and 65,535, where 0 thru 99 might be reserved. Unlike UID, GID=0 has no special meaning.

1.1.9 UNIX standards

The work on Portability Operating Systems Interface (POSIX) started as an effort to standardize UNIX and was performed by a workgroup under the Institute of Electrical and Electronical Engineers (IEEE). What they defined was an application programming interface that could be applied to any operating system.

POSIX is not a product. It is an evolving family of standards describing a wide spectrum of operating system components ranging from C language and shell interfaces to system administration.

The POSIX standard is sponsored by International Organization for Standardization (ISO) and is incorporated into X/Open Portability Guides (XPG). Each element of the standard is defined by a 1003.* number. For example:

1003.1	System Application Program Interface for C
1003.1a	System Application Program Interface Extensions
1003.1b	Real Time Extensions
1003.1c	Threads Extensions (previous 1003.4a)
1003.1e	Security Extensions
1003.1f	Network - Transparent File Access
1003.1g	Protocol Independent Network API
1003.2	Shell and Utilities
1003.5	ADA Bindings (for 1003.1)

1003.9	Fortran Bindings (for 1003.1)
1003.13	Real Time Application Environment Profile
1003.15	Batch System Administration

POSIX defines the interfaces and not the solution or implementation. In this way POSIX can be supported by any operating system. Implementation of POSIX can be different in areas such as performance, availability, and recoverability. Not all POSIX-compliant systems are the same, although they all support basically the same interface.

POSIX and *1003.1* are registered trademarks of the Institute of Electrical and Electronic Engineers, Inc (IEEE).

1.1.10 MVS and UNIX functional comparison

Table 1-1 provides a functional comparison of some of the basic functions of MVS and the equivalent or similar functions with z/OS UNIX.

Table 1-1 MVS and UNIX functional comparison

Function	MVS	UNIX
Background work	Submit batch JCL	sh_cmd &
Configuration parameters	SYS1.PARMLIB	/etc
Data management	DFSMS, HSM	tar, cpio, pax
Debug	TSO TEST	dbx
Editor	ISPF option 2	ed, sed, oedit, ishell
Initiate new task	ATTACH, LINK, XCTL	fork(), spawn()
Interactive access	Logon to TSO	telnet/rlogin to sh/tcsh
Job management	SDSF	ps, kill
List files	ISPF option 3.4, LISTC	ls
Long running work	Started task (STC)	daemon
Post IPL commands	COMMNDxx	/etc/rc
Power user	RACF OPERATIONS	superuser or root
Primary configuration	IEASYSxx	BPXPRMxx
Primary data index	Master Catalog	root (“/”) directory
Procedural language	CLIST, REXX	shell scripts, REXX
Program products	LNKLST	/usr
Resident programs	LPA	sticky bit
System logging	SYSLOG	SYSLOGD
System programs	LNKLST	/bin
Test programs	STEPLIB	/sbin
User data	&SYSUID or &SYSPREF	/u/<username>
User identity	user/group	UID/GID

1.2 z/OS UNIX System Services fundamentals

z/OS UNIX System Services is a UNIX operating environment, implemented within the z/OS operating system. z/OS UNIX System Services is also referred to by its shorter name z/OS UNIX.

The z/OS support for z/OS UNIX enables two open systems interfaces on the z/OS operating system:

- An application program interface (API). The application interface is composed of C interfaces. Some of the C interfaces are managed within the C Run-Time Library (RTL), and others access kernel interfaces to perform authorized system functions on behalf of the unauthorized caller.
- An interactive z/OS shell interface.

Figure 1-3 shows the API and interactive shell open systems interfaces and their relationship with z/OS.

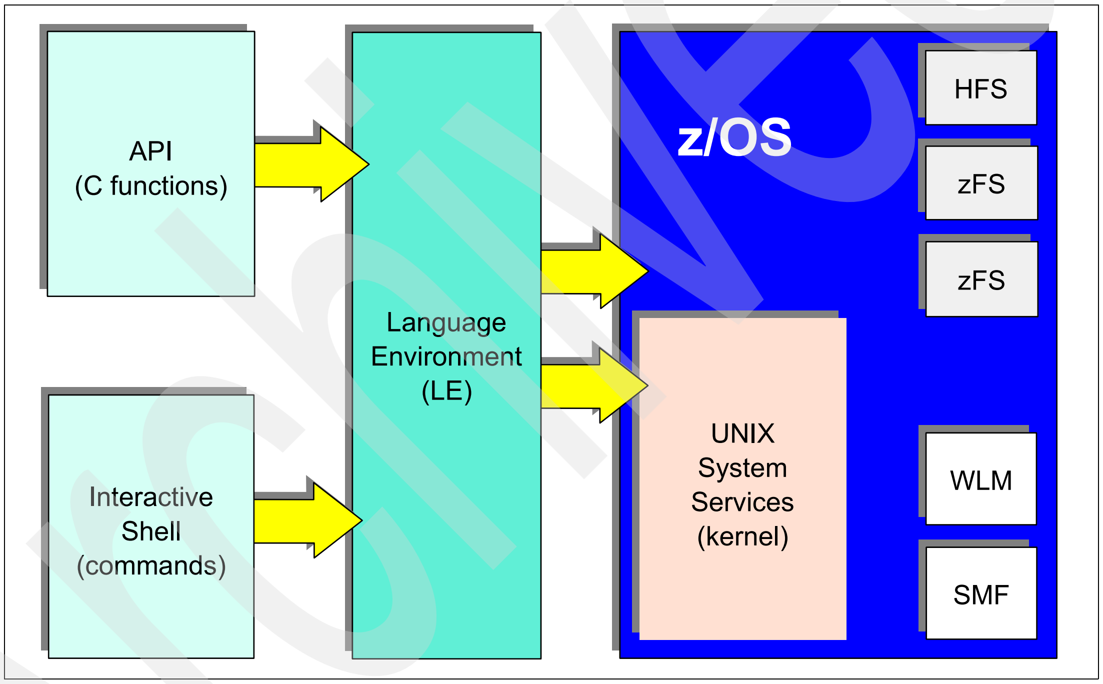


Figure 1-3 z/OS UNIX with open systems interfaces

With the APIs, programs can run in any environment, including in batch jobs, in jobs submitted by TSO/E users, and in most other started tasks, or in any other MVS application task environment. The programs can request:

- Only MVS services
- Only z/OS UNIX
- Both MVS and z/OS UNIX

The shell interface is an execution environment analogous to TSO/E, with a programming language of shell commands analogous to the REXX language. The shell work consists of:

- Programs run by shell users
- Shell commands and scripts run by shell users
- Shell commands and scripts run as batch jobs

z/OS UNIX has two shells, the z/OS shell and the tcsh shell. They are collectively called the z/OS UNIX shells.

z/OS shell

The z/OS shell is modeled after the UNIX System V shell with some of the features found in the Korn shell. As implemented for z/OS UNIX services, this shell conforms to POSIX standard 1003.2, which has been adopted as ISO/IEC International Standard 9945-2: 1992. The z/OS shell is upward-compatible with the Bourne shell.

tcsh shell

The *tcsh* shell is an enhanced but completely compatible version of the Berkeley UNIX C shell, *csh*. It is a command language interpreter usable both as an interactive login shell and a shell script command processor. It includes a command-line editor, programmable word completion, spelling correction, a history mechanism, job control, and a C-like syntax.

1.2.1 Dub and undub

Dub is a term that means to make an MVS address space known to z/OS UNIX System Services. Once dubbed, an address space is considered to be a “process”. Address spaces created by the fork() function are automatically dubbed when they are created; other address spaces become dubbed if they invoke a z/OS UNIX service. Dubbing also applies to MVS tasks. A dubbed task is considered to be a “thread”. Tasks created by pthread_create() are automatically dubbed threads; other tasks are dubbed if they invoke a z/OS UNIX service.

Undub is a term that means the inverse of dub. Normally, a task (dubbed a thread) is undubbed when it ends. An address space (dubbed a process) is undubbed when the last dubbed thread ends.

1.2.2 z/OS UNIX support

z/OS UNIX, which responds to requests from programs and the z/OS UNIX shells, is made up of system and application services.

System Services

System Services provide:

- XPG4 UNIX 1995 conformance
- Assembler callable services
- TSO/E commands to manage the file system
- ISPF shell environment

Application Services

Application Services (FMID HOTxxxx) interprets commands from users or programs, called shell scripts, and requests MVS services in response to the commands. The dbx debugger enables the application programmer to debug source programs written in C or C/C++. Application Services provide:

- A TSO/E command to enter the shell environment
- A shell environment for developing and running applications
- Utilities to administer and develop in a UNIX environment
- The dbx debugger

- Support for socket applications
- rlogin (remote login) and inetd functions
- Direct telnet based on TCP/IP protocol
- Support for full-screen applications (curses support)

It also contains the code that was provided in the optional Shell and Utilities and the Debugger features prior to z/OS.

1.2.3 Interaction with elements and features of z/OS

z/OS UNIX interacts with the following elements and features of z/OS:

- BCP (WLM and SMF components)
- C/C++ Compiler, to compile programs
- Language Environment®, to execute the shell and utilities or any other XPG4-compliant shell application
- Data Facility Storage Management Subsystem (DFSMS) (HFS is a component of DFSMS)
- Security Server for z/OS (RACF is a component of the Security Server)
- Resource Measurement Facility (RMF™)
- System Display and Search Facility (SDSF)
- Time Sharing Option Extensions (TSO/E)
- z/OS Communications Server (TCP/IP)
- ISPF, to use the dialogs for OEDIT, or ISPF/PDF for the ISPF shell
- BookManager® READ/MVS, to use the OHELP online help facility
- Network File System (NFS)
- z/OS Distributed File Service zSeries File System (zFS)

Figure 1-3 on page 9 shows how z/OS UNIX, the shell interface, and the API relate to the rest of the z/OS operating system.

Workload Manager (WLM)

The Workload Manager is a component of the BCP element. The kernel uses WLM to create child processes. When programs issue fork() or spawn(), as shown in Figure 1-4 on page 12, the BPXAS PROC found in SYS1.PROCLIB is used to provide a new address space. For a fork(), the system copies one process, called the parent process, into a new process, called the child process. The forked address space is provided by WLM.

Processes can be created by a fork or spawn. Existing MVS address space types such as TSO, STC, batch, and APPC can request z/OS UNIX services. When one of those address spaces makes its first request to the z/OS kernel, the kernel dubs the task; that is, it identifies the task as a z/OS UNIX process.

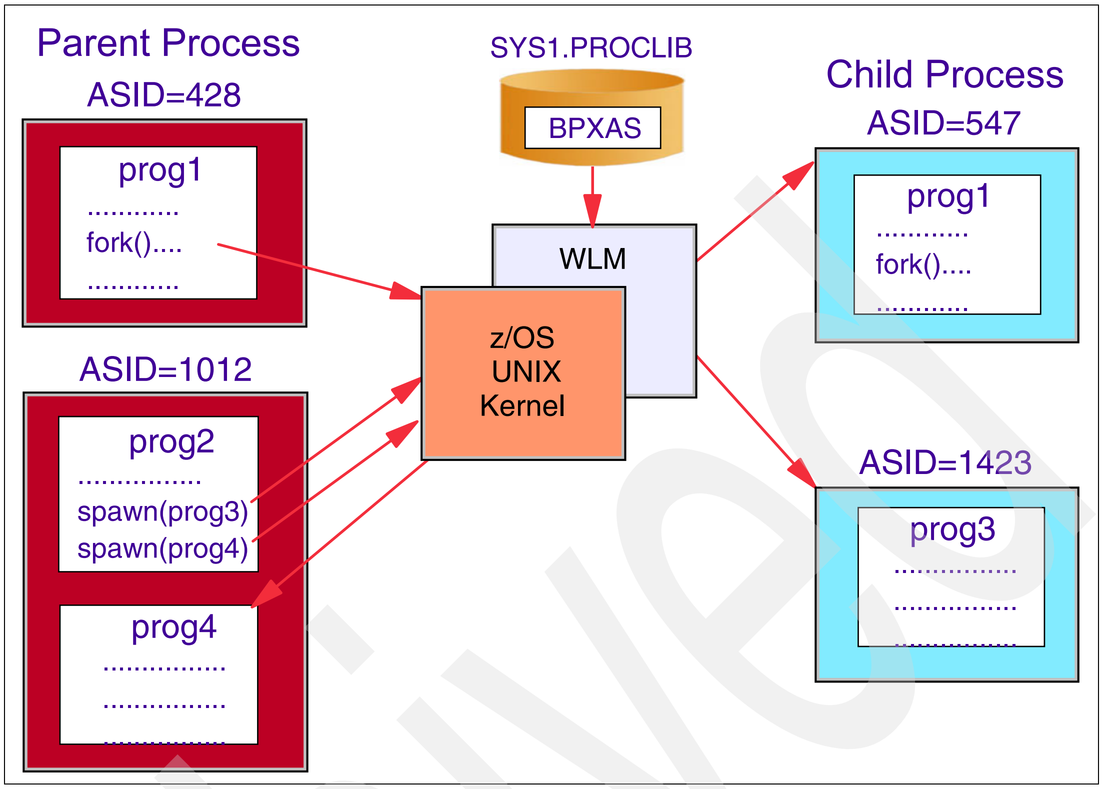


Figure 1-4 Examples of a parent process issuing fork() and spawn()

The types of processes are:

- User processes, which are associated with a user.
- Daemon processes, which perform continuous or periodic system-wide functions, such as a Web server. Daemons are programs that are typically started when the operating system is initialized and remain active to perform standard services. Some programs are considered daemons that initialize processes for users even though these daemons are not long-running processes. Examples of daemons are:
  - cron, which starts applications at specific times.
  - inetd, which provides service management for a network.
  - rlogind, which starts a user shell session when one is requested, using a remote rlogin command.

  In similar systems, initialization usually starts a telnet daemon to perform terminal services. In addition to using a cron daemon, z/OS installations can use Operations Planning and Control/ESA (OPC/ESA) to set up a timed event.

  Daemons are not restarted if they stop. You can restart them in any of several ways:
  - The z/OS operator can restart daemons using a cataloged procedure.
  - A system programmer can restart the daemon from a shell.
  - You can use automation products such as NetView® to notice daemons terminating and then restart them using cataloged procedures.

A process can have one or more threads; a thread is a single flow of control within a process. Application programmers create multiple threads to structure an application in independent sections that can run in parallel for more efficient use of system resources.

System Management Facilities (SMF)

System management facilities (SMF), which is a component of the BCP element, collects data for accounting. SMF job and job-step accounting records identify processes by user, process, group, and session identifiers. Fields in these records also provide information on resources used by the process. SMF file system records describe file system events such as file open, file close, and file system mount, unmount, quiesce, and unquiesce.

Use the JWT value in the SMF parmlib SMFPRMxx to specify when to time-out an idle address space. SMF/WLM does the tracking.

C/C++

To compile C code using the c89 command, or to compile C/C++ code using cxx, you need the C/C++ compiler that is available with z/OS.

Language Environment (LE)

To run a shell command or utility, or any user-provided application program written in C or C++, you need the C/C++ run-time library provided with Language Environment.

Data Facility System Managed Storage (DFSMS)

Data Facility System Managed Storage (DFSMS) can be used to manage the data sets used for processing the Hierarchical File System (HFS). These HFS data sets make up a file hierarchy. A file hierarchy can consist of:

- Files, which contain data or programs. A file containing a load module or shell script or REXX program is called an executable file. Files are kept in directories.
- Directories, which contain files, other directories, or both.
- Additional local or remote file systems, which are mounted within the file hierarchy.

To the MVS system, the file hierarchy is a collection of HFS data sets. Each HFS data set is a mountable file system.

Security Server (RACF)

The RACF component of the Security Server authenticates users and verifies whether they are allowed to access certain resources. An equivalent security product (such as CA-ACF2) can be used to do those tasks.

A user is identified by a UID, which is kept in the RACF user profile, and a GID, which is kept in the RACF group profile.

Resource Measurement Facility (RMF)

Resource Measurement Facility (RMF) collects data used to describe z/OS UNIX performance. RMF reports support an address space type of OMVS for address spaces created by fork or spawn callable services, and support two swap reason codes.

When an installation specifies an OMVS subsystem type in the Workload Manager service policy, RMF shows the activity of forked address spaces separately in the RMF Workload Activity report.

RMF monitors the use of resources in an OMVS Kernel Activity report.

System Display and Search Facility (SDSF)

Shell users can enter TSO/E sessions and use SDSF to:

- Monitor printing

- Monitor and control a batch job
- Monitor and control forked address spaces
- Find out which users are logged on to TSO/E sessions

Time Sharing Options Extensions (TSO/E)

One way to enter the shell environment is using TSO/E. A user logs on to a TSO/E session and enters the TSO/E OMVS command.

The z/OS environment has other TSO/E commands, for example, to logically mount and unmount file systems, create directories in a file system, and copy files to and from MVS data sets. Users can switch from the shell to their TSO/E session, enter commands or do editing, and switch back to the shell.

z/OS Communications Services (TCP/IP Services)

Another way to enter the shell environment is using rlogin or telnet from a workstation in the TCP/IP network.

User-written socket applications can use TCP/IP Services as a communication vehicle. Both client and server socket applications can use the socket interface to communicate over the Internet (AF_INET and AF_INET6) and between other socket applications by using local sockets (AF_UNIX). An assembler interface is also provided for those applications that do not use the C/C++ run-time library.

ISPF

Users of ISPF can use the ISPF shell environment to create, edit, browse, and perform other functions for files and directories in the HFS.

BookManager READ/MVS

You can invoke the online help facility with the TSO/E OHELP command and view online publications in BookManager format.

Network File System (NFS)

Network File System (NFS) enables users to access files on other systems in a network.

zSeries File System (zFS)

zSeries File System (zFS) is a UNIX file system that can be used, along with HFS.

1.2.4 Hardware considerations

You can use the same hardware as the other components of the z/OS system. Use the same network connections that TSO/E uses and the processor and network connections that JES uses.

Additional hardware considerations are:

- If you want to use rlogin, the connections are different from those for TSO/E users.
- The optional Suppression on Protection feature, if not present, negates certain functions such as mmap() and fork() copy-on-write.
- For improved TCP/IP performance, install the CHECKSUM hardware improvement.
- To take advantage of improved performance in semaphore processing, you must be running on hardware that supports the PLO (Perform Locked Operation) instruction.

1.2.5 Configuration parameters

The z/OS implementation of UNIX is different from other implementations because it is part of something else (the z/OS operating system) rather than an independent entity. A non-z/OS implementation of UNIX is typically an operating system that is booted (or IPLed) within its own physical machine (see *z/OS UNIX System Services Planning,* GA22-7800). But with z/OS, UNIX is just an environment within the z/OS operating system, which also processes other non-UNIX (or in this case MVS) workloads (such as CICS, IMS™, MQ, TSO, batch, etc). In fact, on any MVS system today, the UNIX workload is typically the minor workload.

Defining the external configuration of z/OS UNIX

Since z/OS UNIX System Services is an environment within the z/OS operating system, there must be some way to define the environment and the file systems, within the greater z/OS operating system. The solution for this is the BPXPRMxx member of SYS1.PARMLIB. See *z/OS MVS Initialization and Tuning Reference,* SA22-7592 for detailed information about the contents of BPXPRMxx.

Defining the internal configuration of z/OS UNIX

The internal configuration of z/OS UNIX System Services is defined the same way as any other UNIX implementation, such as via the config files within the /etc directory. See *z/OS UNIX System Services Planning,* GA22-7800 for detailed information about the contents of the /etc directory.

1.2.6 z/OS UNIX file system

z/OS UNIX allows you to install virtual file system servers (VFS servers) and physical file systems (PFSs).

- A VFS server makes requests for file system services on behalf of a client. A VFS server is similar to a POSIX program that reads and writes files, except that it uses the lower-level VFS callable services API instead of the POSIX C-language API. An example of a VFS server is the Network File System.
- A PFS controls access to data. PFSs receive and act upon requests to read and write files that they control. The format of these requests is defined by the PFS interface. The following are all PFSs:
  - Hierarchical File System (HFS)

    z/OS UNIX files are organized in a hierarchical file system (HFS), as in other UNIX systems. Files are members of a directory, and each directory is in turn a member of another directory at a higher level.
  - Network File System (NFS)

    Using NFS client on z/OS UNIX, you can mount a file system, directory, or file from any system with an NFS server within your user directory. You can edit or browse the files.
  - Distributed File System (DFS™)

    A DCE component, DFS joins the local file systems of several file server machines making the files equally available to all DFS client machines. DFS allows users to access and share files stored on a file server anywhere in the network, without having to consider the physical location of the file.
  - Temporary File System (TFS)

    The TFS is an in-memory physical file system that delivers high-speed I/O. To take advantage of that, the system programmer (superuser) can mount a TFS over the /tmp directory so it can be used as a high-speed file system for temporary files (normally, the TFS is the file system that is mounted instead of the HFS if z/OS UNIX is started in minimum setup mode).

– zSeries File System (zFS)

  zFS is a UNIX file system that can be used in addition to HFS. It contains files and directories that can be accessed with APIs.

– Pipe

  A program creates a pipe with the pipe() function. A pipe typically sends data from one process to another; the two ends of a pipe can be used in a single program task. A pipe does not have a name in the file system, and it vanishes when the last process using it closes it.

– Socket

  A program creates a socket with the socket() function. A socket is a method of communication between two processes that allows communication in two directions, in contrast to pipes, which allow communication in only one direction. The processes using a socket can be on the same system or on different systems in the same network.

Another name for a PFS is an *installable file system*.

Figure 1-5 on page 17 shows how user-written programs use the POSIX API to issue file requests. VFS servers use the VFS callable services API to issue file requests (see “1”). These requests are routed by the logical file system (LFS) to the appropriate PFS through the PFS interface (see “2”).

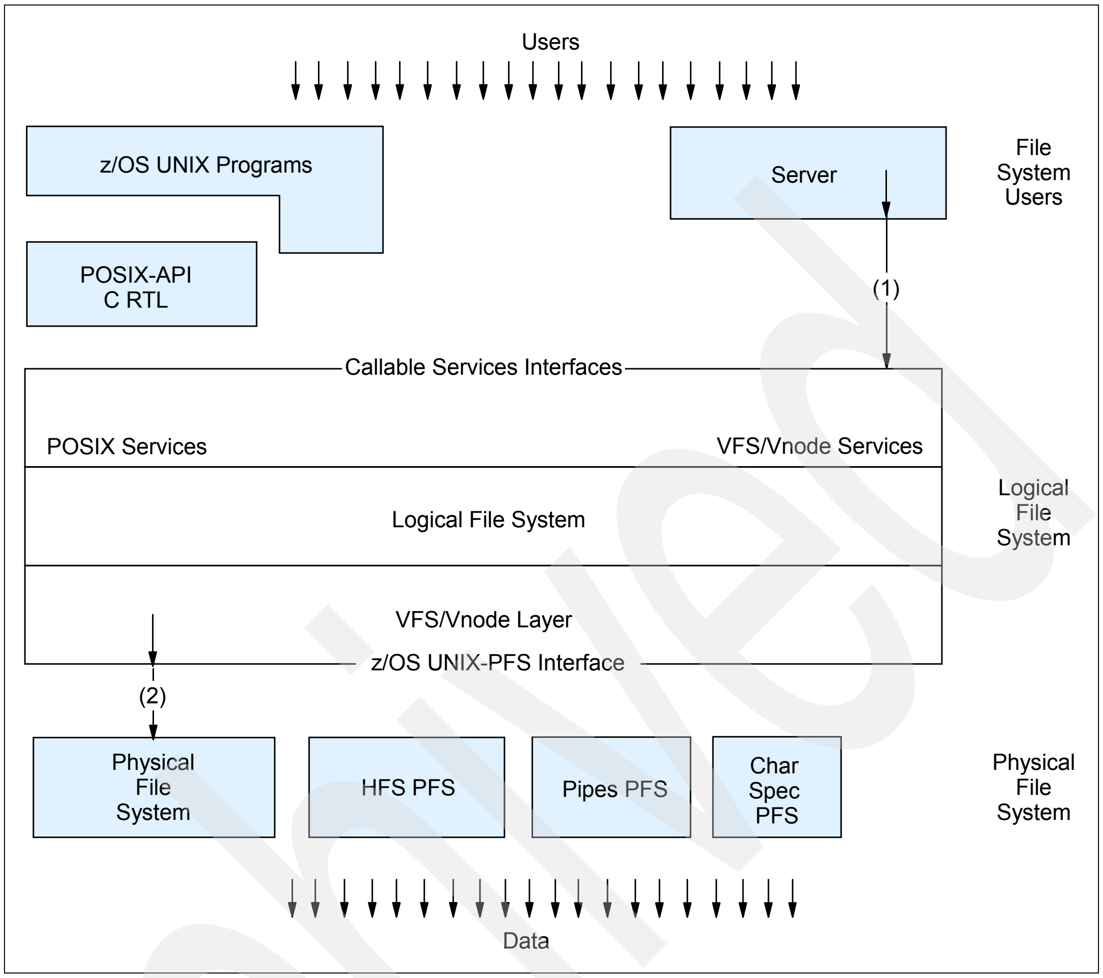


Figure 1-5 VFS server and PFS structure

Physical file systems are defined to z/OS UNIX with the FILESYSTYPE statements in the BPXPRMxx member of SYS1.PARMLIB.

File system organization

z/OS UNIX files are organized in a hierarchy, as in a UNIX system. All files are members of a directory, and each directory is in turn a member of another directory at a higher level in the hierarchy. The highest level of the hierarchy is the root directory.

MVS views an entire file hierarchy as a collection of hierarchical file system data sets (HFS or zFS data sets). Each HFS or zFS data set is a mountable file system. DFSMS facilities can be used to manage HFS data sets.

The root file system is the first file system mounted. Subsequent file systems can be mounted on any directory within the root file system or on a directory within any mounted file system.

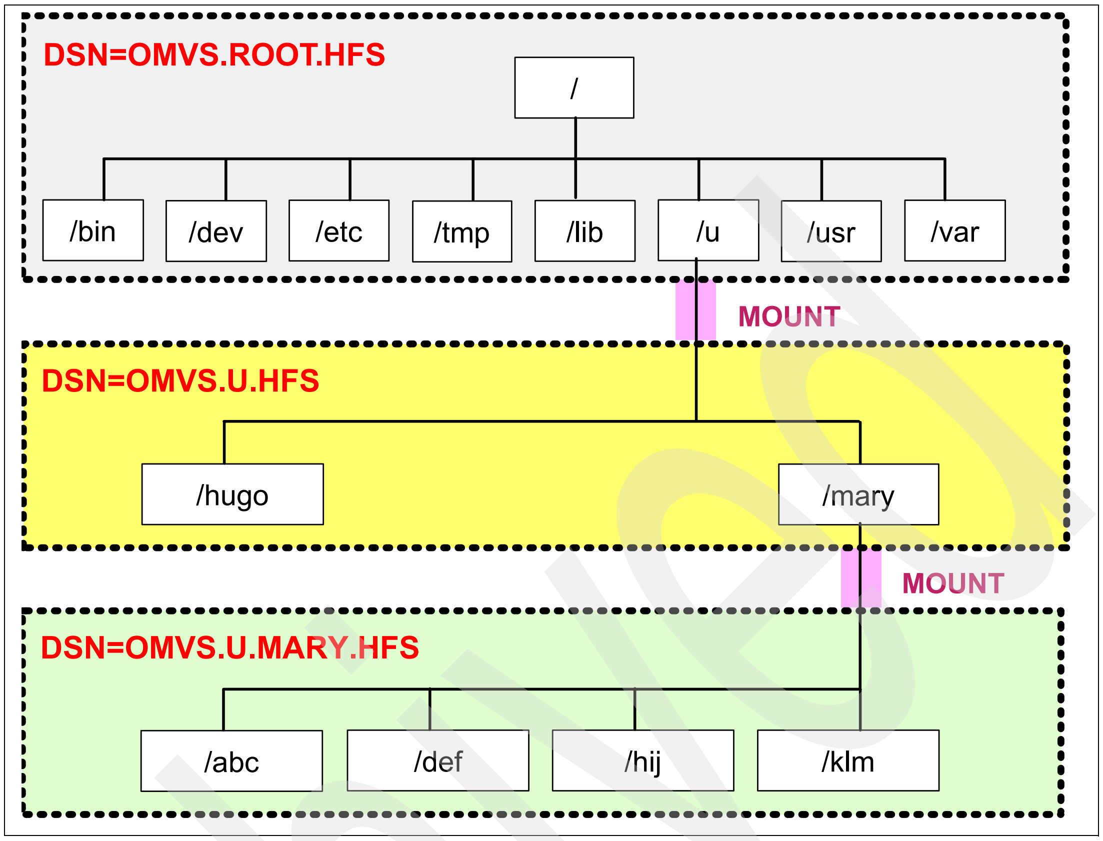


Figure 1-6 Mounted HFS data sets

Figure 1-6 shows an example where HFS data set OMVS.U.MARY.HFS is mounted at directory /mary in HFS data set OMVS.U.HFS. This HFS in turn is mounted at directory **/u** in the root HFS OMVS.ROOT.HFS. Users who access /u/mary/abc can do so seamlessly and have no knowledge that they have passed through two HFS data sets to access the third.

File types

There are four other types of files that can exist in the HFS, in addition to directories:

- A *regular file* is an identifiable (named) unit of text or binary data information. A file can be C source code, a list of names or places, a printer-formatted document, a string of numbers organized in a certain way, an employee record containing smaller information units in fields, a memo, and many other possible things. A user or an application program must understand how to access and use the individual increments of information (such as employee record fields) within a file.
- A *character special file* defines one of these:
  - A terminal (/dev/ptypnnnn and /dev/ttypnnnn). Only a superuser can create this file.
  - The default controlling terminal for a process (/dev/tty).
  - A null file (/dev/null). Data written to this file is discarded; hence, it is known as “the bit bucket”. Only a superuser can create this file.
  - A file descriptor file (/dev/fdn or /dev/fd/n). Only a superuser can create this file.
  - A system console file (/dev/console). Data written to this file is sent to the console using a write-to-operator (WTO) that displays the data on the system console. Only a superuser can create this file.

  - A UNIX domain socket name file. This is a pathname that specifies the socket address for a UNIX domain socket. The pathname is assigned by the application programmer; there is no convention for the name. The operating system creates the file.
  - A Communications Server remote tty file (for example, rtynnnn) that corresponds to the requesting terminal on the originating Communications Server node. The name is assigned by the Communications Server administrator.
  - The Communications Server character special file (/dev/ocsadmin) that supports ioctl functions for Communications Server administrative functions.
- A *FIFO special file* is a file typically used to send data from one process to another so that the receiving process reads the data first-in-first-out (FIFO). A FIFO special file is also known as a named pipe.
- A is a file that contains the pathname for another file, in essence a reference to the original file. Only the original pathname is the real name of the original file. You can create a symbolic link to a file or a directory. In OS/390 V2R9 and later, /etc, /tmp, /dev, and /var are symbolic links.

  An external link is a type of symbolic link, a link to an object outside of the HFS. Typically, it contains the name of an MVS data set.

Users and programs create regular files, FIFO special files, symbolic links, and external links.

File security packet

Each z/OS UNIX file and directory has a 64-byte file security packet (FSP) associated with it to control access. The FSP is created when a file or directory is created, and is stored in the file system for the life of the file/directory, until the file/directory is deleted, at which time the FSP is also deleted. Figure 1-7 shows the structure of the FSP.

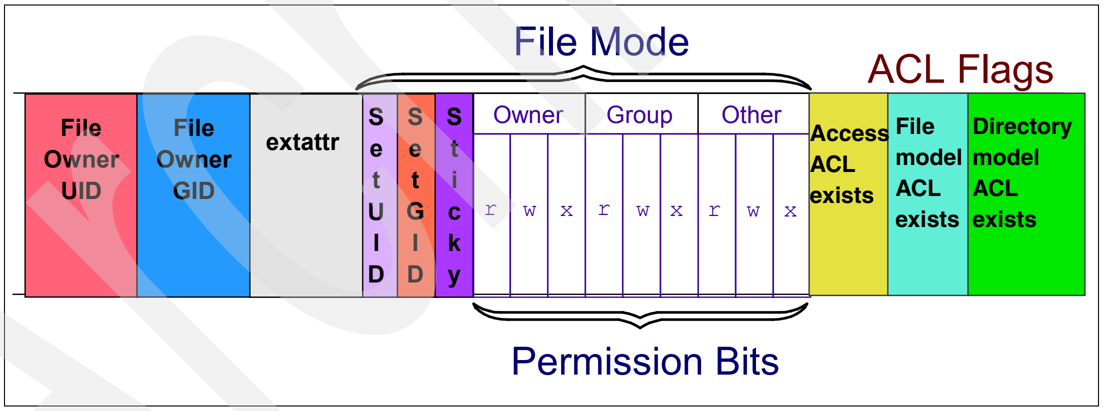


Figure 1-7 File security packet

The FSP includes the following fields:

UID	File owner UID.
GID	File owner GID.
extattr	Extended program attributes. See “Executable modules in the file system” on page 20.
SetUID	This bit only relates to executable files. If on, it causes the UID of the user executing the file to be set to the file's UID.
SetGID	This bit only relates to executable files. If on, it causes the GID of the user executing the file to be set to the file's GID.

Sticky Bit	This bit only relates to executable files. If on, it causes the file to be retained in memory for performance reasons. The implementation of this varies between platforms. In z/OS UNIX, it means programs are loaded from LPA (or LNKLST as per normal MVS program search) instead of a HFS file. For a directory, the sticky bit causes UNIX to permit files in a directory or subdirectories to be deleted or renamed only by the owner of the file, or by the owner of the directory, or by a superuser. See "Symbolic and external links with a sticky bit" on page 21.
Permission bits	Owner, group and other permission bits, where owner is the UID that owns the file, group is the GID of the owning user, and other is anybody else. See "File and directory permissions" on page 5. Note that in z/OS UNIX, these three permissions (r/w/x) are not hierarchical. For example, a user with write permission who does not have read permission, can only write over existing data or add data to a file, and cannot look at the contents of the file or print the file. Similarly, write and read permission does not allow a user to execute a file or search a directory.
ACL bits	ACLs are used together with the permission bits in the FSP in order to control the access to z/OS UNIX files and directories by individual users (UIDs) and groups (GIDs).

Executable modules in the file system

You can have an executable module in HFS. To run a shell script or executable, a user must have read and execute permissions to the file. Use chmod to set the permissions.

For frequently used programs in the HFS, you can use the **`chmod`** command to set the sticky bit. This reduces I/O and improves performance. When the bit is set `on`, z/OS UNIX searches for the program in the user's STEPLIB, the LPALST, or the LNKLST concatenation.

The **`extattr`** command is used to set, reset and display extended attributes for files to allow executable files to be marked so they run APF authorized, as a program controlled executable, or not in a shared address space. The **`ls`** shell command has an option that displays these attributes:

`-E` Displays extended attributes for regular files:

`a`	Program runs APF authorized if linked AC=1
`p`	Program is considered program controlled
`s`	Program runs in a shared address space
`-`	Attribute not set

When the **`extattr`** attribute **`l`** is set (`+l`) on an executable program file, it will be loaded from the shared library region.

Path and pathname

The set of names required to specify a particular file in a hierarchy of directories is called the path to the file, which you specify as a pathname. Pathnames are used as arguments for commands.

An *absolute pathname* is a sequence that begins with a slash for the root, followed by one or more directory names separated with slashes, and ends with a directory name or a filename. The search for the file begins at the root and continues through the elements in the pathname until it gets to the final name. For example:

```
/u/smitha/projectb/plans/ldft
```

is the absolute pathname for 1dft, the first draft of the plans for a particular project that a user named Alice Smith (smitha) is working on.

Instead of using the absolute pathname with shell commands, you can specify a pathname as relative to the working directory; this is called the *relative pathname*. In most cases, a user can specify a particular file without having to use its absolute pathname. A relative pathname does not have a slash (/) at the beginning, and the search for the file begins in the working directory. For example, if Alice Smith is working in the directory projectb, she can specify the relative pathname for the file /u/smitha/projectb/plans/1dft as:

```
/plans/1dft
```

A pathname can be up to 1023 characters long, including all directory names, file names, and separating slashes. For pathnames and filenames, use characters from the POSIX portable character set. Using DBCS data in these names is not recommended; it may cause unpredictable results.

The system performs pathname resolution to resolve a pathname to a particular file in a file hierarchy. The system searches from element to element in a pathname in order to find the file.

Symbolic and external links with a sticky bit

DLLs, and all flavors of spawn() and exec(), follow the same processing as described below. Where it says exec(), it covers all forms of module loading.

External links

exec() does a stat() on the passed filename. stat() does the search, not exec(). If the filename is an external link, then stat() fails with a unique reason code which causes exec() to read the external link. If the external link name is a valid PDS member name (1-8 alphanumeric/special characters), then exec() will attempt to locate the module in the MVS search order. If it cannot be found, exec() fails.

The external link is normally used when you want to set the sticky bit `on` for a file name that is longer than 8 characters or contains characters unacceptable for a PDS member name.

Symbolic links

If the file name you specify is a symbolic link, and exec() sees the sticky bit `on`, then it will truncate any dot qualifiers. So, as long as the base file name is an acceptable PDS member name, the need to set up links in order to get exec() to go to the MVS search order should not be an issue.

For example, if you have a file named java.jll, when you put the sticky bit `on`, exec() will attempt to load JAVA. If exec() cannot find JAVA, it reverts to using the java.jll file in the file system.

The important thing to understand is that exec() never sees the name that the symbolic link resolves to, even though it can see the stat() data for the final file.

If you define /u/user1/name1 as a symbolic link to /u/user1/name2, and then invoke name1:

1. The shell will spawn name1.
2. spawn() will access the file for name1 unaware that there is a symbolic link already established. It will access the name2 file by its underlying vnode, not the name2 handle.
3. If the sticky bit is `on` for the name2 file, spawn() will do the MVS search for name1 (the only name it has to work with).

1.2.7 Address spaces

z/OS UNIX employs a number of z/OS address spaces, depending on the configuration features that are enabled.

OMVS

The OMVS address space runs a program that initializes the kernel. The STARTUP_PROC statement in the BPXPRMxx member of SYS1.PARMLIB specifies the name of the OMVS cataloged procedure. It is strongly recommended that this procedure name remain its default value of OMVS—changing it is likely to cause some impact with related functions such as TCP/IP.

BPXOINIT

The BPXOINIT address space runs the initialization process. BPXOINIT is also the jobname of the initialization process.

The BPXOINIT address space has two categories of functions:

1. It behaves as PID(1) of a typical UNIX system. This is the parent of /etc/rc, and it inherits orphaned children so that their processes get cleaned up using normal code in the kernel. This task is also the parent of any MVS address space that is *dubbed* and not created by fork() or spawn(). Therefore, TSO/E commands and batch jobs have a parent PID of 1.
2. Certain functions that the kernel performs need to be able to make normal kernel calls. This address space is used for these activities (for example, mmap() and user ID alias processing).

The STEPLIB DD statement is propagated from OMVS to BPXOINIT. Therefore, if there is a STEPLIB DD statement in the BPXOINIT procedure, it will not be used if a STEPLIB DD statement was specified in the OMVS procedure.

BPXAS

The BPXAS address spaces are those started by WLM when programs use the fork() or spawn() C function or callable services.

Colony address spaces

Physical file systems are sometimes initialized in an address space called a *colony* address space. You can think of these address spaces as extensions of the kernel address space. The NFS Client and DFS Client physical file systems must be set up in a colony address space because they need to use socket sessions to talk to their remote servers and this cannot be done from the kernel. You can choose to set up the TFS in a colony address space also. Some physical file systems cannot be initialized in colonies; for example, the INET or CINET sockets file systems and HFS.

1.2.8 Accessing z/OS UNIX

To access z/OS UNIX, the user must first have a valid UID and GID. Under z/OS, this information is stored in the OMVS segments of the user and group RACF (or functionally equivalent Security product) profiles. Information about the user's home directory and shell choice is also stored in these segments.

It is possible to access UNIX without personal OMVS segments defined, if the BPX.DEFAULT.USER facility has been defined. In this case, a user without personal OMVS segments inherits the OMVS segments of the default user.

Once a user has a valid UID and GID configuration, then choices to access z/OS UNIX include:

- rlogin or telnet

  rlogon and telnet are interfaces that heritage UNIX users will find most comfortable. Access should be via an ASCII terminal.

- The TSO OMVS command

  The TSO OMVS command provides a telnet-like interface, subject to the limitations of 3270 technology.

- The ISPF shell

  The ISPF shell is an interface that heritage MVS users will find most comfortable. It exploits the full-screen capabilities of ISPF.

- BPXBATCH

  BPXBATCH allows UNIX work to be executed from batch JCL.

1.2.9 What people like about z/OS UNIX

Many people prefer to use z/OS UNIX because of the following:

- It is a standard part of the z/OS operating system. Users who already have a z/OS operating system only need to customize z/OS UNIX to suit their needs.
- Accessing heritage MVS data is relatively easy using standard supplied interfaces. That means it is cost effective to do things such as "add a new face" (Web enable) to an old MVS application.
- Security is strong utilizing z/OS access control software such as RACF. Many tasks that may be impossible or difficult to control under other implementations of UNIX can be controlled under z/OS UNIX.
- Workload can be effectively managed, making resources available whenever they are needed.
- Almost infinite disk capacity is available using the virtual disk capabilities of DFSMS storage management. With automount, data not used for a chosen period of time can be migrated to cheaper media, but recalled seamlessly when needed.
- z/OS address space structure means a failing process cannot impact other processes.

1.2.10 What people don't like about z/OS UNIX

It is an EBCDIC implementation. Traditional UNIX solutions have been written for ASCII platforms, so there may be some porting issues on the z/OS platform. But this issue continues to diminish as new features are implemented, such as Enhanced ASCII functionality in z/OS V1R2. See "Improved application flexibility: Enhanced ASCII" on page 35.

1.3 z/OS UNIX System Services release history

In 1991, the US Federal Information Processing Standards (FIPS) Document 151 stated that MVS must incorporate support for popular UNIX interfaces. So began the challenge of including UNIX functionality into the MVS operating system. The first implementation was known as OpenEdition (or OE, or OMVS), then it became OS/390 UNIX System Services, and then finally z/OS UNIX System Services, as we know it today.

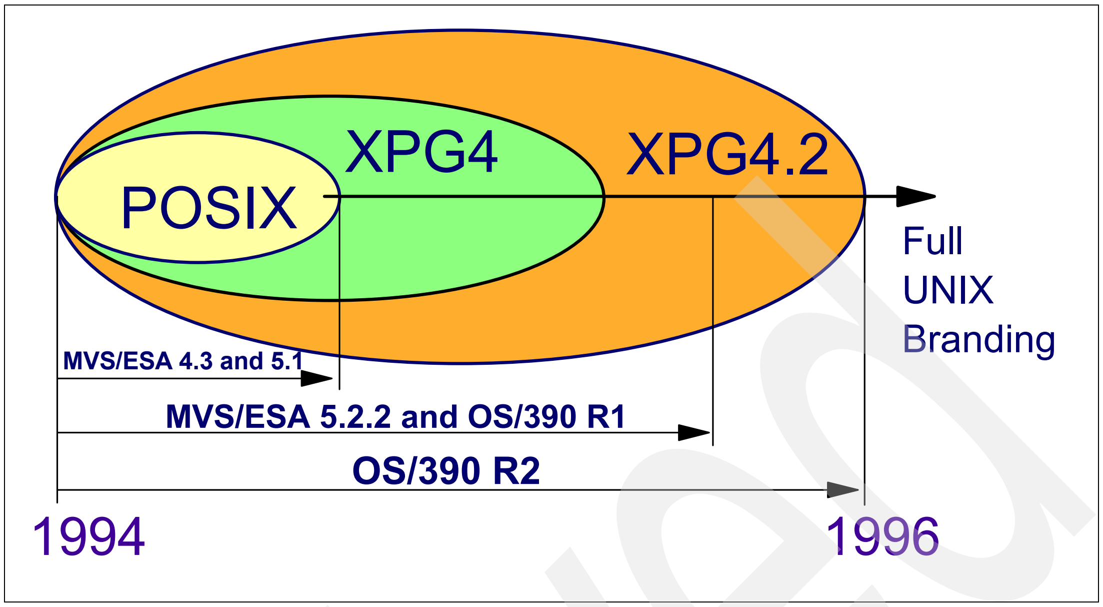


Figure 1-8 z/OS UNIX standards relationship

Figure 1-8 shows that z/OS UNIX has been UNIX branded since 1996.

1.3.1 MVS/ESA V4R3 - 1994

MVS/ESA™ V4R3 introduced:

- OpenEdition Services Support Feature
  - C language API, HFS files, Extended User Interface
  - Uses APPC/ASCH to supply address spaces
- OpenEdition Shell and Utilities
  - Mortice Kern System's InterOpen (TM) POSIX Shell
- OpenEdition dbx Debugger
- Standards:
  - ISO/IEC 9945-1:1990/IEEE (2) POSIX 1003.1-1990
  - A subset of IEEE POSIX 1003.1a
  - ISO/IEC DIS 9945-2:1992/IEEE POSIX 1003.2-1992
  - A subset of IEEE POSIX 1003.4a

1.3.2 MVS/ESA V5R1 - 1994

Changes for MVS/ESA V5R1 include:

- OpenEdition Services integrated in MVS
- NFS (TM Sun™ Microsystems™) Server
- AD/Cycle® C/370™ Language Support
- REXX under OpenEdition
- Integrated Sockets Support
- DCE Base Services
- DCE Application Support
- DCE User Data Privacy Feature
- TCP/IP V3.1

1.3.3 MVS/ESA V5R2M2 - 1995

Changes for MVS/ESA V5R2M2 include:

- DCE/DFS
- Extended Sockets - TCP/IP and SNA
- Standards:
  - POSIX, X/Open XPG4 Base Profile
  - A subset of the X/Open Single UNIX Specification

1.3.4 OS/390 V1R1 - 1996

Changes for OS/390 V1R1 include:

- Internet BonusPak - ICS

1.3.5 OS/390 V1R2 - 1996

Changes for OS/390 V1R2 include:

- Officially branded UNIX
- Internet BonusPak II - ICSS
- NFS Client
- UNIX-to-UNIX Copy Program (UUCP)
- Standards:
  - X/Open XPG4.2 UNIX specification
  - Improved performance

1.3.6 OS/390 V1R3 - 1997

Changes for OS/390 V1R3 include:

- OE System Services merged into BCP
- Permanent Kernel
- OE services always available
- OMVS started automatically during IPL
- Temporary File System (TFS) introduced
- Minimum & Full Function Modes
- S/390® Firewall - proxy, socks, DNS
- JAVA for S/390
- TCP/IP V3.2
- Improved performance

1.3.7 OS/390 V2R4 - 1997

Changes for OS/390 V2R4 include:

- Uses WLM to supply address spaces.
- APPC/ASCH no longer required.
- New extended attributes for executable files.
- Cached read-only files.
- Parallel Environment.
- Message Passing Interface (MPI).
- Domino® Go Webserver.
- BookManager BookServer.
- Enhanced security.
- Improved performance.

1.3.8 OS/390 V2R5 - 1998

Changes for OS/390 V2R5 include:

- C interface to WLM
- Print server
- Firewall technologies
- LDAP server
- Improved operations
- F BPXOINIT,SHUTDOWN=FORKINIT
- Improved performance

1.3.9 OS/390 V2R6 - 1998

Changes for OS/390 V2R6 include:

- Name changed from OpenEdition to OS/390 UNIX System Services
- Single HFS
- Improved performance

1.3.10 OS/390 V2R7 - 1999

Changes for OS/390 V2R7 include the BPXTIINT statement in the BPXPRMxx parmlib member, dynamic creation of character special files, inetd and rlogind daemons, man pages, Parallel Environment (new release), security enhancements for system programming and installation, and the UNIXMAP class.

BPXTIINT statement in BPXPRMxx parmlib member

References to BPXTIINT have been deleted because TCP/IP no longer runs at that level.

Dynamic creation of character special files

The character special files found under /dev are now created dynamically. Files such as /dev/fdxx and /dev/ptyzzzz are created based on the MAXFILEPROC and MAXPTYS setting in BPXPRMxx, respectively.

MAXFILEPROC is the upper bound on the VALUE of n in the name of both forms (/dev/fdn and /dev/fd/n). Both versions of n can be used in a single process.

For /dev/ptypnnnn files, MAXPTYS is the upper bound on the VALUE of nnnn.

*i*netd and rlogind daemons

The way the inetd and rlogind daemons are shipped has been changed. You will no longer find a load module in SYS1.LINKLIB called INETD and RLOGIND, respectively. If system programmers have created started procedures to start INETD via the START operator command using BPXBATCH, they will need to change those procedures.

man pages

Previously, the man pages were automatically enabled. Now BookManager is used and an optional task must be performed to enable them.

Parallel Environment (new release)

The second release of OS/390 UNIX System Services Parallel Environment is installed as part of your OS/390 Release system.Compared to OS/390 V2R5, new features for Parallel Environment in OS/390 V2R7 are:

- Parallel debuggers
- MPI-2 I/O (subset)
- MPMD support
- Multiple user thread support
- Enhanced WLM selection
- New utilities
- Online documentation (man pages)

Security enhancements for system programming and installation

System programmers who use SMP/E to install products and maintenance no longer require a UID=0 user ID to perform these actions. Changes have been made to SMP/E to check the BPX.SUPERUSER FACILITY class and to execute with superuser authority when the respective user IDs are permitted to this facility class. Similar changes have also been made to the TSO/E MOUNT and UNMOUNT utilities. Permission to the BPX.FACILITY class allows sufficient authority to execute these utilities.

UNIXMAP class

The RACF UNIXMAP class makes it quicker for the system to look up a user ID from a UID, or a group name from a GID.

Miscellaneous enhancements

- DFSMS 1.5 enhancement for HFS data sets.
- ServerPac install IPL eliminated.
- Mounts without security.
- New chroot command for testing fixes.
- Users can correct a bad home directory.

1.3.11 OS/390 V2R8 - 1999

Changes for OS/390 V2R8 include magic number support, OS/390 UNIX user limits, protected user ID, SETOMVS RESET operator command, and superuser granularity.

Magic number support

Most UNIX systems support a feature called the magic number (#!).The magic number is a numeric or string constant in a file that indicates the file name of the executable program to be run. When a script file starts with #!, the kernel invokes the specified file name as the script file interpreter.

For example, the HFS file /u/userid/util1 contains the following line at the beginning of the file:

```
#! /u/userid/othershell
```

When /u/userid/util1 is executed via either spawn or exec, the kernel recognizes the magic number and invokes /u/userid/othershell as the interpreter to process the /u/userid/util1 file. Prior to OS/390 V2R8, the OS/390 UNIX kernel did not support the magic number, so it treated it as a comment.

If the kernel cannot locate the program specified in the magic number, the shell attempts to process the file as a shell script. Make sure that any magic number specifies a valid file name or else eliminate the magic number.

OS/390 UNIX user limits

You can control the amount of resources that are consumed by individual OS/390 UNIX users. Resource limits for most OS/390 UNIX users are determined by the BPXPRMxx

PARMLIB member. Use the RACF ADDUSER and ALTUSER commands to specify and adjust the following limits, which are stored in the OMVS segment of the user profile: MAXCPUTIME, MAXASSIZE, MAXFILEPROC, MAXPROCUSER, MAXTHREADS, and MAXMMAPAREA. To shorten the names of the commands to be typed, RACF changed the names of those limits by putting MAX at the end. For example, the ADDUSER and ALTUSER commands support CPUTIMEMAX. This allows the abbreviation of CPU instead of MAXCPU.

Protected user ID

You can define RACF user IDs that cannot be used for activities such as logging on to TSO or signing on to CICS. As such, the user IDs that are defined for OS/390 UNIX daemons and other important subsystems or started tasks can be protected from being used for other purposes. They can also be protected from being revoked after several unsuccessful attempts to enter a password.

SETOMVS RESET operator command

You can dynamically add the FILESYSTYPE, NETWORK, and SUBFILESYSTYPE statements to the BPXPRMxx parmlib member without having to re-IPL. However, if you change the existing values, a re-IPL will be necessary.

Superuser granularity

You can reduce the number of people who have superuser authority at your installation by defining profiles in the UNIXPRIV class that grant RACF authorization for certain OS/390 UNIX privileges. Normally, these privileges are automatically defined for all users who are defined with OS/390 UNIX superuser authority. But you can use UNIXPRIV to grant certain superuser privileges, with a high degree of granularity, to users who do not have superuser authority.

1.3.12 OS/390 V2R9 - 2000

Changes for OS/390 V2R9 include support for shared HFS, an additional UNIX C shell, shared library support, new shell commands, and improvements in application enablement, systems management and debugging.

Support for shared HFS

Shared HFS allows read/write data to be shared transparently among participating systems across a sysplex. Before Release 9, you could have read/write access only to data in file systems mounted on your own system. The Shared HFS capability will become available some time in the Release 9 time frame.

A new chapter in *OS/390 UNIX System Services Planning,* SC28-1890, “OS/390 in a Sysplex” contains information about how to set up shared HFS in a sysplex, including how to create the sysplex root HFS data set and the system-specific HFS data set, and how to format an OS/390 UNIX Couple Data Set (CDS). It also describes how to update BPXPRMxx, move file systems, control security, and tune performance in a sysplex.

Specific changes in support of the shared HFS capability include:

- Changes to the BPXPRMxx parmlib member:
  - New BPXPRMxx parameters: SYSPLEX(YES|NO) and VERSION('nnnn'). SYSPLEX(YES) indicates whether a system running OS/390 UNIX System Services is to be initialized in a sysplex environment or operate in local mode. The first system entering the sysplex with SYSPLEX(YES) initializes a Couple Data Set (CDS), which controls shared HFS mounts. The CDS eventually contains sysplex-wide data about

the systems that use Cross-System Coupling Facility (XCF) services. The value of this parameter cannot be changed dynamically.

VERSION('nnnn') indicates the release or version of root HFS.

- New BPXPRMxx optional keywords on the ROOT and MOUNT parameters: SYSNAME(sysname) and AUTOMOVE|NOAUTOMOVE.

  On the ROOT parameter, SYSNAME(sysname) is the name of a system in a sysplex that was IPLed with SYSPLEX(YES). The AUTOMOVE|NOAUTOMOVE parameters indicate whether, if the specified root file system owner goes down, the root file system can be automatically moved to another system, which then becomes the owner for that root.

  On the MOUNT parameter, SYSNAME(sysname) specifies the particular system on which a mount should be performed.

- New information fields on the df-v shell command are displayed for file systems whose owner is part of a sysplex: the file system ID (owner/mounted file system server) and the file system ID issuing a quiesce request.
- New keywords on the TSO MOUNT command:
  - SYSNAME(system_name) specifies the specific system on which a mount should be performed (this system must be IPLed with SYSPLEX(YES)).
  - AUTOMOVE|NOAUTOMOVE keywords indicate whether the ownership of a file system is to be transferred if the file system's owner goes down.

A UNIX C shell

The new shell, tcsh, is an enhanced version of the Berkeley UNIX C-shell, which is commonly available on other UNIX systems. The tcsh shell is specifically designed to have a syntax similar to the C programming language, and has a number of commands designed especially for C programmers. It also has many general tools that can help any programmer. The tcsh shell commands are documented in OS/390 UNIX System Services Command Reference, and their usage is documented in OS/390 UNIX System Services User's Guide.

Support for WLM multi-system enclaves

Support for WLM multi-system enclaves simplifies tasks related to the management of multi-system transactions in a parallel sysplex. It provides the capability for managing and reporting on work requests that are executed in parallel on multiple MVS images as single entities.

The BPX1WLM (__wlm) callable service is enhanced to support the following new function codes:

- WLM_EXPORT_WORKUNIT
- WLM_UNDOEXPORT_WORKUNIT
- WLM_IMPORT_WORKUNIT
- WLM_UNDOIMPORT_WORKUNIT
- WLM_QUERY_ENCLAVECLASS
- WLM_CONNECT_EXPORTIMPORT

Shared library support

Support is added for shared object libraries. Shared object libraries contain subroutines that can be shared by multiple processes. Programs using shared libraries contain references to the library routines that are resolved by the loader at run time. Shared library modules reside in the shared library region in memory. System-shared object libraries are indicated by a new file attribute, and user-shared by a new file suffix, `.so`.

New shell commands

The UNIX fuser utility lists the process IDs of all processes running on the local system that have one or more named files opened.

New shell commands **`mount`**, **`chmount`**, **`unmount`**:

- **`mount`** mounts a file system or lists all mounts over a file system.
- **`chmount`** changes the mount attributes of a specified file system in a sysplex.
- **`unmount`** removes file systems from the file hierarchy.

Application enablement

Megabyte mapping services greatly reduce the excessive amounts of ESQA required to support servers that need to access more than 2 GB of storage. Two new callable services, BPX1MMI (__map_init) and BPX1MMS(__map_service), enable applications to manage an unlimited number of data blocks, each of which can hold some number of megabytes of data. They provide a fast way to connect to persistent memory for applications that need more shared memory than will fit in the address space.

You can now have access to system variables from the shell. The new sysvar command allows you to obtain substitution text for system variables that are defined in the IEASYMxx PARMLIB member or in the system IPL parameters. Shell scripts that run on multiple systems can use variable names such as SYSNAME.

Systems management features

- The **`D OMVS`** command has two new operands:
  - **`PFS`** displays information about the current configuration of the physical file system.
  - **`CINET`** displays routing information in effect for active transport providers using the Common INET Pre-Router.
- An alternative entry point, BPXBATSL, is provided for BPXBATCH. BPXBATSL, which is an alias for BPXBATCH and behaves exactly like BPXBATCH, except that it does not require the resetting of environment variables. This allows for more accurate measurement and analysis of the system.
- The pax utility now supports hard link and symbolic link names that are over 100 characters in length.
- This release provides a controlled way for a PFS to terminate and restart so that its kernel-resident load module can be deleted and reloaded for APAR service without a re-IPL. A superuser can recycle a kernel-resident PFS by using two calls to pfsctl (BPX1PCT), one to stop the PFS and one to start it.

Debugging improvements

- This release introduces a BPXPRMxx syntax checker. A new SETOMVS command parameter, SYNTAXCHECK=(xx), allows you to check the syntax of a BPXPRMxx parmlib member before you use it to IPL. Any syntax errors are sent to the hardcopy log.
- With JOBLOG to STDERR support, WTO messages normally targeted to the JES JESYSMSG file can be redirected to a joblog in the HFS with a new environment variable, _BPXK_JOBLOG. You can specify the joblog to receive messages. This helps in the diagnosis of system issues for UNIX applications.

- dbx now supports the long long data type. Programmers can debug C/C++ programs that use the long long and unsigned long long data types.
- dbx supports Language Environment debug events for read/write locks and shared mutexes (LE CEEEVDBG). A new **`readwritelock`** subcommand displays read/write lock information.

Changes for OS/390 C/C++

New C functions are added:

- ConnectExportImport()
- __cpl()
- ExportWorkUnit()
- __getuserid()
- ImportWorkUnit()
- __ipDomainName()
- __map_init()
- __map_service()
- __mount()
- QueryWorkUnitClassification()
- sigqueue()
- strtoll()
- strtoull()
- UnDoExportWorkUnit()
- UnDoImportWorkUnit()

The following compiler options and suboptions are new:

- The CHECKOUT option has a new suboption, CAST, which checks for the potential violation of ANSI type-based aliasing rules in explicit pointer type castings.
- COMPRESS suppresses the generation of function names in the function control block, thereby reducing the size of your application's load module.
- The DIGRAPH option is now supported for C as well as C++.
- IGNERRNO informs the compiler that your application is not using errno to return error conditions. This allows the compiler to explore additional optimization opportunities for certain library functions.
- INITAUTO directs the compiler to generate code to initialize automatic variables. Automatic variables require storage only while the function in which they are declared are active.
- PHASEID specifies that each compiler module (phase) is to issue an informational message as the phase begins execution. This helps you determine the maintenance level of each compiler component (phase).
- RECONST informs the compiler that the const qualifier is respected by the program. Variables defined with the const keyword will not be overridden by a casting operation.
- ROSTRING directs the compiler to place string literals into read-only memory, and not in the Writeable Static Area (WSA). This reduces the memory requirements for DLLs.
- STRICT_INDUCTION instructs the compiler to disable loop induction variable optimizations. These optimizations have the potential to alter the semantics of your program. Such optimizations can change the result of a program if truncation or sign

extension of a loop induction variable occurs as a result of variable overflow or wraparound. This option provides information to the compiler that enables it to explore additional opportunities for optimization.

- The TARGET option has been extended so that you can specify the OS/390 release for your program's object module that OS/390 C/C++ generates. This lets you generate code that is backward compatible with earlier levels of the operating system. You can compile and link an application on a higher level system, and run the application on a lower level system. You can also use the RTLIB suboption to inform the compiler whether a complete C run-time library is available. For example, use the NORTLIB suboption when building a System Programmer C (SPC) application.

The following #pragma directives are new:

- `leaves` specifies that a named function never returns to the instruction following the call to that function. This pragma provides information to the compiler that enables it to explore additional opportunities for optimization.
- `option_override` directs the compiler to optimize functions at different optimization levels from the one specified on the command line by the OPTIMIZE option. With this pragma directive, you can leave specified functions unoptimized, while optimizing the rest of your application. This eases the debugging effort of functions that are problematic under optimization, by allowing you to isolate those functions.
- `reachable` specifies that you can reach the instruction after a specified function from a point in the program other than the return statement in the named function. This pragma provides information to the compiler that enables it to explore additional opportunities for optimization.

1.3.13 OS/390 V2R10 - 2000

Changes for OS/390 V2R10 include support for C/C++ applications, RAS enhancements, performance improvements, and other miscellaneous enhancements.

Support for C/C++ applications

- Large file support

  Support is added to some of the utilities that perform file operations for large (2GB or larger) HFS files.
- Kernel support for Language Environment XPLINK

  OS/390 UNIX provides support for Language Environment XPLINK (eXtra Performance Linkage), which improves the execution performance and compile times of OS/390 applications written in C/C++.
- Shell and utilities support for new long long data types

  Long long support eases the task of porting programs that use 64-bit integers (such as JAVA Virtual Machine).
- dbx support of long long compiler symbolic and arithmetic

  dbx supports the debugging of C/C++ applications that include long long and unsigned long long data types.
- dbx support of XPLINK

  dbx supports the debugging of new code associated with XPLINK.

Reliability, availability, and serviceability enhancements

Diagnostics and serviceability of the OS/390 UNIX environment are improved with tools that identify problems in setup, enable you to gather better dumps, and improve the analysis of dumps. These enhancements include:

- Descriptions for shell and utilities messages.

  Descriptions have been added for over 150 Shell and Utilities messages.
- Shell script for removing old files

  The skulker shell script removes files over a certain age from user-specified directories, based on the date a file was last accessed.
- Zombie cleanup for the init process

  A mechanism is introduced to ensure that zombie processes are cleaned up on a regular basis for the init process.
- Enhanced program control support

  Enhanced program control support is provided for authorized address spaces, such as daemon and server address spaces, to enable better integrity for those address spaces and better problem determination information for programs that require program control. A new SAF service is used to better maintain program control and provide better problem determination information. Aids in the protection and problem determination for these address spaces.
- Debugging support for byte-range lock waits

  DISPLAY OMVS has a new operand, BRL, which displays thread-level information for any thread that is in a byte-range lock wait.
- Security enhancements to AF_UNIX PFS

  These enhancements allow an AF_UNIX datagram server to receive the identity of the sender of each message it receives, providing for better troubleshooting of data passed from the syslog daemon to the joblog.
- sysconf() performance enhancement

  The performance of sysconf(), a valuable tool that allows application programs to retrieve data from the system, is improved, and new flags are added to meet UNIX98 standards.

Performance improvements

- Enhanced reporter support

  This support allows more kernel-related data to be made available to report applications like RMF, improving the ability of the OS/390 UNIX platform to manage UNIX workloads.
- Kernel generic timeout service

  This new time-out function enhances the performance of Lotus® and other UNIX-based applications, greatly increasing the number of Notes clients that can be supported on a single server.
- Application notification of stack recycle

  Common Inet is enhanced to notify servers when a new transport provider stack is initialized, so that servers do not have to be manually recycled.
- Relative addressing exploitation

  The performance of heavily used kernel modules is improved through conversion to compiler/assembler relative addressing support, which reduces the size of the kernel modules in LPA.

- Spawn of OS/390 shell pipeline commands

  The performance of pipeline commands is improved by the replacement of fork and exec calls with spawn calls. In addition, spawn allows more sharing of processes within the parent address space, making more efficient use of system resources.

Other enhancements

- Message routing capability for the _console() service

  Routing and descriptor codes can be specified for messages issued with the _console() service. A DOM (delete operator message) capability is added to delete held messages from the console.

- New features for binary semaphores

  The UNDO feature is provided for binary semaphores, allowing them to be freed when they are not freed by the exiting process. The short semaphore feature allows semaphores to be held for very short intervals of time. Short-duration requesters can bypass the default first-in-first-out ordering of semaphore obtain requesters and cut to the front of the wait chain.

1.3.14 OS/390 V2R10 - 2000 Software Refresh

Changes for OS/390 V2R10 Software Refresh include support for C/C++ applications, RAS enhancements, performance improvements, and other miscellaneous enhancements.

Support for C/C++ applications

Support is added for 64-bit real addressing, which improves the performance and response time of applications that have very large memory and DASD storage demands, use Data in Memory, or need to access very large databases. New functions are added to the ptrace callable service (BPX1PTR) to support callers using the new general-purpose “high” registers.

Reliability, availability, and serviceability enhancements

Introductory paragraph:

- UNIX System Services Parmlib Limits Checking

  This support provides enhancements for monitoring and managing UNIX System Services parmlib limits, including:

  – Console messages that indicate the status of USS parmlib limits, allowing an installation to react quickly when limits are reaching critical levels.

  – A new BPXPRMxx parmlib statement, LIMMSG(NONE|SYSTEM|ALL), which controls the displaying of these console message. A new SETOMVS operand, PID=, which dynamically changes a parmlib limit for a process.

  – A new keyword, LIMITS, on the DISPLAY OMVS command, which displays information about USS parmlib limits and current system usage. With the PID= keyword, LIMITS displays information for an individual process. With the RESET keyword, LIMITS resets the high-water marks for system limits to 0.

- Sysplex CDS Repair Tool

  Corruption of the Couple Data Set (CDS) can prevent the USS file system from performing key sysplex operations. The Sysplex CDS Repair Tool, used under the direction of an IBM service representative, makes it possible to correct or isolate the scope of a defect in the CDS, reducing the need for a sysplex-wide IPL. The MODIFY operator command is enhanced to support these shared HFS diagnostic and repair functions.

- Support for MVS Dump Debugging

  Post-mortem analysis is now possible on MVS dumps. A new dbx command line option, -C, puts dbx in full source-level debug mode.

Performance improvements

- Fast **`pthread_quiesce`**

  A new **`pthread_quiesce`** interface, BPX1PQG, is introduced, which freezes or unfreezes a set of threads and returns state information for them, without requiring that signals be sent to the target threads. This improves the performance and reliability of applications, such as JAVA, that use **`pthread_quiesce`** or signalling to stop threads.

1.3.15 z/OS V1R1 - 2001

There were no new or changed z/OS UNIX System Services functions in z/OS V1R1. Changes in the documentation reflect only the change in product name, such as OS/390 UNIX System Services becoming z/OS UNIX System Services.

1.3.16 z/OS V1R2 - 2001

Changes for z/OS V1R2 include improved application flexibility, new tools for managing e-business, and greater ease of use.

Improved application flexibility: Enhanced ASCII

Enhanced ASCII functionality makes it easier to port internationalized applications developed on (or for) ASCII platforms to z/OS platforms, by providing conversion from ASCII to EBCDIC and from EBCDIC to ASCII.

Enhanced ASCII introduces:

- A file tagging mechanism, which allows programmers to tag files with a file attribute that describes the contents of the file. The file tag contains a Coded Character Set Identifier (CCSID) that identifies the character set of the text data within the file, and indicates whether the file is eligible for automatic conversion.
- Support for automatic data conversion between character sets when the CCSIDs of a program and a file it is reading or writing to are different.

Enhanced ASCII support applies only to z/OS UNIX files; it does not apply to MVS files, even if they can be accessed by z/OS UNIX. For more information about the limitations of Enhanced ASCII, see z/OS C/C++ Programming Guide.

Specific changes in support of Enhanced ASCII include:

- A new statement in the BPXPRMxx parmlib member, AUTOCVT(ON|OFF), globally enables and disables the automatic text conversion of I/O data between code sets. AUTOCVT can be turned on or off with the SETOMVS and SET OMVS operator commands. Automatic conversion can also be overridden by individual programs at a thread level using flags in the thread control block.
- The MOUNT statement in BPXPRMxx has a new keyword, TAG, which specifies whether files should be converted during reading and writing.
- The TSO/E MOUNT command has a new operand, TAG(NOTEXT|TEXT,ccsid) to support file tagging.
- Shell commands are added or changed to support file tagging and automatic file conversion:

- A new shell command, **chtag**, assigns, changes, and removes the file tag on existing files.
- The new **-T** option on the **cksum** shell command enables the automatic conversion of tagged files.
- The new **-B** option on the **cmp** shell command disables the automatic conversion of tagged files.
- The new **tag** option on the **automount** command specifies whether file tags for untagged files in the mounted file systems are to be implicitly set.
- New options on the following shell commands support file tagging and/or automatic file conversion:
  - cp
  - df
  - file
  - find
  - head
  - icnov
  - localedef
  - ls
  - mount
  - mv
  - od
  - pack
  - pax
  - strings
  - tail
  - tcsh
  - test

- Two new environment variables are added to support file tagging and automatic file conversion: BPXK_AUTOCVT enables the conversion of data between EBCDIC and ASCII code sets, and _BPXK_CCSIDS identifies an EBCDIC or ASCII pair of corresponding CCSIDs (only one pair is supported with this release: EBCDIC 1047 and ISO-8859-1).
- New shell variables (_TAG_REDIR_IN, _TAG_REDIR_OUT, and _TAG_REDIR_ERROR) control the conversion of untagged files. (See the descriptions of the **sh** and **tcsh** commands in *z/OS UNIX System Services Command Reference,* SA22-7802.)
- The BPX1FCT callable service controls the automatic conversion of file data with two new arguments. F_SETTAG sets the file tag, and F_CONTROL_CVT controls automatic file conversion.

New tools for managing e-business

- HFS control

  The new FACILITY class profile BPX.DAEMON.HFSCTL enforces program control for HFS programs only. When users are given permission to this profile, z/OS UNIX bypasses program control rules for programs loaded from MVS libraries, but enforces the rules for HFS programs.

- Soft shutdown for mounted file systems

  This enhancement allows file systems to be unmounted without the loss of data. A new keyword on the MODIFY operator command, SHUTDOWN=FILESYS, specifies that all active file systems on this system are to be unmounted and the data synched to disk. In a sysplex environment, AUTOMOVE(YES) file systems that are owned by this system are moved to another system.

- A new file system, zFS, which can be used in addition to HFS

  zServer File System (zFS) is a new file system for z/OS UNIX System Services that can improve performance for many applications, especially those that access very large sequential files.

Greater ease of use

- Support for the TCP/IP Services resolver enhancement

  The TCP/IP Services resolver enhancement provides common functionality across native MVS and z/OS UNIX environments. The key functions for the various z/OS resolver libraries, which are used by TCP/IP Services applications for name-to-address or address-to-name resolution, are consolidated into a new, single resolver component.

  – A new BPXPRMxx statement, RESOLVER_PROC, specifies the name of a cataloged procedure in SYS1.PROCLIB that will be used to start the resolver address space during z/OS UNIX initialization.

  – DISPLAY OMVS,O displays the RESOLVER_PROC specification.

  – Two new callable services, gethostbyname(BPX1GHN) and gethostbyaddr (BPX1GHA), provide access to the system resolver functions.

- Enhancement to *uname* utility for OS/390 to z/OS compatibility.

  The uname utility has a new option that allows for continued support of the name OS/390 within the uname() field.

1.3.17 z/OS V1R3 - 2002

Changes for z/OS V1R3 include Managed System Infrastructure for Setup (msys), improved system management features, a greater level of security for HFS files and directories with access control lists (ACLs), and numerous enhancements to ISHELL.

Managed System Infrastructure for Setup support

z/OS Managed System Infrastructure for Setup (msys for Setup) significantly reduces the complexity of setting up the z/OS UNIX environment. It uses a series of customization dialogs to help you establish the basic definitions and values used by the TFS and HFS file systems and set limits on z/OS UNIX system resources. Default settings are based on best practices and current experience. Each panel supplies extensive help.

Improved system management features

- New functions for the automount facility

  – System symbolics are supported.

  – New keywords on automount generic entries support the use of automount to allocate HFS data sets: allocany, allocuser, and lowercase.

  – A new flag option on the automount shell command, -q, displays the current automount policy.

- Sysplex mount table limit monitoring

  A new eventual action console message warns when the mount table limit in the Coupled Data Set (CDS) reaches critical limits of 85%, 90%, 95%, and 100%. Another new message is issued when the resource shortage has been relieved.

These messages are issued only when the installation has set up system limit messaging (with the LIMMSG= statement in BPXPRMxx).

- OMVS outage avoidance

  With this support you can recycle the OMVS address space and its associated workload without having to re-IPL mission-critical systems.

  - The F OMVS operator command has a new keyword, SHUTDOWN, that shuts down the entire z/OS UNIX system and all processes.
  - The output of the D OMVS operator command indicates which processes are registered as permanent or blocking.
  - A SIGTERM signal is sent to each eligible process to indicate that a system shutdown is imminent. Applications that use SIGTERM for other purposes can specify that a new signal, SIGDANGER, can be used as the initial indication of an imminent shutdown. This is done with the new environment variable, _BPXK_SIGDANGER.
  - The SHUTDOWN_REG parameter on the BPX1ENV callable service registers the caller for special treatment at OMVS shutdown time.
  - The BPX1SDD (set_dub_default) callable service has three new options: DUBJOBPERM, DUBABENDCALLS, and DUBNOJSTUNDUB, which handle the behavior of the calling task and its subtasks during a shutdown and restart of OMVS.

- Automatic removal of mounted file systems when a system leaves the sysplex

  You can specify that a file system is to be automatically unmounted when the system leaves the sysplex. This includes any file systems mounted on that file system.

  - The UNMOUNT keyword is added to the AUTOMOVE | NOAUTOMOVE keyword on the MOUNT statement in BPXPRMxx. When specified, it indicates that the file system should be unmounted whenever the system leaves the sysplex.
  - The SETOMVS operator command has a new UNMOUNT operand.
  - The MOUNT TSO/E command has a new UNMOUNT option.
  - The **`mount`** and **`chmount`** shell commands have new unmount options.
  - The output of the DISPLAY OMVS operator command reflects the new UNMOUNT option.
  - File system information displayed by the **`df -v`** shell command provides unmount information.
  - The _mount (BPX2MNT) callable service supports unmount with an unmount bit defined in the mnte control block.
  - The **`getmntent`** syscall command has a new variable for unmount requests, MNT_MODE_AUNMOUNT.

- Colony address spaces started outside of JES.

A new start parameter on the ASNAME keyword of the FILESYSTYPE statement of BPXPRMxx, SUB=MSTR, specifies that an address space is not to be started under JES. This allows you to recycle JES without affecting the DFS or NFS clients. APAR OW48709 is required for this support.

Access control to files and directories by individual UIDs and GIDs

Access control lists (ACLs) extend the security provided by permission bits, by allowing you to control access to files and directories by individual user (UID) and group (GID). Previously, HFS files were protected only with POSIX permission bits, which are contained within the File Security Packet (FSP) in the file system. You could only specify permissions for file owner (user), group owner, and everyone else. ACLs behave much like RACF profile access lists, but they are contained within the file system. The currently participating file systems are HFS and zFS.

Shell commands are added or modified to support ACLs:

- Two new shell commands, **`setfacl`** and **`getfacl`**, define and display ACLs.
- **`cp`** options **`-p`** and **`-Z`** preserve the ACLs of files and directories, and specify that error messages are not displayed when ACLs are being set on the target, respectively.
- **`df`** displays ACL information.
- **`find`**, **`test`**, and the **`test`**, **`[...]`**, and **`[[...]]`** reserved-word commands have new ACL primary operators.
- **`getconf`** displays ACL information.
- **`ls`** indicates the presence of ACLs.
- The **`mv`** option **`-Z`** specifies that error messages are not displayed when ACLs are being set on the target.
- **`pax`** has a new keyword, **`-o`**, which displays extended ACL data.
- **`tar`** has a new **`-L`** *type* option, which displays extended ACL entries.
- **`tcsh`** has new file inquiry operators to support ACLs.

Callable services are modified to support ACLs:

- The BPX1FPC (fpathconf) and BPX1PCF (pathconf) callable services support new pathname variables: _ACL and _ACL_ENTRIES_MAX.
- The BPX1IOC (w_ioctl) and BPX1PIO (w_pioctl) callable services accept two new commands: SetfACL and GetfACL.

REXX syscall commands are added or modified to support ACLs:

- New REXX syscall commands are added:
  - aclupdateentry
  - acldelete
  - acldeleteentry
  - aclfree
  - aclget
  - aclgetentry
  - aclinit
  - aclset
- New variables are added to **`stat`**, **`fstat`**, and **`lstat`** REXX syscall commands: ST_ACCESSACL, ST_DMODELACL, and ST_FMODELACL.
- New variables are added to the **`pathconf`** REXX syscall command: PC_ACL and PC_ACL_MAX.

ISHELL enhancements

Numerous enhancements have been made to ISHELL in response to customer requests:

- Many changes have been made to the directory list:
  - Most areas of the directory list are cursor sensitive. You can, for instance, click on a file name and get a panel showing the full path name for that file.
  - The directory list panel contains brief instructional information and an action bar specific to the directory list. These can be turned on or off.

  - The current directory path name was previously shown as selected, with dots, dot-dots, and symlinks. It is now fully resolved. In addition, the directory name is preceded with the effective UID of the process.
  - Sort options are extended beyond the file name, and a secondary sort column can be specified.
  - File names in the directory list can be displayed in different colors, based on selected criteria (such as file type, setuid or setgid bit on, sticky bit on, file marked as executable, etc.) Colors are specified with the colors command.
- The main panel shows the effective UID of the process, and it remembers the last path name that was entered.
- The **su** command entered on the command line allows a UID or user name to be specified. The **su** command from the pull-down on the action bar cannot switch to a UID.
- The **execute** command no longer executes the selected file. A panel is displayed that allows you to enter a command and select the method for command execution. The command can be executed directly (local spawn), as a shell command through a login shell (sh -Lc), or as a TSO command. The selected path is automatically inserted at the end of the command line by default. You can also use {} anywhere within the command, any number of times, and it will be replaced with the selected path name.
- Time stamps on all panels that contain time stamps display local time based on the TZ setting for that user. The time stamp format is changed to be consistent with the ISO 8601 standard (yyy-mm-dd hh:mm). The directory list can also be configured to display the last changed time for files.
- The two panels that allow you to create HFS file systems have two new optional fields, Volume and Unit. When either is specified, it is added to the allocation command that gets issued.
- ISHELL can now be run with the option -d, and ISHELL will not suppress ISPF severe dialog errors, but terminate. This should only be used at the direction of IBM support.
- The oedit shell utility and OEDIT TSO command have an -r xx option to set the record length to be edited for fixed length text files.
- The **oedit** and **obrowse** shell utilities now pass the effective user ID of the process to the TSO session. If the effective user ID does not match that of the TSO process, the OEDIT or OBROWSE TSO commands attempt to set the effective user ID of the TSO process to that of the shell command before loading the file.

1.3.18 z/OS V1R4 - 2002

Changes for z/OS V1R4 include support for Internet Protocol Version 6 (IPv6), improved UNIX security management, greater application flexibility with zFS and REXX enhancements, and new system management features.

z/OS UNIX IPv6 support

UNIX System Services offers CINET support for Internet Protocol Version 6 (IPv6). IPv6 increases the size of IP addresses from 32 bits to 128 bits. IPv6 can be added to, and is interoperable with, IPv4 systems. Local INET (BPXTLINT) is no longer supported.

You activate IPv6 on a system by adding a second NETWORK statement to the definition of the INET or CINET configuration, using AF_INET6 as the domain value. You can also add the second NETWORK statement dynamically with SETOMVS RESET=(), although the TCP/IP stacks have to be recycled in order for IPv6 to be activated.

Other changes to support IPv6:

- The CINET operand on the DISPLAY OMVS operator command displays 16-byte IP addresses, where appropriate, if IPv6 is in use.
- The inetd and rlogind daemons are enhanced to support IPv6 connections.
- The socket callable service (BPX1SOC) supports the use of AF_INET6 as the domain value to create IPv6 sockets.
- Three new callable services provide for protocol-independent name resolution services:
  - BPX1GAI (Get the IP address and information of a service name or location.)
  - BPX1FAI (Free Addr_Info structures)
  - BPX1GNI (Get the host name and service name from a socket address.)

For more information about IPv6, see the home page for Playground.Sun.Com, a server operated by the Internet Engineering group of Solaris™ Software, a division of Sun Microsystems, Inc.

UNIX security management enhancements

- UID/GID enhancements

  Enhancements to the way UIDs and GIDs can be assigned by RACF make managing UNIX identities for users and groups easier and less error prone. Administrators can:

  - Use the new RACF facility class profile, BPX.NEXT.USER FACILITY, to have UIDs and GIDs automatically assigned to new users. The AUTOUID and AUTOGID keywords of ADDUSER/ALTUSER allow RACF to automatically assign an unused UID or GID to a user or group.
  - Assign shared (non-unique) UIDs and GIDs to z/OS UNIX groups, or prevent them from being shared, using the SHARED keyword of ADDUSER, ALTUSER, ADDGROUP and ALTGROUP. The SHARED.IDS profile must be defined in the UNIXPRIV class.
  - Determine the user, or set of users, currently assigned a given UID, using the UID keyword of the SEARCH command.
  - Determine the group, or groups, currently assigned a given GID value, using the GID keyword of the SEARCH command. This provides an alternative to using the UNIXMAP class.
  - Use the FILE.GROUPOWNER.SETGID class profile in the UNIXPRIV class to specify that the group owner of a new HFS file is to come from the effective GID of the creating process. Previously, only the group owner of the parent directory could be the group owner of a new HFS file.
- Enhanced program security

  RACF provides enhanced program control checking for privileged z/OS UNIX programs that require a program-controlled environment. The BPX.MAINCHECK security profile allows files to be defined to RACF as trusted, or “MAIN”. HFS programs must be moved to an MVS library before they can be defined to MAIN.
- Sanction lists

  Sanction lists provide additional security for APF or program-controlled programs. You can compile a single list to contain the lists of pathnames and program names that are sanctioned by the installation for use by APF-authorized or program-controlled calling programs.

  - A new BPXPRMxx statement, AUTHPGMLIST, specifies that a sanction list is to be used and points to the sanction list HFS file. The value in AUTHPGMLIST is the pathname of the HFS file that contains the sanction list.

  - SETOMVS has a new keyword, AUTHPGMLIST, which specifies that a sanction list is to be used.
  - The output of DISPLAY OMVS contains the AUTHPGMLIST value.

- Transport Layer Security (TLS) certificate support

  The BPX1SEC callable service has a new function code, SECURITY_CERTAUTH#, that allows users to supply a digital certificate for authentication of a specified user ID. Once the authentication is provided, a setuid() can be used to change the MVS/UNIX identify to that of the specified user ID.

Application flexibility

- zFS enhancements

  With z/OS Distributed File Service zSeries File System (zFS), you can put multiple mountable file systems into a single data set, called an aggregate. You can display the names of aggregate file systems with:

  - The w_getmountent (BPX1GMN) file system interface
  - The DISPLAY OMVS,FILESYSTEM operator command
  - The MODIFY BPXOINIT,FILESYS=DISPLAY operator command
  - The **df** shell command
  - ISHELL file system attributes

  You can use the ISHELL to create HFS-compatible zFS file systems.

- REXX enhancements
  - Level-1 support for REXX functions that extend the REXX language in the z/OS UNIX environment. Some of these were previously available on the UNIX System Services Tools & Toys Web page.
  - Functions for standard REXX I/O:
    - charin()
    - charout()
    - linein()
    - lineout()
    - stream()
  - Functions for accessing common file services and environment variables:
    - bpxwunix()
    - chars()
    - chmod()
    - convd2e()
    - directory()
    - environment()
    - exists()
    - getpass()
    - lines()
    - outtrap()
    - procinfo()
    - rexxopt()
    - sleep()
    - submit()
    - syscalls()
  - BPXWDYN, which makes dynamic allocation and dynamic output services easily accessible to programs running outside of a TSO environment. (It also functions in a

TSO environment.) It supports data set allocation, unallocation, concatenation, and the addition and deletion of output descriptors.

  BPXWDYN is designed to be called from REXX, but it may be called from several other programming languages, including Assembler, C, and PL/I.

  – A TSO host command environment that permits a REXX program to run TSO/E commands.
  – Support for "immediate commands", TSO/E REXX commands that change characteristics that control the execution of an exec or program.

  For additional information, see *z/OS Using REXX and z/OS UNIX System Services,* SA22-7806.

- /dev enhancements

  The /dev/fd/n file is supported, and can be created dynamically.

System management features

- Enhanced **`pthread_quiesce`**

  Two new thread-scope signals, SIGTHSTOP and SIGTHCONT, allow individual threads to be stopped and resumed. These signals can only be used with the **`pthread_kill`** command.

- z/OS UNIX process start/end exits

  Applications can use four installation exit points to monitor the creation and termination of processes:

  – Pre-process initiation exit (BPX_PREPROC_INIT), which receives control immediately before the creation of a new z/OS UNIX process.
  – Post-process initiation exit (BPX_POSPROC_INIT), which receives control immediately after the creation of a new z/OS UNIX process.
  – Process image initiation exit (BPX_IMAGE_INIT), which receives control immediately before the initiation of a new z/OS UNIX process image.
  – Pre-process termination exit (BPX_PREPROC_TERM), which receives control immediately before the termination of a z/OS UNIX process.

- Automove system list

  You can compile an automove system list to indicate where specified file systems should and should not be moved when a system is taken out of a sysplex. Until now, file systems defined with AUTOMOVE=YES have been moved randomly.

  – The MOUNT statement of BPXPRMxx has a new keyword, AUTOMOVE, which indicates where the specified file systems should be moved when systems leave the sysplex.
  – The SETOMVS operator command has a new keyword, AUTOMOVE, which indicates where the specified file systems should be moved when systems leave the sysplex.
  – The DISPLAY OMVS,F operator command displays the list of file systems that will be moved.
  – A new variable for mount requests, MNTE_SYSLIST, is added to the **`getmntent`** and **`mount`** syscall commands.

- Distributed byte range lock manager (BRLM)

  This release allows for distributed, rather than centralized, BRLM. With distributed BRLM, each system in a sysplex is started with BRLM, and each BRLM maintains locks for files in files systems that are locally owned. When a remote sysplex member dies, many applications that lock locally mounted files are unaffected.

Conversion to distributed BRLM is enabled with a new parameter (DISTRBRLM) on the CDS format utility IXCL1DSU. In a future release, distributed BRLM will be the default.

- Signal during socket suspends

  Signal termination processing is enhanced to terminate threads in a TCP/IP socket suspend.

  (Currently, fastpath processing for TCP/IP socket calls causes TCP/IP to wait for the task during a kernel syscall. This places the task in a state in which signal delivery cannot be executed, and defers indefinitely any signal sent to the suspended thread. The only way to terminate the process when the thread is hung in this wait is with the operator's CANCEL command.)

1.3.19 z/OS V1R5 - 2004

Changes for z/OS V1R5 include:

- Support for Multilevel security (MLS)
- Extended functionality of some z/OS UNIX commands
- BPXPRMxx parmlib enhancements
- Changed operator and TSO/E commands
- Support for symlink symbolics

1.3.20 z/OS V1R6 - 2004

In z/OS V1R6 there are a number of enhancements that cover various services of the z/OS UNIX environment, as follows:

- Shared condition variables
- RAS improvements
- Spooled output constraint relief
- Automove system list wildcard support
- Increase the 64K per process file descriptor limit
- Automount enhancements
- Fork() accounting
- Superkill function
- Shell and utility enhancements
- BPXWPERM environment variable
- Mount utility enhancements
- USS REXX BPXWDYN enhancements
- Logical file system support of zFS
- Distributed BRLM enhancement

1.3.21 z/OS V1R7 - 2005

Changes for z/OS V1R7 include:

- Support for Latch Contention Analysis
- Support for mounting file systems with SET OMVS
- Ease of use ISHELLenhancements
- HFS to zFS migration tool BPXWH2Z

1.4 IBM exploitation of z/OS UNIX System Services

Table 1-2 lists products that use z/OS UNIX for customization and exploitation.

Table 1-2 Products using z/OS UNIX

Program Name	Program Number	Path in the HFS
CICS Transaction Server for z/OS	5697-E93	/usr/lpp/cicsts
DB2 OLAP Server™ for OS/390	5655-OLP	/usr/lpp/db2olap
IBM CICS Transaction Gateway	5724-D12	/usr/lpp/ctg
IBM Cloud 9 for Software Configuration and Library Manager for z/OS	5655-G93	/usr/lpp/Cloud9
IBM CM OnDemand For z/OS and OS/390	5655-H39	/usr/lpp/ars
IBM DB2 Universal Database™ Server for OS/390 and z/OS with National Language Versions	5675-DB2	/usr/lpp/db2/db2710 /usr/lpp/db2ext_07_01_00 /usr/lpp/db2tx
IBM Database 2™ Universal Database Server for OS/390 and z/OS Net.Data® with National Language Version	5675-DB2	/usr/lpp/netdata
IBM DB2 Net Search Extender for OS/390 and z/OS	5675-DB2	/usr/lpp/db2/db2nx
IBM DB2 Warehouse Manager for z/OS and OS/390	5655-H34	/usr/lpp/DWC
IBM DB2 Warehouse Manager Sourcing Agent for OS/390	5655-F36	/usr/lpp/DWC
IBM Developer Kit for OS/390, Java™(TM) 2 Technology Edition	5655-D35	/usr/lpp/java
IBM Electronic Service Agent™ for IBM zSeries and IBM S/390	5655-F17	/usr/lpp/esa
IBM Enterprise COBOL for z/OS and OS/390	5655-G53	/usr/lpp/cobol
IBM DB2 Intelligent Miner™ for Data for OS/390	5655-IM3	/usr/lpp/IMiner
IBM Enterprise PL/I for z/OS and OS/390	5655-H31	/usr/lpp/pli
IBM IMS Connect for z/OS	5655-E51	/usr/lpp/imsico
IBM OS/390 Foreign File System	5639-I44	/usr/lpp/ffsserver
IBM Tivoli® Data Protection for Lotus Domino, S/390 Edition	5697-ILD	/usr/lpp/Tivoli/tsm/client/domino
IBM Tivoli Web Access for Information Management	5698-WAI	/usr/lpp/InfoMan/web
IBM WebSphere® Host On-Demand	5733-A59	/usr/lpp/HOD
IBM WebSphere Studio Asset Analyzer for Multiplatforms	5655-I49	/usr/lpp/dmh
Information Management System Transaction and Database Servers	5655-B01	/usr/lpp/ims/imsjava71
Tivoli Distributed Monitoring for OS/390	5697-F05	/usr/lpp/Tivoli
Tivoli Distributed Monitoring Agent for OS/390	5698-EMN	/usr/lpp/Tivoli
Tivoli Information Management for z/OS	5697-SD9	/usr/lpp/InfoMan
Tivoli Management Framework for OS/390 Server and Gateway	5697-D10	/usr/lpp/Tivoli

Program Name	Program Number	Path in the HFS
Tivoli Management Framework for OS/390 Framework Endpoint	5697-D10	/usr/lpp/Tivoli
Tivoli NetView for z/OS	5697-ENV	/usr/lpp/netview
Tivoli NetView Performance Monitor	5655-043	/usr/lpp/NetviewPM
Tivoli Storage Manager, S/390 Edition Application Program Interface	5697-ISM	/usr/lpp/Tivoli/tsm/client/api
Tivoli Storage Manager, S/390 Edition Backup-Archive Client	5697-ISM	/usr/lpp/Tivoli/tsm/client/ba
Tivoli Workload Scheduler for z/OS	5697-WSZ	/usr/lpp/TWS
VisualAge® for Java, Enterprise Edition for OS/390	5655-JAV	/usr/lpp/hpj
VisualAge Generator Server for MVS	5648-B02	/usr/lpp/vgwgs31
WebSphere Application Server for z/OS and OS/390	5655-F31	/usr/lpp/WebSphere
WebSphere Commerce Suite Pro Edition for OS/390	5697-G05	/usr/lpp/CommerceSuite /usr/lpp/PaymentManager
WebSphere MQ for z/OS	5655-F10	/usr/lpp/internet/server_root/csq /usr/lpp/mqm
WebSphere MQ Integrator for z/OS	5655-G97	/usr/lpp/wmqi
WebSphere MQ Integrator for z/OS New Era of Networks Feature	5655-G97	/usr/lpp/wmqi
XML Toolkit for z/OS and OS/390	5655-J51	/usr/lpp/ixm

1.5 Additional material for this Redbook

Some procedures and samples referenced or listed in this Redbook, together with further documents and procedures, are available as additional material related to this book on the Internet.

Attention: To access these softcopy files on the Internet, point your browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG247035/`

Note: SG must be uppercase.

Alternatively, you can go to:

`http://www.redbooks.ibm.com`

and select **Redbooks Online**, and then **Additional Materials**.

2

Installation

This chapter describes sourcing and installing z/OS UNIX System Services. It discusses various z/OS UNIX configurations ranging from small and simple to large and complex.

2.1 Introduction

z/OS UNIX System Services is a *base element* and *exclusive feature* of the z/OS operating system. This chapter does not discuss installation of the z/OS operating system, but it may highlight z/OS installation information where it specifically relates to z/OS UNIX.

Note: Starting with the September 2004 z/OS release (V1R5), IBM intends to deliver z/OS and z/OS.e releases annually.

Since z/OS UNIX is part of the z/OS operating system, the first prerequisite for using z/OS UNIX is to have a z/OS system available. Such a system should have the general z/OS installation work completed, and be either already IPLed, or ready to be IPLed. It is assumed that a root file system already exists and is populated with the elements that were created as part of the z/OS installation process.

The topics that follow only focus on the activities necessary to activate (meaning *enable*, or *implement*—terminology for separation from the SMP/E *installation* process that is outside the scope of this book) the z/OS UNIX components within the z/OS operating system. The sequence of topics is:

- Activating z/OS UNIX in *minimum mode*. Minimum mode is activated if there is no requirement to exploit z/OS UNIX. This might be suitable for a system that only runs traditional MVS workloads. Note that if you want to use any z/OS UNIX service, TCP/IP, or other functions that require the kernel services, then full function mode is required. This is also the case if service is to be applied to the HFS.
- Activating z/OS UNIX in *full function mode*. Full function mode is activated if there is a requirement to exploit z/OS UNIX. This might be suitable for a system that runs UNIX workloads possibly in addition to traditional MVS workloads.

The implementation of UNIX on the z/OS operating system offers a huge variety of customization options. These may be to enhance security, make data available to multiple systems in a SYSPLEX, improve performance, or any number of other choices. These issues are dealt with in separate chapters from the minimum mode and full function mode topics, so that the basic activation process can be understood in simple terms, without having to deal with the burden of complex considerations at every step.

Notes:

- References to SYS1.PROCLIB are intended to indicate a system procedure data set (library) from where started tasks (STCs) may be initiated (with SUB=MSTR). If another data set is more appropriate for this purpose, then its name may be substituted for SYS1.PROCLIB.
- References to SYS1.PARMLIB are intended to indicate a system parameter data set (library) where system parameters may be found by z/OS. If another data set is more appropriate for this purpose (and defined in the PARMLIB statement of LOADxx), then its name may be substituted for SYS1.PARMLIB.
- The activation processes documented in 2.2, “Activating z/OS UNIX in minimum mode” on page 49 and 2.3, “Activating z/OS UNIX in full function mode” on page 53 are for a simple single-system configuration. This is an attempt to convey understanding of the activation process, without concern for the complexities that could be encountered depending on the z/OS UNIX options chosen. Later chapters attempt to address the full range of possible z/OS UNIX configurations.

2.2 Activating z/OS UNIX in minimum mode

In minimum mode, the kernel cannot support some functions, such as the z/OS shell and TCP/IP. When the system is IPLed, the kernel services start up in minimum mode and use the default values for all BPXPRMxx PARMLIB statements. See *z/OS MVS Initialization and Tuning Reference,* SA22-7592 for information about the default values.

In minimum mode, a temporary file system named SYSROOT is used as the root file system. It is initialized and primed with a minimum set of files and directories. Any data written to this file system is not written to DASD. The temporary file system does not have any executables; that is, the shell will not be available.

The steps required to activate z/OS UNIX in minimum mode within a simple single-system configuration are summarized in Table 2-1, then explained in more detail afterwards.

Table 2-1 Activating z/OS UNIX in minimum mode

Step	Activity	Reference
1	Create the OMVS procedure	Page 49
2	Create the BPXOINIT procedure	Page 49
3	Establish security	Page 50
4	Customize IEASYSxx	Page 51
5	IPL	Page 52

2.2.1 Step 1 - Create the OMVS procedure

The OMVS cataloged procedure runs a program that initializes the kernel.

Create a JCL procedure for OMVS and place it in SYS1.PROCLIB (if it is not already there as a result of the z/OS installation process). The content of this started task (STC) member is shown in Figure 2-1.

```
VIEW       SYS1.PROCLIB(OMVS) - 01.01                      Columns 00001 00072
Command ===>                                                   Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 //OMVS PROC
000002 //OMVS EXEC     PGM=BPXINIT,REGION=0K,TIME=NOLIMIT
****** **************************** Bottom of Data ****************************
```

Figure 2-1 SYS1.PROCLIB(OMVS)

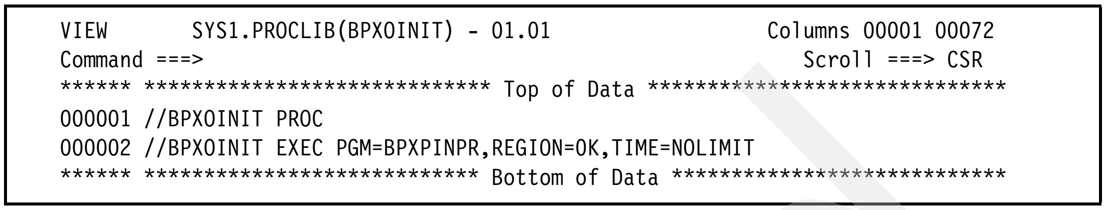
Note that program BPXINIT resides in SYS1.LINKLIB, so it will be found by default; no STEPLIB is required.

Failure to create the OMVS proc results in a JCL error during IPL, rendering z/OS UNIX inoperative.

2.2.2 Step 2 - Create the BPXOINIT procedure

BPXOINIT is the started procedure that runs the initialization process.

Create a JCL procedure for BPXOINIT and place it in SYS1.PROCLIB (if it is not already there as a result of the z/OS installation process). The content of this started task (STC) member is shown in Figure 2-2.



```
VIEW       SYS1.PROCLIB(BPXOINIT) - 01.01                Columns 00001 00072
Command ===>                                                Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 //BPXOINIT PROC
000002 //BPXOINIT EXEC PGM=BPXPINPR,REGION=0K,TIME=NOLIMIT
****** **************************** Bottom of Data ****************************
```

Figure 2-2 SYS1.PROCLIB(BPXOINIT)

Note that program BPXPINPR resides in SYS1.LINKLIB, so it will be found by default; no STEPLIB is required.

Failure to create the BPXOINIT proc results in a JCL error during IPL, rendering z/OS UNIX inoperative. The following SYSLOG message is issued:

```
BPXP006E OMVS IS CREATING THE BPXOINIT ADDRESS SPACE
```

2.2.3 Step 3 - Establish security

To enable the OMVS and BPXOINIT procedures to execute, some basic security requirements must be met. For RACF, both of these procedures require:

- An MVS group ID associated with a UNIX GID.
  - The MVS group ID is shared by both the OMVS and BPXOINIT procedures, and can be any valid group ID according to local naming conventions. For the purposes of this book, a group ID of OMVSGRP is used.
  - The UNIX GID is shared by both the OMVS and BPXOINIT procedures, and can be any valid GID according to local naming conventions. RACF allows for GIDs within the range of 0-2,147,483,647 (however, the pax and tar utilities cannot handle values above 16,777,216). For the purposes of this book, a GID of 1 is used.
- An MVS user ID associated with a UNIX UID.
  - The MVS user ID is shared by both the OMVS and BPXOINIT procedures, and can be any valid user ID according to local naming conventions. For the purposes of this book, a user ID of OMVSKERN is used.
  - The UNIX UID is shared by both the OMVS and BPXOINIT procedures, and must be 0 (zero = superuser).
- STARTED class profiles or ICHRIN03 definitions.
  - STARTED class profiles associate a user ID and group ID with a started task.
  - ICHRIN03 associates a user ID and group ID with a started task if there is no STARTED class profile.

Systems that have an alternative security product (such as ACF2 or Top Secret) have different security requirements. Check with the product vendor to determine their support for z/OS UNIX System Services.

Defining group ID OMVSGRP

A TSO user with RACF SPECIAL authority should enter a command similar to the following example:

```
ADDGROUP OMVSGRP OMVS(GID(1))
```

This creates an MVS group ID called OMVSGRP and associates a UNIX GID of 1 to it.

Defining user ID OMVSKERN

A TSO user with RACF SPECIAL authority should enter a command similar to the following example:

```
ADDUSER OMVSKERN DFLTGRP(OMVSGRP)
    OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
    NOPASSWORD
```

This creates an MVS user ID called OMVSKERN, and associates a UNIX UID of 0 to it (note that UID=0 is the UNIX superuser UID, with special administrative powers that other UIDs do not have). The default RACF group is OMVSGRP, the home path is the root directory (/), and the default shell is /bin/sh.

Associating OMVSKERN/OMVSGRP with the procedures

A TSO user with RACF SPECIAL authority should enter commands similar to the following example:

```
SETROPTS GENERIC(STARTED)
RDEFINE STARTED OMVS.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(YES))
RDEFINE STARTED BPXOINIT.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(NO))
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
SETROPTS RACLIST(STARTED) REFRESH
```

If STARTED class profiles cannot be used, then ICHRIN03 should be changed similar to the following example:

```
DC CL8'OMVS'     PROCEDURE NAME
DC CL8'OMVSKERN' USERID (ANY RACF-DEFINED USER ID)
DC CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC XL1'40'       TRUSTED
DC XL7'00'       RESERVED

DC CL8'BPXOINIT' PROCEDURE NAME
DC CL8'OMVSKERN' USERID (ANY RACF-DEFINED USER ID)
DC CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC XL1'00'       NOT TRUSTED
DC XL7'00'       RESERVED
```

This associates user ID OMVSKERN and group ID OMVSGRP with started task (STC) procedures OMVS and BPXOINIT.

2.2.4 Step 4 - Customize IEASYSxx

Define the following statement in the IEASYSxx member of SYS1.PARMLIB:

```
OMVS=DEFAULT
```

Specifying OMVS=DEFAULT is actually the same as omitting the OMVS statement from IEASYSxx. Defining this statement in IEASYSxx is recommended so that the intent of the system configuration is clear to other support people.

2.2.5 Step 5 - IPL

IPL the system so that the modified IEASYSxx member is implemented.

After IPL, you will observe that the OMVS and BPXOINIT address spaces start, and the following SYSLOG message is issued:

```
BPXI004I OMVS INITIALIZATION COMPLETE
```

If you check the status of z/OS UNIX using the D OMVS console command, in SYSLOG you can expect to see a response similar to what is shown in Figure 2-3.

```
D OMVS
BPXO042I 02.36.02 DISPLAY OMVS 205
OMVS     000D ACTIVE          DEFAULT
```

Figure 2-3 Successful minimum mode configuration - status

If you check the status of z/OS UNIX address spaces using the D OMVS,A=ALL console command, in SYSLOG you can expect to see a response similar to Figure 2-4.

```
D OMVS,A=ALL
BPXO040I 02.52.00 DISPLAY OMVS 211
OMVS     000D ACTIVE          DEFAULT
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN BPXOINIT 0020          1          0 MR---- 02.28.36       .04
  LATCHWAITPID=        0 CMD=BPXPINPR
  SERVER=Init Process                   AF=    0 MF=00000 TYPE=FILE
RMFTASK  RMFGAT   0023          2          1 1R---P 02.31.40      4.17
  LATCHWAITPID=        0 CMD=ERB3GMFC
```

Figure 2-4 Successful minimum mode configuration - address spaces

If you check the status of z/OS UNIX file systems using the D OMVS,F console command, in SYSLOG you can expect to see a response similar to Figure 2-5.

```
D OMVS,F
BPXO045I 02.48.31 DISPLAY OMVS 207
OMVS     000D ACTIVE          DEFAULT
TYPENAME   DEVICE ----------STATUS----------- MODE
BPXTFS          3 ACTIVE                      RDWR
  NAME=ROOT
  PATH=/
```

Figure 2-5 Successful minimum mode configuration - file system

This means that z/OS has used a temporary file system for the ROOT (BPXTFS = temporary file system). The content of this temporary file system is shown in Figure 2-6 on page 53.

```
Directory List

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=0   /
  Type  Perm  ------Size  Filename                               Row 1 of 6
_ Dir    777        4000  .
_ Dir    777        4000  ..
_ Dir    755        4000  bin
_ Dir    755        4000  dev
_ Dir    755        4000  etc
_ Dir    777        4000  tmp
```

Figure 2-6   Successful minimum mode configuration - file system content

All of these directories are empty.

2.3 Activating z/OS UNIX in full function mode

The steps required to activate z/OS UNIX in full function mode within a simple single-system configuration are summarized in Table 2-2, then explained in more detail afterwards.

Table 2-2   Activating z/OS UNIX in full function mode

Step	Activity	Reference
1	Create the OMVS procedure	Page 54
2	Create the BPXOINIT procedure	Page 54
3	Create the BPXAS procedure	Page 55
4	Establish security	Page 55
5	Create HFS data sets	Page 57
6	Customize BPXPRMxx	Page 58
7	Customize ALLOCxx	Page 67
8	Customize COFVLFxx	Page 67
9	Customize CTnBPXxx	Page 68
10	Customize IEADMR00	Page 68
11	Customize SMFPRMxx	Page 68
12	Customize IEASYSxx	Page 69
13	IPL	Page 69
14	Customize /etc/init.options	Page 72
15	Customize /etc/rc	Page 76
16	Customize /etc/profile	Page 78

2.3.1 Step 1 - Create the OMVS procedure

The OMVS cataloged procedure runs a program that initializes the kernel. The STARTUP_PROC statement in the BPXPRMxx PARMLIB member specifies the OMVS cataloged procedure. The default name is OMVS. Though not recommended, you can replace the OMVS procedure with a procedure that has a different name. If you use a started procedure other than OMVS, the replacement started procedure must also be a single jobstep procedure that invokes the BPXINIT program (EXEC PGM=BPXINIT). If it invokes any other program, OMVS initialization will fail.

Create a JCL procedure for OMVS and place it in SYS1.PROCLIB (if it is not already there as a result of the z/OS installation process). The content of this started task (STC) member is shown in Figure 2-7.

```
VIEW       SYS1.PROCLIB(OMVS) - 01.01                      Columns 00001 00072
Command ===>                                                  Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 //OMVS PROC
000002 //OMVS EXEC     PGM=BPXINIT,REGION=0K,TIME=NOLIMIT
****** **************************** Bottom of Data ****************************
```

Figure 2-7 SYS1.PROCLIB(OMVS)

Note: Program BPXINIT resides in SYS1.LINKLIB, so it will be found by default and no STEPLIB is required.

Failure to create the OMVS procedure results in a JCL error during IPL, rendering z/OS UNIX inoperative.

2.3.2 Step 2 - Create the BPXOINIT procedure

BPXOINIT is the started procedure that runs the initialization process.

Create a JCL procedure for BPXOINIT and place it in SYS1.PROCLIB (if it is not already there as a result of the z/OS installation process). The content of this started task (STC) member is shown in Figure 2-8.

```
VIEW       SYS1.PROCLIB(BPXOINIT) - 01.01                  Columns 00001 00072
Command ===>                                                  Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 //BPXOINIT PROC
000002 //BPXOINIT EXEC PGM=BPXPINPR,REGION=0K,TIME=NOLIMIT
****** **************************** Bottom of Data ****************************
```

Figure 2-8 SYS1.PROCLIB(BPXOINIT)

Note: Program BPXPINPR resides in SYS1.LINKLIB, so it will be found by default; no STEPLIB is required.

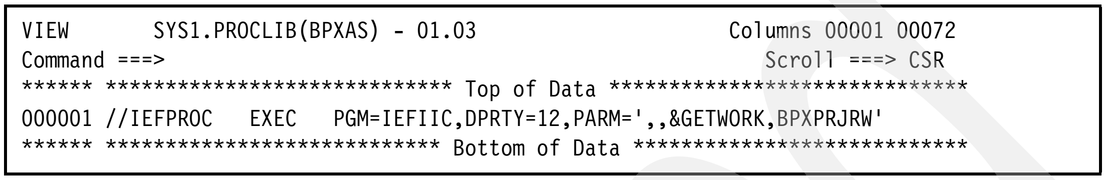
Failure to create the BPXOINIT procedure results in a JCL error during IPL, rendering z/OS UNIX inoperative. The following SYSLOG message is issued:

```
BPXP006E OMVS IS CREATING THE BPXOINIT ADDRESS SPACE
```

2.3.3 Step 3 - Create the BPXAS procedure

When programs issue fork() or spawn(), the BPXAS procedure is used to provide a new address space. For a fork(), the system copies one process, called the parent process, into a new process, called the child process. The forked address space is provided by WLM.

Create a JCL procedure for BPXAS and place it in SYS1.PROCLIB (if it is not already there as a result of the z/OS installation process). The content of this started task (STC) member is shown in Figure 2-8.



```
VIEW       SYS1.PROCLIB(BPXAS) - 01.03                  Columns 00001 00072
Command ===>                                               Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 //IEFPROC   EXEC   PGM=IEFIIC,DPRTY=12,PARM=',,&GETWORK,BPXPRJRW'
****** **************************** Bottom of Data ****************************
```

Figure 2-9 SYS1.PROCLIB(BPXAS)

Note: Program IEFIIC resides in SYS1.LPALIB, so it will be found by default; no STEPLIB is required.

2.3.4 Step 4 - Establish security

To enable the OMVS, BPXOINIT, and BPXAS procedures to execute, some basic security requirements must be met. For RACF, these procedures require:

- An MVS group ID associated with a UNIX GID.
  - The MVS group ID is shared by the OMVS, BPXOINIT and BPXAS procedures, and can be any valid group ID according to local naming conventions. For the purposes of this book, a group ID of OMVSGRP is used.
  - The UNIX GID is shared by the OMVS, BPXOINIT and BPXAS procedures, and can be any valid GID according to local naming conventions. RACF allows for GIDs within the range of 0-2,147,483,647 (however, the pax and tar utilities cannot handle values above 16,777,216). For the purposes of this book, a GID of 1 is used.
- An MVS user ID associated with a UNIX UID.
  - The MVS user ID is shared by the OMVS, BPXOINIT and BPXAS procedures, and can be any valid user ID according to local naming conventions. For the purposes of this book, a user ID of OMVSKERN is used.
  - The UNIX UID is shared by the OMVS, BPXOINIT and BPXAS procedures, and must be 0 (zero = superuser).
- STARTED class profiles or ICHRIN03 definitions.
  - STARTED class profiles associate a user ID and group ID with a started task.
  - ICHRIN03 associates a user ID and group ID with a started task if there is no STARTED class profile.

Systems that have an alternative security product (such as ACF2 or Top Secret) will have different security requirements. Check with the product vendor to ascertain their support for z/OS UNIX System Services.

Defining group ID TTY

Certain shell commands, such as **mesg**, **talk**, and **write** require pseudo terminals to have a group name of TTY. When a user logs in, or issues the OMVS command from TSO/E, the group name associated with these terminals is changed to TTY.

A TSO user with RACF SPECIAL authority should enter a command similar to the following example:

```
ADDGROUP TTY OMVS(GID(0))
```

This creates an MVS group ID called TTY, and associates a UNIX GID of 0 to it. Unlike UID=0 (superuser), GID=0 has no special properties.

Defining group ID OMVSGRP

A TSO user with RACF SPECIAL authority should enter a command similar to the following example:

```
ADDGROUP OMVSGRP OMVS(GID(1))
```

This creates an MVS group ID called OMVSGRP, and associates a UNIX GID of 1 to it.

Defining user ID OMVSKERN

A TSO user with RACF SPECIAL authority should enter a command similar to the following example:

```
ADDUSER OMVSKERN DFLTGRP(OMVSGRP)
   OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
   NOPASSWORD
```

This creates an MVS user ID called OMVSKERN, and associates a UNIX UID of 0 to it (note that UID=0 is the UNIX superuser UID, with special administrative powers that other UIDs do not have). The default RACF group is OMVSGRP, the home path is the root directory (/), and the default shell is /bin/sh.

Associating OMVSKERN/OMVSGRP with the procedures

A TSO user with RACF SPECIAL authority should enter commands similar to the following example:

```
SETROPTS GENERIC(STARTED)
RDEFINE  STARTED OMVS.*     STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(YES))
RDEFINE  STARTED BPXOINIT.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(NO))
RDEFINE  STARTED BPXAS.*    STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(NO))
SETROPTS CLASSACT(STARTED) RACLIST(STARTED)
SETROPTS RACLIST(STARTED) REFRESH
```

If STARTED class profiles cannot be used, then ICHRIN03 should be changed similar to the following example:

```
DC CL8'OMVS'     PROCEDURE NAME
DC CL8'OMVSKERN' USERID (ANY RACF-DEFINED USER ID)
DC CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC XL1'40'       TRUSTED
DC XL7'00'       RESERVED

DC CL8'BPXOINIT' PROCEDURE NAME
DC CL8'OMVSKERN' USERID (ANY RACF-DEFINED USER ID)
DC CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC XL1'00'       NOT TRUSTED
DC XL7'00'       RESERVED
DC CL8'BPXAS'    PROCEDURE NAME
DC CL8'OMVSKERN' USERID (ANY RACF-DEFINED USER ID)
DC CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC XL1'00'       NOT TRUSTED
DC XL7'00'       RESERVED
```

This associates user ID OMVSKERN and group ID OMVSGRP with started task (STC) procedures OMVS, BPXOINIT, and BPXAS.

Note that information about the security requirements of BPXAS is inconclusive. As a result, it is typical to define BPXAS with the same attributes as BPXOINIT.

Adding USER/GROUP OMVS segments for a TSO user

To complete the activation of z/OS UNIX in full function mode, a TSO user ID needs to be available with the correct authorities to perform z/OS UNIX administration work. This could be any existing user but for the purposes of this book we will use the commonly known IBMUSER user ID. This may be substituted by any appropriate user ID.

A TSO user with RACF SPECIAL authority should enter commands similar to the following example:

```
CONNECT (IBMUSER) GROUP(OMVSGRP)
ALTUSER IBMUSER DFLTGRP(OMVSGRP)
   OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
```

This alters existing TSO user ID IBMUSER to have UNIX UID(0) associated to it (note that UID=0 is the UNIX superuser UID, with special administrative powers that other UIDs do not have). The default RACF group is OMVSGRP, the home path is the root directory (/), and the default shell is /bin/sh.

Normally a TSO user would not have a UID(0), or / as their HOME directory, but for the user performing initial z/OS UNIX activation work, it is appropriate.

2.3.5 Step 5 - Create HFS data sets

IBM requires that you maintain a separate HFS data set for each of the following directories:

/etc	/etc contains customization data. Keeping the /etc file system in an HFS data set separate from other file systems allows you to separate your customization data from IBM's service updates. It also makes migrating to another release easier.
/dev	/dev contains character-special files that are used when logging into the OMVS shell environment and also during c89 processing. You have the option of mounting a temporary file system (TFS) on /tmp.
/tmp	/tmp contains temporary data that are used by products and applications. /tmp, is created empty, and temporary files are created dynamically by different elements and products. You have the option of mounting a temporary file system (TFS) on /tmp.
/var	/var contains dynamic data that is used internally by products and by elements and features of z/OS. Any files or directories that are needed are created during execution of code. An example of this is caching data. In addition, you can be assured that IBM products will only create directories or files when code is executed.

For our simple single-system configuration, we will use a temporary file system (TFS) for /dev and /tmp. For a TFS, no physical HFS is required; the definition occurs using statements in the SYS1.PARMLIB(BPXPRMxx) member.

For /etc and /var, build and submit JCL similar to that shown in Figure 2-10.

```
//ALLOCHFS JOB ,'ALLOC',USER=IBMUSER,PASSWORD=????????
//*
//STEP01   EXEC PGM=IEFBR14
//*
//ETC      DD DSN=OMVS.ETC,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=SYSALLDA,
//            SPACE=(TRK,(15,15,1)),
//            DCB=(DSORG=PO),
//            DSNTYPE=HFS
//*
//VAR      DD DSN=OMVS.VAR,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=SYSALLDA,
//            SPACE=(TRK,(15,15,1)),
//            DCB=(DSORG=PO),
//            DSNTYPE=HFS
```

Figure 2-10 JCL to create ETC and VAR HFS data sets

If these data sets are not allocated under SMS control, then VOL=SER information should be included in the JCL.

2.3.6 Step 6 - Customize BPXPRMxx

The BPXPRMxx member in SYS1.PARMLIB contains the statements and parameters that define the z/OS UNIX configuration to z/OS. To activate z/OS UNIX in full function mode, a BPXPRMxx member must exist. The syntax of BPXPRMxx statements is documented in *z/OS MVS Initialization and Tuning Reference,* SA22-7592.

To simplify initial setup, a sample BPXPRMxx is supplied with z/OS in SYS1.SAMPLIB(BPXPRMxx) (see Example B-4 on page 531). It is recommended that this member be copied and used as the basis for your customized BPXPRMxx. For your first BPXPRMxx member, it is recommended that a suffix of 00 be used (for example, BPXPRM00); this is typically used for simple single-system configurations.

Tip: Check the SYS1.SAMPLIB(BPXPRMXX) for the latest enhancements, by searching the member for FMID HBB7708.

As previously mentioned, BPXPRMxx controls the parameters that control the z/OS UNIX environment. To specify which BPXPRMxx Parmlib member to start with, the systems programmer can include OMVS=xx in the IEASYSxx parmlib member. You can also specify multiple OMVS Parmlib members in IEASYSxx. For example:

```
OMVS=(AA,BB,CC)
```

Note: To modify BPXPRMxx parmlib settings without re-IPLing, you can use the **`SETOMVS`** operator command. Or you can dynamically change the BPXPRMxx parmlib members that are in effect by using the **`SET OMVS`** operator command.

Defining file systems

The following sections explain how to customize the FILESYSTYPE, ROOT, MOUNT, NETWORK, and SUBFILESYSTYPE statements in BPXPRMxx, to specify your file systems.

FILESYSTYPE

FILESYSTYPE specifies the type of file system that is to be started. BPXPRMxx can contain more than one FILESYSTYPE statement. The syntax of the FILESYSTYPE statement is shown in Figure 2-11.

```
FILESYSTYPE
   TYPE(type_name)
   ENTRYPOINT(entry_name)
   PARM('parm')
   ASNAME(proc_name[,'start_parms'])
```

Figure 2-11   FILESYSTYPE syntax

Typical FILESYSTYPEs are:

- HFS for a hierarchical file system (HFS)
- TFS for a temporary file system (TFS)
- UDS for z/OS UNIX domain (AF_UNIX) sockets
- INET for network (AF_INET and AF_INET6) sockets
- CINET for common INET (AF_INET and AF_INET6) sockets
- AUTOMNT for an automounted file system
- DFSC for accessing global namespace
- NFS for accessing remote files
- zFS for a DFS zSeries file system

Figure 2-12 shows FILESYSTYPE statements as supplied in SYS1.SAMPLIB(BPXPRMxx).

```
  FILESYSTYPE TYPE(HFS)           /* Type of file system to start  */
              ENTRYPOINT(GFUAINIT)  /* Entry Point of load module    */
              PARM(' ')           /* Null PARM for physical file
                                     system                        */

/*FILESYSTYPE TYPE(AUTOMNT)       *//* Type of file system to start  */
/*            ENTRYPOINT(BPXTAMD) *//* Entry Point of load module    */

/*FILESYSTYPE TYPE(TFS)           *//* Type of file system to start  */
/*            ENTRYPOINT(BPXTFS)  *//* Entry Point of load module    */

/*FILESYSTYPE TYPE(NFS)           *//* Type of file system to start  */
/*            ENTRYPOINT(GFSCINIT)*//* Entry Point of load module   */
/*            ASNAME(MVSNFSC,'start_parms')*/ /*               @DCC*/
/*            PARM('biod(6)')     *//* Parameter to pass in         */

  FILESYSTYPE TYPE(UDS) ENTRYPOINT(BPXTUINT)

  FILESYSTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)

/*FILESYSTYPE TYPE(CINET) ENTRYPOINT(BPXTCINT) */
```

Figure 2-12   ‘SYS1.SAMPLIB(BPXPRMXX) - FILESYSTYPE

It is recommended that in addition to TYPE(HFS) and TYPE(UDS), the entries for TYPE(AUTOMNT) and TYPE(TFS) be uncommented for the initial full function configuration.

We do not need TYPE(INET) at this stage, so that should be commented out. This should result in FILESYSTYPE definitions in SYS1.PARMLIB(BPXPRM00) as shown in Figure 2-13.

```
  FILESYSTYPE TYPE(HFS)            /* Type of file system to start  */
              ENTRYPOINT(GFUAINIT)  /* Entry Point of load module    */
              PARM(' ')             /* Null PARM for physical file
                                       system                        */

  FILESYSTYPE TYPE(AUTOMNT)         /* Type of file system to start  */
              ENTRYPOINT(BPXTAMD)   /* Entry Point of load module    */

  FILESYSTYPE TYPE(TFS)             /* Type of file system to start  */
              ENTRYPOINT(BPXTFS)    /* Entry Point of load module    */

/*FILESYSTYPE TYPE(NFS)           *//* Type of file system to start  */
/*            ENTRYPOINT(GFSCINIT)*//* Entry Point of load module   */
/             ASNAME(MVSNFSC,'start_parms')*/ /*               @DCC*/
/*            PARM('biod(6)')     *//* Parameter to pass in         */

  FILESYSTYPE TYPE(UDS) ENTRYPOINT(BPXTUINT)

/*FILESYSTYPE TYPE(INET) ENTRYPOINT(EZBPFINI) */

/*FILESYSTYPE TYPE(CINET) ENTRYPOINT(BPXTCINT) */
```

Figure 2-13 SYS1.PARMLIB(BPXPRM00) - FILESYSTYPE

ROOT

ROOT specifies a file system that z/OS UNIX is to logically mount as the root file system. The syntax of the ROOT statement is shown in Figure 2-14.

```
ROOT FILESYSTEM('fsname')|DDNAME(ddname)
   TYPE(type_name)
   MODE(access)
   PARM('parameter')
   SETUID|NOSETUID
   AUTOMOVE|NOAUTOMOVE
   SYSNAME(sysname)
   TAG(NOTEXT|TEXT,ccsid)
```

Figure 2-14 ROOT syntax

Figure 2-15 shows the ROOT statement as supplied in SYS1.SAMPLIB(BPXPRMxx).

```
ROOT     FILESYSTEM('OMVS.ROOT') /* Either FILESYSTEM or DDNAME must
                                    be specified, but not both.
                                    FILESYSTEM must be entered in
                                    quotes.                       */
         TYPE(HFS)              /* Type of File system           */
         MODE(RDWR)             /* (Optional) Can be READ or RDWR.
                                    Default = RDWR                */
```

Figure 2-15 SYS1.SAMPLIB(BPXPRMxx) - ROOT

The name of the HFS ROOT was determined during the z/OS installation process. For the purposes of this book we assume that the ROOT HFS name was as defined in the supplied sample. This should result in a ROOT definition in SYS1.PARMLIB(BPXPRM00) as shown in Figure 2-16.

```
ROOT     FILESYSTEM('OMVS.ROOT') /* Either FILESYSTEM or DDNAME must
                                    be specified, but not both.
                                    FILESYSTEM must be entered in
                                    quotes.                       */
         TYPE(HFS)               /* Type of File system           */
         MODE(RDWR)              /* (Optional) Can be READ or RDWR.
                                    Default = RDWR                */
```

Figure 2-16 SYS1.PARMLIB(BPXPRM00) - ROOT

MOUNT

MOUNT specifies a file system that z/OS UNIX is to logically mount onto the root file system or another file system. Mount statements are processed in the sequence in which they appear. If they are cascading, the system will mount the first file system first. Make sure that a mount point exists before the file system is mounted. If you mount a file system over an existing directory containing files, you will cover up the existing files. The syntax of the MOUNT statement is shown in Figure 2-17.

```
MOUNT FILESYSTEM('fsname')|DDNAME(ddname)
   TYPE(type_name)
   MOUNTPOINT('pathname')
   MODE(access)
   PARM('parameter')
   TAG(NOTEXT|TEXT,ccsid)
   SETUID|NOSETUID
   SECURITY|NOSECURITY
   AUTOMOVE[(INCLUDE|EXCLUDE,sysname1,sysname2,...,sysnamen)]|NOAUTOMOVE|UNMOUNT
   SYSNAME(sysname)
```

Figure 2-17 MOUNT syntax

Figure 2-18 shows the MOUNT statement as supplied in SYS1.SAMPLIB(BPXPRMxx).

```
/*MOUNT FILESYSTEM('OMVS.USER.JOE')*/ /* Either FILESYSTEM or DDNAME
                                         must be specified, but not both.
                                         FILESYSTEM must be entered in
                                         quotes.                        */
/*      TYPE(HFS)            */    /* Type of FileSystem            */
/*      MODE(RDWR)           */    /* Can be READ or RDWR           */
/*      MOUNTPOINT('/u/joe') */    /* Must be entered in quotes.    */
/*      NOSETUID             */    /* ignore setuid/gid mode bits   */
/*                                                              @PAD */
/*      SECURITY             */    /* enforce security checks       */
/*      TAG(NOTEXT,0)        */    /*                           @D9A*/
```

Figure 2-18 SYS1.SAMPLIB(BPXPRMXX) - MOUNT

MOUNT definitions need to be included for the /dev, /etc, /tmp, and /var directories. The ETC and VAR HFS data sets were allocated in 2.3.5, “Step 5 - Create HFS data sets” on page 57, so MOUNT statements are needed for them. The /dev and /tmp directories require MOUNT statements to indicate they are TYPE(TFS). This should result in MOUNT definitions in SYS1.PARMLIB(BPXPRM00) as shown in Figure 2-19.

```
MOUNT FILESYSTEM('/DEV')
      TYPE(TFS)
      MOUNTPOINT('/dev')
      MODE(RDWR)
      PARM('-s 5') /* 5M virtual */

MOUNT FILESYSTEM('OMVS.ETC')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/etc')

MOUNT FILESYSTEM('/TMP')
      TYPE(TFS)
      MOUNTPOINT('/tmp')
      MODE(RDWR)
      PARM('-s 20') /* 20M virtual */

MOUNT FILESYSTEM('OMVS.VAR')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/var')
```

Figure 2-19 SYS1.PARMLIB(BPXPRM00) - MOUNT

For those of you running z/OS V1R5 or higher, an additional MKDIR() parameter can be used for the MOUNT or ROOT statement in the BPXPRMxx parmlib member. This allows you to specify a directory, or mountpoint, which is to be created during parmlib processing at OMVS initialization time.

You may have experienced failed parmlib mounts because the mountpoint did not preexist. Now it is possible to create one or more directory entries in the file system associated with the ROOT or MOUNT parameter, or to create other directory entries in another file system that is already mounted.

```
MOUNT FILESYSTEM(‘OMVS.&SYSNAME..ITSO’)
      MOUNTPOINT(‘/u/itso’)
      TYPE(HFS) MODE(RDWR)
      MKDIR(‘resident’)

MOUNT FILESYSTEM(‘OMVS.&SYSNAME..RESIDENT’)
      MOUNTPOINT(‘/u/itso/resident’)
      TYPE(HFS) MODE(RDWR)
      MKDIR(‘patrick’)
```

Figure 2-20 MKDIR() use in the BPXPRMxx parmlib member

There are some things that you have to be aware of when using MKDIR():

- With the MKDIR() support, permissions are set to 755 (-rwxr-xr-x).
- Do not use it with file systems that mount asynchronously, like NFS clients.
- The total length of the MKDIR() and its mountpoint cannot exceed 1023 characters.
- If sharing parmlib members between shared HFS members is being used, this MKDIR() statement should be omitted unless all are running at V1R5 or above.

SWA above

In z/OS V1R5, an additional parameter was introduced for the BPXPRMxx parmlib member. This parameter lets you control the Scheduler Work Area (SWA) control blocks allocation below or above the 16 megabyte line.

SWA control blocks reside in a user's address space and are used by initiated tasks to contain information about the job and its job steps. The SWA control blocks for z/OS UNIX are by default allocated below the 16 megabyte line. However, when a large number of file systems are mounted, this can cause storage contraints.

To resolve these contraints, a new BPXPRMxx parmlib parameter was added. Now you can specify where the SWA control blocks are allocated—above or below the 16 megabyte line.

To put the SWA below the 16 megabyte line, which is the default, specify:

```
SWA(BELOW)
```

To put the SWA above the 16 megabyte line, the command is:

```
SWA(ABOVE)
```

You can check the current settings by issuing the following operator command:

```
D OMVS,O
```

Keep in mind that changes to the SWA setting only become available after OMVS initialization.

NETWORK

NETWORK defines address families for sockets. It is necessary if the facility needs the socket domains. The syntax of the NETWORK statement is shown in Figure 2-21.

```
NETWORK DOMAINNAME(sockets_domain_name)
   DOMAINNUMBER(sockets_domain_number)
   MAXSOCKETS(number)
   TYPE(type_name)
   INADDRANYPORT(starting_port_number)
   INADDRANYCOUNT(number_of_ports_to_reserve)
```

Figure 2-21 NETWORK syntax

Figure 2-22 shows the NETWORK statements as supplied in SYS1.SAMPLIB(BPXPRMxx).

```
   NETWORK DOMAINNAME(AF_UNIX)
           DOMAINNUMBER(1)
           MAXSOCKETS(200)
           TYPE(UDS)

   NETWORK DOMAINNAME(AF_INET)
           DOMAINNUMBER(2)
           MAXSOCKETS(64000)
           TYPE(INET)

/* NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) */   /* For IPv6   */
/*         TYPE(INET) */

/* NETWORK DOMAINNAME(AF_INET)                  */
/*         DOMAINNUMBER(2)                      */
/*         MAXSOCKETS(64000)                    */
/*         TYPE(CINET)                          */
/*         INADDRANYPORT(2000)                  */
/*         INADDRANYCOUNT(325)                  */

/* NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) */   /* For IPv6   */
/*         TYPE(CINET)                           */
```

Figure 2-22 SYS1.SAMPLIB(BPXPRMXX) - NETWORK

We do not need TYPE(INET) at this stage, so that NETWORK definition should be commented out. This should result in NETWORK definitions in SYS1.PARMLIB(BPXPRM00) as shown in Figure 2-23.

```
   NETWORK DOMAINNAME(AF_UNIX)
           DOMAINNUMBER(1)
           MAXSOCKETS(200)
           TYPE(UDS)

/* NETWORK DOMAINNAME(AF_INET)
           DOMAINNUMBER(2)
           MAXSOCKETS(64000)
           TYPE(INET) */

/* NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) */   /* For IPv6   */
/*         TYPE(INET) */

/* NETWORK DOMAINNAME(AF_INET)                  */
/*         DOMAINNUMBER(2)                      */
/*         MAXSOCKETS(64000)                    */
/*         TYPE(CINET)                          */
/*         INADDRANYPORT(2000)                  */
/*         INADDRANYCOUNT(325)                  */

/* NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) */   /* For IPv6   */
/*         TYPE(CINET)                           */
```

Figure 2-23 SYS1.PARMLIB(BPXPRM00) - NETWORK

SUBFILESYSTYPE

SUBFILESYSTYPE specifies an AF_INET or AF_INET6 physical file system that is to run underneath the CINET socket file system. The TYPE() value is usually CINET and matches the TYPE operand on a previous FILESYSTYPE and NETWORK statement. In the case of TCP/IP, the NAME() value is the procname. The syntax of the SUBFILESYSTYPE statement is shown in Figure 2-24.

```
SUBFILESYSTYPE
   NAME(transport_name)
   TYPE(type_name)
   ENTRYPOINT(entry_name)
   PARM('parameter')
   DEFAULT
```

Figure 2-24   SUBFILESYSTYPE syntax

Figure 2-25 shows the SUBFILESYSTYPE statements as supplied in SYS1.SAMPLIB(BPXPRMxx).

```
/*SUBFILESYSTYPE NAME(TCPIP)     */  /* Name of file system         */
/*               TYPE(CINET)     */  /* Type matching Cinet's TYPE  */
/*           ENTRYPOINT(EZBPFINI)*/  /* Entry point of load module  */
/*               DEFAULT         */  /* <- The Default Socket PFS   */

/*SUBFILESYSTYPE NAME(TCPIP2)    */  /* Name of file system         */
/*               TYPE(CINET)     */  /* Type matching Cinet's TYPE  */
/*           ENTRYPOINT(EZBPFINI)*/  /* Entry point of load module  */
```

Figure 2-25   SYS1.SAMPLIB(BPXPRMXX) - SUBFILESYSTYPE

For the purposes of configuring a simple single-system configuration, we will use the SUBFILESYSTYPE definitions as supplied in the sample. This should result in SUBFILESYSTYPE definitions in SYS1.PARMLIB(BPXPRM00), as shown in Figure 2-26.

```
/*SUBFILESYSTYPE NAME(TCPIP)     */  /* Name of file system         */
/*               TYPE(CINET)     */  /* Type matching Cinet's TYPE  */
/*           ENTRYPOINT(EZBPFINI)*/  /* Entry point of load module  */
/*               DEFAULT         */  /* <- The Default Socket PFS   */

/*SUBFILESYSTYPE NAME(TCPIP2)    */  /* Name of file system         */
/*               TYPE(CINET)     */  /* Type matching Cinet's TYPE  */
/*           ENTRYPOINT(EZBPFINI)*/  /* Entry point of load module  */
```

Figure 2-26   SYS1.PARMLIB(BPXPRM00) - SUBFILESYSTYPE

Defining system limits

We have seen in this chapter that the BPXPRMxx Parmlib member contains the parameters that control z/OS UNIX file systems, such as FILESYSTYPE and MOUNT. But BPXPRMxx can contain more parameters to control z/OS UNIX processing.

Note: IBM recommends that you have two BPXPRMxx members, one that specifies file system setup and one that specifies system limits. This makes it easier to migrate from one release to another.

Figure 2-27 on page 66 shows an example of BPXPRMxx Parmlib member parameters that you can set up to influence user logon, active processes, file handling, and storage requirements.

```
MAXPROCSYS(300)
MAXPROCUSER(10125)
MAXUIDS(50)
MAXFILEPROC(65535)
MAXTHREADTASKS(32768)
MAXTHREADS(100000)
MAXPTYS(256)
MAXCORESIZE(4194304)
MAXASSIZE(2147483647)
MAXCPUTIME(2147483647)
MAXMMAPAREA(4096)
MAXSHAREPAGES(32768000)
```

Figure 2-27 BPXPRMxx parameters that control active processes

It is important to handle these parameters with care because a number of these statements are interrelated. For example, it makes no sense to allow more users to access z/OS UNIX when you do not provide enough TTYs. Each user or process entering z/OS UNIX needs a pseudo-terminal (pseudo-TTY). A common rule is to allow four pseudo-TTY pairs for each user (MAXPTYS = MAXUIDS * 4). For more information about pseudo-TTY see “Customize the number of pseudoterminal files” on page 209.

BPXPRMxx controls the complete environment of z/OS UNIX. A number of the MAX-parameters shown in Figure 2-27 can also be set for the individual user. This can be done in the RACF OMVS segment of the user, as shown in Figure 2-28. For this individual user only the ASSIZEMAX parameter is set, which maximizes the address space size for this user to 20MB.

```
USER=PATRICK

OMVS INFORMATION
----------------
UID= 0000068216
HOME= /u/patrick
PROGRAM= /bin/sh
CPUTIMEMAX= NONE
ASSIZEMAX= 0020480000
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE
```

Figure 2-28 Listing of the user’s OMVS segment

Note: The parameters set in the OMVS Segment overrule the corresponding settings in the BPXPRMxx Parmlib member.

Figure 2-27 also shows the MAXFILEPROC parameter, which sets the maximum number of file descriptors that a single process can use concurrently. File descriptors are used for open files, directories, sockets, and pipes. By limiting the number of open files that a process can have, you limit the amount of system resources a single process can use at one time.

Before z/OS V1R6, the maximum you could specify was 64K, which in effect limited the number of connected clients at any one time. To relieve this constraint the limit was increased to 128K (131072) descriptors.

The IBM-supplied BPXPRMXX member in SYS1.SAMPLIB shows a MAXFILEPROC value of 2000, which is generally a good starting point for normal TN3270 client usage. Some software products like SAP R/3 or Websphere Application Server require you to set a higher value for MAXFILEPROC. This is generally well documented.

Checking the syntax of BPXPRMxx

You can use the SETOMVS SYNTAXCHECK operator command to check the syntax of a BPXPRMxx PARMLIB member before doing an IPL (but you cannot use SETOMVS to verify whether HFS data sets or mount points are valid). As an example, to check the syntax of SYS1.PARMLIB(BPXPRM00) the following operator command can be used:

```
SETOMVS SYNTAXCHECK=(00)
```

If the syntax is correct, the following message will be issued:

```
BPXO039I SETOMVS SYNTAXCHECK COMMAND SUCCESSFUL
```

2.3.7 Step 7 - Customize ALLOCxx

Forked address spaces are perceived to be batch jobs by MVS allocation. If a forked address space attempts to allocate a data set on a volume that is not mounted, the request either waits (with or without an operator prompt) or it fails. The ALLOCxx parmlib member controls the behavior of allocation requests of this type. If you do not want the request to wait, specify the following ALLOCxx statements:

```
VOLUME_ENQ POLICY (CANCEL)
VOLUME_MNT POLICY (CANCEL)
```

Use this policy so that forked addresses do not go into allocation waits. Be aware that using this policy can disrupt your system, because it will cause a failure rather than a wait.

2.3.8 Step 8 - Customize COFVLFxx

If you are using the virtual lookaside facility (VLF), update the VLF member SYS1.PARMLIB(COFVLFxx). Add CLASS and EMAJ statements to activate a RACF performance option for z/OS UNIX. The required statements are shown in Figure 2-29.

```
CLASS NAME(IRRGMAP)     /* OpenMVS-RACF GMAP table     */
EMAJ(GMAP)              /* Major name = GMAP           */
CLASS NAME(IRRUMAP)     /* OpenMVS-RACF UMAP table     */
EMAJ(UMAP)              /* Major name = UMAP           */
CLASS NAME(IRRGTS)      /* RACF GTS table              */
EMAJ(GTS)               /* Major name = GTS            */
CLASS NAME(IRRACEE)     /* RACF saved ACEEs            */
EMAJ(ACEE)              /* Major name = ACEE           */
CLASS NAME(IRRSMAP)     /* Security packet             */
EMAJ(SMAP)              /* Major name = SMAP           */
```

Figure 2-29 COFVLFxx statements

If your RACF database has been converted to stage 3 of application identity mapping (AIM), then the UNIXMAP classes no longer exist, and this step with VLF is not required.

2.3.9 Step 9 - Customize CTnBPXxx

The SYS1.PARMLIB(CTnBPXxx) member specifies the tracing options for a component trace of z/OS UNIX events. This causes trace records to be retained in a buffer, which could be read if a dump is written. Two members are recommended:

- One member should control initial tracing, which automatically starts when the OMVS address space is started (this member should be considered the operating system's default member). The CTRACE statement in SYS1.SAMPLIB(BPXPRMxx) specifies CTIBPX00. When z/OS is installed SYS1.PARMLIB(CTIBPX00) is supplied as shown in Figure 2-30 on page 68.

```
TRACEOPTS
          ON
          BUFSIZE(128K)
```

Figure 2-30 SYS1.PARMLIB(CTIBPX00)

- One member should be set up to trace all z/OS UNIX events. When z/OS is installed SYS1.PARMLIB(CTIBPX01) is supplied as shown in Figure 2-30.

```
TRACEOPTS
           ON
           BUFSIZE(4M)
           OPTIONS(
                   'ALL     '   */     /* ALL OPTIONS TRACED        */
                   )            */
```

Figure 2-31 SYS1.PARMLIB(CTIBPX01)

IBM support would normally tell you which options to select when attempting to debug a reported problem.

2.3.10 Step 10 - Customize IEADMR00

You should change PARMLIB member IEADMR00 (SYSMDUMP and core dump defaults) to specify at least the following values:

```
SDATA=(RGN,SUM,TRT,LPA)
```

This gathers adequate data without an excessive dump size.

2.3.11 Step 11 - Customize SMFPRMxx

The JWT value in SYS1.PARMLIB(SMFPRMxx) specifies how long an idle address space is allowed to wait before it is terminated. When an address space is dubbed a process, or when a forked or spawned process is created, the process may go into signal-enabled waits. In a signal-enabled wait, the address space is made exempt from long-wait time-outs as specified by the JWT value in SYS1.PARMLIB(SMFPRMxx).

This enables parent processes to wait forever while child processes are running. Otherwise, if the parent process is terminated due to job wait timeout, a SIGHUP signal is sent to the running process and work is lost.

However, shell users, whether logged on through TSO/E and the OMVS command, or via rlogin or telnet, are exempt from job wait timeout because the shell is in a signal-enabled wait while waiting for a command from the user. To have shell users be timed out and logged off, you need to specify the TMOUT environment variable in /etc/profile. The TMOUT environment variable contains the number of seconds before user input times out. If user input is not received, the shell ends.

If a shell started by the TSO/E OMVS command times out, then the TSO address becomes enabled for job wait timeout processing. This means that if you have JWT=30 (30 minutes) and you have TMOUT=600 (10 minutes), then TSO users who leave their terminals in the shell will time out and be logged off in about 40 minutes.

2.3.12 Step 12 - Customize IEASYSxx

After the BPXPRMxx member has been customized, you need to specify the OMVS statement in the IEASYSxx member of SYS1.PARMLIB. This links the BPXPRMxx member into the z/OS IPL process, so that z/OS UNIX is initiated as desired. If you do not specify the OMVS statement (or you specify OMVS=DEFAULT), the kernel is started in minimum mode with all BPXPRMxx statements set to their default values.

The OMVS statement in IEASYSxx can be specified as follows:

- OMVS=xx if only a single BPXPRMxx member has been defined. In this case, xx represents the suffix of the BPXPRMxx member. For example, if BPXPRM00 is the member name, then IEASYSxx should define:

  ```
  OMVS=00
  ```

- OMVS=(nn,mm,...) if multiple BPXPRMxx members have been defined. In this case, nn represents the suffix of one BPXPRMxx member, mm represents the suffix of another, and so forth. For example, if BPXPRM00 contains z/OS UNIX statements that are common to all systems, while BPXPRMZD only contains statements suitable for system EGZD (&SYSNAME=EGZD), then IEASYSxx for system EGZD might specify:

  ```
  OMVS=(ZD,00)
  ```

  or

  ```
  OMVS=(&SYSCLONE.,00)
  ```

  Note that with concatenated members, the first value set for a parameter is the one that is used; if a later member in the list specifies a different value, that value is ignored.

  See *z/OS MVS Initialization and Tuning Reference,* SA22-7592 for detailed information about the format of IEASYSxx statements.

2.3.13 Step 13 - IPL

IPL the system so that the modified IEASYSxx member is implemented.

After IPL, you will observe that the OMVS, BPXOINIT and BPXAS address spaces start, and the SYSLOG message shown in Figure 2-32 are produced.

```
BPXF013I FILE SYSTEM OMVS.ROOT 468
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM /DEV 469
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM OMVS.ETC 471
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM /TMP 472
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM OMVS.VAR 474
WAS SUCCESSFULLY MOUNTED.
BPXF203I DOMAIN AF_UNIX WAS SUCCESSFULLY ACTIVATED.
BPXF225I THE RESOLVER_PROC, DEFAULT, WAS NOT STARTED.  THERE IS NO
AF_INET OR AF_INET6 DOMAIN TO SUPPORT THE RESOLVER FUNCTION.

BPXI027I THE ETCINIT JOB ENDED IN ERROR, EXIT STATUS 00000500
BPXI004I OMVS INITIALIZATION COMPLETE
```

Figure 2-32 Successful full function mode configuration - messages

The BPXF225I message occurs because INET is not yet configured. The BPXI027I message occurs because /etc does not have any contents yet.

If you check the status of z/OS UNIX using the D OMVS console command, in SYSLOG you can expect to see a response similar to Figure 2-33.

```
D OMVS
BPXO042I 09.29.27 DISPLAY OMVS 192
OMVS     000D ACTIVE          OMVS=(00)
```

Figure 2-33 Successful full function mode configuration - status

If you check the status of z/OS UNIX address spaces using the D OMVS,A=ALL console command, in SYSLOG you can expect to see a response similar to Figure 2-34.

```
D OMVS,A=ALL
BPXO040I 09.42.23 DISPLAY OMVS 194
OMVS     000D ACTIVE          OMVS=(00)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSTASK BPXOINIT 001F          1          0 MRI--- 09.11.18       .07
  LATCHWAITPID=         0 CMD=BPXPINPR
  SERVER=Init Process                    AF=    0 MF=00000 TYPE=FILE
RMFTASK  RMFGAT   0023   16777218          1 1R---P 09.15.01      5.43
  LATCHWAITPID=         0 CMD=ERB3GMFC
```

Figure 2-34 Successful full function mode configuration - address spaces

If you check the status of z/OS UNIX file systems using the D OMVS,F console command, in SYSLOG you can expect to see a response similar to Figure 2-35.

```
D OMVS,F
BPXO045I 09.44.04 DISPLAY OMVS 196
OMVS     000D ACTIVE          OMVS=(00)
TYPENAME   DEVICE ----------STATUS----------- MODE
TFS             6 ACTIVE                      RDWR
  NAME=/TMP
  PATH=/SYSTEM/tmp
  MOUNT PARM=-s 20
TFS             4 ACTIVE                      RDWR
  NAME=/DEV
  PATH=/SYSTEM/dev
  MOUNT PARM=-s 5
HFS             7 ACTIVE                      RDWR
  NAME=OMVS.VAR
  PATH=/SYSTEM/var
HFS             5 ACTIVE                      RDWR
  NAME=OMVS.ETC
  PATH=/SYSTEM/etc
HFS             3 ACTIVE                      RDWR
  NAME=OMVS.ROOT
  PATH=/
```

Figure 2-35 Successful full function mode configuration - file system

If you logon to TSO and at the READY prompt enter OMVS, you will enter the z/OS shell as shown in Figure 2-36.

```
IBM
Licensed Material - Property of IBM
5694-A01 (C) Copyright IBM Corp. 1993, 2001
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

#
```

Figure 2-36 Successful full function mode configuration - z/OS Shell

Note: The "#" prompt indicates "superuser". A non-superuser would see a "$" prompt.

If you enter the **`ls -al`** command, you can see the structure of the root file system as shown in Figure 2-37.

```
# ls -al
total 264
drwxr-xr-x  13 U701503  STCGRP      8192 Jul 17 22:54 .
drwxr-xr-x  13 U701503  STCGRP      8192 Jul 17 22:54 ..
dr-xr-xr-x   2 U701503  STCGRP      8192 May 29  2002 ...
-rw-------   1 U701503  STCGRP         9 Jul 17 22:55 .sh_history
drwxr-xr-x   5 U701503  STCGRP      8192 Oct  9  2002 SERVICE
drwxr-xr-x   6 U701503  STCGRP      8192 May 29  2002 SYSTEM
drwxr-xr-x   4 U701503  STCGRP     32768 Nov  8  2002 bin
lrwxrwxrwx   1 U701503  STCGRP        12 Jul 22  2002 dev -> $SYSNAME/dev
lrwxrwxrwx   1 U701503  STCGRP        12 Jul 22  2002 etc -> $SYSNAME/etc
lrwxrwxrwx   1 U701503  STCGRP        16 Jul 22  2002 krb5 -> etc/dce/var/krb5
drwxr-xr-x   2 U701503  STCGRP      8192 Jul 31  2002 lib
drwxr-xr-x   2 U701503  STCGRP      8192 Jun 10  2002 opt
drwxr-xr-x   4 U701503  STCGRP      8192 May 29  2002 samples
lrwxrwxrwx   1 U701503  STCGRP        12 Jul 22  2002 tmp -> $SYSNAME/tmp
drwxr-xr-x   3 U701503  STCGRP      8192 Jul  4  2002 u
drwxr-xr-x  12 U701503  STCGRP      8192 Jan 13  2003 usr
lrwxrwxrwx   1 U701503  STCGRP        12 Jul 22  2002 var -> $SYSNAME/var
```

Figure 2-37 ls -al output

If you enter the **env** command, you will see the environment variables that are defined to your session; see Figure 2-38 on page 72.

```
# env
_BPX_TERMPATH=OMVS
PATH=/bin
SHELL=/bin/sh
COLUMNS=80
_=/bin/env
LOGNAME=IBMUSER
TERM=dumb
HOME=/
LINES=28
TZ=UTC0
```

Figure 2-38 env output

2.3.14 Step 14 - Customize /etc/init.options

The /usr/sbin/init file is the initialization program that is run when the OMVS address space is initialized. This program invokes a shell to execute an initialization shell script that customizes the environment. When this shell script finishes, or when a time interval established by /usr/sbin/init expires, kernel services become available for general batch and interactive use.

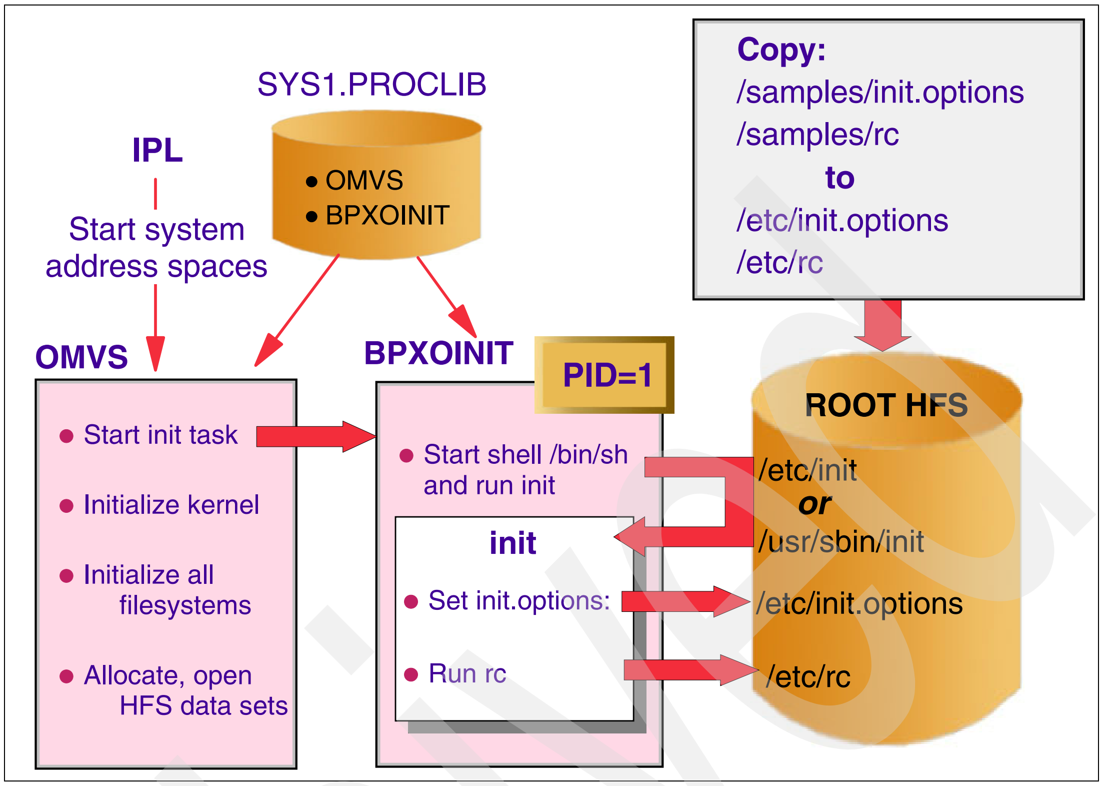


Figure 2-39 z/OS UNIX initialization processing shell scripts

Standard output (stdout) and standard error output (stderr) are redirected to /etc/log.

The files associated with the system initialization program /usr/sbin/init are as follows:

`/bin/sh`	Default shell that /usr/sbin/init invokes to execute **`/etc/rc`** or another shell script specified in the /etc/init.options file.
`/etc/init.options`	Initialization options file, which is read by /usr/sbin/init.
`/etc/rc`	Default initialization shell script.
`/etc/log`	The file that output is written to.
Other utilities	Services that are called by the initialization shell script

/usr/sbin/init and the customized /etc/init.options and /etc/rc are run at IPL, as shown in Figure 2-39 on page 73. There is no other way to invoke them explicitly.

Before /usr/sbin/init invokes the shell to execute the system initialization shell script, it reads the file /etc/init.options for values of various options. The IBM-supplied default is in /samples/init.options. Copy this file to /etc/init.options using the following command in the z/OS shell:

```
cp /samples/init.options /etc
```

and make the appropriate changes. /usr/sbin/init treats all lines in /etc/init.options that do not start with a hyphen (-) as comment lines. Lines that start with a hyphen are used to specify options. The format of lines specifying options is as follows:

```
-oo vvvvv comment
```

Where:

oo is a field of one or more nonblank characters immediately following the hyphen that identify the option. The end of the option field is delimited by one or more blanks.

vvvvv is a field of one or more nonblank characters that specify an option value. These characters are numeric, alphabetic, or a combination of both, depending on the option being specified. The end of the value field is delimited by one or more blanks.

Option and option value characters must appear in columns 1 through 79 of an option line in `/etc/init.options`. `/usr/sbin/init` ignores characters beyond column 79. However, a backslash (\) immediately following nonblank value field characters is recognized as a continuation character. If the continuation character is found, nonblank characters at the beginning of the next line are treated as option value characters. The first blank character delimits the end of the value field.

Option value characters on a continuation line are limited to columns 1 through 79.

The continuation character is recognized on continuation lines as well as the option line.

Any characters after a blank delimiting the end of the option value field on the same line are treated as comment characters.

Options and option value ranges are listed below:

-a nnnn Alarm option: nnnn are digits that specify the maximum time in seconds /usr/sbin/init will wait for execution of the initialization shell script to complete.

Default: 9999 seconds

Maximum: 9999 seconds

If the shell does not signal completion of the script before this time elapses, /usr/sbin/init writes the timeout error message, FSUM4013I, in /etc/log and exits with status indicating: Timeout waiting for shell script to complete. You must specify enough time for the system initialization script to complete if this is a requirement at your installation.

Note: If the value 0 is specified for the alarm option, no timeout interval is set. The decision to specify the value 0 for the alarm option should be made carefully and only after it is known that the initialization script is error-free.

-t n Terminate option: n is a digit indicating whether to terminate execution of the initialization shell script if the timeout specified by the alarm option (-a) occurs.

n = 0: Allow the shell script to continue

n not = 0: End the shell script

Default: n=1 (terminate the shell script)

Maximum: 1 digit

If you specify terminate and the timeout waiting for the initialization shell script occurs, /usr/sbin/init sends a stop signal to the shell process group.

It is your responsibility to decide if the initialization shell script can continue concurrent with batch and interactive use of the shell.

-sh pathname Initialization shell pathname option: pathname specifies the shell that /usr/sbin/init should invoke to run the initialization script. /usr/sbin/init cannot set environment variables for the rest of the system.

Default: `/bin/sh`

Maximum length: 255 characters

The line -sh pppp\ in /etc/init.options specifies the first four characters of a shell pathname pppp. It also indicates that the pathname is continued on the next line (starting in column 1). Comment characters can appear after -.

The line -sh <blanks in /etc/init.options tells /usr/sbin/init not to run the shell. If you select this option, /usr/sbin/init does not invoke the shell to execute an initialization script. Instead, /usr/sbin/init signals that multiuser mode be entered and then exits.

-sc pathname Initialization script pathname option: pathname specifies the initialization shell script.

Default: /etc/rc

Maximum length: 255 characters

The line -sc pppp\ in /etc/init.options specifies the first four characters of an initialization script name pppp, and indicates that the pathname is continued on the next line (starting in column 1).

-e string Environment variable option: string in the form name=value specifying the environment variable name and the value that /usr/sbin/init passes to the shell it invokes.

Maximum length: 255 characters

The line -e ssss\ in /etc/init.options specifies the continuation of the environment variable name or value on the next line.

/etc/init.options can contain up to 25 -e option lines specifying names and values for different environment variables. /usr/sbin/init passes the resultant environment variable array to the shell that it invokes. In turn, the shell uses this array to set up an execution environment for the initialization shell script that is appropriate for the installation. TZ is an example of an environment variable that should be considered.

These environment variables should also be set up in /etc/profile or $HOME/.profile for each interactive user. Examples of variables that you could specify are TZ, LANG, and NLSPATH.

Figure 2-40 is a sample /etc/init.options file showing the time zone, the Japanese language, and the locale.

```
-e TZ=JST-9
-e LANG=Ja_JP
-e NLSPATH=/usr/lib/nls/msg/%L/%N
```

Figure 2-40 Sample /etc/init.options

Figure 2-41 shows the /etc/init.options suppled in the /samples directory supplied by z/OS.

```
-a  9999                           timeout = 9999 seconds
-t  1                              terminate shell = yes
-sc /etc/rc                        shell script = /etc/rc
-e  TZ=EST5EDT                     TZ environment variable
```

Figure 2-41 /etc/init.options from /samples

/etc/init opens the message catalog fsumucat.cat in directory /usr/lib/nls/msg/C unless an NLSPATH environment variable naming a different directory is specified in the /etc/init.options file.

For more information on environment variables for the shell, refer to *z/OS UNIX System Services Command Reference,* SA22-7802.

Using REXX execs as an alternative to /etc/init

You can use a REXX exec in an MVS data set as an alternative to running the /etc/init initialization program. To activate the REXX exec for initialization, you must specify its name on the `STARTUP_EXEC` statement in SYS1.PARMLIB(BPXPRMxx). Note that /etc/rc is initiated from /etc/init.options (via “-sc”), so replacing /etc/init.options with STARTUP_EXEC also means /etc/rc is replaced.

2.3.15 Step 15 - Customize /etc/rc

The /etc/rc file contains customization commands for z/OS UNIX. The file is invoked by /etc/init.options during startup of z/OS UNIX. An IBM supplied default file is in /samples/rc. Copy this file to /etc/rc using the following command in the z/OS shell:

```
cp /samples/rc /etc
```

Figure 2-42 on page 76 shows the active contents of the supplied file.

```
export _BPX_JOBNAME='ETCRC'
set -v -x
>/etc/utmpx
chmod 644 /etc/utmpx
chmod 666 /dev/tty*
chown 0 /dev/tty*
chmod 1777 /tmp
chmod 1777 /var
chmod 1755 /dev
mkdir -m 777 /etc/recover
/usr/lib/exrecover
sleep 5
echo /etc/rc script executed, `date`
```

Figure 2-42 /etc/rc from /samples

Regarding the content of the sample /etc/rc file:

- The `export _BPX_JOBNAME='ETCRC'` statement sets the z/OS jobname for this script to ETCRC. `_BPX_JOBNAME` is an environment variable for this purpose.
- The `set -v -x` statement specifies that a verbose shell command trace of /etc/rc is to be written to /etc/log.
- The `>/etc/utmpx` statement is a short form of redirection `0>/etc/utmpx`. This takes null input from standard input (stdin), and writes it to file /etc/utmpx—an easy way to create file /etc/utmpx if it does not already exist. Another way to achieve the same result might be to use the **`touch`** command, but the provider of the /samples/rc file chose to use redirection.
- The `chmod 644 /etc/utmpx` statement sets the initial permission bits of file /etc/utmpx so that the owner has read/write authority (6 = x’110’ = read[Y], write[Y], execute[N]), while group and other have read authority (4 = x’100’ = read[Y], write[N], execute[N]).

- The `chmod 666 /dev/tty*` statement resets the initial permission bits of pseudoterminal slave files so that the owner, group and other have read/write authority (6 = x'110' = read[Y], write[Y], execute[N]).
- The `chown 0 /dev/tty*` statement resets the owner of pseudoterminal slave files to 0.
- The `chmod 1777 /tmp` statement sets the sticky bit on for the /tmp directory so that users cannot delete each other's files. It also sets the initial permission bits of the /tmp directory so that the owner, group and other have read/write/execute authority (7 = x'111' = read[Y], write[Y], execute[Y]). For a directory, this means:
  - READ: Permission to read, but not search the contents.
  - WRITE: Permission to change, add, or delete directory entries.
  - EXECUTE: Permission to search the directory.
- The `chmod 1777 /var` statement sets the sticky bit on for the /var directory so that users cannot delete each other's files. It also sets the initial permission bits of the /var directory so that the owner, group and other have read/write/execute authority (7 = x'111' = read[Y], write[Y], execute[Y]). For a directory, this means:
  - READ: Permission to read, but not search the contents.
  - WRITE: Permission to change, add, or delete directory entries.
  - EXECUTE: Permission to search the directory.
- The `chmod 1755 /dev` statement sets the sticky bit on for the /dev directory so that users cannot delete each other's files. It also sets the initial permission bits of the /dev directory so that the owner has read/write/execute authority (7 = x'111' = read[Y], write[Y], execute[Y]), while group and other have read/execute authority (5 = x'101' = read[Y], write[N], execute[Y]). For a directory, this means:
  - READ: Permission to read, but not search the contents.
  - WRITE: Permission to change, add, or delete directory entries.
  - EXECUTE: Permission to search the directory.
- The `mkdir -m 777 /etc/recover` statement creates the /etc/recover file for use by the exrecover directory and sets the initial permission bits so that the owner, group and other have read/write/execute authority (7 = x'111' = read[Y], write[Y], execute[Y]). For a directory, this means:
  - READ: Permission to read, but not search the contents.
  - WRITE: Permission to change, add, or delete directory entries.
  - EXECUTE: Permission to search the directory.

  The exrecover daemon recovers text files from working files created by vi and ex. These working files are in one or more temporary directories. It is normally invoked from a system startup file before these working files are purged.
- The `/usr/lib/exrecover` statement starts the exrecover daemon.
- The `sleep 5` statement follows the start of the exrecover daemon. When starting daemons in the background environment, it is very important to include a sleep command at the end of the script. This command gives the background processes time to get started and set up to ignore SIGHUP so that when the shell exits, the daemons keep running when the shell script completes. The amount of time required can be determined empirically. A value of 5 seconds is suggested for a start.
- The `echo /etc/rc script executed` date statement writes information to standard output (stdout).
- The setup section for the mailx utility has been commented out in the /etc/rc file because the mailx utility no longer requires this.

- The setup section for creating the terminfo database has been commented out in the /etc/rc file because IBM ships the individual files that make up the terminfo database.
- The setup section for the mesg, talk, write, and uucp utilities has been commented out in the /etc/rc file because this customization is now done when running the FOMISCHO sample job.

2.3.16 Step 16 - Customize /etc/profile

The /etc/profile file is the system-wide profile for the z/OS shell users. It contains environment variables and commands used by most shell users. An IBM-supplied default file is in /samples/profile. Copy this file to /etc/profile using the following command in the z/OS shell:

```
cp /samples/profile /etc
```

Figure 2-43 on page 78 shows the active contents of the supplied file.

```
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=none
    exec sh -L
fi
TZ=EST5EDT
export TZ
LANG=C
export LANG
readonly LOGNAME
PATH=/bin
export PATH
LIBPATH=/lib:/usr/lib:.
export LIBPATH
NLSPATH=/usr/lib/nls/msg/%L/%N
export NLSPATH
MANPATH=/usr/man/%L
export MANPATH
MAIL=/usr/mail/$LOGNAME
export MAIL
umask 022
```

Figure 2-43 /etc/profile from /samples/profile

Regarding the content of the sample /etc/profile file:

STEPLIB=none	Indicates that STEPLIBs should be not propagated. Running the shell with `STEPLIB=none` assumes that the Language Environment run-time library resides in LINKLIST or in LPA.
exec sh -L	Reruns the SHELL command in the current address space with the environment variables just defined. Both STEPLIB=none and exec sh -L are run only on the first invocation of an interactive shell. The tty -s test prevents the shell from being run by noninteractive invocations. e.g. those started with the BPXBATCH and OSHELL utilities.
	The `fi` statement is the end of the shell script `if` statement.
TZ=EST5EDT	Sets the time zone as appropriate. In the sample profile, TZ is set to EST5EDT, which is US Eastern Daylight Time.
LANG=C	Specifies the name of the default locale. C specifies the POSIX locale.

`readonly LOGNAME`	Prevents subsequent changes in the value of variable LOGNAME.
`PATH=/bin`	Sets a default command search path to search only the /bin directory.
`LIBPATH=/lib:/usr/lib`	Specifies the directory to search for a dynamic link library (DLL) filename. If this is not set, only the working directory is searched.
`NLSPATH=/usr/lib/nls/msg/%L/%N`	Sets the path for message catalogs.
`MANPATH=/usr/man/%L`	Sets the path for the man pages.
`LANG=C`	Specifies the name of the default locale. C specifies the POSIX locale and Ja_JP specifies the Japanese locale.
`MAIL=/usr/mail/$LOGNAME`	Sets the name of the system mail file and enables mail notification. The `export` statements make the values available to the system.
`umask 022`	Sets the default file creation mask (umask). In the sample, the mask is set to 022. This causes a file created with mode 777 to have permissions of 755. The creator cannot set the group write or other write bits on in the file mode field, because the mask sets them off.

Note: For how to use the umask, see "Using the umask" on page 124.

3

Establish security for z/OS UNIX

This chapter provides a comprehensive description of all necessary controls and techniques to establish security for UNIX System Services (z/OS UNIX) on the zSeries operating system (z/OS). Assuming that the external security product is RACF, we discuss all RACF classes and profiles needed to protect z/OS UNIX resources, as well as some traditional UNIX commands related to security. Although we emphasize the new security features for z/OS UNIX introduced from OS/390 V2R7 to z/OS V1R4, this chapter contains all RACF security controls introduced since the launch of the MVS OpenEdition subsystem with MVS/ESA 5.2.2 in 1994.

We expect that the readers have, in their RACF databases, the equivalent of member BPXISEC1 from SYS1.SAMPLIB at the level of OS/390 V2R6.

After studying this chapter, you should be able to:

- Understand superuser mode in z/OS UNIX
- Understand z/OS UNIX security concepts
- Convert to Application Identity Mapping
- Define new z/OS UNIX users and groups.
- Change existing z/OS UNIX users and groups
- Manage superusers
- Define and use permission bits and ACLs.
- Set up security for z/OS UNIX daemons and servers
- Set up security for operations in z/OS UNIX
- Set up auditing for z/OS UNIX events

3.1 Superuser authority

The concept of superuser comes from UNIX. Sometimes it is also referred to as *root* authority. A superuser can:

- Pass all z/OS UNIX security checks, so that the superuser can access any file in the hierarchical file system. A superuser does not get any additional authorities to access MVS/ESA resources. The authority is limited to the z/OS UNIX component.
- Manage z/OS UNIX processes and files.
- Have an unlimited number of processes running concurrently.
- For a started procedure, this is true only if it has a UID of 0. It is not true for a trusted or privileged process with a different UID.
- Change identity from one UID to another.
- Use **`setrlimit`** to increase any of the system limits for a process.

A superuser is usually a system administrator, or it can be a started procedure that is authorized by the RACF started procedures table or the RACF STARTED class.

The UID of a parent process and the UID's trusted or privileged attributes are propagated to a forked child process. Thus, a UID of 0 is propagated to a forked child.

3.1.1 Defining superusers with appropriate privileges

As you are defining users, you might want to define some of them with appropriate superuser privileges. There are three ways of assigning superuser privileges:

- Using the RACF UNIXPRIV class profiles—the preferred way.

  Access to profiles from the UNIXPRIV class allows you to perform various privileged functions, such as mounting a file system or changing ownership of files.
- Using the BPX.SUPERUSER profile in the FACILITY class.

  BPX.SUPERUSER allows you to request full superuser authority, perform tasks requiring such authority, and then switch back to ordinary user authority. You do not have superuser status unless you make the request.
- Assigning a UID of 0 should be given to the most important administrators.

  Superusers are special users in a z/OS UNIX environment and they are identified by a UID value of 0. One way of defining superusers is to set the UID to 0 in a user's OMVS segment. Using this method, the user always runs as a superuser. Multiple users can be defined with a UID of 0.

While some functions require a UID of 0, in most cases you can choose among the three ways. When choosing, try to minimize the number of "human" user IDs (as opposed to started procedures) set up with UID(0) superuser authority.

Do not confuse superuser authority with the MVS supervisor state. Being a superuser is not related to supervisor state, PSW key 0, and using APF-authorized instructions, macros, and callable services.

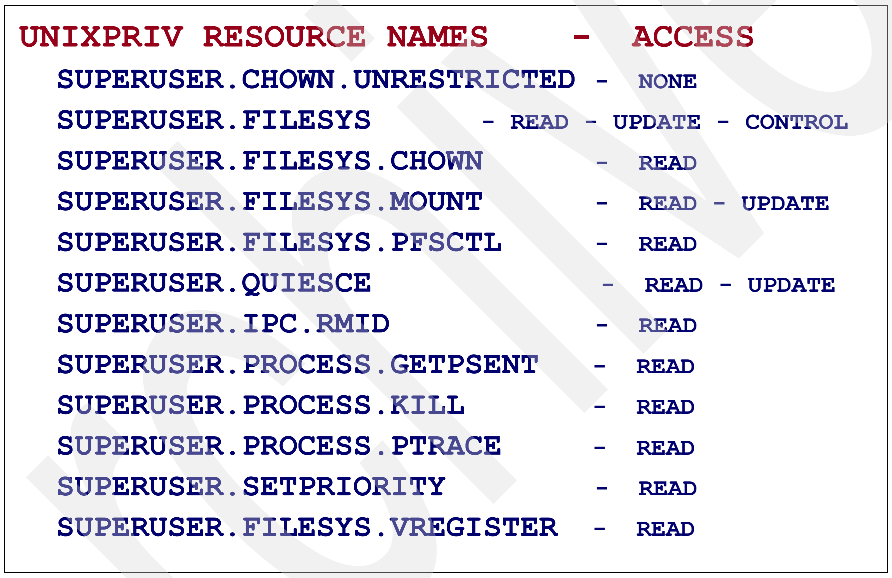
3.1.2 Using the UNIXPRIV class profiles

You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. These privileges are automatically granted to all users with z/OS UNIX superuser authority. By defining profiles in the UNIXPRIV class, you may specifically grant certain superuser privileges with a high degree of granularity to users who do not have superuser authority. This allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

Resource names in the UNIXPRIV class are associated with z/OS UNIX privileges. You must define profiles in the UNIXPRIV class protecting these resources in order to use RACF authorization to grant z/OS UNIX privileges. The UNIXPRIV class must be active and the SETROPTS RACLIST command must be in effect for the UNIXPRIV class when you define new profiles. Global access checking is not used for authorization checking to UNIXPRIV resources. To activate the UNIXPPRIV class, issue:

```
SETROPTS CLASSACT(UNIXPRIV)
```

The UNIXPRIV profiles shown in Figure 3-1 were introduced in UNIX System Services in OS/390 V2R8.



UNIXPRIV RESOURCE NAMES	- ACCESS
SUPERUSER.CHOWN.UNRESTRICTED	- NONE
SUPERUSER.FILESYS	- READ - UPDATE - CONTROL
SUPERUSER.FILESYS.CHOWN	- READ
SUPERUSER.FILESYS.MOUNT	- READ - UPDATE
SUPERUSER.FILESYS.PFSCTL	- READ
SUPERUSER.QUIESCE	- READ - UPDATE
SUPERUSER.IPC.RMID	- READ
SUPERUSER.PROCESS.GETPSENT	- READ
SUPERUSER.PROCESS.KILL	- READ
SUPERUSER.PROCESS.PTRACE	- READ
SUPERUSER.SETPRIORITY	- READ
SUPERUSER.FILESYS.VREGISTER	- READ

Figure 3-1 UNIXPRIV profile names introduced with OS/390 V2R8

UNIXPRIV class example

Normally, these privileges are automatically defined for all users who are defined with z/OS UNIX superuser authority (UID=0). But you can use the UNIXPRIV class to grant certain superuser privileges, with a high degree of granularity, to users who do not have superuser authority. For example, if users have READ access to SUPERUSER.FILESYS.MOUNT, they can issue a **mount** and **unmount** command without being a defined superuser with all superuser capabilities, as follows:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.MOUNT UACC(NONE)
PERMIT  SUPERUSER.FILESYS.MOUNT CLASS(UNIXPRIV) ID(JANE) ACCESS(READ)
SETROPTS  RACLIST(UNIXPRIV) REFRESH
```

Now user JANE (UID=35) can issue mount, which is a superuser function. This is the only superuser function JANE can do.

3.1.3 Using the BPX.SUPERUSER profile

Using the BPX.SUPERUSER resource name in the FACILITY class is another way for users to get full superuser authority for a limited time when performing tasks requiring such authority. These users do not have a UID(0). You can assign users this authority as follows:

- As an individual user
- As a group of users all belonging to the same RACF group

Since the user who has this authority is a normal user, the user must switch his authority to superuser mode using the `su` shell command.

Set up BPX.SUPERUSER profiles

Determine the users that may require this authority; they may be:

- z/OS UNIX system programmers
- MVS system programmers
- RACF system programmers
- RACF administrators

If many of the users are in the same group, you can define a RACF group with any users who need to use superuser mode. This group should not be used for any other access, for example:

```
AG UNIXSUP OWNER(SECADM)
```

Define the BPX.SUPERUSER profile in class FACILITY:

```
RDEFINE FACILITY BPX.SUPERUSER OWNER(SECADM) UACC(NONE)
```

Use the PERMIT command to permit all groups and individual users needing temporary superuser authority to this profile. Ensure that all users have OMVS segments with nonzero UIDs and a HOME keyword with the value of /u/userid.

```
PE BPX.SUPERUSER CLASS(FACILITY) ID(UNIXSUP) ACCESS(READ)
PE BPX.SUPERUSER CLASS(FACILITY) ID(JANE) ACCESS(READ)
```

When users need to perform superuser tasks, they can switch to superuser mode with the `su` command.

Deleting superuser authority is done by deleting the profile. If the installation determines that a group or a user no longer requires superuser authority, the RACF administrator can remove the user from the access list with the PERMIT or REMOVE command:

```
PE BPX.SUPERUSER CLASS(FACILITY) ID(UNIXSUP) DELETE
RE JANE GROUP(PROG1)
```

3.1.4 Switch to superuser mode

For users that are given access through the BPX.SUPERUSER profile in the FACILITY class, you can use any of the methods described in this section to gain superuser authority.

SU command

Enter the shell using the OMVS command and then issue the `su` command with no operands. This creates a nested shell (a new process) that runs with superuser authority.

When running in this manner, editing a file with the OEDIT command (OEDIT with PF6) returns you to the TSO/E address space where your original authority is still in place.

New to z/OS V1R5 is an **su** option that starts a new shell as a login shell.

```
su [-] [-s] [userid [arg ...]]
```

Using the **su** command as shown in Figure 3-2, a child shell will be started with the login environment of the admin user ID.

```
su admin
```

Figure 3-2 su command

By using the **su** command this way, you will get:

- Admin's default shell
- Admin's HOME directory
- The ability to run /etc/profile and admin's .profile to set the environment variables

In another example, shown in Figure 3-3, you can see it is also possible to issue commands under a surrogate user ID. The command runs the remove **rm** shell command under the admin user ID and returns to the invoker when the command ends.

```
su admin -c "rm -rf /tmp/"
```

Figure 3-3 su command

To switch to another userid without having to specify a password, you have to be granted RACF SURROGAT authority. Figure 3-4 shows an example of the RACF statements used to obtain this authority.

```
RDEFINE SURROGAT BPX.SRV.HERING UACC(NONE)
PERMIT BPX.SRV.HERING CLASS(SURROGAT) ID(PATRICK) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

Figure 3-4 RACF commands for granting SURROGAT authority

Without the proper authorization, user PATRICK will receive the message shown in Figure 3-5 when trying to switch to user ID HERING.

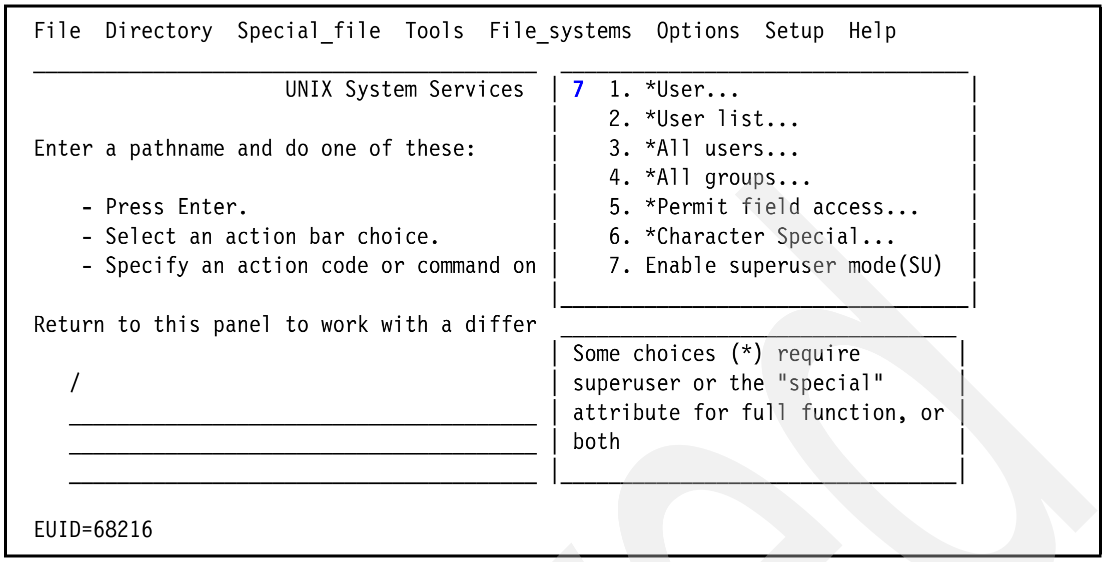
```
PATRICK @ SC64:/u/patrick>su -s hering
FSUM5027 su: User is not a surrogate of "hering".
```

Figure 3-5 Switching a user ID

Note: It is also possible to enter the **su** command after you logged on to the shell using **rlogin** or **telnet**.

Enable superuser mode(SU) panel option

Enter the ISPF shell using the ISHELL command or a dialog selection. From the ISPF shell, click **Setup** and specify option **7**, Enable superuser mode(SU) to switch to superuser state (Figure 3-6 on page 86). You can then manage the file system using ISPF shell functions while in the superuser state.



```
File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
__________________________________________  ___________________________________
                     UNIX System Services  | 7  1. *User...                    |
                                           |    2. *User list...               |
Enter a pathname and do one of these:      |    3. *All users...               |
                                           |    4. *All groups...              |
    - Press Enter.                         |    5. *Permit field access...     |
    - Select an action bar choice.         |    6. *Character Special...       |
    - Specify an action code or command on |    7. Enable superuser mode(SU)   |
                                           |___________________________________|
Return to this panel to work with a differ  __________________________________
                                           | Some choices (*) require         |
   /                                       | superuser or the "special"       |
   _______________________________________ | attribute for full function, or |
   _______________________________________ | both                             |
   _______________________________________ |__________________________________|

EUID=68216
```

Figure 3-6 ISHELL panel

If you enter the ISPF shell, switch to superuser and then exit the ISPF shell, you may lose superuser authority. If the ISPF shell is the only process in the address space, you will lose all connection to kernel services when the ISPF shell terminates. If there is another dubbed process in this address space (for example, another ISPF shell, or a local shell), it will share the UID with the ISPF shell process. For example, you can open an ISPF shell on both sides of a split screen. When you toggle to superuser in one ISPF shell, it affects the address space; therefore, both ISPF shells are now superuser. Regardless of which ISPF shell terminates first, the address space retains its UIDs until the ISPF shell is used to toggle back, or the last process is undubbed.

Note: Notice the effective user ID (EUID) on the left bottom of the ISHELL panel in Figure 3-6. This will change to a UID of zero when the user switches to superuser mode using this ISPF panel.

After gaining superuser authority in the ISPF shell, you can split the screen in ISPF and enter the OMVS command. The shell that is started inherits the superuser authority set up in the ISPF shell.

Note: For privileged shells (when the effective UID does not match the real UID, or the effective GID does not match the real GID) $HOME/.profile is not run. If the file /etc/suid_profile exists, it will be run.

REXX

If you are permitted to the BPX.SUPERUSER resource you can get superuser access through REXX.

Some REXX examples to gain superuser access can be found on the internet at the zSeries “tools & toys” Web site:

http://www-1.ibm.com/servers/eserver/zseries/zos/unix/bpxa1toy.html

In particular, some of the tools written by Robert Hering can be very useful.

http://www-1.ibm.com/servers/eserver/zseries/zos/unix/toys/usstools.html

BPXBATCH

Use the **su** command from a BPXBATCH submitted job. An example of a job using BPXBATCH is shown in Figure 3-7.

The command shown in the **PARM=** statement pipes (symbol **|**) the result of the **echo** command (that is, the **copy** command) into the **su** command. Thus, you are able to change file ownership by becoming superuser.

```
//STEP01 EXEC PGM=BPXBATCH, PARM='SH echo chown antoff /u/antoff | su'
//STDIN DD PATH '/yourpath/input.stuff',PATHOPTS=(ORDONLY)
```

Figure 3-7 BPXBATCH job using the su command

With no parameters coded at all, create a file that has the **su** command in it and run it as a script.

```
su
chown antoff /u/antoff
```

In the suinput.stuff file, shown in Figure 3-8, you would have the **su** command followed by the **chown** command. These are commands as you would have entered them from the console if you had been running in the z/OS UNIX shell.

```
//STEP01 EXEC PGM=BPXBATCH
//STDIN DD PATH='/yourpath/suinput.stuff',PATHOPTS=(ORDONLY)
```

Figure 3-8 BPXBATCH job using the su command

Important: When you set up your own $HOME/.profile as superuser, specify the /usr/sbin directory in your PATH environment variable because certain superuser utilities are in that directory instead of the /bin directory, such as automount. For more information about the profile, see *z/OS UNIX System Services Planning,* GA22-7800.

3.1.5 Assigning a UID of 0

Although sometimes appropriate, it is often desirable to have as few superusers with a UID of 0 as possible in the UID parameter in the OMVS segment of the ADDUSER or ALTUSER commands.

Also, consider assigning a UID of 0 to a user for superuser capability and assign a secondary user ID with a nonzero UID for activities other than system management. For example, you could assign:

- User ID CONWAY UID(0) - used for system maintenance
- User ID CONWAY1 UID(7) - used for regular programming

Assign a user to be a superuser

In the following example, the ALTUSER command gives the TSO/E user ID CONWAY superuser authority, makes the root directory the home directory, and causes invocation of the shell in response to a TSO/E OMVS command.

```
ALTUSER CONWAY OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
```

3.1.6 Managing UIDs

There have been enhancements to UID assignment in z/OS Version 1 Release 4. You may want to evaluate all of your current UID assignments. The changes in z/OS V1R4 are as follows:

- Prevention of shared UIDs and GIDs

  This support allows an installation to prevent more than one user from having the same UID. Using this support does not eliminate UIDs that are currently being shared, especially those that are UID(0). For more detail on prevention of shared UIDs, see 3.10.2, “Shared UID and GID prevention” on page 119.

- Automatic UID and GID assignment

  For non UID(0) users, you can have RACF choose the next available UID or GID. For more detail, see 3.10.1, “Automatic UID and GID assignment” on page 115.

Search for users with the same UID

z/OS V1R4 introduces a search for users with the same UID or GID. The RACF **search** command has the following changes to display user and group profiles with z/OS V1R4:

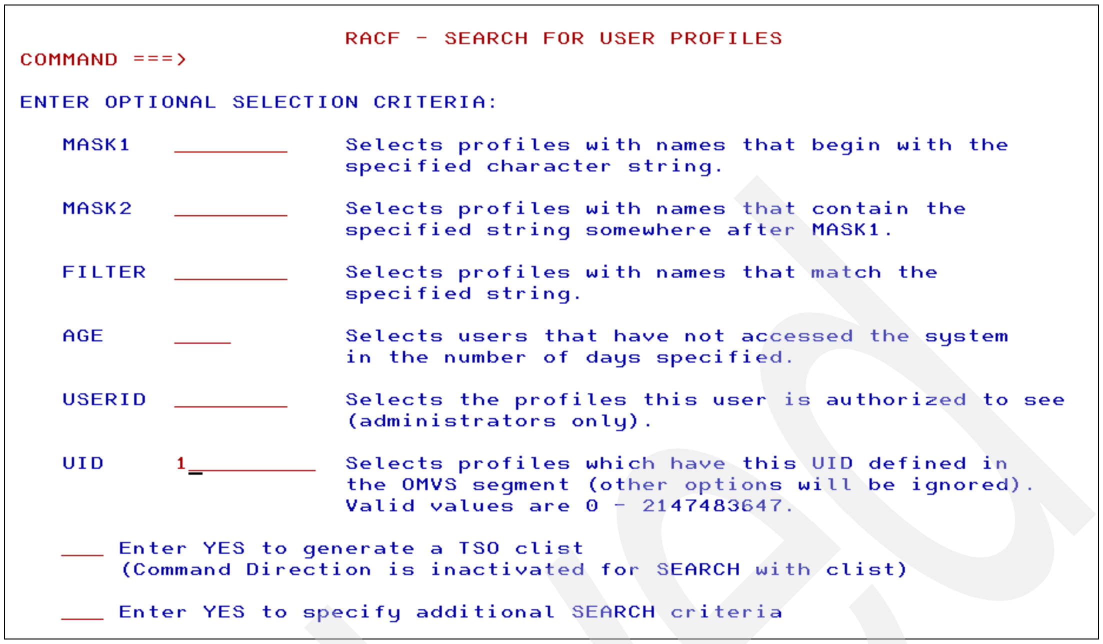
- User profiles that contain an OMVS UID equal to the UID you specify.
- Group profiles that contain an OMVS GID equal to the GID you specify.

Search command examples

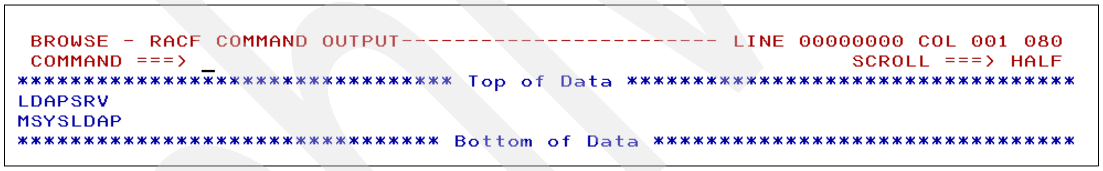
The following commands are examples followed by the response:

```
search class(group) gid(1)
OMVSGRP

search class(user) uid(0)
HAIMO
BOBH
BPXROOT
CONWAY
LDAPSRV
MSYSLDAP
```

If you use RACF panels for using the search function, there is a new option for displaying duplicate UIDs, as shown in Figure 3-9 to display all UID(1).


```
                              RACF - SEARCH FOR USER PROFILES
COMMAND ===>

ENTER OPTIONAL SELECTION CRITERIA:

   MASK1     ________     Selects profiles with names that begin with the
                          specified character string.

   MASK2     ________     Selects profiles with names that contain the
                          specified string somewhere after MASK1.

   FILTER    ________     Selects profiles with names that match the
                          specified string.

   AGE       ____         Selects users that have not accessed the system
                          in the number of days specified.

   USERID    ________     Selects the profiles this user is authorized to see
                          (administrators only).

   UID       1_________   Selects profiles which have this UID defined in
                          the OMVS segment (other options will be ignored).
                          Valid values are 0 - 2147483647.

   ___ Enter YES to generate a TSO clist
       (Command Direction is inactivated for SEARCH with clist)

   ___ Enter YES to specify additional SEARCH criteria
```
Figure 3-9 RACF panel to display user IDs having the same UID

The output from the display is displayed on the screen, as shown in Figure 3-10.


```
 BROWSE - RACF COMMAND OUTPUT------------------------ LINE 00000000 COL 001 080
 COMMAND ===> _                                                SCROLL ===> HALF
********************************* Top of Data **********************************
LDAPSRV
MSYSLDAP
******************************** Bottom of Data ********************************
```
Figure 3-10 Output from RACF panel display of duplicate UIDs

Reassign UIDs of existing users

If you wish to remove users with UID(0) or any other UID and use automatic UID assignment, do the following:

Delete the UID from the user's OMVS segment and then issue the ALTUSER command with the AUTOUID keyword:

```
ALU JOHN OMVS(NOUID)
ALU JOHN OMVS(AUTOUID)
```

3.2 Creating a RACF environment for products and subsystems

Good naming standards for all resources in the z/OS environment are a necessary condition, but not a sufficient condition to have a neat and tidy, easy to manage RACF database. The sufficient condition is the permanent, everyday enforcement of these standards without any compromise.

For the purpose of our further presentation, we assume that you have naming standards for several types of groups and user IDs. Also we assume that you have a policy for ownership of resources.

3.2.1 RACF group structure

An important condition for a good RACF database is its group structure. It is outside the scope of this book to deal with the best way to design such a structure. However, it is sufficient to recommend a very simple structure, as shown in Figure 3-11.

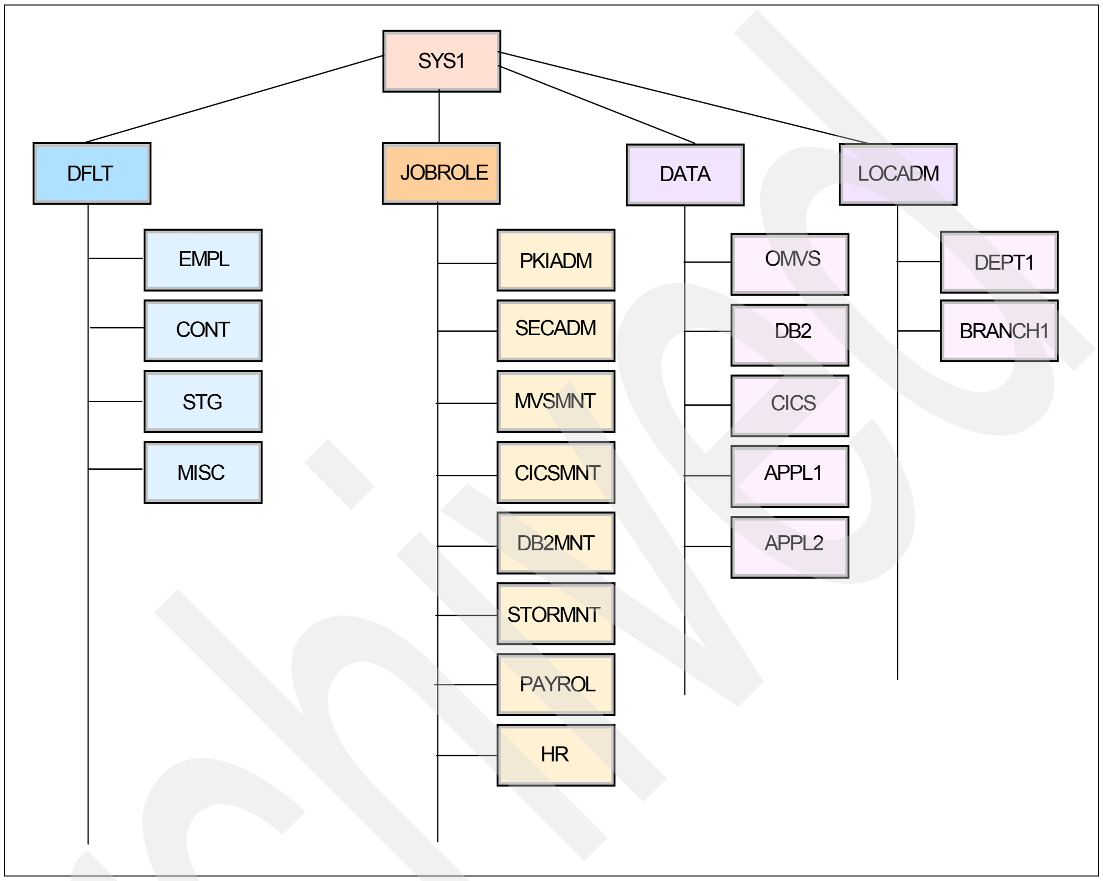


Figure 3-11 Very simple RACF group structure example

In the example in Figure 3-11, the subgroups of group SYS1 are as follows:

DFLT	The subgroups are all default groups.
JOBROLE	The subgroups are all jobrole groups).
DATA	The subgroups are all groups matching high-level qualifiers for data sets.
LOCADM	The subgroups are all groups for decentralized RACF administration, containing users with authority of GROUP-SPECIAL.

Note: You should not connect any user IDs to these four groups.

Default group for started task user IDs

As part of the job role group structure implementation, we created a RACF default group for all started task user IDs on our systems. You may have such a group with names STC, STCG, and STCGRP. Our preferred name for the example is STG and the command to create it is:

```
AG STG SUPGROUP(DFLT) OWNER(DFLT)
```

Note: The group DFLT is a subgroup of group SYS1 and has as subgroups all default groups, for example default group EMPL for all employees, default group CONT for all contractors and vendors, default group STG for all started task user IDs, and default group MISC for miscellaneous machine user IDs). Default groups should never be permitted to any resource.

Jobrole groups

In this redbook, we introduce the concept of the *jobrole* RACF group as an example. The user IDs connected to such a group need the same access to various resources in order to do their everyday jobs. For example, the group of MVS systems programmers, let's call it MVSMNT, needs access ALTER to all operating system data sets and all data sets related to software products. However, they do not even need READ access to data sets storing data from financial, human resources, or other business applications. When an MVS systems programmer joins or leaves the team, the RACF administrator needs only to connect or remove him to or from the group MVSMNT. In order for such a concept to be implemented, a lot of preliminary work is needed, such as interviews with representatives of all areas of the enterprise, logical assessment by a RACF architect, and security policy reviews, followed by the difficult task of actually reengineering the whole RACF group structure.

If you have applications running in your z/OS UNIX environment, such as PKI server or Web servers, you have to extend the jobrole concept into the z/OS UNIX environment in order to work out which jobrole groups will be the *owning groups* of HFS files, and their level of access.

We assume that a jobrole group structure is implemented at least in the Information technology department of your organization. We will refer to the z/OS UNIX system groups as follows:

- The UNIX systems programmers group as USSMNT
- The security administrators group as SECADM
- The decentralized security administrators groups, all subgroups of LOCADM, as LOCADM

Note: To repeat: you should not connect users to group LOCADM, but to its subgroups, for example DEPT1 and BRANCH1, as shown in Figure 3-11 on page 90.

3.2.2 Creating user IDs

It is necessary to create user IDs for the various subsystems, programs, and procedures used in a z/OS environment.

Started task user IDs

In this section we refer to the started task user IDs (STUs) necessary to run procedures (programs for various subsystems and products). In the z/OS UNIX environment, the STUs are called daemons. We assume that your site has a naming standard for started task user IDs (STUs), as shown in Figure 3-12 on page 91.

```
productnameTASK
STCproductname
productnameSTC
productnameSTU
```

Figure 3-12 Possible naming standards for started task user IDs

We prefer the last naming standard. In the framework of the job role concept for group structure, we recommend that STUs are never connected to job role groups, but permitted individually to RACF profiles, instead.

In your z/OS UNIX environment you may need to make some STU owners of HFS files and directories and permit them to these files and directories with a suitable level of access.

Miscellaneous user IDs

Here we refer to various miscellaneous user IDs such as batch submission ids, CICS default users, console IDs, and surrogate user IDs. They should have their own default group, for example MISC, and should be permitted individually to RACF profiles.

3.2.3 System data set profiles

Most installations have used well established naming standards for data sets for many years. Usually all IBM system software resides in data sets prefixed with SYS1 while in-house modifications to software, as well as system data, reside in data sets prefixed with SYS2, or other user-defined names. Our preferred prefix is SYSU.

3.2.4 Ownership

Although OWNER may not have any significance in RACF except when a decentralized administration is in place, we recommend that all resources have a meaningful OWNER (always a RACF group), for example:

- User IDs are owned by their default groups.
- System data set profiles are owned by MVSMNT.
- CICS software data sets are owned by CICS systems programmers.
- Application data sets are owned by respective business groups.
- Security profiles (BPX, IRR in the RACF FACILITY class) are owned by SECADM.

3.3 The RACF database and z/OS UNIX

Associating RACF user IDs and groups to UIDs and GIDs has important performance considerations.

There are two considerations to improve performance for access to UIDs and GIDs, the most important one being Application Identity Mapping (AIM) because some new functions in z/OS UNIX require AIM, beginning with z/OS V1R4:

- Identity Mapping using the Virtual Lookaside Facility (VLF) and the UNIXMAP class

  If your installation shares the RACF database with systems running releases prior to OS/390 Version 2 Release 10, it is important to use the VLF classes IRRUMAP and IRRGMAP and the RACF UNIXMAP class to improve performance by avoiding sequential searches of the RACF database for UID and GID associations. RACF uses the VLF caching to search for UIDs or GIDs, which is known as identity mapping, as shown in Figure 3-13. Identity mapping for RACF user IDs and groups started with the introduction of the RACF UNIXMAP class in OS/390 V2R7 with the intention to improve system performance. The UNIXMAP class is used to allow the system to quickly look up a user ID from a UID, or a group name from a GID.

- Application Identity Mapping

  If your installation shares the RACF database only with systems running z/OS, or OS/390 Version 2 Release 10 or above, you may be able to achieve improved performance without using UNIXMAP and VLF. However, before you can avoid using UNIXMAP and VLF, you must ask your systems programmer if your installation has reached stage 3 of Application Identity Mapping by running the IRRIRA00 conversion utility. There are four stages you must complete to convert the RACF database to AIM, stages 0, 1, 2, and 3.

Note: If your installation is new to RACF and you are not running any releases prior to OS/390 Version 2 Release 10, you will automatically take advantage of application identity mapping at the stage 3 level without running the IRRIRA00 conversion utility, and you will not need to use VLF and UNIXMAP to achieve improved performance.

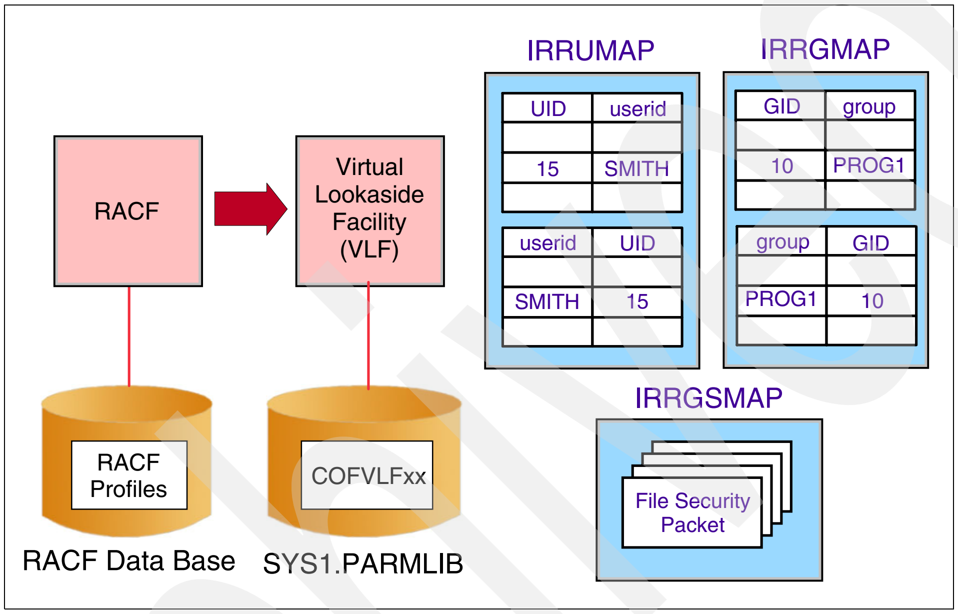


Figure 3-13 RACF and VLF caching to improve performance

The introduction of AIM in OS/390 V2R10 and the modification of the RACF database with the new IRRIRA00 utility were the first attempt to map RACF user IDs using various applications across the enterprise to one entity (alias index).

3.4 Identity mapping with VLF and UNIXMAP

If your installation shares the RACF database with systems running releases prior to OS/390 V2R10, it is important to use Virtual Lookaside Facility (VLF) and the UNIXMAP class to improve performance. You may also need to use the VLF and UNIXMAP class if your system programmer has not yet converted your systems for stage 3 of AIM.

Both VLF and the UNIXMAP class can be either active or inactive. Table 3-1 on page 93 shows how these states affect performance.

Table 3-1 The UNIXMAP class and VLF: effects on performance

State	Performance
Active class UNIXMAP Active VLF	Running in this state at all times will give you the best performance.

State	Performance
Active class UNIXMAP Inactive VLF	If VLF is inactive, requests for UID-to-user-ID mapping and GID-to-group-name mapping must access a UNIXMAP class profile in the database, which degrades performance. Running with VLF inactive should be done only when you need to stop VLF to make changes to it.
Inactive class UNIXMAP Active VLF	If the UNIXMAP class is inactive, requests for UID-to-user-ID mapping and GID-to-group-name mapping must search the entire RACF database when the UID or GID specified is not found in VLF. Running in this state degrades performance severely. The inactive state for the UNIXMAP class is provided as a migration aid. After migration is complete, you should never need to run with the UNIXMAP class inactive.
Inactive class UNIXMAP Inactive VLF	Running with both VLF inactive and the UNIXMAP class inactive causes requests for UID-to-user-ID mapping and GID-to-group-name mapping to default to searching the RACF database on each request. Running in this state significantly degrades performance of these functions. It could also affect other systems in a complex sharing the RACF database because of the increased I/O to the database. It is recommended that you never run in this state.

3.4.1 VLF PARMLIB definitions

It is recommended that both the UNIXMAP class and VLF remain active, and that the VLF classes IRRUMAP and IRRGMAP should be defined to VLF by updating the COFVLFxx member of SYS1.PARMLIB to include the following:

```
CLASS NAME(IRRGMAP)          /* GMAP table for z/OS UNIX System Services */
EMAJ (GMAP)                  /* Major name = GMAP */
CLASS NAME(IRRUMAP)          /* UMAP table for z/OS UNIX System Services */
EMAJ (UMAP)                  /* Major name = UMAP */
```

You have the option to cache additional z/OS UNIX security information in VLF. This capability allows RACF to avoid accessing the RACF database when called to create a security environment for z/OS UNIX users. To use the cached User Security Packet (USP), the IRRSMAP class, shown in Figure 3-13 on page 93, must be defined to VLF by updating the COFVLFxx member of SYS1.PARMLIB to include the following:

```
CLASS NAME(IRRSMAP)          /* SMAP table for z/OS UNIX System Services */
EMAJ (SMAP)                  /* Major name = SMAP */
```

Note: Do not confuse the VLF classes with RACF general resource classes. They are totally unrelated.

3.4.2 Populating and activating the UNIXMAP class

The UNIXMAP class is used for UID and GID lookups. Run the RACF database unload utility (IRRDBU00) against the RACF database having OMVS segments to unload all of the profiles in the database which produces a file containing RDEFINE and PERMIT commands that will be used to populate the UNIXMAP class.

Note: Make sure that you issue a `setr list` command and look for: ADDCREATOR IS NOT IN EFFECT. If ADDCREATOR IS IN EFFECT, then issue the command SETR NOADDCREATOR.

When you run the RACF commands, you may see messages ICH408I and ICH10102I, indicating that some profiles are already defined to the UNIXMAP class. This occurs if a UID maps to more than one user ID or if a GID maps to more than one group.

You must activate the UNIXMAP class to cause it to be used, as follows:

```
SETROPTS CLASSACT(UNIXMAP)
```

z/OS UNIX can be active while the initial population takes place. From this point, RACF automatically keeps the UNIXMAP profiles synchronized with the user and group profiles.

UNIXMAP class profiles to map UIDs and GIDs

For each UID that is defined in the OMVS segment of a USER profile, a profile named Uuid in the UNIXMAP class is automatically created. The access list of the Uuid profile contains all user IDs that have been assigned this UID.

For each GID that is defined in the OMVS segment of a GROUP profile, a general resource profile named Ggid in the UNIXMAP class is automatically created. The access list of the Ggid profile contains all groups that have been assigned this GID.

These mapping profiles are used to provide a cross reference to USER and GROUP profiles. They provide RACF with a performance-sensitive method of returning information for a given UID or GID when requested by z/OS UNIX or application programs.

Changing UIDs and GIDs

RACF automatically maintains these mapping profiles when UIDs and GIDs are added, changed, or deleted. The UNIXMAP class does not have to be active for this to happen. RACF does this by modifying UNIXMAP class profiles appropriately when ADDUSER, ALTUSER, DELUSER, ADDGROUP, ALTGROUP, or DELGROUP commands are issued. When RACF creates UNIXMAP profiles as a result of an ADDUSER, ALTUSER, ADDGROUP, or ALTGROUP command, the user ID of the command issuer becomes the owner of the UNIXMAP profile.

For example, if the following command is issued:

```
ADDUSER ANTOFF OMVS(UID(340))
```

RACF creates a UNIXMAP profile named U340 with ANTOFF on the access list. If the following command is subsequently issued:

```
ALTUSER ANTOFF OMVS(UID(341))
```

RACF deletes the U340 profile and creates a U341 profile with ANTOFF on the access list.

Problems with profiles

In general, you should not alter these profiles. However, it is possible that they might get inadvertently deleted, or damaged by database corruption. If a profile is deleted, or if the user is not on its access list, RACF will not be able to retrieve information for the UID or GID that the profile represented. RACF will be unable to locate the mapping profile and will send z/OS UNIX a return code indicating that the UID or GID is not known.

If this happens, a RACF administrator needs to repair the damage. First, see if the user name associated with the UID or the group name associated with the GID can be determined from a message displayed by RACF. For example, suppose you received an error message associated with user ANTOFF. You should display the UID associated with the user profile for ANTOFF by entering:

```
LU ANTOFF OMVS NORACF
```

If, for example, LU displays a UID of 340, you would then enter:

```
RDEF UNIXMAP U340 UACC(NONE)
PE U340 CL(UNIXMAP) ID(ANTOFF) ACCESS(NONE)
```

If you are unable to determine the user name or group name from a RACF message, look at the output from the database unload utility (IRRDBU00) to find the user ID or group associated with a given UID or GID. The mapping profiles should then be added, changed, or deleted as appropriate to be consistent.

3.5 Application identity mapping

This section discusses the application identity mapping (AIM) function, and how to migrate RACF user IDs. Application identity mapping provides an improved method for associating identities defined by the following:

- z/OS UNIX
- Novell Directory Services for OS/390
- Lotus Notes® for z/OS applications to RACF user IDs

The IRRIRA00 utility migrates the UNIXMAP, NOTELINK, and NDSLINK mapping profiles to alias entries in four stages (from 0 to 3). Updates to the ADDUSER and ALTUSER commands prevent you from associating application user ID entities for Lotus Notes for OS390 and Novell Directory Services for OS/390 with more than one RACF user ID.

AIM requires the following z/OS UNIX functions:

- Shared UID prevention requires at least AIM stage 2.
- Automatic UID and GID assignment requires at least stage 2.
- Removal of VLF and UNIXMAP requires AIM stage 3.

AIM in its final stage, stage 3, is an alternative to the use of mapping profiles to associate RACF user and group names with z/OS UNIX, Lotus Notes, and Novell Directory Services identifiers. For these associations, the IRRIRA00 utility converts the database mapping profile information into an alias index, which uses less space. This conversion is accomplished through a series of stage transitions from an initial stage 0 to the completed conversion in stage 3. It is important to verify that your applications relying on the alias information continue to execute properly through the interim stages. Changes made to RACF user and group commands and callable services to support the alias indexes are intended to be transparent. However, you need to modify any application code that references or manipulates the mapping profiles directly to use the standard interfaces.

3.5.1 RACF IRRIRA00 utility

The conversion utility, IRRIRA00 (the Internal Reorganization of Aliases utility) can process an existing RACF database in four stages controlled by the installation and lists the current stage of its input RACF database. The utility updates all active RACF data sets, including active backup data sets; all data sets making up a RACF database must be at the same stage.

The IRRIRA00 utility converts a RACF database created before OS/390 V2R10 to a database that supports the new alias identity mapping function. The utility runs only against an active primary and backup database and it serializes against RACF resources to prevent disruptive competing updates. You can run the utility while the database is shared between systems. The sysplex Coupling Facility is updated as needed.

Important:

- Before advancing the stage of your database, make a copy of the database for recovery purposes.
- If you are sharing a database with a lower level system, review “Using Application Identity Mapping” in *z/OS Security Server RACF System Programmer’s Guide,* SA22-7681 before proceeding with the conversion.

The conversion of an existing RACF database can take place in the following four stages:

Stage 0 The database does not have an alias index and the RACF database manager does not attempt to use or maintain the alias index. It continues to use the mapping profiles. Any database created earlier than OS/390 Release 10 exists in stage 0 automatically until you convert it with IRRIRA00.

Stage 1 In stage 1, database contains the existing mapping profiles and the new alias index. RACF uses VLF and the mapping profiles to locate a base USER or GROUP profile name that has been given another product’s identity information. The RACF database manager maintains an alias index but does not use it to locate user or group names. RACF user commands (AU/ALU/DU) and GROUP commands (AG/ALG/DG) maintain both the mapping profiles and the alias index entries during addition, modification, or deletion of USER and GROUP profiles.

Note: Before entering this stage, run IRRMIN00 PARM=UPDATE and IPL the system if it was not done during previous migration steps.

Stage 2 In stage 2, RACF maintains both alias index entries and mapping profiles. The RACF database manager can use the alias index to locate user and group names. At this stage, the identity mapping callable services look up application IDs in an alias index to retrieve corresponding RACF user or group names. If the entry is not found in the index, RACF searches through VLF, mapping IRRIRA00 utility profiles, or base profiles depending on the alias type and active classes. You must resolve any problems before continuing to stage 3.

Stage 3 In stage 3, RACF uses only alias index entries, not mapping profiles, for UID, GID, SNAME, and UNAME associations. Commands such as ADDUSER no longer maintain the old mapping profiles. Entries are not placed in VLF, which is no longer used to locate profiles. You can remove the IRRUMAP and IRRGMAP VLF classes from the COFVLFxx member of SYS1.PARMLIB. You can also deactivate the RACF classes UNIXMAP, NOTELINK, and NDSLINK.

Attention: Before advancing to stage 3

- You can advance to stage 3 after successfully operating in stage 2. Before entering stage 3, check the LOGREC entries and correct any errors that might have occurred when the callable services searched for alias index entries during stage 2.
- You should enter stage 3 only when all sharing systems have the OS/390 V2R10 Security Server or later installed. If you are sharing a RACF database with a system that is at a lower level, you might receive unpredictable results.

Table 3-2 gives a summary of the processing done for each stage and the processing that can be done at each level.

Table 3-2 IRRIRA00 stage summary

Stage	RACF Manager	Commands	Callable Services
0	Does not maintain alias index Purges VLF Does not allow alias index entry locates	Maintains VLF and mapping profiles	Identity search order: 1. VLF 2. Mapping profile or database search
1	Maintains alias index Purges VLF Does not allow alias index entry locates	Maintains VLF and mapping profiles	
2	Maintains alias index Purges VLF Allows alias index entry locates	Maintains VLF and mapping profiles	Identity search order: 1. Alias index entry locate 2. VLF 3. Mapping profile or database search
3	Maintains alias index Allows alias index entry locates	Does not maintain VLF and mapping profiles	Identity search order: 1. Alias index entry locate

Notes:

1. Mapping profiles are used if the appropriate class is active (for example, UNIXMAP, NOTELINK, NDSLINK). If UNIXMAP is not active, RACF searches through all the user and group profiles in the database with an OMVS segment until a match is found for the GID or UID.
2. VLF is applicable only for an OMVS UID or GID. The IRRUMAP or IRRGMAP class must be defined on COFVLFxx member in SYS1.PARMLIB.

3.5.2 AIM conversion considerations

If you create a new RACF database in OS/390 V2R10 or later, this database is automatically set to stage 3.

The IRRIRA00 utility does not provide RACF database diagnostic information. If you suspect a RACF database error, you should start your problem determination by running the IRRUT200 utility and requesting the INDEX and MAP ALL functions. For details, see "RACF Database Verification Utility Program (IRRUT200)" in *z/OS Security Server RACF System Programmer's Guide,* SA22-7681.

Note: To enter stage 3, all systems sharing the RACF database must be at RACF V2R10 or higher.

This utility produces AIM for RACF databases created before OS/390 V2R10. You do not need to run the utility against databases created with IRRMIN00 PARM=NEW for OS/390 V2R10 or later because they are already initialized for the final stage of application identity mapping.

IRRIRA00 processing

IRRIRA00 opens the master primary RACF data set and, if active, the master backup RACF data set. Access UPDATE to each data set is required to allow the data set to be opened. IRRIRA00 obtains serialization to prevent activities such as RVARY and SETROPTS commands from being processed while the utility is running. Processing of RACF commands that add, alter, and delete user and group profiles might also be delayed.

Note: You should avoid RACF administration while the utility is running. We also recommend that when you plan for the migration using our installation procedures, plan to do the changes at a quiet time or during a scheduled outage.

All primary RACF data sets must be active to allow the utility to complete successfully.

- If the primary RACF data sets are active but the backup data sets are inactive, the utility updates only the primary data sets. A message is issued to indicate that the backup database was not changed.
- If some backup data sets are active and some are inactive, an error message is issued and processing ends without updating the primary database.

IRRIRA00 runs faster when there is minimal activity on the system. For a database with a large number of mapping profiles, the utility converts from stage 0 to stage 1 in about half the time if you set the backup database inactive and run IRRIRA00 against the primary database only. You can use IRRUT200 or IRRUT400 to copy the primary database to the backup database after the utility completes successfully.

IRRIRA00 does not propagate the new alias index entries or the deleted mapping profiles to other databases with RRSF. You need to run the utility for each database when that system is ready to enter a new stage. RACF databases do not need to be at the same stage to be part of the same RRSF network unless specific code is used to manipulate mapping class profiles using RACROUTE or ICHEINTY. Command propagation works correctly between systems whose RACF databases are at different stages.

Important: If RACF is enabled for sysplex communication, whenever you need to run IRRIRA00 against a database that is active on a system that is a member of the RACF data sharing group, always run the utility from a system in the group. If you do not, you might damage your RACF database, or receive unpredictable results from the utility.

Input for IRRIRA00

In Figure 3-14 on page 99, IRRIRA00 converts an existing RACF database from stage 1 to stage 2 for the AIM function.

```
//AIMSTAGE JOB
//STEP EXEC PGM=IRRIRA00,PARM=STAGE(2)
//SYSPRINT DD SYSOUT=A
```

Figure 3-14 Sample JCL for IRRIRA00 utility

IRRIRA00 expects the following parameter on the JCL EXEC statement:

PARM=STAGE(n), with n=1,2,3 to specify the desired level of the system.

The utility does the following:

1. Checks the current level of the system to be sure it is at the n-1 level.
2. Performs the necessary actions to enable the specified state n. If no parameter is specified, the current stage is listed.

Output from IRRIRA00

A return code greater than 4 indicates that the stage conversion did not complete successfully. If appropriate, correct the errors indicated by the messages and run the utility again. IRRIRA00 issues no message when the return code is x'14' (20 decimal) because SYSPRINT cannot be opened to write the message. In this case, you should verify that the SYSPRINT DD statement is correct and that the utility can access the specified file.

3.5.3 Recovering from errors with AIM

With application identity mapping enabled at stage 3, RACF uses an alias index rather than mapping profiles to associate users and groups with z/OS UNIX, Lotus Notes, and Novell Directory Service identities. It is possible that an unexpected error could cause an association mismatch that you can identify by comparing IRRUT200 alias index output with profile information returned from LISTUSER, LISTGRP, or DBUNLOAD. This section suggests methods to correct such inconsistencies.

At stages below application identity mapping stage 3, RACF maintains mapping profiles and functionality to ensure mapping compatibility with systems running RACF at the OS/390 V2R10 level or below that share a database with higher-level systems. You should use program control to be sure that USER and GROUP commands can only be issued from systems running RACF on OS/390 V2R10 or higher. After all systems sharing the database are migrated OS/390 V2R10 or higher, run IRRIRA00 to advance the database to stage 3, thereby reducing the likelihood of mapping errors.

Mapping profile exists

If your database is at application identity mapping stage 3, no generic profiles in class UNIXMAP, NOTELINK, or NDSLINK should exist. If you find one, you can ignore it just as RACF does, or you can delete it using RDELETE. For example:

```
RDELETE UNIXMAP U1
```

If the mapping profile contains lowercase letters, you cannot specify them on the RDELETE command. You must use BLKUPD or RACROUTE to delete the profile.

Missing alias index entry

If your database is at stage 0, you should not expect to see any alias index entries. If your database is at a higher stage and you do not find an alias index entry corresponding to a specific UID, GID, SNAME, or UNAME, you can regenerate the entry by altering the user or group profile with the desired entry. For example:

```
ALTUSER YOURID OMVS(UID(1))
```

User or group associated with an alias index entry does not exist

If the profile associated with and alias index entry does not exist, you can remove the entry by temporarily adding the referenced profile with the indicated alias, then deleting the profile. For example:

```
ADDUSER YOURID OMVS (UID(1))
DELUSER YOURID
```

Profile and alias index mismatch

If an alias index entry references the incorrect user or group, you can correct the index by altering the incorrect profile that references the given alias entry, altering it again to reference another alias entry, and finally altering the desired profile to reference the given alias entry.

For example, if the alias index entry for UID 1 references MYID rather than the desired YOURID:

```
ALTUSER MYID OMVS(UID(1))
ALTUSER MYID OMVS(UID(2))
ALTUSER YOURID OMVS(UID(1))
```

3.6 RACF utilities and IRRIRA00

The following RACF utilities are useful before running the IRRIRA00 utility and for the management of UIDs and GIDs in your z/OS UNIX environment. For the conversion to AIM, run these utilities in the order shown before running the IRRIRA00 utility, for the following reasons:

- IRRDBU00 to download the RACF database to a flat file.
- RACFICE to create a report on UIDs and GIDs to determine the number of shared UIDs or GIDS.
- IRRUT400 to create a backup copy of the RACF database.
- ITTUT200 to obtain statistics on UIDs and GIDs and the number of UNIXMAP entries for reference purposes for the migration to AIM.

A description of these utilities follows:

IRRDBU00 The RACF database unload utility unloads the RACF database to a sequential file. For information on how to use IRRDBU00, see *z/OS Security Server RACF Macros and Interfaces*, SA22-7682 and *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683.

RACFICE The RACFICE reporting tool allows an installation to create tailored RACF reports without requiring a relational database management product, and provides an alternative to the RACF report writer. It makes use of the DFSORT™ ICETOOL reporting facility. RACF makes several ICETOOL-based reports available in SYS1.SAMPLIB. The RACJCL member of SYS1.SAMPLIB provides sample JCL to allocate a report data set and add the RACFICE reports in IEBUPDTE format. The RACFICE member provides the IEBUPDTE-format ICETOOL and DFSORT control statements that implement the RACFICE reports. This utility reports GIDS and UIDs from member IRRICE of SYS1.SAMPLIB to obtain information about the number of shared GIDs/UIDs. For a complete description of the RACFICE reporting tool, see *OS/390 Security Server 1999 Updates: Installation Guide,* SG24-5629.

IRRUT400 IRRUT400 copies a RACF database to a larger or smaller database, provided there is enough space for the copy. IRRUT400 also can redistribute data from RACF databases. For example, IRRUT400 can split a single data set in the RACF database into multiple data sets, merge multiple data sets in the RACF database (previously split) into fewer data sets, or rearrange RACF profiles across the same number of input and output RACF data sets. Though the utility allows a maximum of 255 input data sets and 255 output data sets, MVS allows RACF to have up to 90 data sets in the primary database and up to 90 corresponding data sets in the backup database.

IRRUT200 IRRUT200 is a RACF utility program that you can use to identify inconsistencies in the internal organization of each data set comprising a RACF database and to make an exact copy of a RACF data set. You can also use it to monitor the usable space in a data set.

IRRUT200 can be used to obtain statistics on UID/GIDs and the number of UNIXMAP, NOTELINK and NDSLINK for reference purposes before and after the migration to an AIM converted RACF database.

3.7 Defining and managing z/OS UNIX users and groups

With z/OS UNIX, a security product is required for the management of users and groups. The following considerations for the security definitions when using RACF to manage z/OS UNIX group identifiers (GIDs) and z/OS UNIX user identifiers (UIDs) include:

- User validation
- File access checking
- Privileged user checking
- User limit checking

z/OS UNIX users can be defined using RACF commands. When a job starts or a user logs on, the user ID and password are verified by RACF. When an address space requests a z/OS UNIX function for the first time, RACF does the following:

- Verifies that the user is defined as a z/OS UNIX user.
- Verifies that the user's current connect group is defined as a z/OS UNIX group.
- Initializes the control blocks needed for subsequent security checks.

The user’s security environment for making access decisions is shown in Figure 3-15 on page 102.

- The accessor environment element (ACEE) is a RACF control block that contains a description of the current user's security environment, including user ID, current connect group, user attributes, and group authorities. An ACEE is constructed during user identification and verification.
- The effective UID and effective GID (user security packet (USP)) of the process is used in determining access decisions. The only exception is that if file access is being tested, rather than requested, the real UID and GID are used instead of the effective UID and GID. The real and effective IDs are generally the same for a process, but if a setuid or setgid program is executed, they can be different.
- The FSP packet, consisting of UID, GID and permission bits (base and extended entries).

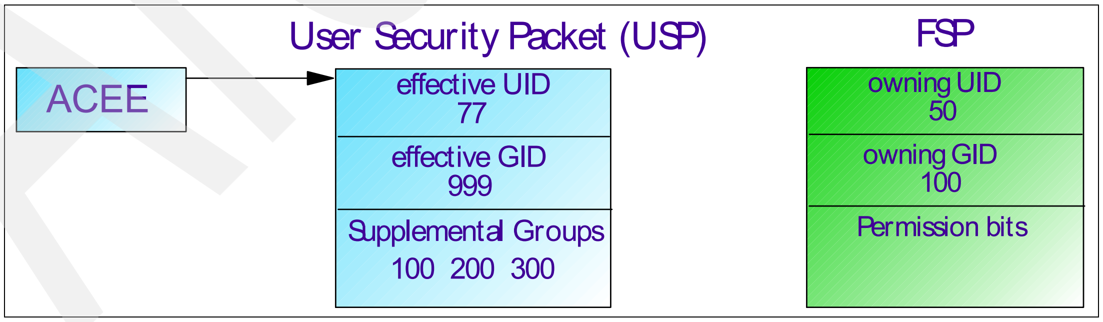


Figure 3-15 User’s security environment for access checking

z/OS UNIX users are TSO/E user IDs with a RACF segment called OMVS defined for z/OS UNIX use. All users that want to use z/OS UNIX services must be defined as z/OS UNIX users. Similar to users in a UNIX system, z/OS UNIX users are identified by a UID (user identification). The UID has a numerical value.

There are two types of users:

- A user (regular user)
  - Identified by a non-zero UID
- A superuser (an authorized or privileged user) can be any of the following:
  - A z/OS UNIX user with a UID=0
  - A started procedure with a trusted or privileged attribute in the RACF started procedures table

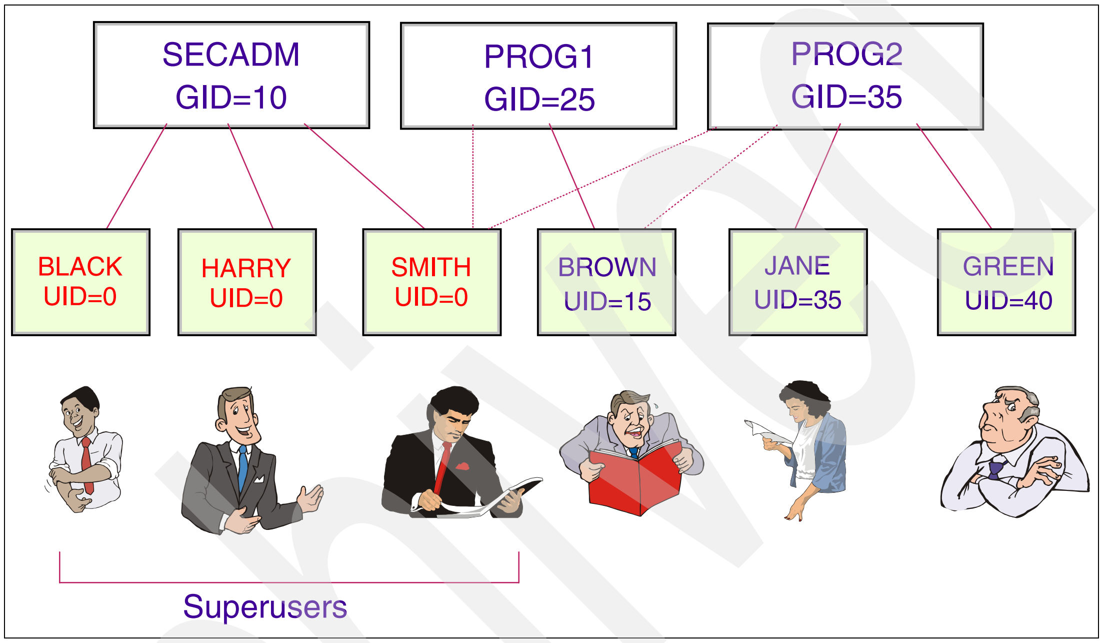


Figure 3-16 Regular users and superusers

The concept of superuser comes from other UNIX platforms. It is also referred to as *root* authority.

3.7.1 Superuser authority

An installation defines certain systems programmers, administrators and started task user IDs as superusers. These users can:

- Change the contents of any file
- Install products
- Manage processes
- Perform administrative activities

When not doing activities that require superuser authority, that person may switch from superuser status to an ordinary user status, which permits access to their own files and certain files to which they have access, according to permission bits.

A UID of 0 is used to define a z/OS UNIX as a superuser. It is good security practice to seek to minimize the assignment of superuser authority to personal user IDs and even to started task user IDs in your installation. You can accomplish this by setting z/OS UNIX user limits above the installation defaults in OMVS segments, and by managing superuser privileges through profiles in FACILITY and UNIXPRIV RACF classes.

We suggest that you have a policy for assigning superuser authority, privileges, and switching from a superuser to an ordinary user status.

3.7.2 Authentication and authorization of users to z/OS UNIX

RACF provides security for z/OS UNIX by authenticating a user and verifying that a user or program can access a process or file. It authenticates the users through user IDs and passwords when they log on to a TSO/E session or when they start a job.

Types of users requiring a RACF user profile and an OMVS segment in the user profile are as follows:

- TSO logon users who will use the z/OS UNIX shell using the OMVS command
- Remote users who will access the z/OS UNIX shell using rlogin or telnet
- Programs that use z/OS UNIX services
- Daemons
- Started procedures associated with z/OS UNIX having protected user IDs

The users must be defined in the RACF database with a user profile, an OMVS segment, and be defined to a RACF with a group profile, as shown in Figure 3-17 on page 107.

RACF user profile

The RACF user profile definition was expanded with a segment called OMVS for z/OS UNIX support. All users and programs that need access to z/OS UNIX must have a RACF user profile defined with an OMVS segment which has, as a minimum, a UID specified. A user without a UID cannot access z/OS UNIX.

You can assign a user identifier (UID) to a RACF user by specifying a UID value in the OMVS segment of his RACF user ID profile. When assigning a UID to a user, make sure that the user’s default group has an assigned GID. A user with a UID and a default group with a GID can use z/OS UNIX functions and access z/OS UNIX files based on permissions for the assigned UID and GID values.

Although you can assign the same UID to multiple users, it is not recommended. If you assign the same UID to multiple users, control at an individual user level is lost because the UID is used in z/OS UNIX security checks. Users with the same UID assignment are treated as a single user during z/OS UNIX security checks.

An OMVS segment in the user profile has nine fields. The first three fields are as follows:

UID	A number from 0 to 2147483647 that identifies an z/OS UNIX user. An z/OS UNIX user must have a UID defined.
HOME	The name of a directory in the file system. This directory is called the home directory and becomes the current directory when the user accesses z/OS UNIX. This field is optional.The home directory is the current directory when a user invokes z/OS UNIX. During z/OS UNIX processing, this can be changed temporarily by using the **`cd`** (change directory) shell command. The command will not change the value in the RACF profile. The directory specified as home directory in the RACF profile must exist (be pre-allocated) before a user can invoke z/OS UNIX. If a home directory is not specified in RACF, the root (/) directory will be used as default.

Attention: The recommended home directory for a user is /u followed by the user ID; for example, /u/antoff would be the home directory for the user ID ANTOFF. If a REXX exec or CLIST extracts the user ID with a &userid variable, the value returned is in uppercase: ANTOFF. If the REXX exec or CLIST appends the returned value to /u, the result is /u/ANTOFF. /u/antoff and /u/ANTOFF are two different directories. You should consider this behavior in using REXX execs, CLISTs, C programs, or programs using the callable services where the functions return user IDs.

PROGRAM	The name of a program. This is the program that will be started for the user when the user begins a z/OS UNIX session. Usually this is the program name for the z/OS UNIX shell. This field is optional.

The other six fields on the OMVS segment are:

CPUTIMEMAX	Maximum CPU time (RLIMIT_CPU)
ASSIZEMAX	Maximum address space size (RLIMIT_AS)
FILEPROCMAX	Maximum number of files per process
PROCUSERMAX	Maximum number of processes for this UID
THREADSMAX	Maximum number of threads per process
MMAPAREAMAX	Maximum memory map size

These options are discussed in 3.8.5, “Setting z/OS UNIX resource limits for users” on page 112.

RACF group profile

We call all RACF groups having an OMVS segment z/OS UNIX groups. The RACF group profile also has an OMVS segment for z/OS UNIX groups. It contains only one field, as follows:

GID	A number from 0 to 2147483647 which identifies a z/OS UNIX group.

Authority checks for access to hierarchical file system files and directories use the GID of the default group and up to 300 supplementary groups to make group access decisions. This type of user does not require a home directory or a program specified in the OMVS segment. The home directory and program are important for people's user IDs.

Although the same GID can be assigned to multiple RACF groups, it is not recommended. If you assign the same GID to multiple groups, control at an individual group level is lost because the GID is used in z/OS UNIX security checks. RACF groups that have the same GID assignment are treated as a single group during z/OS UNIX security checks. You can enforce identity uniqueness when assigning UNIX identifiers.

Important: If you use `pax` or `tar` commands to copy files with a UID or GID above 16777216, UIDs or GIDs may be incorrectly assigned to the restored files. Because they are commonly used utilities, you should take this problem into consideration before assigning UIDs or GIDs above 16777216.

3.7.3 Defining users and groups

RACF commands can be used to define the users and the groups using the RACF ADDUSER (AU) and ADDGROUP (AG) commands. If a RACF defined user or group is already defined in the database, you can use the ALTUSER (ALU) and ALTGROUP (ALG) commands to add an OMVS segment. Use the CONNECT (CO) command to connect a user to a group, modify a user's connection to a group, or assign the group-related user attributes.

The example in Figure 3-17 on page 107 shows a user profile for TSO/E user ID SMITH which is connected to two groups, PROG1 and PROG2. SMITH is defined as an z/OS UNIX user because he has a UID specified. His home directory is /u/smith and he will get into the shell when he issues the OMVS command because the name of the shell, /bin/sh is specified as the program name.

Assigning OMVS segments

To add an OMVS segment to an existing user profile:

```
alu smith omvs(uid(0) home('/u/smith') program('/bin/sh'))
```

Important: To define or change information in the OMVS segment of a user profile, including your own, you must have the RACF SPECIAL attribute or at least UPDATE authority to the segment through field-level access checking. To allow authorization to the entire OMVS segment of a user profile, the user would need authority to the USER.OMVS.* profile in the FIELD class. Individual fields in the OMVS segment can be defined such as USER.OMVS.UID. You can allow users to change their own HOME or PROGRAM values by creating USER.OMVS.HOME and USER.OMVS.PROGRAM in the FIELD class and permitting &RACUID to the profiles. See "Setting up field access level for OMVS segment" on page 114 for more information.

Figure 3-17 on page 107 shows that group PROG1 is also a z/OS UNIX group with a GID value of 25. A user can be connected to more than one group, and the connect group PROG2 does not have to have an OMVS segment and therefore is not an z/OS UNIX group.

To add an OMVS segment for an existing group profile:

```
alg prog1 omvs(gid(25))
```

To add a group without an OMVS segment:

```
ag prog2
```

To connect a user to defined groups:

```
connect (smith) group(prog1 prog2)
```

To list a user or group and its OMVS segment:

```
lu smith omvs
lg prog1 omvs
```

To remove the OMVS segment for a user or group:

```
alu smith noomvs
alg prog1 noomvs
```

Note: The commands **`lu smith omvs`** and **`lg prog1 omvs`** produce output that is unrelated to the OMVS segment information (it may be massive) such as subgroups and connected user IDs. If you wish to find the GID only for groups or the UID, HOME, and PROGRAM for user IDs, use:

`lu smith omvs noracf` or `lg sys1 omvs noracf`

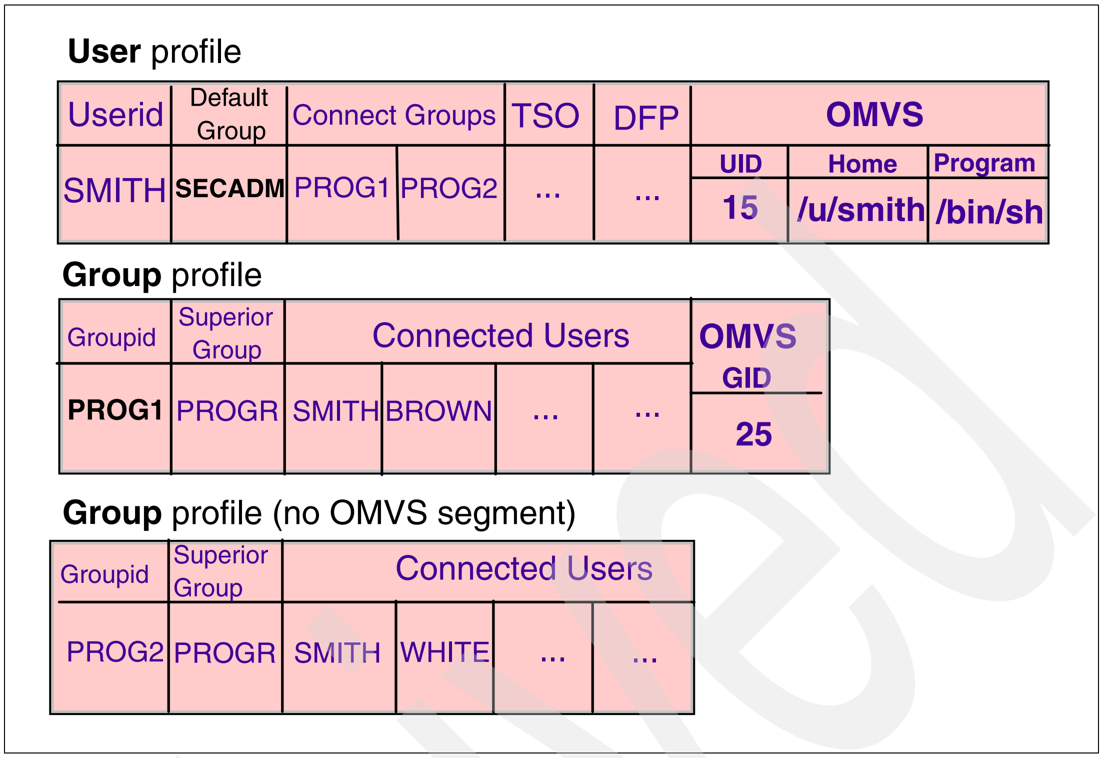


Figure 3-17 RACF-defined user with a user profile, OMVS segment, and group profile

Defining user ID considerations

To create a new RACF user ID having an OMVS segment:

```
AU ANTOFF DFLTGRP(CONT) OWNER(CONT) NAME(‘THEO ANTOFF’) PASS(*****) +
OMVS(UID(340) HOME(‘/u/antoff’) PROG(‘/bin/sh’))
```

Note: The command above is not complete. For example, the definition for a TSO segment is missing. Your procedures for creating user IDs for various subsystems like CICS and products like NetView surely will include the definitions of the other user profile corresponding segments.

Same UID and GID assignment

If you assign users the same UID, you should warn them of the effects. For UID(0), the effects are less significant, because superusers have access to all processes and files and because most BPXPRMxx limits are not enforced against superusers.

Attention: If you want to enforce not to have multiple users have the same UID or GID, see 3.10.2, “Shared UID and GID prevention” on page 119.

Creating a WORKATTR segment

When defining the users, you may wish to use the WORKATTR segment to specify the user’s name and address. The name and address appear on the user’s SYSOUT output. For example, specifying the WORKATTR for user ID ANTOFF allows daemons to create processes with the correct accounting and SYSOUT defaults. If user ANTOFF logs into the system using a `rlogin` command from a workstation, a new process will be created for ANTOFF using the attributes from his WORKATTR segment. You may add the WORKATTR segment as follows:

```
ALU ANTOFF WORKATTR( WAACCNT(12345678) WAADDR1(ITSO)+ WAADDR2(POUGHKEEPSIE)WAADDR3(NEW
YORK) WAADDR4(12601) WABLDG(BLDG 8 SOUTH RD)+ WADEPT(ZOS) WANAME('THEO ANTOFF')
WAROOM(2C03))
```

3.7.4 Group access considerations

Although the same GID can be assigned to multiple RACF groups, it is not recommended. If you assign the same GID to multiple groups, control at an individual group level is lost because the GID is used in z/OS UNIX security checks. RACF groups that have the same GID assignment are treated as a single group during z/OS UNIX security checks.

Default groups

The security administrator needs to prepare RACF to provide security and to define users to RACF. For a user to be a z/OS UNIX user, the user's default group must be a z/OS UNIX group.

You can change the default group of a user ID having an OMVS segment to a default group without a GID by using the following command:

```
ALU SMITH DFLTGRP(PROG3)
```

Note: If PROG3 is a new group, make sure you have defined an OMVS segment.

User access with connect groups

To authorize a user to access z/OS UNIX and use z/OS UNIX resources (or define a z/OS user) you have to make sure that two conditions are met:

- The default group of the user has an OMVS segment with Group Identifier GID(n).
- At least one of his connect groups is authorized to the UNIX file or directory the user wants to access and has a GID.

When a GID is assigned to a group, all users connected to this group as their default group who have a user identifier (UID) in their user profile can use z/OS UNIX functions and can access z/OS UNIX files based on the GID and UID values assigned.

If only his default group has a GID and none of his connect groups have a GID, then the user still can use z/OS UNIX with access based on the “other” permission bits. For the meaning of “other” permission bits see chapter .... on page .....

RACF list-of-groups checking

When RACF list-of-groups checking is active, a user can access z/OS UNIX resources if they are permitted to any group his user ID is connected to and if the group has a GID. The additional groups are called *supplemental* groups. To activate the RACF list-of-groups checking, specify the GRPLIST keyword on the RACF SETROPTS command:

```
SETROPTS GRPLIST
```

3.7.5 Defining protected user IDs for STCs

A user ID becomes a protected user ID when it is given the NOPASSWORD and NOOIDCARD attributes by an ADDUSER or ALTUSER command. The user IDs that are defined for z/OS UNIX, z/OS UNIX daemons, and other important subsystems or started tasks can be protected from being used for other purposes. These user IDs can also be protected from being revoked after several unsuccessful attempts to enter a password. This support protects these user IDs from being misused if the RACF administrator does not change the password of the user ID from the default group to a more secure value.

Protected user IDs cannot be used to log on to the system, and are protected from being revoked through incorrect password attempts. The following examples show a protected user ID being defined for a CICS region, and an existing user ID used by JES being given the PROTECTED attribute:

```
ADDUSER CICS03 DFLTGRP(STCGROUP) OWNER(STCADMIN) NOPASSWORD
ALTUSER JES    DFLTGRP(STCGROUP) OWNER(STCADMIN) NOPASSWORD
```

You can define protected user IDs for started procedures (STCs) associated with the following z/OS UNIX programs:

- The z/OS UNIX kernel with a user ID OMVSKERN
- The initialization started procedure, BPXOINIT, with a user ID OMVSKERN
- Daemons that are critical to the availability of the z/OS UNIX system with user ID OMVSKERN

This prevents these user IDs from being revoked through inadvertent or malicious incorrect password attempts, or from being used for other purposes, such as logging on to the system.

3.8 User access to the z/OS UNIX shell

z/OS UNIX users can access the z/OS UNIX shell in several ways. To work interactively, the shell user connects to the system in one of the following ways:

- Logs on to TSO/E and enters the OMVS TSO/E command, which invokes a shell. The OMVS command provides a 3270 terminal interface to the shell.
- Issues the **`rlogin`** command, which invokes the shell. It provides an asynchronous terminal interface to the shell, familiar to Unix users.
- Issues the **`telnet`** command, which invokes the shell. It provides an asynchronous terminal interface to the shell, which is familiar to Unix users.

3.8.1 Define a user's file system

Before a user is ready to log on to the z/OS UNIX shell using the TSO commands OMVS or ISHELL, you need to accomplish a few very important steps:

- Allocate space for a user file system in the HFS or zFS file system by creating a data set with a standard naming convention chosen by your installation. In these examples, OMVS.USERID.HFS is being used, where OMVS is the HLQ of all the data set names. This data set is the space for the file system that is defined with the keyword HOME in the user's OMVS segment.
- The data sets that define the file systems should be RACF-protected by creating a profile in the DATASET class and then permitting authorized users access to it, as follows:

  ```
  AD ‘OMVS.**’ UACC(NONE)
  PE ‘OMVS.**’ ID(SECADM) ACC(ALTER)
  ```

  Access has been given to the SECADM group.

> **Note:** For the following administration steps, the administrator must have superuser authority to issue the commands. These commands are needed only for HFS file systems.

- Issue the CHOWN command to make the user owner of his directory.
- Issue the CHGRP command to make his default group the owning group of his directory.
- Issue the CHMOD command to change the permission bits for the user's directory to 700

Note: We should emphasize that the intended results from all three commands above are entirely a matter of the security policy adopted by your organization. You are in no way bound to use these commands in the suggested manner.

Define the user HFS and OMVS segment

See A.2, "JCL example to define a user OMVS segment" on page 512, to define a user HFS and issue the commands to give the user ownership and access to the file system.

List the OMVS segment

Using the JCL example for user ID HARRY from the TSO command line and from the OMVS command line in the shell is shown in Figure 3-18.

```
lu harry omvs noracf

USER=HARRY

OMVS INFORMATION
----------------
UID= 0000001021
HOME= /u/harry
PROGRAM= /bin/sh
CPUTIMEMAX= NONE
ASSIZEMAX= NONE
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE

ANTOFF:/u/antoff: >ls -la /u/harry
total 24
drwx------   2 HARRY    EMPL 8192 Jun 25 13:10 .
```

Figure 3-18 Issuing the lu command to list user HARRY and the ls command from the shell

3.8.2 Entering the shell from TSO/E

When a user invokes the shell from a TSO session, RACF is called to verify that the user is defined as a z/OS UNIX user before the system initializes the shell, as shown in Figure 3-19. This verification consists of checking that the user has an OMVS segment in his user ID profile and that his default group has an OMVS profile.

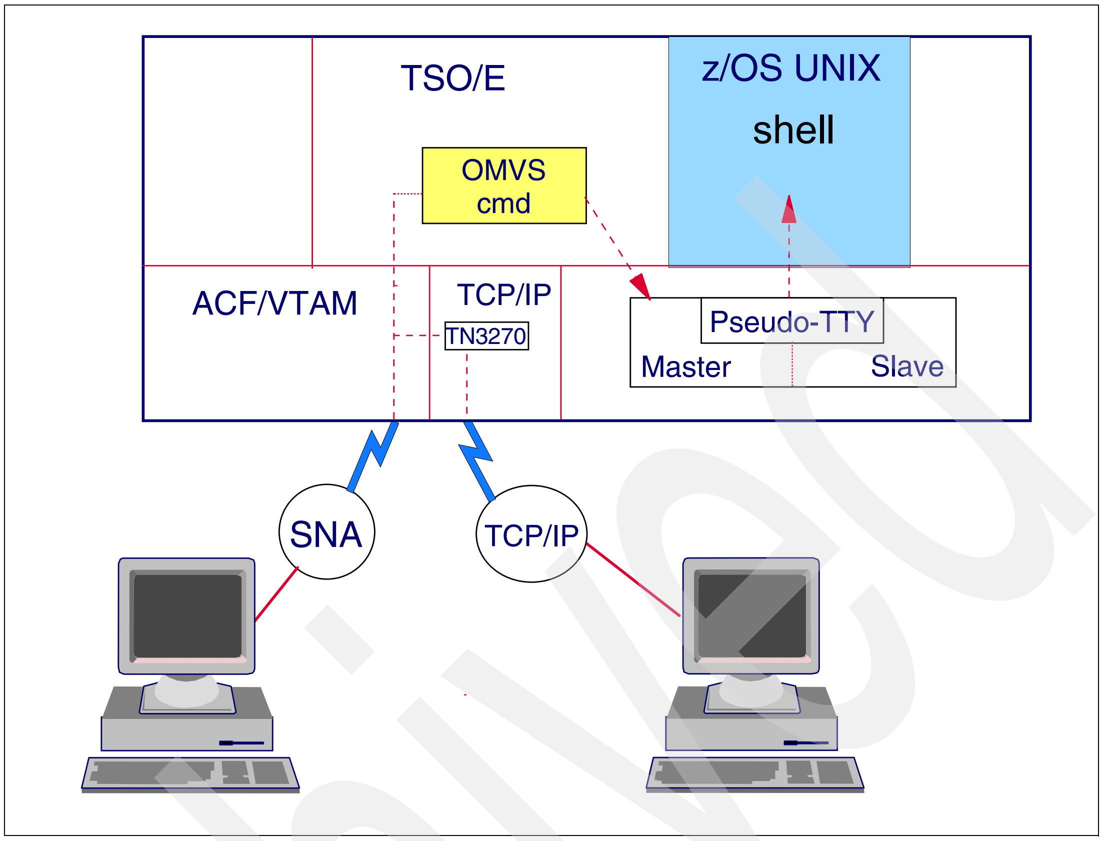


Figure 3-19 TSO user logging on and accessing the z/OS UNIX shell

3.8.3 Entering the shell from rlogin

When a user invokes the shell from rlogin, the rlogin user is authenticated to RACF by the rlogin daemon (rlogind) before entering the shell. Figure 3-20 on page 112 shows an overview of the two methods of logging in directly to the shell, as follows:

- With a remote login, if the inetd daemon is set up and active on the z/OS system, a workstation user with rlogin client support can use the TCP/IP network to log in to the shell without going through TSO/E.
  - When a daemon creates a process for a user, RACF is called to verify that the daemon's user ID is properly defined before the system initializes the process.
  - When a program requests a kernel service for the first time, RACF is called to verify that z/OS UNIX users are running the program before the system provides the service. The types of programs are:
    - Application programs
    - Started procedures
    - Products that use kernel services, such as the Resource Measurement Facility (RMF)

3.8.4 Entering the shell from telnet

The Telnet support comes with the z/OS CS. It also uses the inetd daemon, which must be active and set up to recognize and receive the incoming Telnet requests.

A z/OS system provides asynchronous terminal support for the z/OS UNIX shell. This is different from the 3270-terminal support provided by the TSO/E OMVS command.

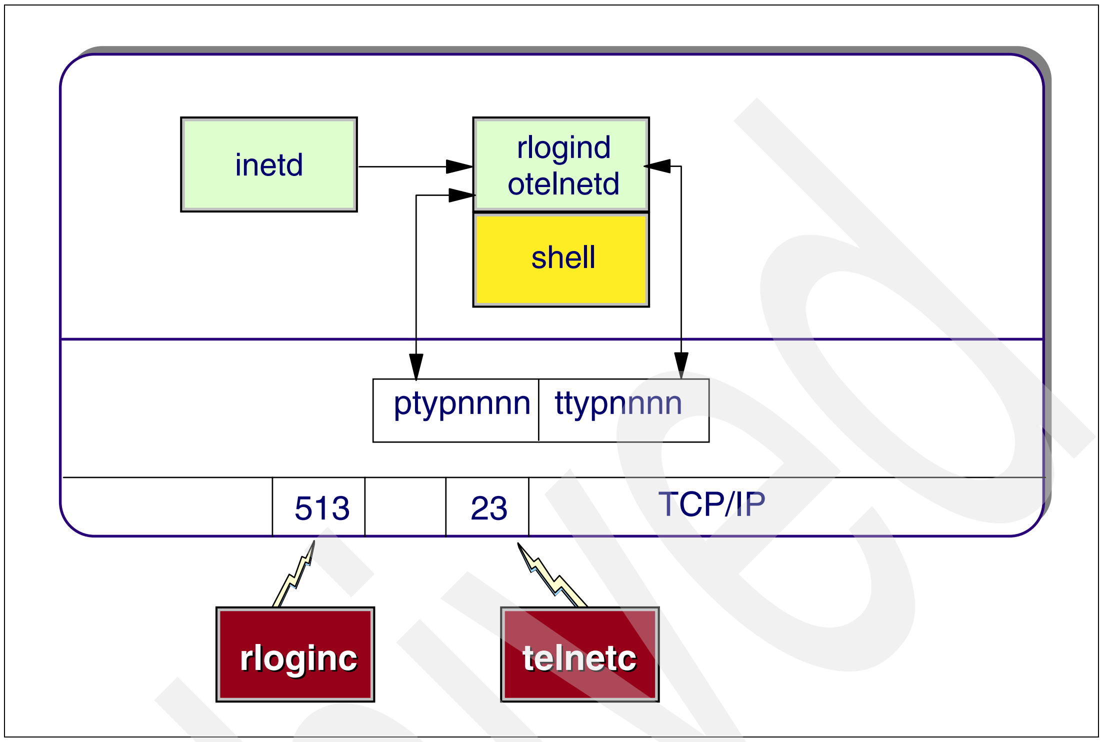


Figure 3-20 Users accessing the z/OS UNIX shell from rlogin and telnet

Remote connection differences

There are some differences between the asynchronous terminal support (direct shell login) and the 3270-terminal support (OMVS command) from ISPF of TSO/E:

- You cannot switch to TSO/E. However, you can use the TSO shell command to run a TSO/E command from your shell session.
- You cannot use the ISPF editor (this includes the **oedit** and TSO/E OEDIT commands, which invoke ISPF edit).

3.8.5 Setting z/OS UNIX resource limits for users

You can control the amount of resources consumed by certain z/OS UNIX users by setting individual limits for these users. The resource limits for all z/OS UNIX users is specified in the BPXPRMxx member of SYS1.PARMLIB. These limits apply to all users except those with UID 0 (superuser authority). Rather than assigning superuser authority to application servers and other users so they can exceed BPXPRMxx limits, you can individually set higher limits to these users in their OMVS segment.

Setting user limits allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

You can specify z/OS UNIX user limits by choosing options on the ADDUSER or ALTUSER commands. The limits are stored in the OMVS segment of the user profile. The following limits may be set in the OMVS user segment:

ASSIZEMAX	Maximum address space size
CPUTIMEMAX	Maximum CPU time

FILEPROCMAX	Maximum number of files per process
MMAPAREAMAX	Maximum memory map size
PROCUSERMAX	Maximum number of processes per UID
THREADSMAX	Maximum number of threads per process

The default values specified in the BPXPRMxx member and their ranges are shown in Table 3-3.

Table 3-3 Parameters and their values in OMVS segments

Parameter	Default value	Range	Unit
ASSIZEMAX	209,715,200	10M-2G	byte
CPUTIMEMAX	1000	7-2,147,483,647	sec
FILEPROCMAX	2,000	3-65535	number
MMAPAREAMAX	40,960	1-16,777,216	page
PROCUSERMAX	25	3-32,767	number
THREADSMAX	200	0-100000	number

Once you have set individual user limits for users who require higher resource limits, you should consider removing their superuser authority. You should also reevaluate your installation's BPXPRM xx limits and consider reducing these limits.

After you set individual user limits for users who require higher resource limits, you should consider removing their superuser authority, if they have any. You should also reevaluate your installation's BPXPRMxx limits and consider reducing these limits. See 2.3.6, "Step 6 - Customize BPXPRMxx" on page 58 for more information.

3.8.6 Support for lowercase user IDs

Supporting case-sensitive user IDs: XPG4 compliance requires the operating system to support case-sensitive user IDs that can optionally contain periods, dashes, and underscores. To provide this capability, the installation can define a user ID alias table.

This user ID alias support allows an XPG4-compliant program to work correctly with a user ID that exploits user ID naming conventions not normally tolerated on z/OS.

However, this support stops at the boundary between XPG4-defined functions and the rest of z/OS.

All security checks done by traditional z/OS services are based on the z/OS user ID. You can only log on to TSO/E using a valid z/OS user ID.

Recommendation: There are many ways in which use of a non-standard user ID conflicts with the running of normal business workloads. It is therefore strongly recommended that installations not define a user ID alias table. If you still believe that it is in your installation's best interest to exploit case-sensitive user IDs, see "Set up a user ID alias table." on page 177.

3.8.7 Setting up field access level for OMVS segment

To allow a user to see or change OMVS fields in a RACF user profile, you can set up field-level access. You can authorize a user to specified fields in any profile or to specified fields in the user's own profile. To authorize users to the OMVS fields in their own profiles, use the ISPF shell, or issue the commands shown in Figure 3-21.

Each profile defines one of the nine fields in the OMVS segment using the RACF RDEFINE command. However, you would only do this if you have different access lists for each of the profiles.

```
RDEFINE FIELD USER.OMVS.UID UACC(NONE)
RDEFINE FIELD USER.OMVS.HOME UACC(NONE)
RDEFINE FIELD USER.OMVS.PROGRAM UACC(NONE)
RDEFINE FIELD USER.OMVS.CPUTIME UACC(NONE)
RDEFINE FIELD USER.OMVS.ASSIZE UACC(NONE
RDEFINE FIELD USER.OMVS.FILEPROC UACC(NONE)
RDEFINE FIELD USER.OMVS.PROCUSER UACC(NONE)
RDEFINE FIELD USER.OMVS.THREADS UACC(NONE)
RDEFINE FIELD USER.OMVS.MMAPAREA UACC(NONE)
```

Figure 3-21 RACF commands to allow user access to change OMVS segment fields

For example, to control access to all fields in the OMVS segment of the user profiles, issue the RDEFINE command and specify USER.OMVS.** as follows:

```
RDEFINE FIELD USER.OMVS.** UACC(NONE)
```

Next decide who should be permitted to the above profiles. Using a user ID specification of &RACUID allows all users to look at their own fields.

READ READ access allows users to find their UID value.

UPDATE UPDATE access allows users to change their home directory in the HOME field or the program invoked in the PROGRAM field.

```
PERMIT USER.OMVS.UID CLASS(FIELD) ID(&RACUID) ACCESS(READ)
PERMIT USER.OMVS.HOME CLASS(FIELD) ID(&RACUID) ACCESS(UPDATE)
PERMIT USER.OMVS.PROGRAM CLASS(FIELD) ID(&RACUID) ACCESS(UPDATE)
PERMIT USER.OMVS.UID CLASS(FIELD) ID(decentralized adm) ACCESS(UPDATE)
PERMIT USER.OMVS.** CLASS(FIELD) ID(decentralized adm) ACCESS(UPDATE)
```

If you have decentralized RACF administration groups, you may decide to allow such groups to have UPDATE access to the UID field and possibly to the user limits fields.

Attention: A user with UPDATE access to profile USER.OMVS.UID can make any user ID in his scope of GROUP-SPECIAL authority a superuser by changing his UID to 0.

To activate the FIELD class with the RACF SETROPTS command:

```
SETROPTS CLASSACT(FIELD) RACLIST(FIELD)
```

3.9 UNIXPRIV class enhancements

With z/OS V1R3 and z/OS V1R4, new resource names were added in the UNIXPRIV class in support of the changes made to z/OS UNIX in those releases. These changes are discussed in some of the following sections. The new resource names are shown in Figure 3-22.

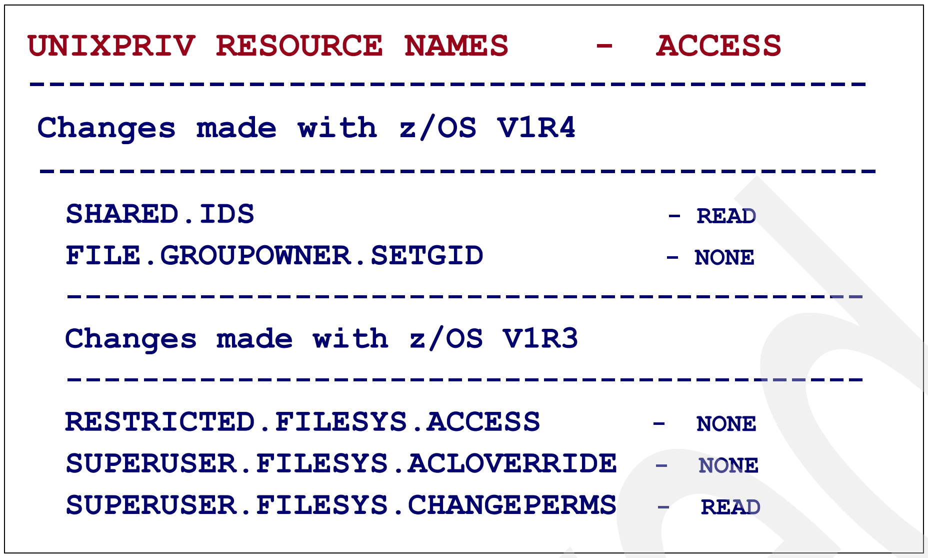

```
UNIXPRIV RESOURCE NAMES    -  ACCESS
-------------------------------------------
 Changes made with z/OS V1R4
-------------------------------------------
   SHARED.IDS                        - READ
   FILE.GROUPOWNER.SETGID            - NONE
   ----------------------------------------
   Changes made with z/OS V1R3
   ----------------------------------------
   RESTRICTED.FILESYS.ACCESS       -  NONE
   SUPERUSER.FILESYS.ACLOVERRIDE   -  NONE
   SUPERUSER.FILESYS.CHANGEPERMS   -  READ
```

Figure 3-22 New resource names in the UNIXPRIV class with z/OS V1R3 and V1R4

3.10 Shared UIDs and GIDs

Enhancements in z/OS V1R4 have been made so that UIDs and GIDs can be assigned by RACF. Two new functions are supported:

- UIDs and GIDs can be automatically assigned to new users
  - UIDs and GIDs can be assigned automatically by RACF to new users, making it easier to manage the process of assigning UIDs and GIDs to users. (Previously, this was a manual process and guaranteed the uniqueness of the UID and GID for every user.)
- UIDs and GIDs can either be prevented from being shared, or allowed to be shared.
  - Controlling the use of shared UIDs and GIDs is a prerequisite to using automatic UID and GID assignment.
  - By default, RACF does not prohibit the sharing of UIDs and GIDs among any number of users or groups. However, you can control enforcement of unique UIDs and GIDs by defining a new profile called SHARED.IDS in the UNIXPRIV class.

Attention: Application identity mapping (AIM) to at least stage 2 must be implemented if you wish to define and use the profile SHARED.IDS.

3.10.1 Automatic UID and GID assignment

Automatic UID and GID assignment is implemented by using a new AUTOUID keyword with the ADDUSER and ALTUSER commands. An unused UID will be assigned to the new or modified user. Using the AUTOGID keyword on ADDGROUP and ALTGROUP commands, a GID will be automatically assigned to the new or modified group.

The use of automatic UID and GID assignment requires the following:

- AIM stage 2 or 3; otherwise, the automatic assignment attempt fails and an IRR52182I message is issued:

```
IRR52182I Automatic UID assignment requires application identity mapping to be
implemented
```

- A SHARED.IDS profile in the RACF UNIXPRIV class; if not:

```
IRR52183I Use of automatic [UID|GID] assignment requires SHARED.IDS to be implemented.
```

- A BPX.NEXT.USER profile in the RACF FACILITY class; if not:

```
IRR52179IThe BPX.NEXT.USER profile must be defined before you can use automatic
[UID|GID] assignment.
```

- Use of the AUTOUID or AUTOGID keyword

SHARED.IDS profile

The SHARED.IDS profile in the UNIXPRIV class, shown in Figure 3-22 on page 115, acts as a system-wide switch to prevent assignment of an ID that is already in use.

If the SHARED.IDS profile in the RACF UNIXPRIV class is not defined, the attempt to use AUTOUID or AUTOGID fails and an IRR52183I message is issued:

```
IRR52183I Use of automatic UID assignment requires SHARED.IDS to be implemented
```

The SHARED.IDS must be defined as follows:

```
RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)
```

BPX.NEXT.USER facility class profile

A BPX.NEXT.USER RACF FACILITY class profile must be defined and RACLISTed for automatic assignment to work. Otherwise, the attempt to do automatic assignment fails and an IRR52179I message is issued:

```
IRR52179I The BPX.NEXT.USER profile must be defined before you can use automatic UID
assignment.
```

The definition of the BPX.NEXT.USER FACILITY in the FACILITY class, which must be RACLISTed, has the following syntax:

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA(UID/GID)
SETROPTS RACLIST(FACILITY) REFRESH
```

The APPLDATA keyword consists of two qualifiers separated by a forward slash (/). The qualifier on the left of the slash character specifies the starting UID value or range of UID values.

The qualifier on the right of the slash character specifies the starting GID value or range of GID values. Qualifiers can be null or specified as NOAUTO to prevent automatic assignment of UIDs or GIDs, as shown in Figure 3-23 on page 117.

The starting value is the value RACF attempts to use in ID assignment, after determining that the ID is not in use. If it is in use, the value is incremented until an appropriate value is found.

The maximum value valid in the APPLDATA specification is 2,147,483,647. If this value is reached or a candidate UID or GID value has been exhausted for the specified range, subsequent automatic ID assignment attempts fail and message IRR52181I is issued.

Note: Keep in mind that APPLDATA is verified at the time of *use*, not when it is defined. If a syntax error is encountered when the auto assignment is used, the IRR52187I message is issued and the attempt fails.

Automatic assignment example

In the following example, we have defined the APPLDATA for a range of values from 5 to 70000 for UIDs, and from 3 to 30000 for GIDs. USERA and USERB are added using the automatic assignment of UID. The range of automatic UID assignment starts with 5, so USERA is assigned to UID(5), which was free. UID(6) and UID(7) were already assigned before we started our examples. The first following free UID is 8. USERB is assigned to UID(8).

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('5-70000/3-30000')

ALTUSER USERA OMVS(AUTOUID)
IRR52177I User USERA was assigned an OMVS UID value of 5.

ALTUSER USERB OMVS(AUTOUID)
IRR52177I User USERB was assigned an OMVS UID value of 6.
```

RACF extracts the APPLDATA from the BPX.NEXT.USER and parses out the starting value. It checks if it is already in use and if so, the value is incremented and checked again until an unused value is found. Once a free value is found, it assigns the value to the user or group and replaces the APPLDATA with the new starting value, which is the next potential value or the end of the range.

In our example, RACF will start assigning from UID(7) in the next assignment. However, you can change the APPLDATA and modify the starting value. The APPLDATA can be changed using the following command:

```
RALTER FACILITY BPX.NEXT.USER APPLDATA('2000/500')
```

Note: Automatic assignment of UIDs and GIDs fails if you specify a list of users to be defined with the same name, or if you specify the SHARED keyword. Also, AUTOUID or AUTOGID is ignored if UID or GID is also specified.

APPLDATA examples

Figure 3-23 shows examples of correct and incorrect APPLDATA specifications.

```
Good data
   1/0
   1-50000/1-20000
   NOAUTO/100000
   /100000
   10000-20000/NOAUTO
   10000-20000/
Bad data
   /
   123B
   2147483648     /* higher than max UID value */
   555/1000-900
```

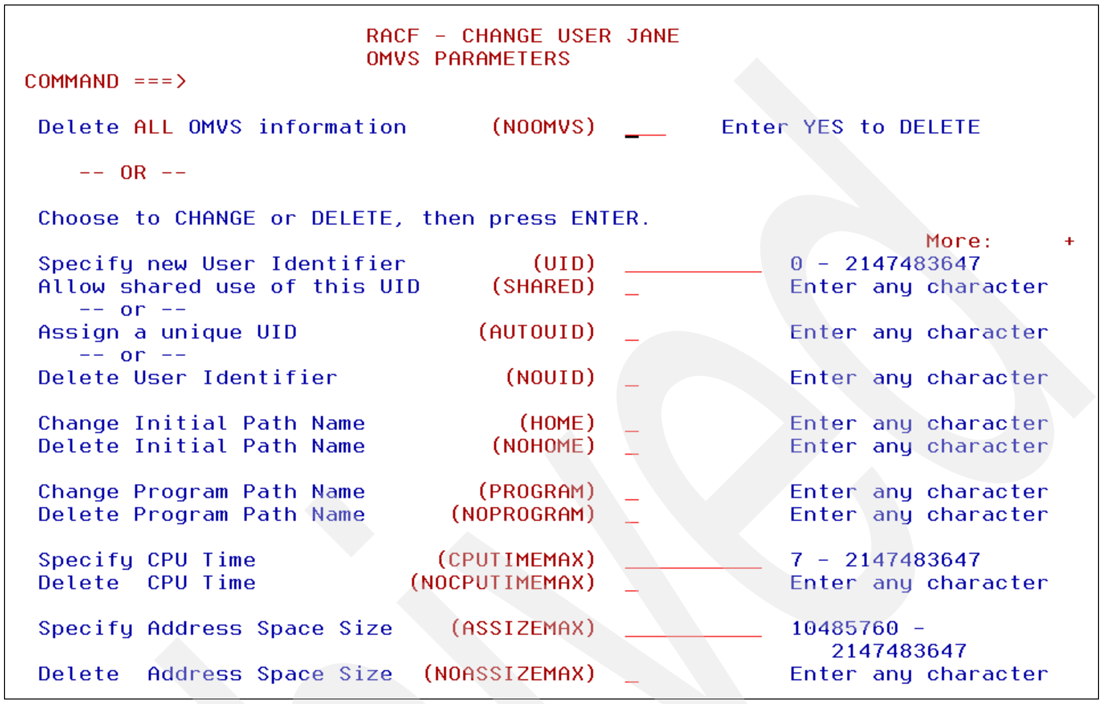
Figure 3-23 Examples of APPLDATA that can be specified

If you have an incorrect specification and attempt to use AUTOUID on an ADDUSER command, the following message is issued:

```
IRR52187I Incorrect APPLDATA syntax for the BPX.NEXT.USER profile.
```

Automatic assignment with RACF panels

You may use the RACF panels to define the OMVS segment. Figure 3-24 indicates how to use the automatic assignment by using the AUTOUID field.



```
                         RACF - CHANGE USER JANE
                         OMVS PARAMETERS
COMMAND ===>

 Delete ALL OMVS information      (NOOMVS) ___    Enter YES to DELETE

    -- OR --

 Choose to CHANGE or DELETE, then press ENTER.
                                                               More:     +
 Specify new User Identifier         (UID) __________  0 - 2147483647
 Allow shared use of this UID     (SHARED) _           Enter any character
    -- or --
 Assign a unique UID             (AUTOUID) _           Enter any character
    -- or --
 Delete User Identifier            (NOUID) _           Enter any character

 Change Initial Path Name           (HOME) _           Enter any character
 Delete Initial Path Name         (NOHOME) _           Enter any character

 Change Program Path Name        (PROGRAM) _           Enter any character
 Delete Program Path Name      (NOPROGRAM) _           Enter any character

 Specify CPU Time             (CPUTIMEMAX) __________  7 - 2147483647
 Delete  CPU Time           (NOCPUTIMEMAX) _           Enter any character

 Specify Address Space Size    (ASSIZEMAX) __________  10485760 -
                                                          2147483647
 Delete  Address Space Size  (NOASSIZEMAX) _           Enter any character
```

Figure 3-24 RACF panel to set OMVS parameters for automatic assignment

Automatic assignment in an RRSF configuration

In an RRSF configuration (see Figure 3-25 on page 119) you may wish to avoid UID and GID duplications. This can be done by using non-overlapping APPLDATA ranges.

You may also wish to make RACF automatically suppress propagation of internal updates. This can be done by specifying the ONLYAT keyword to manage the BPX.NEXT.USER profile, as follows:

```
RDEFINE BPX.NEXT.USER APPLDATA('5000-10000/5000-10000') ONLYAT(NODEA.MYID)
RDEFINE BPX.NEXT.USER APPLDATA('10001-20000/10001-20000') ONLYAT(NODEB.MYID)
```

RRSF automatic assignment considerations

RACF does two things to facilitate automatic ID assignment in an RRSF environment in order to prevent different nodes from arriving at the same ID values independently for different users and then propagating these updates on the network.

1. RACF suppresses propagation of its own internal updates to the BPX.NEXT.USER profile. This prevents RACF from altering the BPX.NEXT.USER profile on other RRSF nodes when you are using automatic direction of application updates for the FACILITY class.
2. RACF alters the command image on the source node before propagating it out to other RRSF nodes. RACF inserts the generated ID value into the command z/OS UNIX image so (from the perspective of the target node) an explicit ID assignment is being requested. This protects you when automatic command direction is in effect for USER and GROUP profiles.

For example, if you issue the following command at NODEA, as shown in Figure 3-25:

```
ADDUSER ROGERS OMVS(AUTOUID)
```

RACF assigns a UID value of 5000 through BPX.NEXT.USER and the APPLDATA. The command propagated out to NODEB is:

```
ADDUSER ROGERS OMVS(AUTOUID UID(5000))
```

The BPX.NEXT.USER profile on the target node is not used as a result of receiving an update that involved automatic ID generation at the source node (when the ADDUSER command runs on the target node, AUTOUID is ignored because UID is specified).

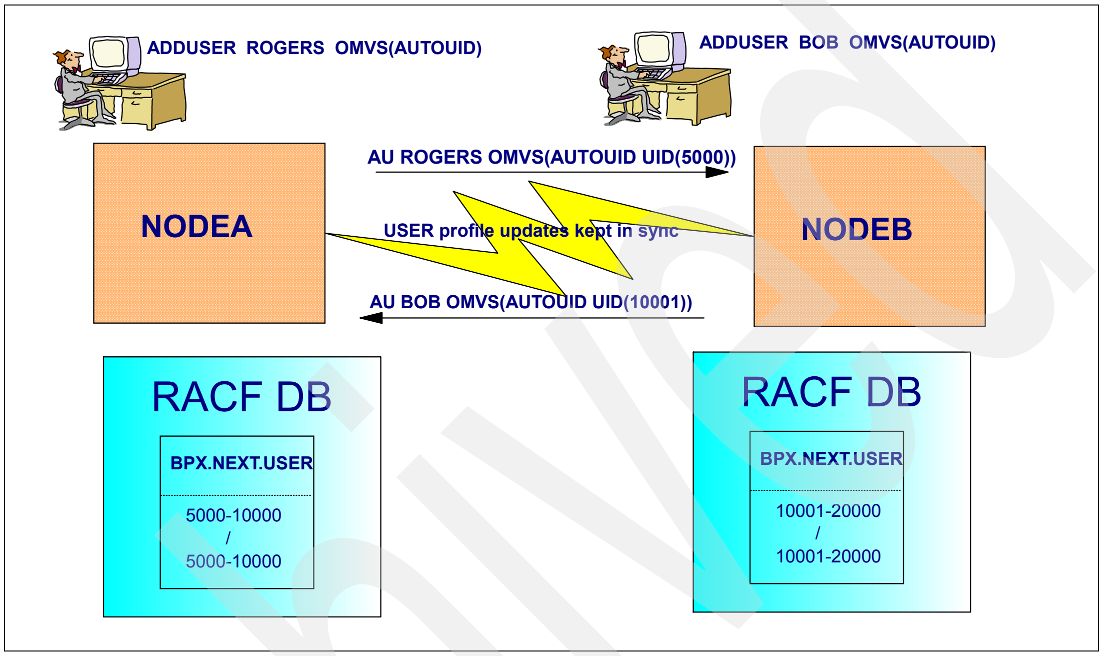

ADDUSER ROGERS OMVS(AUTOUID)
ADDUSER BOB OMVS(AUTOUID)
AU ROGERS OMVS(AUTOUID UID(5000))
NODEA
USER profile updates kept in sync
NODEB
AU BOB OMVS(AUTOUID UID(10001))
RACF DB
BPX.NEXT.USER
5000-10000
/
5000-10000
RACF DB
BPX.NEXT.USER
10001-20000
/
10001-20000

Figure 3-25 Automatic assignment in an RRSF configuration

For more information about automatic assignment in an RRSF configuration, refer to *z/OS V1R4.0 Security Server RACF Security Administrator's Guide,* SA22-7683.

3.10.2 Shared UID and GID prevention

In order to prevent several users from having the same UID number, you must use the RACF SHARED.IDS profile introduced in the UNIXPRIV class with z/OS V1R4. This profile acts as a system-wide switch to prevent assignment of a UID that is already in use.

To enable shared UID prevention, you must define the SHARED.IDS profile in the UNIXPRIV class as follows:

```
RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Once the SHARED.IDS profile has been defined and the UNIXPRIV class refreshed, it will not allow a UID to be assigned if the UID is already in use. The same is true for GIDs; it will not allow a GID to be shared between different groups.

Shared UID examples

We created USER1 with UID(7). Then we tried to define USER2 with the same UID(7), but received the following error message:

```
IRR52174I Incorrect UID 7.  This value is already in use by USER1.
```

You will get the following error message if you try to specify more than one user in an ADDUSER command request:

```
IRR52185I The same UID cannot be assigned to more than one user.
```

Existing shared UIDs and GIDs

The use of this new functionality does not affect preexisting shared UIDs; they will remain as shared once you install the new support. If you want to eliminate sharing of the same UID, you must clean them up separately. z/OS V1R4 provides a new IRRICE report to find the shared GIDs. Shared UIDs were previously reported by the IRRICE report.

3.10.3 SHARED keyword to allow duplicate UID and GID

Even if the SHARED.IDS profile is defined, you may still require some UIDs to be shared and others not to be shared. For example, you may require multiple superusers with a UID(0). It is possible to do this using the new SHARED keyword in the OMVS segment of the ADDUSER, ALTUSER, ADDGROUP, and ALTGROUP commands.

Assigning the same UID or GID

To allow an administrator to assign a non-unique UID or GID using the SHARED keyword, you must grant that administrator at least READ access to SHARED.IDS profile, as follows:

```
PERMIT SHARED.IDS CLASS(UNIXPRIV) ID(ADMIN) ACCESS(READ)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Once user ID ADMIN has at least READ access to the SHARED.IDS profile, ADMIN will be able to assign the same UID or GID to multiple users, using the SHARED KEYWORD, as follows:

```
ALTUSER (USERA USERB) OMVS(UID(7) SHARED)
AU KERNSTU OMVS(UID(0) SHARED)
AG (G1 G2 G3) OMVS(GID(9) SHARED)
```

If a user is not authorized for the SHARED keyword, and the following command is issued:

```
ALU ANTOFF OMVS(UID(0) SHARED)
```

the following message is returned:

```
IRR52175I You are not authorized to specify the SHARED keyword.
```

Note: To specify the SHARED operand, you must have the SPECIAL attribute or at least READ authority to the SHARED.IDS profile in the UNIXPRIV class.

3.11 Protecting files in the file systems

The data administrator or the system programmer must manage the files in the hierarchically organized data that the system and its users use. This overall structure of data is called the hierarchical file system (HFS). It consists of the root file system and all the file systems that are added to it whether it is an HFS files system or a zFS file system. For the security administrator to control access to files and directories in the file systems, it is necessary to understand the following controls that allow access:

- Permission bits that allow access to files and directories
- Change ownership of files and directories
- Change permission bit settings

- Obtain security information about the files and directories
- Controlling access to files with RACF

3.11.1 File and directory access

Access to files and directories in the file systems are controlled by permission bits that are stored in the file security packet (FSP) associated with each file and directory. The files and directories are protected by RACF security rules. You can use control access as follows:

- Permission bit settings
- Access control lists (ACLs) can also be used in conjunction with permission bits.

Note: ACLs support is introduced in z/OS UNIX beginning with z/OS V1R3.

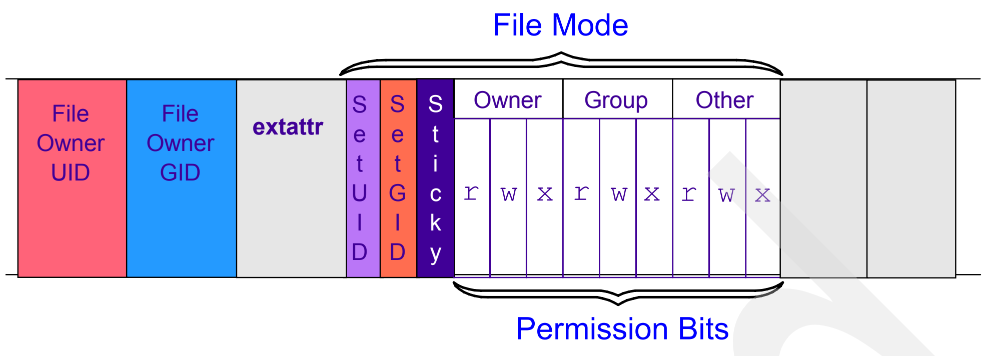
File security packet (FSP)

The system provides security for files and directories by verifying that a z/OS UNIX user can access a directory or file, and which would include every directory in the path to a file.

The FSP of every file and directory contains security information, which consists of:

- UID and GID of the file
- Permission bits
- Setuid, setgid, and set-sticky bits
- Audit options set by file owner
- Audit options set by security auditor
- ACLs, if they exist (see 3.14.3, "FSP and access control lists" on page 135)

The authorization checking for access to z/OS UNIX files and directories in a file system has been done using the FSP. The FSP, shown in Figure 3-26 on page 122, is stored in the file system as part of the attributes of a file or directory and is created when a file or directory is created. If a security authorization is needed for a file or directory, the security packet is passed to RACF for authorization checking.



Access	Permission for File	Permission for Directory
Read (**r**)	Permission to read or print the contents.	Permission to read, but not search, the contents.
Write (**w**)	Permission to change, add to, or delete from the contents.	Permission to change, add, or delete directory entries.
Execute (**x**)	Permission to run the file. This permission is used for executable files.	Permission to search the directory.

Figure 3-26 File security packet with permission bit explanation

File mode

The file mode consists of the following permission bits:

SetUID This bit only relates to executable files. If on, it causes the UID of the user executing the file to be set to the file's UID.

SetGID This bit only relates to executable files. If on, it causes the GID of the user executing the file to be set to the file's GID.

Sticky Bit This bit only relates to executable files. If on, it causes the file to be retained in memory for performance reasons. The implementation of this varies between platforms. In z/OS UNIX, it means programs are loaded from LPA (or LNKLST as per normal MVS program search) instead of a HFS file. For a directory, the sticky bit causes UNIX to permit files in a directory or subdirectories to be deleted or renamed only by the owner of the file, or by the owner of the directory, or by a superuser.

Note: For a description and how to use these bits as an administrator, see 3.12.5, “Setuid, setgid, and sticky bits” on page 129.

Permission bits

Access checking compares the user's UID and GID to the ones stored in the FSP. You control access to a file and directory that you own through its permission bits. The permission bit settings for files and directories are set for the following types of users:

Owner The effective UID of the owner or creator of the file. The owner can be changed by an authorized administrator or by the file owner.

Group Anyone who has the same GID as the owner or belongs to a supplemental group GID of the owner.

Other This includes every user who has a UID or GID that is not an owner.

The access (read, write, or execute) that each z/OS UNIX has is shown in Figure 3-26 on page 122.

The file mode permission bits have the following meaning:

r	Read (r) access to both files and directories
w	Write (w) access to both files and directories
x	Execute (x) has a different meaning for files and directories, as follows:

–For an executable file, an access of x means that the user can execute the file.

–For a directory, an access of x means the user can search the directory.

Both read (r) and execute (x) are required in order to execute a shell script. In order to access HFS files, a user needs the following:

- Search (x) permission to all the directories in the path name of files the user wants to access.
- Write permission to directories where the user will be creating new files and directories.
- Read and/or write permission, as appropriate, to files for access.
- Execute permission for an executable file.

Note: In z/OS UNIX, these three permissions are not hierarchical. For example, a user with write permission who does *not* have read permission, can only write over existing data or add data to a file, and cannot look at the contents of the file or print the file. Similarly, write and read permission does not allow a user to execute a file or search a directory.

3.12 Creating and managing files and directories

When you create directories and files, you can control access to them. Whenever you want, you can change the access permissions that are set when you first create a directory or file.

3.12.1 Setting the permission bits

When you first create a file or directory, the system sets the default read, write, and execute (rwx) permissions. For the commands shown in Figure 3-27 on page 125, the permission settings are changed by the umask. The default file creation mask (umask) is set by an administrator in /etc/profile. If not modified by the administrator, the default is 022.

Setting the file creation mask (umask)

When a file is created, it is assigned initial access permissions. When a z/OS UNIX user or program is creating a file, the final permission bits are set by the umask. Since a user or program can set a umask for that process, the order in which the umask is used is as follows:

- If you want to control the permissions that a program can set when it creates a file or directory, you can set the umask for that session or program with the **`umask`** command.
- The user can place a umask in a file named in the ENV environment variable in $HOME/.profile, which points to a .setup file.
- The user can place a umask in the $HOME/.profile.
- If no umask is set by the user, the default umask from /etc/profile is used.

Using the umask

In this example, the default mask is used and is set to 022. When a file created and the permission bits are set to 777, they are changed by the umask (022) to have permissions of 755. The effect of the umask is to turn off the group write and other write bits. So, a umask of 022 turns off the write bit if it is on in those two positions.

When you set the mask, you are setting limits on allowable permissions. You are implicitly specifying which permissions are not to be set, even though the calling program may allow those permissions. When a file or directory is created, the permissions set by the program are adjusted by the umask value: The final permissions set are the program's permissions minus what the umask values restrict.

umask command

A user can change the default setting when a file is created with the **umask** shell command. The values set by the **umask** command last for the length of the user's session, or the command can be part of the user's login so that the user always has the same default permissions. To use the **umask** command for a single session, enter:

```
umask mode
```

The `mode` is in either of the formats, symbolic (rwx) or octal values. The symbolic form expresses what can be set, what is allowed, while octal values express what cannot be set, what is disallowed. For example, both of the following commands set the same **umask** value. The `a=` specifies owner, group, and other permissions.

```
umask a=rx
umask 222
```

Display the umask

If you just enter the **umask** command, you see the mode displayed in octal values, indicating what cannot be set:

```
umask
0022 .
```

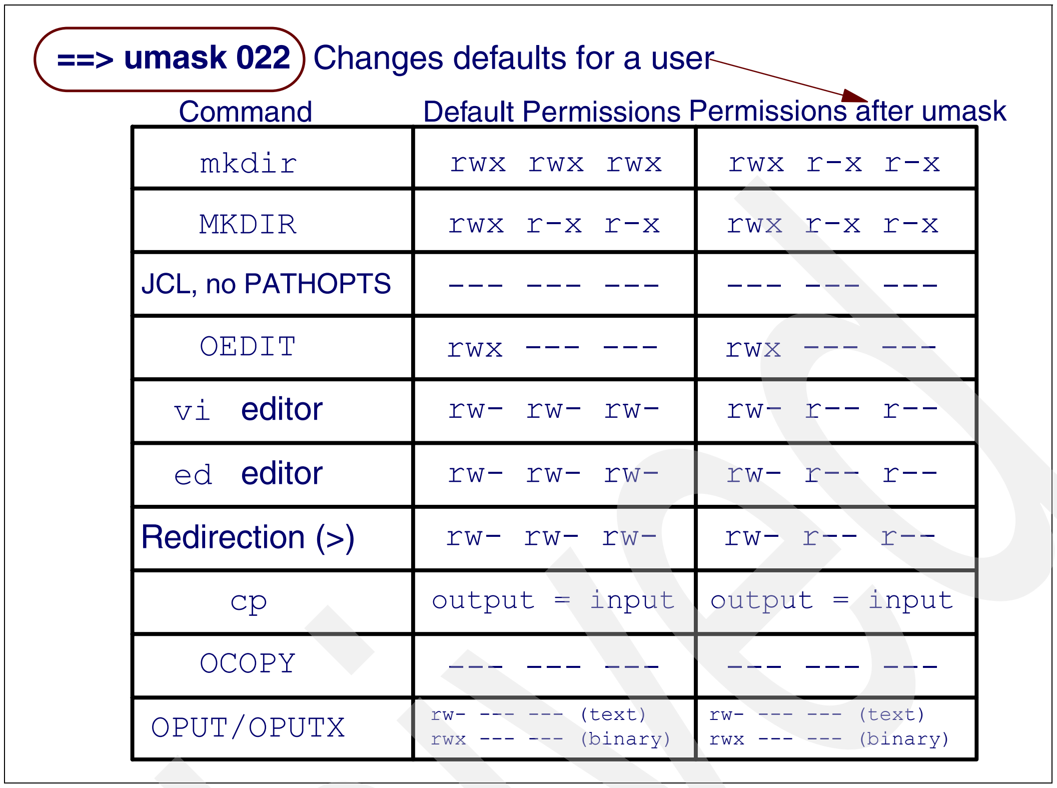
If you enter **umask -S**, you see the mode displayed in symbolic form, indicating what can be set, as follows:

```
umask -S
u=rwx,g=rx,o=rx
```

The shell's initial setting of the mask is 000, which means that read, write, and execute permission can be set on for everyone. But the system-wide profiles provided with the product set the mask to 022 in the umask directive of /etc/profile.

Default permissions set by the system

Figure 3-27 on page 125 shows the default permissions set by the system when the commands are issued and the file is created. After the file is created, the default permissions are set by the function or command that created the file. The permissions are then changed by the umask processing and become the final permissions for the file.


==> umask 022 Changes defaults for a user

Command	Default Permissions	Permissions after umask
mkdir	rwx rwx rwx	rwx r-x r-x
MKDIR	rwx r-x r-x	rwx r-x r-x
JCL, no PATHOPTS	--- --- ---	--- --- ---
OEDIT	rwx --- ---	rwx --- ---
vi editor	rw- rw- rw-	rw- r-- r--
ed editor	rw- rw- rw-	rw- r-- r--
Redirection (>)	rw- rw- rw-	rw- r-- r--
cp	output = input	output = input
OCOPY	--- --- ---	--- --- ---
OPUT/OPUTX	rw- --- --- (text) rwx --- --- (binary)	rw- --- --- (text) rwx --- --- (binary)

Figure 3-27 Permission bit settings using the umask when creating files

3.12.2 Setting the UID and GID

By default, the system sets the UID and GID of the file when the file is created:

- The UID is set to the effective UID of the creating process.
- The GID is set to the GID of the owning directory.

Note: Beginning with z/OS V1R4,you can change the setting of the GID by using the profile FILE.GROUPOWNER.SETGID in the UNIXPRIV class shown in Figure 3-22 on page 115.

Using the FILE.GROUPOWNER.SETGID profile

When a new UNIX file is created on z/OS, by default, the owning GID is copied from the parent directory, as shown in Figure 3-28 on page 126. The POSIX standard allows the owning GID to be taken either from the parent directory in the FSP, or from the effective GID of the creating process (the user security packet (USP). Also, see Figure 3-15 on page 102.

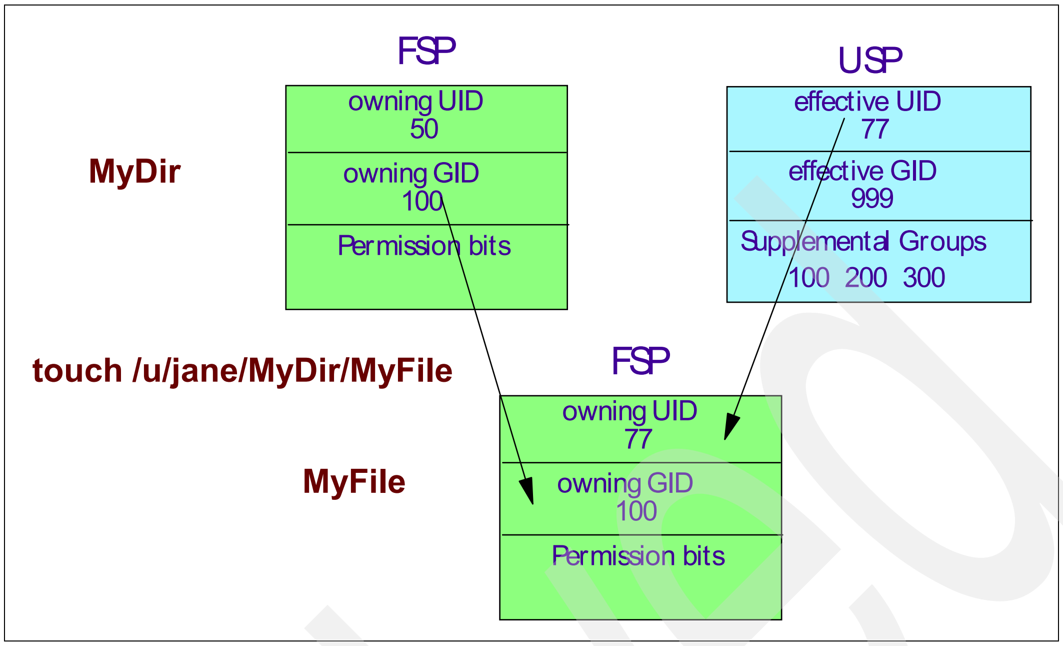


Figure 3-28 Creating or updating a file and setting the owning UID and GID

z/OS V1R4 support for setting the owning GID

Many versions of UNIX and Linux use the setgid bit of the parent directory to determine how to set a new object's group owner. If the parent's setgid bit is on, then the group owner is set to that of the parent directory. If it is off, the group owner is set from the effective GID of the process. Further, the setgid bit for a new directory is inherited from the parent directory.

To specify that the group owner of a new HFS file is to come from the effective GID of the creating process, you need to set up profile FILE.GROUPOWNER.SETGID in the RACF UNIXPRIV class. Generic characters cannot be used in this profile name. Issue the command:

```
RDEF UNIXPRIV FILE.GROUPOWNER.SETGID OWNER(SECADM) UACC(NONE)
```

This change for the setting of the GID is an optional one. You do not need to permit anybody to this profile. The resulting change in the GID assignment is shown in Figure 3-29 on page 127.

Once you have created this profile, in order for newly created files and directories in a particular directory, /u/janeto, to have group owner set from the effective GID of the process, you have to check that the setgid bit for this directory is turned off by displaying, as follows:

```
ROGERS @ SC64:/u/jane>ls -l
total 24
-rwxr-sr---
```

If the third permission bit for group owner is s or S (that means the setgid bit is on; the default is off), you must turn it off with the command:

```
chmod g-s /u/jane
```

Attention: When a new file system is mounted, you must turn on the setgid bit of its root directory if you want new objects within the file system to have their group owner set to that of the parent directory.

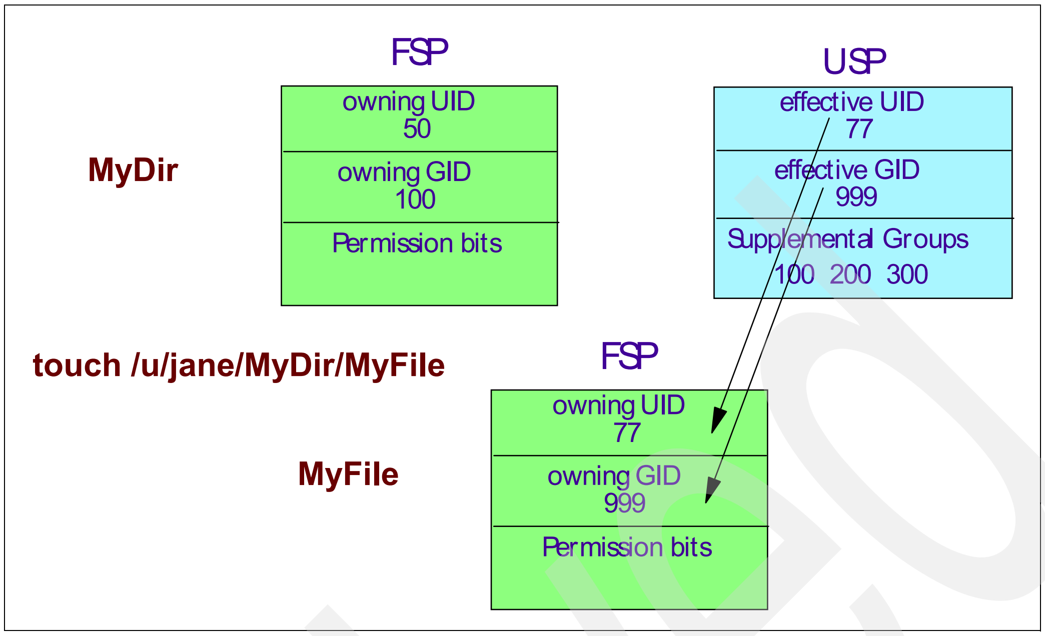


Figure 3-29 z/OS V1R4 change for setting the owning GID

3.12.3 Change of file ownership

Administrators who have superuser authority might need to change file ownership by changing the owning UID or owning GID. To protect files from unauthorized users, on z/OS UNIX systems, RACF enforces rules for the POSIX constant called _POSIX_CHOWN_RESTRICTED. This means that, by default, only superusers or administrators can change the ownership of any file to any user ID or group on the system, and that general users can only change the ownership of files that they own, and only to one of their own supplementary groups.

- To change the owner (UID) of a file, the superuser can enter a **`chown`** command or use the chown() function, specifying a RACF user ID.
- To change the group (GID) of a file, the superuser or the file owner can enter a **`chgrp`** command or use the chgrp() function, specifying a RACF group.

Allow all z/OS UNIX users to change their file ownership

By defining profile CHOWN.UNRESTRICTED in the UNIXPRIV class, you can indicate that _POSIX_CHOWN_RESTRICTED is not in effect. This allows *all z/OS UNIX users* to transfer ownership of files they own to any RACF user ID or group on the system.

CHOWN.UNRESTRICTED must be a discrete profile. Any matching generic profiles will be ignored. RACF checks only for the existence of this profile and any access list will be ignored. Issue the command:

```
RDEF UNIXPRIV CHOWN.UNRESTRICTED OWNER(SECADM) UACC(NONE)
```

Allow selected users to change ownership for all files

To allow selected z/OS UNIX users to transfer ownership of any file to any RACF user ID or group, create profile SUPERUSER.FILESYS.CHOWN in the UNIXPRIV class with the command:

```
RDEF UNIXPRIV SUPERUSER.FILESYS.CHOWN OWNER(SECADM) UACC(NONE)
```

Authorize selected groups as appropriate (best candidates might be groups for decentralized administration):

```
PE SUPERUSER.FILESYS.CHOWN CL(UNIXPRIV) ID(LOCADM) ACCESS(READ)
```

3.12.4 Changing permission bits

To change the permission bits for a file, use one of the following:

- The ISPF shell.
- The **chmod** command can be used to change individual permission bits without affecting the other bits. You can also use the **setfacl** command to change permission bits (see C.1.2, "Set an entire ACL (base and extended)" on page 561.
- The chmod() function in a program. The function changes all permission bits to the values in the mode argument.

chmod command

You can use the **chmod** command to set or change permissions for your files and directories. To change permissions, you must be the owner or a superuser. If you are uncertain about ownership, use the **ls -l** command and look for your TSO/E user ID.

For example, to turn on read, write, and execute permissions, and turn off the set-user-ID and sticky bit attributes for a file, enter the command using either rwx or octal notation:

```
chmod a=rwx filename
chmod 777 filename
```

When specifying permissions for a file or directory, use at least a three-digit octal number, omitting the digit in the first position. When you specify three digits instead of four, the first digit describes owner permissions, the second digit describes group permissions, and the third digit describes permissions for all others, as shown in Table 3-4.

Table 3-4 Permission bit setting using the chmod command

Octal Number	Meaning
666	owner(rw-) group(rw-) other(rw-)
700	owner(r--) group(r--) other(r--)
755	owner(rwx) group(r-x) other(r-x)
777	owner(rwx) group(rwx) other(rwx)

Authorization to the chmod command

You may wish to allow selected groups to use the **chmod** command to change the permission bits of any file or to use the **setfacl** command to manage ACLs for any file. As an enhancement to superuser granularity, when using the **chmod** command, a RACF service (IRRSCF00) has been updated to check the caller's authorization to profile SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class if the caller's user ID is not any of the following:

- The file owner
- A superuser with UID(0)
- A user who has switched to superuser being authorized to profile BPX.SUPERUSER in the FACILITY class

If a user executing the **chmod** command has at least read authority to the resource, that user is authorized to change the file mode in the same manner as a user having UID(0).

```
RDEF UNIXPRIV SUPERUSER.FILESYS.CHANGEPERMS OWNER(SECADM) UACC(NONE)
PE SUPERUSER.FILESYS.CHANGEPAERMS CL(UNIXPRIV) ID(LOCADM) ACCESS(READ)
```

These profiles allow users to use the **chmod** command to change the permission bits of any file and to use the **setfacl** command to manage access control lists for any file. In this example, an authorized selected group (LOCADM) has been chosen as the best candidate for administration:

```
PE SUPERUSER.FILESYS.CHANGEPAERMS CL(UNIXPRIV) ID(LOCADM) ACCESS(READ)
```

Note: The SUPERUSER.FILESYS.CHANGEPERMS resource name was added in z/OS V1R3.

3.12.5 Setuid, setgid, and sticky bits

In the FSP, shown in Figure 3-32 on page 135, there are three bits that control access to executable files in the file systems. An executable file can have an additional attribute displayed in the execute position (x) when you issue the **ls -l** command. This permission setting is used to allow a program temporary access to files that are not normally accessible to other users.

setgid and setuid bits

An s or S can appear in the execute permission position; this permission bit sets the effective user ID or group ID of the user process executing a program to that of the file whenever the file is run. The setuid and setgid bits are only honored for executable files containing load modules. These bits are not honored for shell script and REXX execs that reside in the file system:

s In the owner permissions section, this indicates that both the setuid bit and execute (search) permission are set.

In the group permissions section, this indicates that both the setgid bit and execute (search) permission are set.

S In the owner permissions section, this indicates the setuid bit is set, but the execute (search) bit is not.

In the group permissions section, this indicates the setgid bit is set, but the execute (search) bit is not.

A good example of this behavior is the mailx utility. A user sending mail to another user on the same system is actually appending the mail to the recipient's mail file, even though the sender does not have the appropriate permissions to do this—the mail program does.

A superuser or the file owner can use a **chmod** command or chmod() function to change two options for an executable file. The options are set in two file mode bits:

- Set-user-ID (S_ISUID) with the setuid option
- Set-group-ID (S_ISGID) with the setgid option

If one or both of these bits are on, the effective UID, effective GID, or both, plus the saved UID, saved GID, or both—for the process running the program—are changed to the owning UID, GID, or both, for the file. This change temporarily gives the process running the program access to data the file owner or group can access.

When creating a new file, both bits are set off. Also, if the owning UID or GID of a file is changed or if the file is written to, the bits are turned off. In shell scripts, these bits are ignored.

Sticky bit on a directory to control file access

Using the **mkdir**, **MKDIR**, or **chmod** commands, you can set the sticky bit on a directory to control permission to remove or rename files or subdirectories in the directory. When the bit is set, a user can remove or rename a file or remove a subdirectory only if one of these is true:

- The user owns the file or subdirectory
- The user owns the directory
- The user has superuser authority

If you use the **rmdir**, **rename**, **rm**, or **mv** commands to work with a file, and you receive a message that you are attempting an “operation not permitted”, check to see if the sticky bit is set for the directory the file resides in.

3.13 File and directory access checking

When a security decision is needed, the file system calls RACF and supplies the FSP (and ACL, if one exists). RACF makes the decision, does any auditing, and returns control to the file system. RACF does not provide commands to maintain the FSP (and ACL), but it does provide SAF services that do the FSP (and ACL) maintenance. z/OS UNIX provides commands that invoke these SAF services.

Authorization checking is done for all directories and files (including special files) in the file system. z/OS UNIX calls RACF to perform the authorization checking and passes RACF the FSP (file security packet), and the CRED (security credentials).

Figure 3-30 on page 131 shows the sequence of authorization checks, as follows:

- A superuser (UID of zero) is allowed access to all resources.
- If the effective UID of the process (the accessor) equals the UID of the file, RACF uses the owner permissions in the FSP to either allow or deny access.
- If the effective GID of the process equals the GID of the file, RACF uses the group permissions in the FSP to either allow or deny access. If RACF list-of-groups checking is active (SETROPTS GRPLIST), RACF will look at the user's connect groups that have a GID for a group that matches the GID of the file. If it finds a matching GID, RACF will allow or deny access based on the group permissions specified in the FSP. Note that if a user is connected to more that 300 z/OS UNIX groups, only the first 300 will be used.
- If the effective UID or GID of the process does not match the file UID or GID, then the other permission bits will determine access.

3.13.1 Controlling access to files for administrators

The profile SUPERUSER.FILESYS in the UNIXPRIV class provides the capability to authorize selected users to a large chunk of the superuser privileges, namely access to all local files. Authorization to the SUPERUSER.FILESYS resource provides privileges to access only local files. No authorization to access Network File System (NFS) files is provided by access to this resource.

It is one of 12 profiles in the UNIXPRIV class prefixed with SUPERUSER and was introduced in OS/390 V2R8, as shown in Figure 3-1 on page 83. Each of these profiles provides granular access to superuser privileges to various user groups. The SUPERUSER.FILESYS profile in the UNIXPRIV class has three access levels that allow access to z/OS UNIX files, as follows:

READ	Allows a user to read any local file, and to read or search any local directory.

UPDATE	Allows a user to write to any local file, and includes privileges of READ access.
CONTROL/ALTER	Allows a user to write to any local directory, and includes privileges of UPDATE access.

Depending on your security policy, you have to identify your users and groups and the level of access for usage of profile SUPERUSER.FILESYS. To authorize a user or group access to files and directories, issue the following commands:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS OWNER(SECADM) UACC(NONE)
PERMIT SUPERUSER.FILESYS CL(UNIXPRIV) ID(JANE) ACC(CONTROL)
PERMIT SUPERUSER.FILESYS CL(UNIXPRIV) ID(LOCADM) ACC(CONTROL)
```

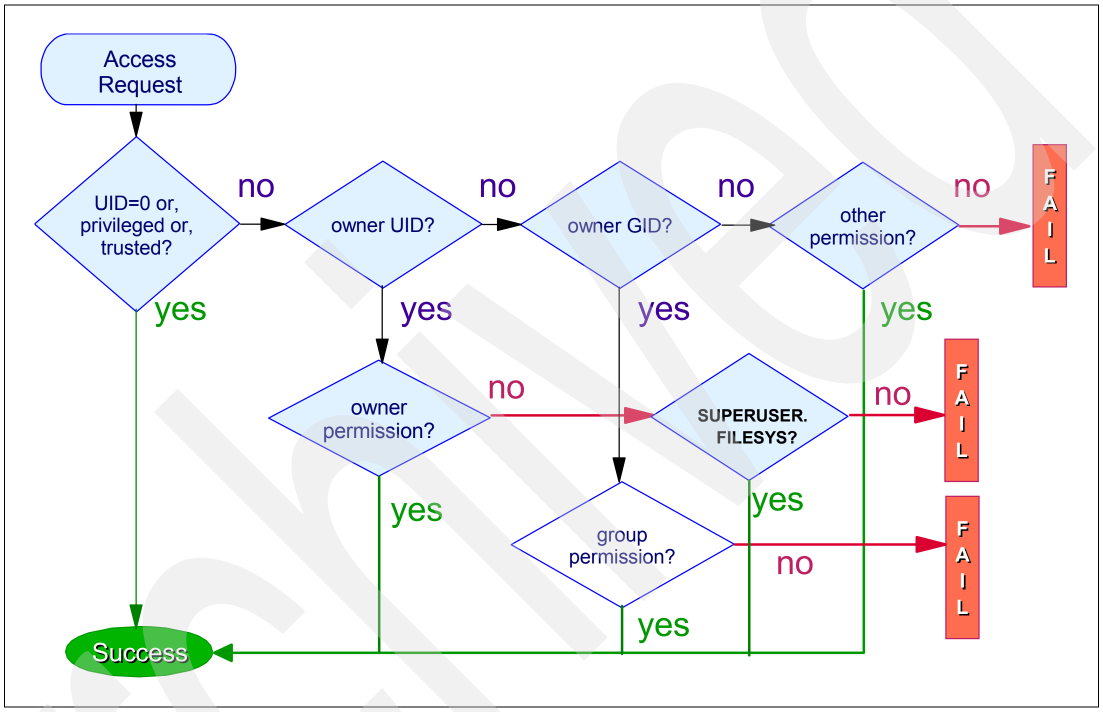


Figure 3-30 Security checking for access to files and directories without ACLs

When given access to the SUPERUSER.FILESYS profile, you may not be the owner UID, have access through the owner's group, or have access with the other permission bits, but you have access to the file or directory, as shown in Figure 3-30.

Note: Figure 3-30 is security checking without any ACLs and is for systems prior to z/OS V1R3. To see the security access flow *with* ACLs, which was introduced with z/OS V1R3, see Figure 3-34 on page 139.

3.13.2 Controlling access to files with RACF

Both the ACL and FSP are maintained by the physical file system (PFS). When a security decision for a z/OS UNIX user or a UNIX program is needed, as shown in Figure 3-31 on page 132, the file system calls the security product, supplying the ACL, if present, and the FSP.

If the security product supports ACLs, it applies its own rules to the file access request. Figure 3-31 on page 132 shows the access checking flow from a UNIX program to the security product. The z/OS UNIX kernel calls the file system iteratively for each directory component of the path name (one is required in order to locate the next), and the file system calls the security product. Next, the base file name is retrieved.

For each directory lookup, the file system calls the security product to make sure the user has search authority. Then the security product is called to insure the user has the requested access to the base file. Once RACF has checked the authorization, it returns control to the file system.

This is the basic architecture of every UNIX file system. The file system never has the complete path name; this is the major roadblock to implementing file protection with RACF profiles, and is the reason why ACLs are now introduced.

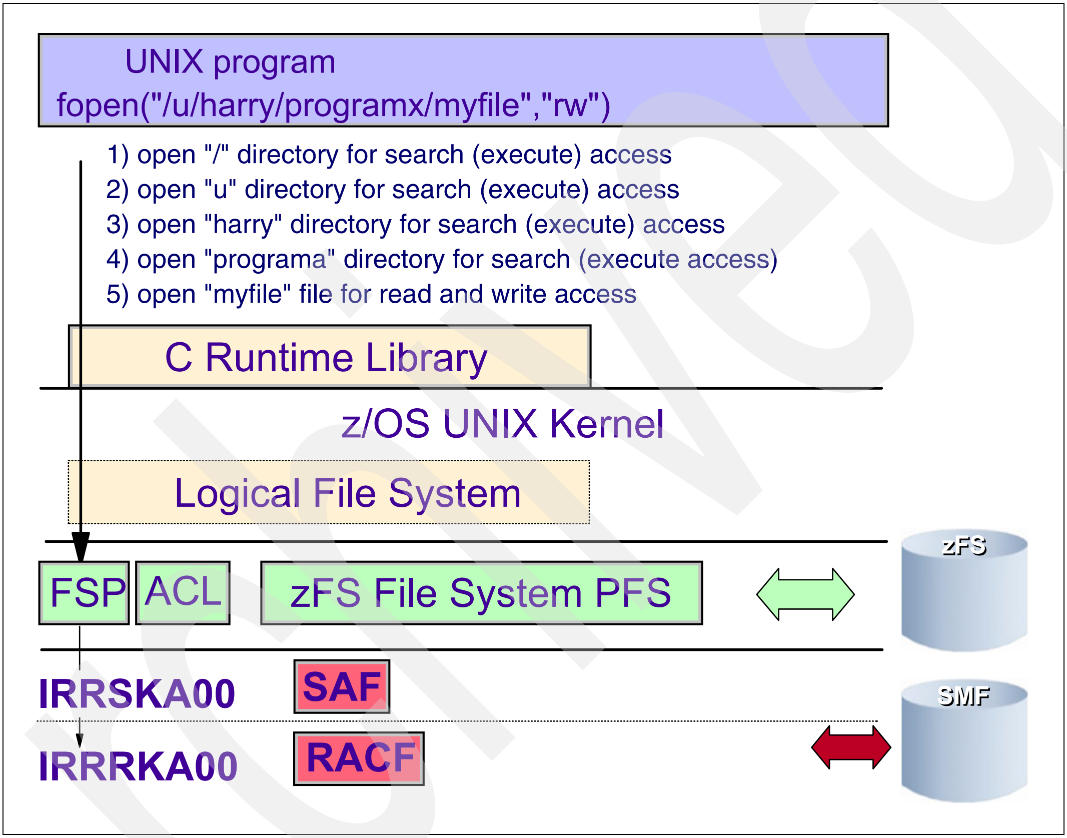

UNIX program
fopen("/u/harry/programx/myfile","rw")
1) open "/" directory for search (execute) access
2) open "u" directory for search (execute) access
3) open "harry" directory for search (execute) access
4) open "programa" directory for search (execute access)
5) open "myfile" file for read and write access
C Runtime Library
z/OS UNIX Kernel
Logical File System
FSP
ACL
zFS File System PFS
zFS
IRRSKA00
SAF
SMF
IRRRKA00
RACF

Figure 3-31 Access checking flow for directory and file access to RACF

UNIXPRIV profiles controlling access

RACF uses the permission bits, the access ACL, and the following UNIXPRIV class profiles, as shown in Figure 3-34 on page 139, to determine whether the user is authorized to access the file with the requested access level:

- SUPERUSER.FILESYS - (as described in 3.13.1, “Controlling access to files for administrators” on page 130)
- RESTRICTED.FILESYS.ACCESS - (new with z/OS V1R3)
- SUPERUSER.FILESYS.ACLOVERRIDE - (new with z/OS V1R3)

Controlling access to file system resources for restricted users

You can define a restricted user ID by assigning the RESTRICTED attribute through the ADDUSER or ALTUSER command, as follows:

```
ALTUSER RSTDUSER RESTRICTED
```

User IDs with the RESTRICTED attribute cannot access protected resources they are not specifically authorized to access. Access authorization for restricted user IDs bypasses global access checking. In addition, the UACC of a resource and an ID(*) entry on the access list are not used to enable a restricted user ID to gain access.

However, the RESTRICTED attribute has no effect when a user accesses a z/OS UNIX file system resource; the file's “other” permission bits can allow access to users who are not explicitly authorized.

To ensure that restricted users do not gain access to z/OS UNIX file system resources through “other” bits, you must define profile RESTRICTED.FILESYS.ACCESS profile in the UNIXPRIV class. If you have any user IDs with the RESTRICTED attribute in your RACF database and you wish to restrict them using HFS or zFS files through the “other” permission bits, issue the command:

```
RDEF UNIXPRIV RESTRICTED.FILESYS.ACCESS OWNER(SECADM) UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Note: An access list is not required to restrict the RESTRICTED user. The security check for this profile is made at (B) in Figure 3-34 on page 139.

Allow a RESTRICTED user access

Specifying UACC(READ) on RESTRICTED.FILESYS.ACCESS does not work, since a RESTRICTED user cannot be granted access via any UACC.

If you wish to override this restriction for a particular RESTRICTED user ID, you can permit this RESTRICTED user (or one of its groups) to RESTRICTED.FILESYS.ACCESS. This permit does not grant the user access to any files. It just allows the “other” bits to be used in access decisions for this user. To permit the RESTRICTED user to use the “other” bits, issue:

```
PERMIT RESTRICTED.FILESYS.ACCESS CL(UNIXPRIV) ID(RSTDUSER) ACCESS(READ)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Note: RESTRICTED.FILESYS.ACCESS is checked, as shown at (BB) in Figure 3-34 on page 139, for RESTRICTED users regardless of whether an ACL exists, so this function can be exploited regardless of whether you use ACLs or not.

If needed, explicitly allow certain restricted users to access certain files using the usual authorization method of adding those users, or one of their groups, to the file’s ACL using the **`setfacl`** command. (See 3.16, “Defining ACLs from the z/OS UNIX shell” on page 139.)

If a RESTRICTED user is authorized to SUPERUSER.FILESYS, his access to HFS will be honored regardless of the existence of the RESTRICTED.FILESYS.ACCESS profile.

For any given z/OS UNIX process, the result of the first check to the RESTRICTED.FILESYS.ACCESS resource will be cached for the life of the process. Therefore, subsequent authorization changes to this resource will not take effect for that process.

3.14 Access control lists (ACLs)

Access control lists have existed on various UNIX platforms for many years, but with variations in the interfaces. ACLs are introduced in z/OS V1R3 to provide a greater granularity for access to z/OS UNIX files and directories by RACF user IDs and groups. ACLs are created, modified, and deleted by using either the `setfacl` shell command or the ISHELL interface.

3.14.1 ACL entries

In the POSIX standard, two different ACLs are referenced as follows:

Base ACL entries	These entries are the same as permission bits (owner, group, other) that have always existed with z/OS UNIX files and directories. You can change the permissions using the `chmod` or the new `setfacl` command. They are not physically part of the ACL although you can use the `setfacl` command to change them and the `getfacl` command to display them.
Extended ACL entries	These entries are for individual users or groups and, like the permission bits, they are stored with the file, not in RACF profiles. Each ACL type (access, file default, directory default) can contain up to 1024 extended ACL entries. Each extended ACL entry specifies a qualifier to indicate whether the entry pertains to a user or a group; the actual UID or GID itself; and the permissions being granted or denied by this entry. The allowable permissions are read, write, and execute. As with other UNIX commands, the `setfacl` command allows the use of either names or numbers when referring to users and groups.

3.14.2 z/OS UNIX V1R3 ACL overview

The ACLs are created and checked by RACF, not by the kernel or file system. If a different security product is being used, you must check their documentation to see if ACLs are supported and what rules are used when determining file access.

ACL types

To reduce administrative overhead, three types of ACLs (extended ACLs) are defined to have the capability to inherit ACLs to newly created files and directories:

Access ACLs	This type of ACL is used to provide protection for a file system object (specific for a file or directory).
File default ACLs	This type is a model ACL that is inherited by files created within the parent directory. The file default ACL is copied to a newly created file as its access ACL. It is also copied to a newly created subdirectory as its file default ACL.
Directory default ACLs	This type is a model ACL that is inherited by subdirectories created within the parent directory. The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL.

Note: The phrases "default ACL" and "model ACL" are used interchangeably throughout z/OS UNIX documentation. Other systems that support ACLs have default ACLs that are essentially the same as the directory default ACLs in z/OS UNIX.

According to the X/Open UNIX 95 specification, additional access control mechanisms may only restrict the access permissions that are defined by the file permission bits. They cannot grant additional access permissions. Because z/OS ACLs can grant and restrict access, the use of ACLs is not UNIX 95-compliant.

3.14.3 FSP and access control lists

Before z/OS v1R3, the level of authorization for the file or directory through the FSP allowed specification of file permission bits for file owner (user), group owner (group), and anybody else (other) but could not permit or restrict the access to other specific users and groups.

The introduction of access control lists in the z/OS UNIX file system (with z/OS v1R3) allows granting and denying access to specific users and groups, in a manner similar to access lists of RACF profiles. The ACL is an SAF-owned construct that resides in the HFS system. ACLs are used in combination with the *traditional permission bits* in the FSP, as shown in Figure 3-32, to allow access to z/OS UNIX files and directories by any individual users (UIDs) and groups (GIDs) in addition to the owning user and the owning group.

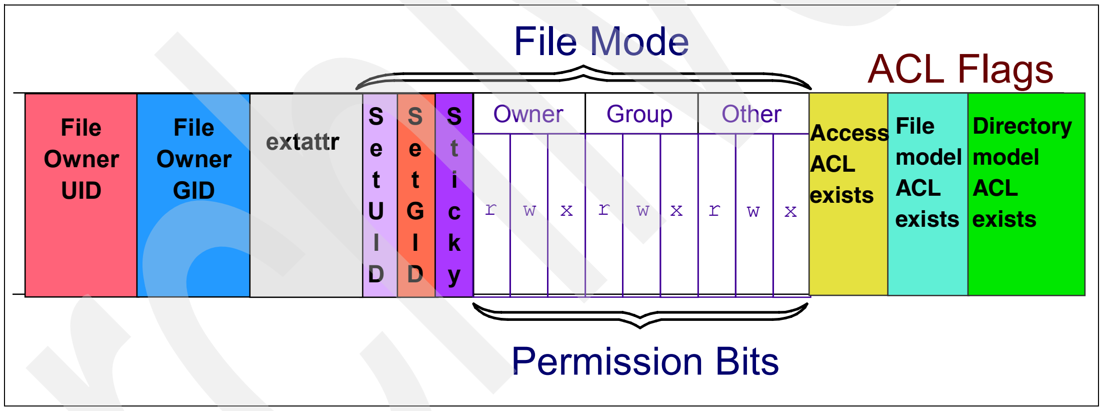


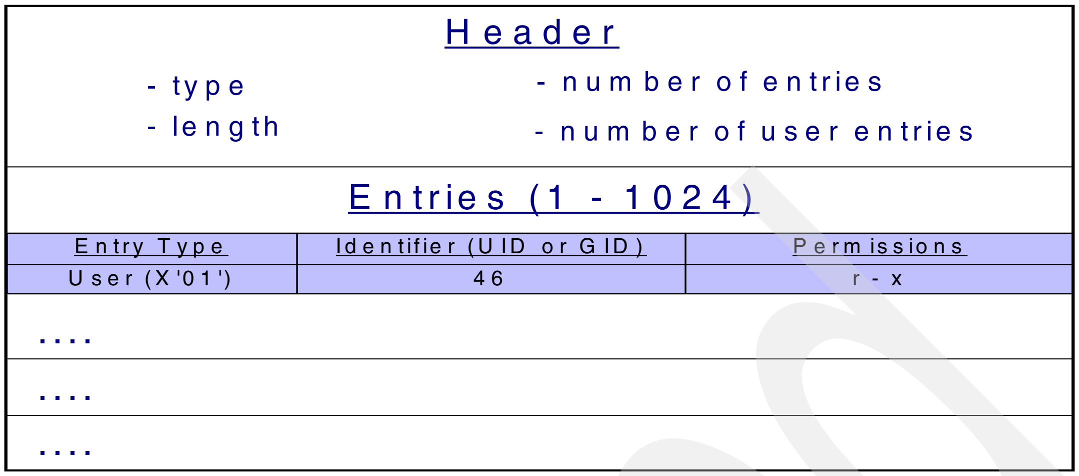
Figure 3-32 FSP updated to contain ACL flags with z/OS V1R3

In the framework of ACL, the traditional permission bits are called *base* ACLs, and the newly introduced ACLs are called *extended* ACLs. For more information about how ACLs are created and used, see 3.15, "Creating and accessing ACLs" on page 136.

3.14.4 ACL mapping

An ACL is mapped by the SAF IRRPFACL macro, as shown in Figure 3-33 on page 136, where the set of user entries is followed by the set of group entries.

The entries are sorted in ascending order by UID and GID to optimize the access checking algorithm. The algorithm consists of a list of entries (with a maximum of 1024) where every entry has information about the type (user or group), identifier (UID or GID), and permissions (read, write, and execute) that apply to a file or directory.


Header

- type
- length
- number of entries
- number of user entries

Entries (1 - 1024)

Entry Type	Identifier (UID or GID)	Permissions
User (X'01')	46	r - x
....		
....		
....		

Figure 3-33 Access control list table

There is no such thing as an empty ACL. If there is only one entry and it is deleted, the ACL table is automatically deleted.

3.14.5 ACL inheritance

ACL inheritance, as shown in Figure 3-38 on page 142, associates an ACL with the newly created file, *myfile*, without requiring administrative action. However, it is not always (and in fact, may seldom be) necessary to apply ACLs on every file or directory within a subtree. If you have a requirement to grant access to an entire subtree (for example, a subtree specific to a given application), then access can be established at the top directory.

If a given user or group does not have search access to the top directory, then no files within the subtree will be accessible, regardless of the permission bit settings or ACL contents associated with these files. The user or group will still need permission to the files within the directory subtree where appropriate. If this is already granted by the “group” or “other” bits, then no ACLs are necessary below the top directory.

3.15 Creating and accessing ACLs

Note: When defining ACLs, we recommend you place ACLs on directories, rather than on each file in a directory.

This support for ACLs allows you to control access to files and directories by an individual user (UID) and group (GID). z/OS UNIX file security on z/OS uses permission bits to control access to files, in accordance with the POSIX standard. However, the permission bit model does not allow for granting and denying access to specific users and groups, such as is possible using RACF profiles. This function will be provided by the introduction of ACLs in the z/OS UNIX file system.

An ACL is an SAF-owned construct that resides within the file system. The RESTRICTED attribute of a user is now applicable to file and directory access, as described in “Controlling access to file system resources for restricted users” on page 132.

3.15.1 Authority to create ACLs

To create an ACL for a file, you must have one of the following security access controls:

- Be the file owner
- Access to resource name BPX.SUPERUSER in the FACILITY class
- Have superuser authority (UID=0)
- Have READ access to profile SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class, as described in "Authorization to the chmod command" on page 128.

Activate RACF class FSSEC

To activate the use of ACLs in z/OS UNIX file authority checks, the following RACF command needs to be issued to activate the RACF FSSEC class:

```
SETROPTS CLASSACT(FSSEC)
```

You can define ACLs prior to activating the FSSEC class and display ACL information—but if the FSSEC class is not active, only the standard base permission bit checks are done, even if an ACL exists.

Activating the RACF FSSEC class causes the ACLs to be used during access checking. In order to set or modify ACLs, the same requirements are needed as for changing the permission bits.

3.15.2 Controlling access to files having ACLs for administrators

Any user who is not a superuser with UID(0), or the file owner—and who is denied access to a file through its ACL—can still access this file if the user has sufficient authority to the SUPERUSER.FILESYS resource in the UNIXPRIV class, as described in 3.13.1, "Controlling access to files for administrators" on page 130. To prevent this, you can force RACF to use ACL authorizations to override a user's SUPERUSER.FILESYS authority by using a UNIXPRIV class resource name, SUPERUSER.FILESYS.ACLOVERRIDE. To do this and prevent all users and not permit any users or groups, define the following profiles:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.ACLOVERRIDE OWNER(SECADM) UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Note: There is a relationship between the existing SUPERUSER.FILESYS profile and the new SUPERUSER.FILESYS.ACLOVERRIDE profile, which is checked at (A) in Figure 3-34 on page 139. Either profile could get checked for a file; it depends upon the presence of an ACL for the file that is checked at (4), and the contents of the ACL for granting access.

If needed, you can grant exceptions to certain groups or individual users to allow them to gain access based on their SUPERUSER.FILESYS authority. Place those groups to the access list of SUPERUSER.FYLESYS.ACLOVERRIDE with the same level of access they have for the SUPERUSER.FILESYS resource:

```
PERMIT SUPERUSER.FILESYS.ACLOVERRIDE CLASS(UNIXPRIV) ID(LOCADM) ACCESS(READ)
PERMIT SUPERUSER.FILESYS.ACLOVERRIDE CLASS(UNIXPRIV) ID(JANE) ACCESS(READ)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

SUPERUSER.FILESYS authority is still checked when an ACL does not exist for the file. This should be done for administrators for whom you want total file access authority. That is, you do not want anyone to deny them access to a given file or directory by defining an ACL entry for them without, or with, limited permission bit access.

3.15.3 RACF authorization checking flow with ACLs

The authorization checking beginning with z/OS V1R3, which replaces the flow shown in Figure 3-30 on page 131, with the changes shown in bold, is as follows:

1. Check owner permission bits.
2. Check user ACL entries.
3. Check the union of group permission bits and group ACL entries.

   All entries are checked until a single entry grants the requested access.
4. Check “other” permission bits.

Note: ACL entries are used only if the RACF FSSEC class is active.

Authorization checking flow for z/OS UNIX files and directories is shown in Figure 3-34 on page 139, and RACF makes the following checks:

- If the UID matches the file owner UID, the file's *user* permission bits are checked. If the user bits allow the requested access, then access is granted.

  If no user bits access is allowed and the FSSEC class is active, and an ACL exists, and there is an ACL entry for the user, then the permission bits of that ACL entry are checked. If at least one matching ACL entry was found for the UID, the processing continues with the ACLOVERRIDE checking.

  If no user ACL matches, then if the UNIXPRIV class is active, the SUPERUSER.FILESYS access is checked.
- If the GID matches the file owner GID, the file's group permission bits are checked. If the group bits allow the requested access, then access is granted.

  If any of the user's supplemental GIDs match the file owner GID, the file's group permission bits are checked. If the group bits allow the requested access, then access is granted.

  If no group bits access is allowed and the FSSEC class is active, and an ACL exists, and there is an ACL entry for any of the user's supplemental GIDs, then the permission bits of that ACL entry are checked. If at least one matching ACL entry was found for the GID, or any of the supplemental GIDs, then processing continues with the ACLOVERRIDE checking.

  If no group ACL matches, then if the UNIXPRIV class is active, the SUPERUSER.FILESYS access is checked.
- SUPERUSER.FILESYS.ACLOVERRIDE is checked only when a user's access was denied by a matching ACL entry based on the user's UID or one of the user's GIDs. If the user's access was denied by the file's permission bits, SUPERUSER.FILESYS is checked.

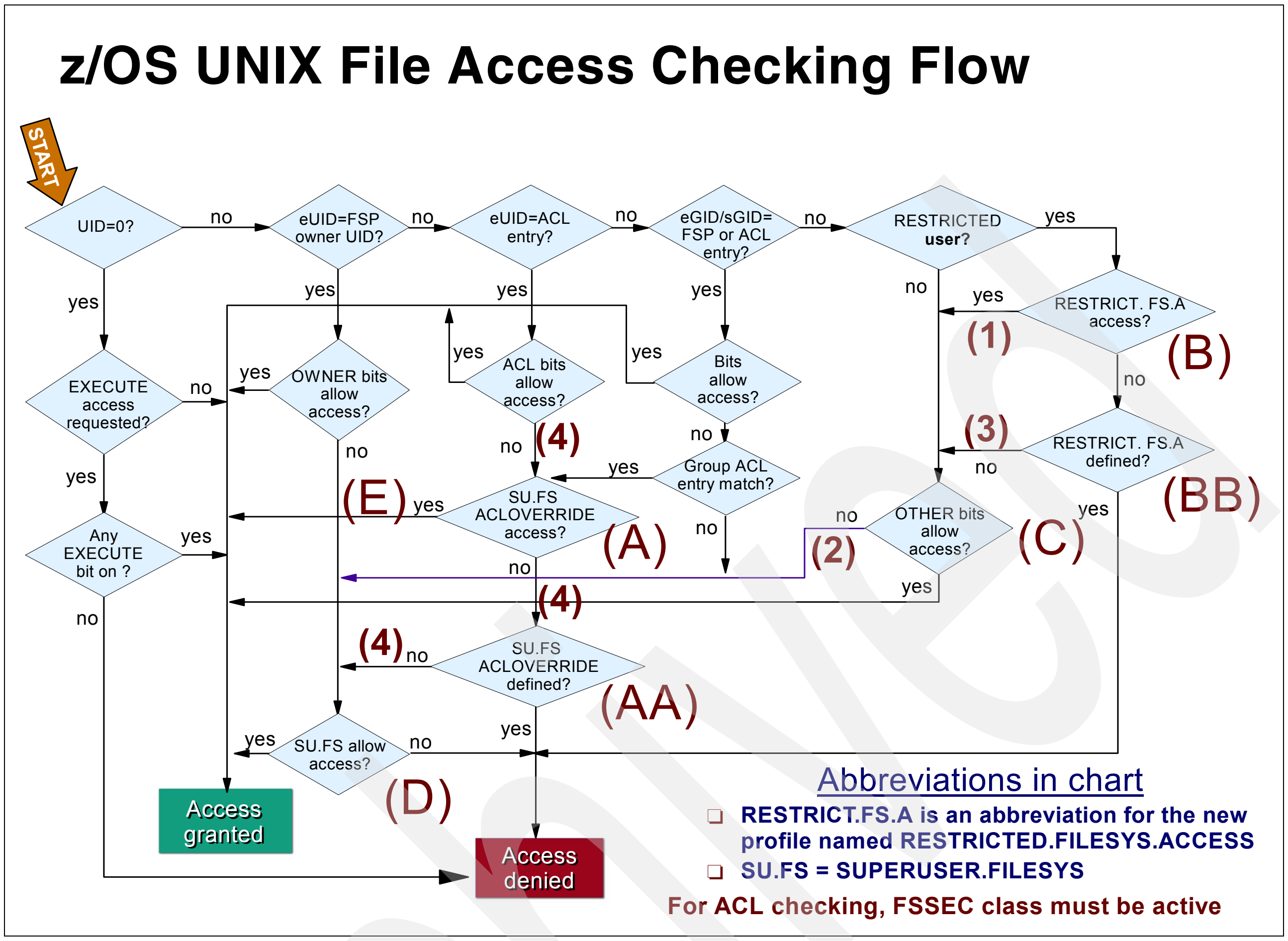


Figure 3-34 z/OS UNIX file access checking algorithm

3.16 Defining ACLs from the z/OS UNIX shell

There are two ways to define ACLs:

- Using the z/OS UNIX shell, after the OMVS command is issued, with the **setfacl** command
- Using the ISHELL and creating ACLs with the use of panels

The shell commands, **setfacl** and **getfacl**, are used to create, modify, delete, and display ACL entries specified by the path, as follows:

setfacl The **setfacl** command creates, modifies, and deletes an ACL definition for a file or directory. **setfacl** has the following syntax:

```
setfacl [–ahqv] -s entries [path ... ]
```

Sets an entire ACL (base and extended entries)

```
setfacl [–ahqv] -S file [path ...]
```

Sets (replaces) all ACLs with the entries specified in a file

```
setfacl [–ahqv] -D type [...][path ... ]
```

Delete some extended ACL entries and delete an entire ACL.

```
setfacl [–ahqv] -m|M|x|X EntryOrFile [...][path ... ]
```

Modify extended ACL entries

getfacl The **`getfacl`** command obtains and displays an ACL entry for a requested file or directory. It has the following syntax:

```
getfacl [–acdfhmos][-e user] file
```

Important: See *z/OS UNIX System Services Command Reference,* SA22-7802, for a complete explanation of the commands and parameters. Also, there are examples of the **`setfacl`** and **`getfacl`** commands in C.1, "Examples of the setfacl and getfacl commands" on page 560.

3.16.1 Define all three ACL types

The access ACL is used to provide protection for a file system directory or file. You have to use the **setfacl** command to define an access ACL from the z/OS UNIX shell. For these definitions, we are using the file system structure shown in Figure 3-35.

setfacl command

With the **`setfacl`** command, to create an ACL, if you use the **-s** option, you must create the entire ACL, which includes the base ACL and extended ACL, as described in 3.14.1, "ACL entries" on page 134. The base ACL (permission bits) is indicated by omitting user or group qualifiers. Since this example is also creating a file default ACL and a directory default ACL, all the ACLs must be created with a single command. Therefore, you should use the **-m** option for directory harry in Figure 3-35, as the following command creates an access ACL that gives user ID jane rwx access to directory harry, and creates the two default ACLs:

```
ROGERS @ SC65:/u>setfacl -m "u:jane:rwx,d:u:jane:rwx,f:u:jane:r--" harry
```

Note: When you are setting the access ACL and using the **`-s`** option, the ACL entries must consist of the three required base ACL entries that correspond to the file permission bits (u::rwx). The ACL entries must also consist of zero or more extended ACL entries (g::---,o::---,u:jane:rwx), which will allow a greater level of granularity when controlling access. The permissions for base entries must be in absolute form. See 3.14.1, "ACL entries" on page 134 for more information.

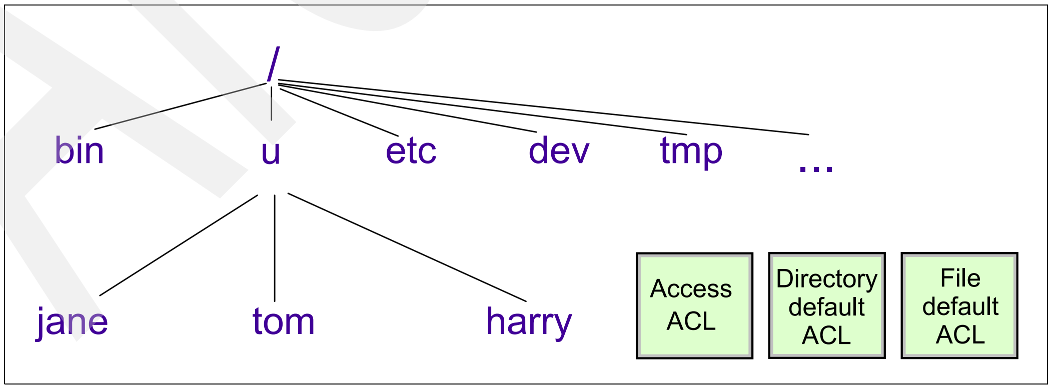


Figure 3-35 Three ACLs created for directory harry

List the directory /u

Issuing the **ls -al** command, shown in Figure 3-36, shows directory harry having a plus (+) sign following the permission bits, which indicates that an ACL exists for directory harry.

```
ROGERS @ SC65:/u>ls -al
total 152
dr-xr-xr-x  11 HAIMO    NOGROUP        0 Aug  2 10:45 .
drwxr-xr-x  48 HAIMO    SYS1       24576 Jul 25 14:44 ..
drwx------+  2 HARRY    SYS1        8192 Aug  2 10:44 harry
drwx------   2 JANE     SYS1        8192 Aug  2 10:44 jane
drwxr-xr-x   2 HAIMO    SYS1        8192 Jun 28 12:23 ldapsrv
drwx------   2 HAIMO    SYS1        8192 Aug  1 11:02 rogers
drwxr-xr-x   2 HAIMO    SYS1        8192 Nov 15  2001 syslogd
drwx------   3 HAIMO    SYS1        8192 May 26 11:03 user1
```

Figure 3-36 Command to show an ACL exists

Display the new ACLs

To display the ACLs just created (shown in Figure 3-37), issue the following command, where **-a** is for the access ACL, **d** is for the directory default ACL, and **f** is for the file default ACL.

```
ROGERS @ SC65:/u>getfacl -adf harry
#file:  harry/
#owner: HARRY
#group: SYS1
user::rwx
group::---
other::---
user:JANE:rwx
fdefault:user:JANE:r--
default:user:JANE:rwx
```

Figure 3-37 Display all the ACL types

3.16.2 Example of ACL inheritance

The directory structure is changed, as shown in Figure 3-38 on page 142, by issuing the following command:

```
mkdir /u/harry/programx
```

The new directory, programx, inherits an access ACL from the directory default ACL of directory harry, and inherits the directory default ACL and file default ACL from directory harry, as shown in Figure 3-38 on page 142.

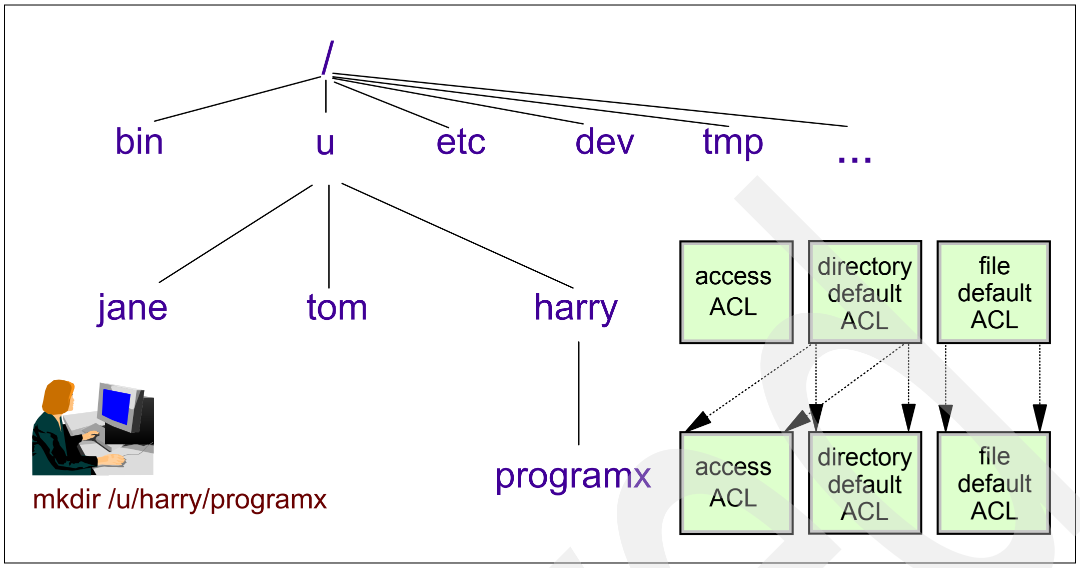


Figure 3-38 ACL inheritance from directory harry ACLs

Using the **getfacl** command (shown in Figure 3-39) you can see the inheritance from directory harry to directory programx.

```
HARRY @ SC65:/u/harry>getfacl -adf programx
#file:  programx/
#owner: HARRY
#group: SYS1
user::rwx
group::r-x
other::r-x
user:JANE:rwx
fdefault:user:JANE:r--
default:user:JANE:rwx
```

Figure 3-39 Display of ACLs for directory programx

The directory structure is changed again, as shown in Figure 3-40 on page 143, by issuing the following command, which creates a file named myfile:

```
oedit /u/harry/programx/myfile
```

The new file, myfile, inherits only the file default ACL from directory programx (Figure 3-40 on page 143).

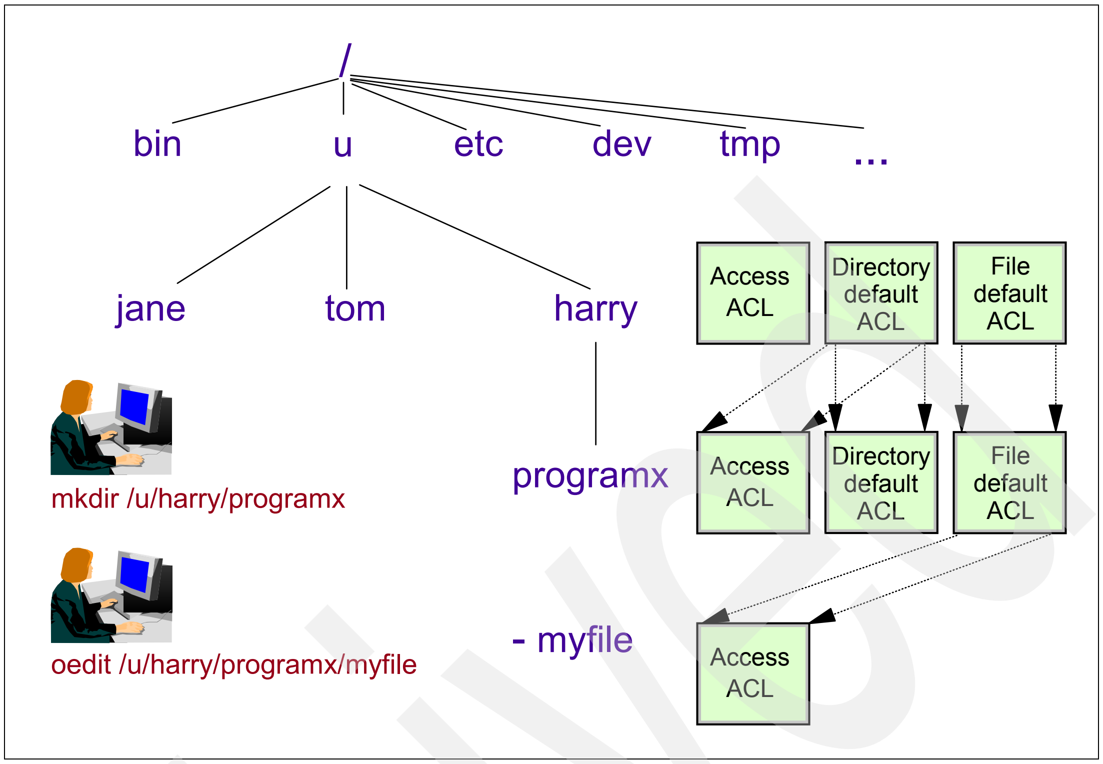


Figure 3-40 ACL inheritance of an access ACL to file myfile

User ID jane now has read access to the file, myfile, through ACL inheritance, and whose owner is harry.

By using the **getfacl** command to display the ACLs for file myfile, Figure 3-41 shows the file default ACL inherited from the directory programx (shown in Figure 3-39 on page 142).

```
HARRY @ SC65:/u/harry/programx>getfacl myfile
#file:  myfile
#owner: HARRY
#group: SYS1
user::rwx
group::---
other::---
user:JANE:r--
```

Figure 3-41 Display of ACL for file myfile

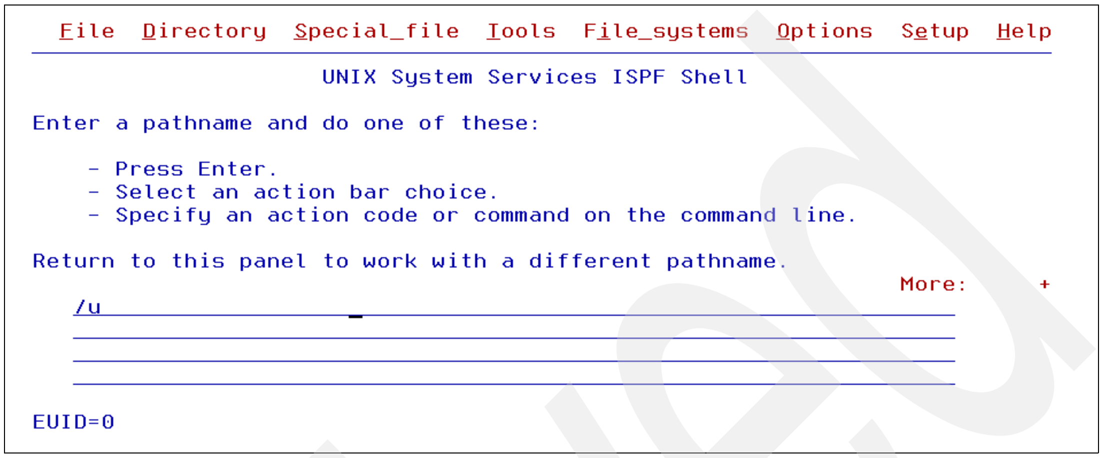
3.17 Defining ACLs from the ISHELL

If you prefer to use the ISHELL rather than the OMVS command line, you can use the ISHELL to display, add, delete, and modify ACLs.

For the example shown in Figure 3-35 on page 140, once you have entered the ISHELL, you should enter **/u** on the command line, as shown in Figure 3-42 on page 144. The following examples of defining ACLs from the ISHELL will repeat the examples using the **setfacl** command from the z/OS UNIX shell.

3.17.1 ISHELL panels to display and define ACLs

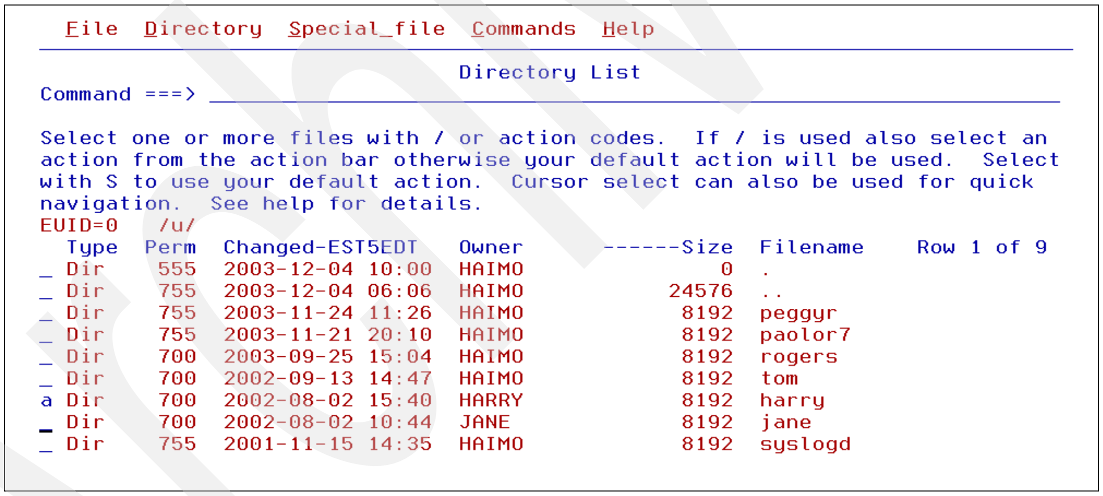
The ISHELL panels that follow will define an access ACL giving user jane access to directory harry, as shown in Figure 3-35 on page 140. First, issue the ISHELL command from Option 6 in ISPF, which displays the panel shown in Figure 3-42.



```
   File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
 ---------------------------------------------------------------------------
                     UNIX System Services ISPF Shell

 Enter a pathname and do one of these:

     - Press Enter.
     - Select an action bar choice.
     - Specify an action code or command on the command line.

 Return to this panel to work with a different pathname.
                                                              More:     +
    /u
    _________________________________________________________________
    _________________________________________________________________
    _________________________________________________________________

 EUID=0
```

Figure 3-42 ISHELL panel

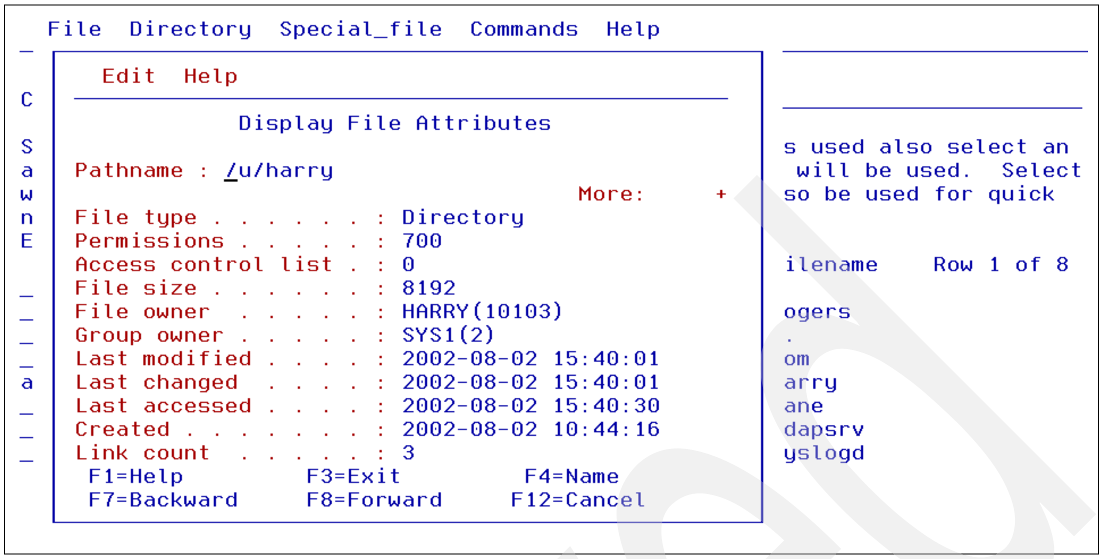
Entering **/u** and pressing Enter displays Figure 3-43, which shows the current directory list.



```
   File  Directory  Special_file  Commands  Help
 ----------------------------------------------------------------------------
                               Directory List
 Command ===> ______________________________________________________________

 Select one or more files with / or action codes.  If / is used also select an
 action from the action bar otherwise your default action will be used.  Select
 with S to use your default action.  Cursor select can also be used for quick
 navigation.  See help for details.
 EUID=0   /u/
   Type  Perm  Changed-EST5EDT   Owner      ------Size  Filename    Row 1 of 9
 _ Dir    555  2003-12-04 10:00  HAIMO               0  .
 _ Dir    755  2003-12-04 06:06  HAIMO           24576  ..
 _ Dir    755  2003-11-24 11:26  HAIMO            8192  peggyr
 _ Dir    755  2003-11-21 20:10  HAIMO            8192  paolor7
 _ Dir    700  2003-09-25 15:04  HAIMO            8192  rogers
 _ Dir    700  2002-09-13 14:47  HAIMO            8192  tom
 a Dir    700  2002-08-02 15:40  HARRY            8192  harry
 _ Dir    700  2002-08-02 10:44  JANE             8192  jane
 _ Dir    755  2001-11-15 14:35  HAIMO            8192  syslogd
```

Figure 3-43 ISHELL directory list panel

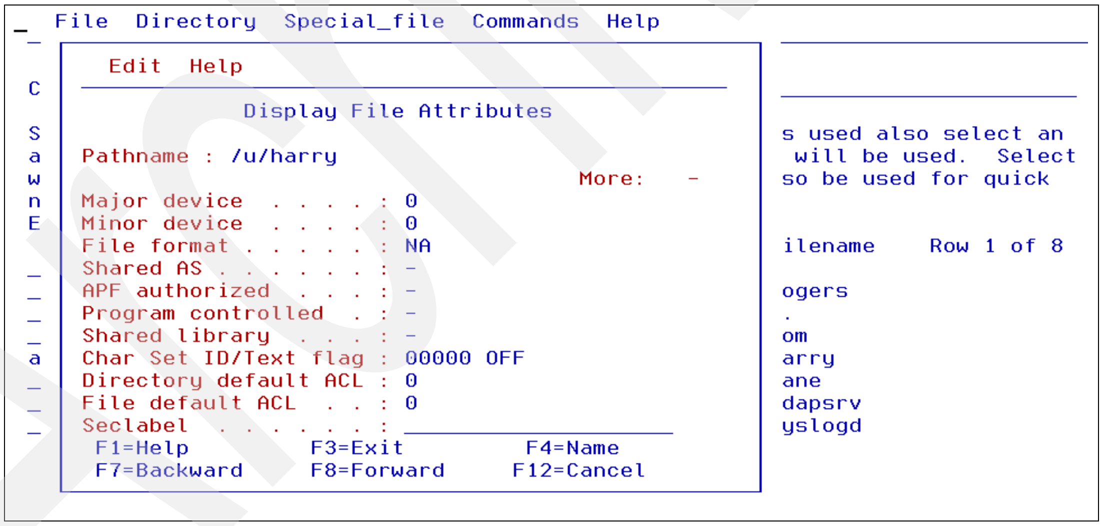
By placing an **a** action code for directory harry, shown in Figure 3-43, the File Attribute panel is displayed, as shown in Figure 3-44 on page 145.



```
  File  Directory  Special_file  Commands  Help
_   ┌──────────────────────────────────────────────┐  ______________________
    │  Edit  Help                                  │
C   │ ───────────────────────────────────────────── │  ______________________
    │             Display File Attributes          │
S   │                                              │  s used also select an
a   │ Pathname : _/u/harry                         │   will be used.  Select
w   │                                  More:     + │  so be used for quick
n   │ File type . . . . . . . : Directory          │
E   │ Permissions . . . . . . : 700                │
    │ Access control list . . : 0                  │  ilename    Row 1 of 8
_   │ File size . . . . . . . : 8192               │
_   │ File owner  . . . . . . : HARRY(10103)       │  ogers
_   │ Group owner . . . . . . : SYS1(2)            │  .
_   │ Last modified . . . . . : 2002-08-02 15:40:01│  om
a   │ Last changed  . . . . . : 2002-08-02 15:40:01│  arry
_   │ Last accessed . . . . . : 2002-08-02 15:40:30│  ane
_   │ Created . . . . . . . . : 2002-08-02 10:44:16│  dapsrv
_   │ Link count  . . . . . . : 3                  │  yslogd
    │  F1=Help         F3=Exit         F4=Name     │
    │  F7=Backward     F8=Forward     F12=Cancel   │
    └──────────────────────────────────────────────┘
```

Figure 3-44 Display File Attributes panel

Note: The field "Access control list" shows if an access ACL exists. If you scroll forward, you will see whether a Directory default ACL and File default ACL is defined, as shown in Figure 3-45 on page 145. These two fields (Directory default and File default ACL) only apply to directory files.

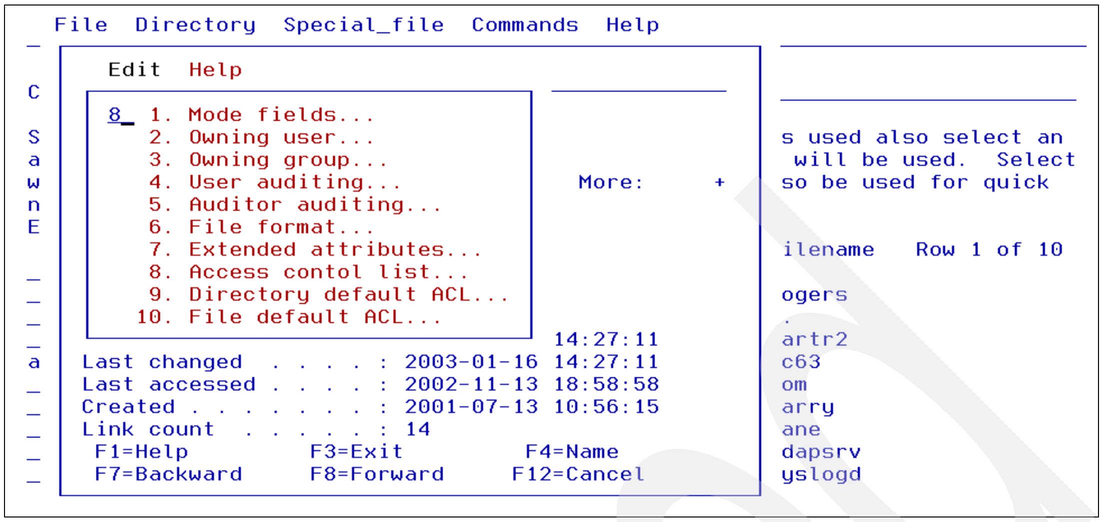


```
_  File  Directory  Special_file  Commands  Help
_   ┌──────────────────────────────────────────────┐  ______________________
    │  Edit  Help                                  │
C   │ ───────────────────────────────────────────── │  ______________________
    │            Display File Attributes           │
S   │                                              │  s used also select an
a   │ Pathname : /u/harry                          │   will be used.  Select
w   │                                  More:   -   │  so be used for quick
n   │ Major device  . . . . . : 0                  │
E   │ Minor device  . . . . . : 0                  │
    │ File format . . . . . . : NA                 │  ilename    Row 1 of 8
_   │ Shared AS . . . . . . . : -                  │
_   │ APF authorized  . . . . : -                  │  ogers
_   │ Program controlled  . . : -                  │  .
_   │ Shared library  . . . . : -                  │  om
a   │ Char Set ID/Text flag : 00000 OFF            │  arry
_   │ Directory default ACL : 0                    │  ane
_   │ File default ACL  . . . : 0                  │  dapsrv
_   │ Seclabel  . . . . . . . : ________________   │  yslogd
    │  F1=Help         F3=Exit         F4=Name     │
    │  F7=Backward     F8=Forward     F12=Cancel   │
    └──────────────────────────────────────────────┘
```

Figure 3-45 Display of directory default ACL and file default ACL for /u/harry

By placing the cursor under the Edit pull-down in Figure 3-44 and pressing Enter, you can choose the option for displaying ACL information, as shown in Figure 3-46 on page 146.

Options 8, 9, and 10 can be selected to display the access ACL, the directory default ACL, and the file default ACL.

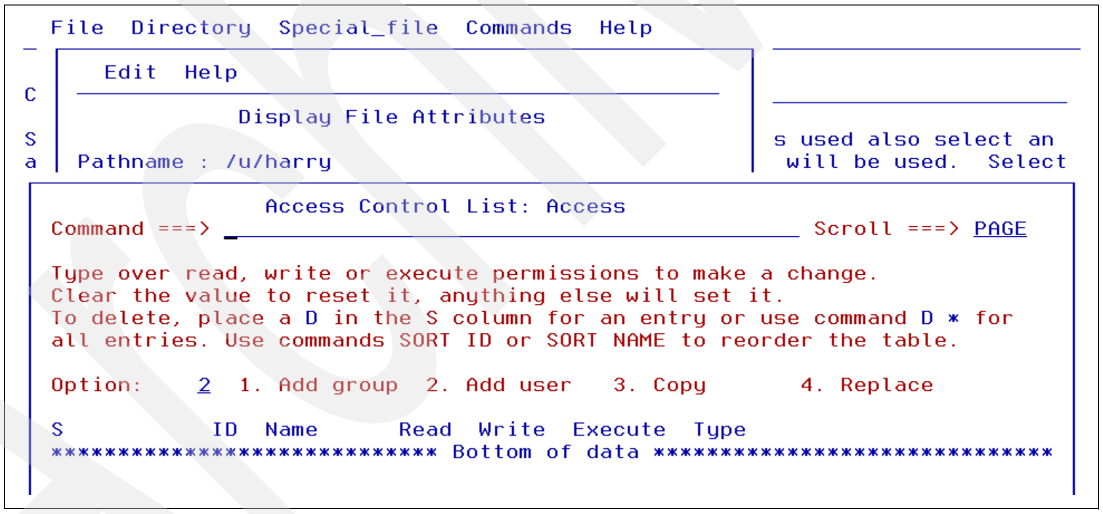


```
  File  Directory  Special_file  Commands  Help
_  ┌──────────────────────────────────────────────┐ ____________________
   │   Edit  Help                                 │
C  │  ┌────────────────────────────┐  ___________ │ ____________________
   │  │ 8_  1. Mode fields...      │              │
S  │  │     2. Owning user...      │              │ s used also select an
a  │  │     3. Owning group...     │              │  will be used.  Select
w  │  │     4. User auditing...    │   More:     +│ so be used for quick
n  │  │     5. Auditor auditing... │              │
E  │  │     6. File format...      │              │
   │  │     7. Extended attributes...             │ ilename   Row 1 of 10
_  │  │     8. Access contol list...              │
_  │  │     9. Directory default ACL...           │ ogers
_  │  │    10. File default ACL... │              │ .
_  │  └────────────────────────────┘  14:27:11    │ artr2
a  │ Last changed  . . . . : 2003-01-16 14:27:11  │ c63
_  │ Last accessed . . . . : 2002-11-13 18:58:58  │ om
_  │ Created . . . . . . . : 2001-07-13 10:56:15  │ arry
_  │ Link count  . . . . . : 14                   │ ane
_  │  F1=Help          F3=Exit         F4=Name    │ dapsrv
_  │  F7=Backward      F8=Forward     F12=Cancel  │ yslogd
   └──────────────────────────────────────────────┘
```

Figure 3-46 Panel showing Options 8, 9, and 10 to define ACLs

Define an access ACL for directory harry

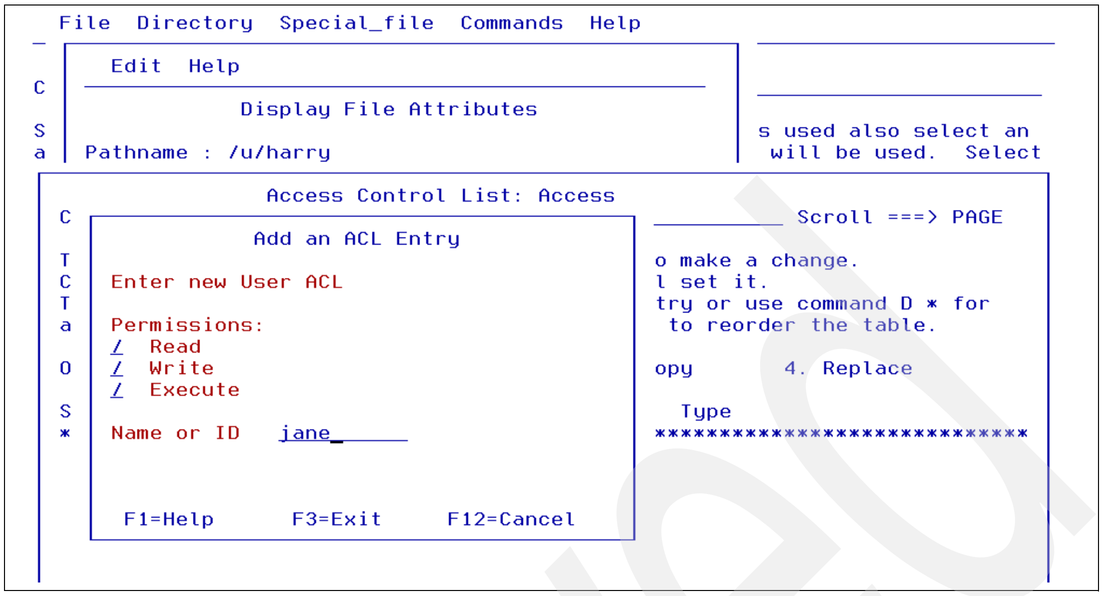
When you press Enter after specifying option 8, 9, or 10, you now have access to modify, add, display, or delete ACL entries, as shown in Figure 3-47. From this panel you can select Option 2 to define an access ACL. If an ACL already exists, you can modify or delete the ACL entry.



```
  File  Directory  Special_file  Commands  Help
_                                                ____________________
   Edit  Help
C  ___________________________________________   ____________________
              Display File Attributes
S                                                s used also select an
a  Pathname : /u/harry                            will be used.  Select

              Access Control List: Access
 Command ===> _______________________________________  Scroll ===> PAGE

 Type over read, write or execute permissions to make a change.
 Clear the value to reset it, anything else will set it.
 To delete, place a D in the S column for an entry or use command D * for
 all entries. Use commands SORT ID or SORT NAME to reorder the table.

 Option:    2  1. Add group  2. Add user   3. Copy      4. Replace

 S            ID  Name      Read  Write  Execute  Type
 ***************************** Bottom of data ******************************
```

Figure 3-47 Access Control List panel to change ACL definitions for Option 8

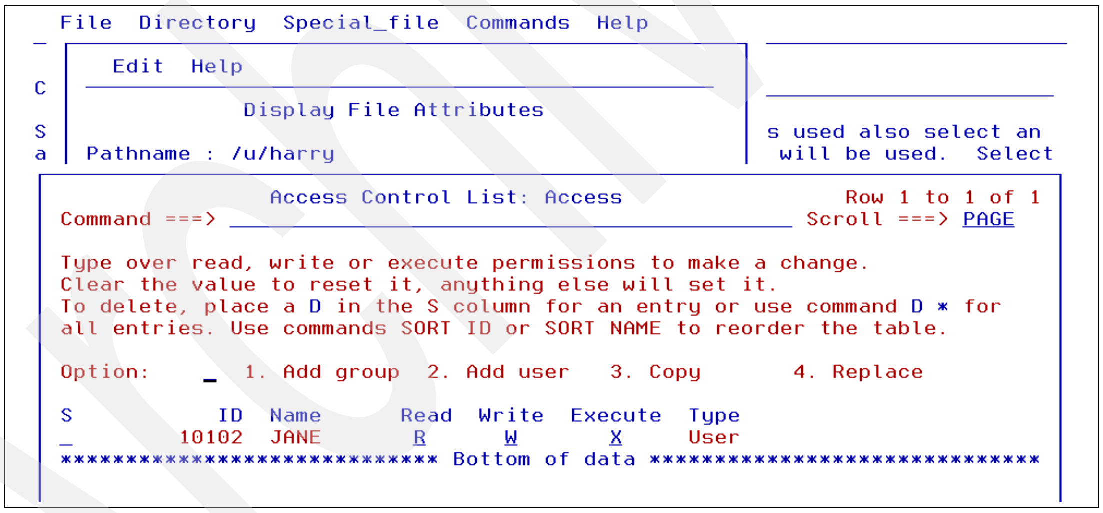
To add user jane by creating an access ACL for directory harry, select Option 2 (Add user) and press Enter, and Figure 3-48 on page 147 displays the panel for the defining of the access ACL for directory harry, giving user jane rwx access.



```
  File  Directory  Special_file  Commands  Help
 _  ┌──────────────────────────────────────────────────┐  ______________________
    │ Edit  Help                                       │
 C  │ ──────────────────────────────────────────────── │  _____________________
    │              Display File Attributes             │
 S  │                                                  │  s used also select an
 a  │ Pathname : /u/harry                              │   will be used.  Select
   ┌─────────────────────────────────────────────────────────────────────────┐
   │                Access Control List: Access                              │
   │ C ┌──────────────────────────────────────────┐ _________  Scroll ===> PAGE
   │   │           Add an ACL Entry               │
   │ T │                                          │ o make a change.
   │ C │ Enter new User ACL                       │ l set it.
   │ T │                                          │ try or use command D * for
   │ a │ Permissions:                             │  to reorder the table.
   │   │ /  Read                                  │
   │ O │ /  Write                                 │ opy       4. Replace
   │   │ /  Execute                               │
   │ S │                                          │   Type
   │ * │ Name or ID   jane_____                   │ *****************************
   │   │                                          │
   │   │                                          │
   │   │  F1=Help      F3=Exit      F12=Cancel    │
   │   └──────────────────────────────────────────┘
```

Figure 3-48 Define the access ACL for directory harry

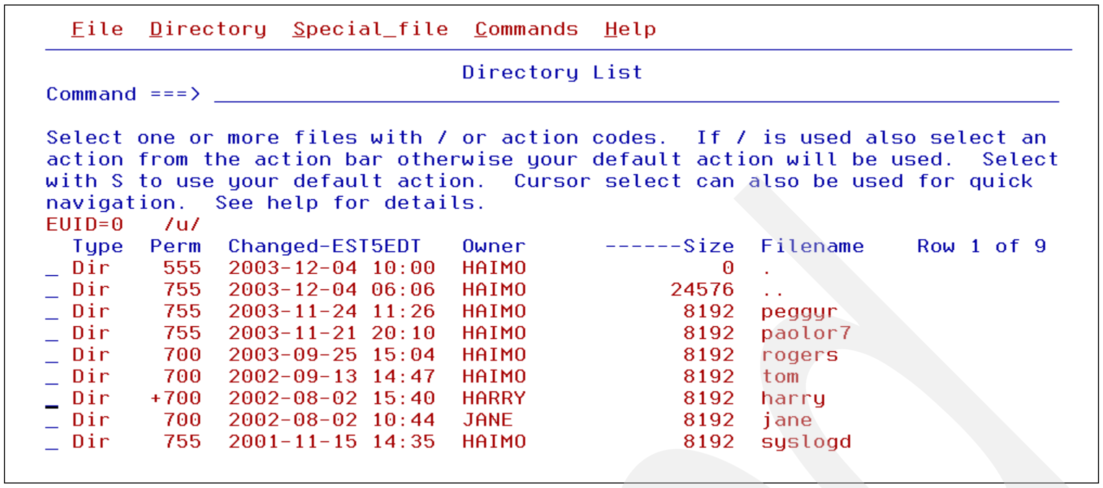
When you press Enter after adding the access ACL, the panel shown in Figure 3-49 shows user jane with UID(10102) and the access ACL access jane has to directory harry.



```
  File  Directory  Special_file  Commands  Help
 _  ┌──────────────────────────────────────────────────┐  ______________________
    │ Edit  Help                                       │
 C  │ ──────────────────────────────────────────────── │  _____________________
    │              Display File Attributes             │
 S  │                                                  │  s used also select an
 a  │ Pathname : /u/harry                              │   will be used.  Select
   ┌──────────────────────────────────────────────────────────────────────────┐
   │               Access Control List: Access                Row 1 to 1 of 1
   │ Command ===> ____________________________________________ Scroll ===> PAGE
   │
   │ Type over read, write or execute permissions to make a change.
   │ Clear the value to reset it, anything else will set it.
   │ To delete, place a D in the S column for an entry or use command D * for
   │ all entries. Use commands SORT ID or SORT NAME to reorder the table.
   │
   │ Option:     _   1. Add group  2. Add user   3. Copy      4. Replace
   │
   │ S          ID   Name       Read  Write  Execute  Type
   │ _       10102   JANE        R      W       X     User
   │ ***************************** Bottom of data ******************************
```

Figure 3-49 Panel showing the result of defining the access ACL

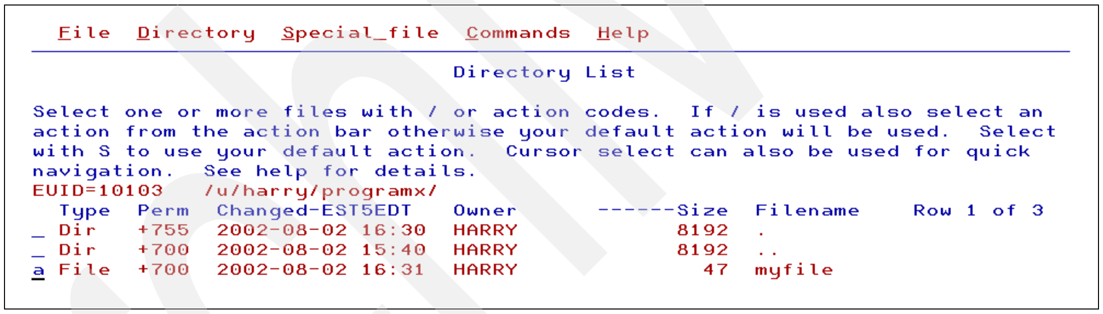
Figure 3-50 on page 148 shows the directory list panel with directory harry now having a + sign indicating that an ACL exists for the directory.



```
  File  Directory  Special_file  Commands  Help
 ------------------------------------------------------------------------------
                               Directory List
Command ===> _________________________________________________________________

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=0   /u/
  Type  Perm  Changed-EST5EDT  Owner      ------Size  Filename    Row 1 of 9
_ Dir    555  2003-12-04 10:00 HAIMO               0  .
_ Dir    755  2003-12-04 06:06 HAIMO           24576  ..
_ Dir    755  2003-11-24 11:26 HAIMO            8192  peggyr
_ Dir    755  2003-11-21 20:10 HAIMO            8192  paolor7
_ Dir    700  2003-09-25 15:04 HAIMO            8192  rogers
_ Dir    700  2002-09-13 14:47 HAIMO            8192  tom
_ Dir   +700  2002-08-02 15:40 HARRY            8192  harry
_ Dir    700  2002-08-02 10:44 JANE             8192  jane
_ Dir    755  2001-11-15 14:35 HAIMO            8192  syslogd
```

Figure 3-50 Directory list panel showing a + sign for directory harry

You can create a directory default ACL by choosing Option 9 and a file default ACL by choosing Option 10 on the panel shown in Figure 3-46 on page 146. If you complete the example shown previously in Figure 3-40 on page 143 and then use the ISHELL, Figure 3-51 and Figure 3-52 on page 149 show the access ACL that was inherited by the file myfile from directory programx.



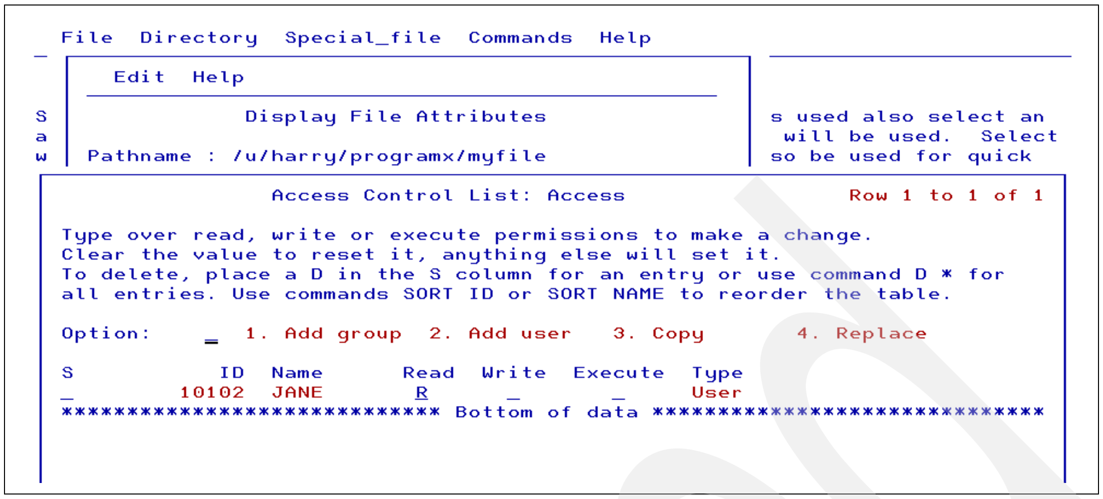
```
  File  Directory  Special_file  Commands  Help
 ------------------------------------------------------------------------------
                               Directory List

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=10103   /u/harry/programx/
  Type  Perm  Changed-EST5EDT  Owner      ------Size  Filename    Row 1 of 3
_ Dir   +755  2002-08-02 16:30 HARRY            8192  .
_ Dir   +700  2002-08-02 15:40 HARRY            8192  ..
a File  +700  2002-08-02 16:31 HARRY              47  myfile
```

Figure 3-51 ISHELL display of file myfile showing an ACL exists

To display Figure 3-52 on page 149, do the following:

Place an **a** action character for file myfile as shown in Figure 3-51.

By placing the cursor under the Edit pull-down and pressing Enter, you can choose the option for displaying ACL information, Option 8. When you press Enter, Figure 3-52 on page 149 is displayed, showing the access jane has to the file myfile.



```
  File  Directory  Special_file  Commands  Help
_ ┌──────────────────────────────────────────────────┐ ──────────────────────
  │   Edit  Help                                     │
  │ ──────────────────────────────────────────────── │
S │              Display File Attributes             │ s used also select an
a │                                                  │  will be used.  Select
w │ Pathname : /u/harry/programx/myfile              │ so be used for quick
 ┌──────────────────────────────────────────────────────────────────────────┐
 │                 Access Control List: Access                 Row 1 to 1 of 1 │
 │                                                                          │
 │ Type over read, write or execute permissions to make a change.           │
 │ Clear the value to reset it, anything else will set it.                  │
 │ To delete, place a D in the S column for an entry or use command D * for │
 │ all entries. Use commands SORT ID or SORT NAME to reorder the table.     │
 │                                                                          │
 │ Option:   _  1. Add group  2. Add user   3. Copy       4. Replace        │
 │                                                                          │
 │ S           ID  Name       Read  Write  Execute  Type                    │
 │ _        10102  JANE        R     _       _      User                    │
 │ **************************** Bottom of data ****************************** │
```

Figure 3-52   Access control list window

User ID jane now has read access to the file myfile, whose owner is user ID harry.

3.17.2 The IRRHFSU utility and ACLs

The IRRHFSU utility unloads the UNIX System Services Hierarchical File System file security information in a manner compatible with the IRRDBU00 utility. RACF provides the IRRDBU00 utility to unload the contents of the RACF database into a flat file suitable for viewing or loading into a relational database for querying. Similarly, the IRRHFSU utility downloads data contained in the FSP (such as file permission bits, owning UID and GID, owner- and auditor-specified logging options) into a UNIX file or an MVS data set.

The **find** command can locate files with ACLs containing “orphaned” ACL references; that is, entries for UIDs ad GIDs that cannot be mapped to RACF user or group profiles. However, the **find** output is not useful for removing these references, because the UID or GID is not reported as part of the output. The IRRHFSU utility can be invoked with a parameter, which results in deletion of orphaned ACL entries. For more additional information, see:

http://www-1.ibm.com/servers/eserver/zseries/zos/racf/goodies.html

3.17.3 Modified commands with ACL support

The following commands have been modified in order to support ACL entries:

The getconf command

This changed command returns configuration values associated with the file at the specified path name, as follows:

_PC_ACL - Indicates whether an access control mechanism is supported by the file system owning the file specified by “pathname”. A value of 1 indicates that it is supported, as shown in Figure 3-53, and a value of 0 indicates it is not supported.

_PC_ACL_ENTRIES_MAX - Maximum number of entries in an ACL for a file or directory, as shown in Figure 3-53, which indicates a value of 1024.

```
@ SC63:/>getconf _PC_ACL /u/user1/test
1
@ SC63:/>getconf _PC_ACL_ENTRIES_MAX /u/user1/test
1024
```

Figure 3-53 Examples of the getconf command

The ls command

The **ls** command indicates the existence of ACLs by adding a plus sign (+) character after the permission bits, as follows:

```
SC63:/>ls -l /u/user1
drwxr-xr-x+  2 HAIMO    SYS1        8192 May 26 09:42 test find
```

The find command

The **find** command has new options for supporting ACL entries.

- The **find** command finds all files or directories with an ACL of a given type. In the next example, the command displays all the files and directories in the /u/user1 directory that have any type of ACL (access, default file, or default directory):

```
AYVIVAR @ SC63:/>find /u/user1 -acl a -o -acl d -o -acl f
/u/user1/test
```

- Find files with ACL entries for a specific user or group. In the next example, **find** displays all the files and directories under the /u/user1 directory that have ACL entries for SYS1 group:

```
@ SC63:/>find /u/user1 -acl_group SYS1
/u/user1/test
```

- Find files with more than the specified amount of ACL entries:

```
@ SC63:/>find /u/user1 -acl_count +1
/u/user1/test
```

- In the following example, the **find** command is useful in command substitution, as it can produce file lists that are used as input to the **setfacl** command:

```
setfacl -m g:OMVSGRP:rwx $(find /u/user1 -acl_group SYS1)
```

The cp command

The **cp -p** command preserves ACLs from source to target, if possible. The ACLs are not preserved if a file system does not support ACLs, or if you are copying files to MVS.

The mv command

The **mv** command preserves an ACL from source to target.

The pax command

When using the **pax** command, ACL data is automatically stored in USTAR formatted archives using special headers. The following options are not required:

- Extracted files will restore ACLs when **-p A** or **-p e** is specified.
- Copy preserves ACL when **-p A** or **-p e** is specified.
- Verbose output adds a plus sign (+) character after the permission bits when an extended ACL exists.

```
@ SC65:/u/user1>pax -vf test.pax
-rwx------   1 STC      SYS1      620000 May  3 15:28 /u/user1/test/file1
-rwx------+  1 STC      SYS1      660000 May  3 15:29 /u/user1/test/file2
```

The tar command

The **tar -U** command (with USTAR format) will preserve ACLs in archives as follows:

- Extracted files will restore ACLs when **-A** is specified.
- For verbose output (**tar -v**), a + character is added to the end of the file permission bits for all files with extended ACLs (as for the **pax** command).

The df command

The **df -v** command indicates whether the file system and security product supports ACLs, as shown in Figure 3-54.

```
@ SC63:/>df -v /u/user1/test
Mounted on     Filesystem               Avail/Total    Files      Status
/u/user1       (OMVS.USER1.HFS)         14208/14400    4294967293 Available
HFS, Read/Write, Device:241, ACLS=Y
File System Owner : SC63        Automove=Y      Client=N
Filetag : T=off   codeset=0
```

Figure 3-54 Example of df -v command

Notes:

- ACLS=Y does *not* mean that the FSSEC class profile is active. It means that the file system will store ACLs and pass them to the security product.
- Using ACLs must be supported by the file system that the file or directory belongs to. It is supported in z/OS V1.3 by zFS and HFS. ACLs are not currently supported for a temporary file system (TFS) in z/OS V1R3.

3.17.4 Using ACLs in a sysplex

Using ACLs should be no different on a sysplex client than on a sysplex server system if all the participating systems are running at V1R3 or higher.

In a sysplex environment, all participating nodes must be on a release level that has ACL support. If any of the participating nodes are at a release level that does not contain ACL support and you have enabled the FSSEC class on an up-level node, then files that are protected by ACLs will not be accessible on down-level nodes (assuming that the compatibility APAR has been applied) except perhaps by a superuser or file owner. The APAR is OW50655 for SAF and OW49334 for RACF.

3.18 Daemons and security

MVS, traditional UNIX, and z/OS UNIX systems manage user identities differently. A daemon is a long-lived process that runs unattended to perform continuous or periodic system-wide functions, such as network control. Some daemons are triggered automatically to perform their task; others operate periodically. Daemons have superuser authority and can issue authorized functions such as setuid(), seteuid(), and spawn() to change the identity of a user's process.

In many cases a daemon program is started from the kernel and inherits the kernel user ID, OMVSKERN. The daemon can have a separate user ID as long as the user ID is defined as a superuser. This superuser must be defined with a UID=0 in RACF, which means that this user cannot become a superuser by using the **su** command.

Daemons typically encountered on z/OS UNIX systems include:

cron	The batch scheduler. The cron daemon is a clock daemon that runs commands at specified dates and times. You can specify regularly scheduled commands with the `crontab` command. Jobs that are to be run only once can be submitted using the `at` or `batch` commands. cron runs commands with priorities and limits set by a queuedefs file.
ftpd	The file transfer daemon supplied with z/OS Communications Server (CS).
inetd	The Internet daemon. The inetd daemon provides service management for a network. It starts the rlogind program or otelnetd program whenever there is either a remote login request or a remote Telnet login from a workstation.
rlogind	The remote login daemon. The rlogind program is the server for the remote login command `rlogin`. It validates the remote login request and verifies the password of the target user. It starts an z/OS UNIX shell for the user and handles translation between ASCII and EBCDIC code pages as data flows between the workstation and the shell.
syslogd	The syslog daemon supplied with z/OS Communications Server. syslogd is a server process that has to be started as one of the first processes in your z/OS UNIX environment. Other servers and stack components use syslogd for logging purposes and can also send trace information to syslogd.
otelnetd	The remote logon daemon supplied with z/OS Communications Server.
orexecd	The remote execution protocol daemon supplied with z/OS Communications Server. The Remote Execution Protocol Daemon (REXECD) is the server for the REXEC routine. REXECD provides remote execution facilities with authentication based on user names and passwords.
uucpd	The UNIX-to-UNIX copy program daemon introduced with z/OS V1R2. The uucpd daemon is used to communicate with any UNIX system that is running a version of the UNIX-to-UNIX copy program. UUCP functions are used to automatically transfer files and requests for command execution from one UUCP system to another usually in batch mode at particular times. Other daemons associated with UUCP included with z/OS V1R2 are: **uucico** Processes uucp file transfer requests. **uuxqt** Runs commands from remote UUCP systems.
Orouted	The Orouted daemon is supplied with z/OS Communications Server. The route daemon is a server that implements the Routing Information Protocol (RIP) (RFC 1058). It provides an alternative to the static TCP/IP gateway definitions.
lpd	The line printer daemon supplied with z/OS Communications Server. It enables printers from any TCP/IP host that are attached to the MVS spooling system.
timed	The time daemon supplied with the z/OS Communications Server. It provides clients with UTC time. Network stations without a time chip obtain clocks from this daemon.
httpd	The http daemon supplied with the IBM HTTP Server.

3.18.1 Security environment for daemons

You can run daemons with regular UNIX security, as shown in Figure 3-55, or with z/OS UNIX security. A z/OS UNIX daemon could also be described as a classical server process.

For administrators, controlling daemons requires some extra considerations:

- How and when is a daemon process started (or restarted if it fails)?
- Daemons often need initialization options customized to installation requirements.
- Daemons have the ability to issue **setuid()**. Access to this type of power needs to be controlled, by controlling which programs can be a daemon.
- The special user security profile BPXROOT must be created in order to support some daemon operations.

3.18.2 UNIX-level security

Initially, the daemon process is started by an external command or event. Once started, the daemon *listens* for work requests from clients. When a request is received, the daemon notes the UID of the requester, and then forks a child process to carry out the request. The forked child process inherits UID(0) from the daemon process. Before executing the request, the daemon uses a special SYSCALL **setuid** to reset the security environment to match the UID of the requester. UNIX-level security for daemons means that all daemon programs execute as superusers, and all superusers are allowed to use the `setuid()` and `seteuid()` functions to change the identity of a process to any other UID. Their MVS identity will be changed to the one corresponding to the UID value; for example, the cron daemon in Figure 3-55 changes its identity to UID=25, which is the MVS user ID BOB.

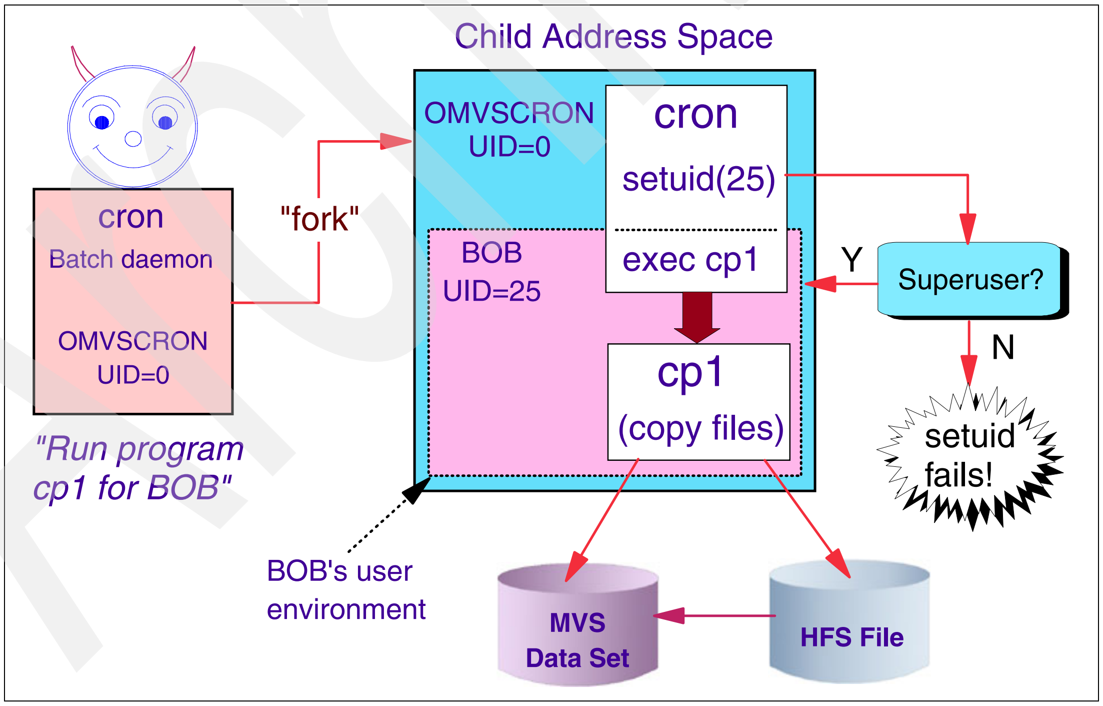


Figure 3-55 Normal UNIX security with daemon processing

Note: The important point about the **setuid** instruction is that, in an z/OS environment, it resets the whole security profile of the forked address space. The UID is set to the requester's UID and the current RACF user ID information (the ACEE) is changed to BOB to complement the UID. The requester's task therefore runs with access to both the UNIX and z/OS resources (data sets) owned by BOB.

3.19 z/OS UNIX level security for daemons

With z/OS UNIX, there are two levels of security you can provide that are a higher level than UNIX-level security, as follows:

- BPX.DAEMON defined in the RACF FACILITY class; see 3.19.1, "BPX.DAEMON FACILITY class profile" on page 154.
  - RACF program control protection
- RACF running with enhanced program security, BPX.DAEMON defined and BPX.MAINCHECK defined. BPX.MAINCHECK is introduced with z/OS V1R4. You can use BPX.MAINCHECK for any privileged z/OS UNIX application that requires a program controlled environment, because the application uses a privileged z/OS UNIX service that requires one. An example is the __passwd() service, which is used by applications such as telnet and rlogin. See 3.19.3, "Enhanced program security mode z/OS V1R4" on page 158.
  - RACF program control protection

3.19.1 BPX.DAEMON FACILITY class profile

If the BPX.DAEMON FACILITY class is defined, your system has z/OS UNIX security. Your system can exercise more control over your superusers, as follows:

- Any superuser permitted to this profile has the daemon authority to change MVS identities via z/OS UNIX services without knowing the target user ID's password. This identity change can only occur if the target user ID has an OMVS segment defined. If BPX.DAEMON is not defined, then all superusers (UID=0) have daemon authority. If you want to limit which superusers have daemon authority, define this profile and permit only selected superusers to it.
- Any program loaded into an address space that requires daemon level authority must be defined to program control. If the BPX.DAEMON profile is defined, then z/OS UNIX will verify that the address space has not loaded any executables that are uncontrolled before it allows any of the following services that are controlled by z/OS UNIX to succeed:
  - seteuid
  - setuid
  - setreuid
  - pthread_security_np()
  - auth_check_resource_np()
  - _login()
  - _spawn() with user ID change
  - _password()

This level of security is for customers with stricter security requirements who need to have some superusers maintaining the file system but want to have greater control over the z/OS resources that these users can access. Although BPX.DAEMON provides some additional control over the capabilities of a superuser, a superuser should still be regarded as a privileged user because of the full range of privileges the superuser is granted.

The additional control that BPX.DAEMON provides involves the use of kernel services such as setuid() that change a caller's z/OS user identity. With BPX.DAEMON defined, a superuser process can successfully run these services if the following are true:

- The caller's user identity has been permitted to the BPX.DAEMON FACILITY class profile.
- All programs running in the address space have been loaded from a library controlled by a security product. A library identified to RACF Program Control is an example. Individual files in the HFS can be identified as controlled programs.

Kernel services that change a caller's z/OS user identity require the target z/OS user identity to have an OMVS segment defined. If you want to maintain this extra level of control at your installation, you will have to choose which daemons to permit to the BPX.DAEMON FACILITY class. You will also have to choose the users to whom you give the OMVS security profile segments.

Give daemon authority to the kernel. Most daemons that inherit their identities from the kernel address space are started from /etc/rc. To authorize the OMVSKERN user ID for the daemon FACILITY class profile, issue the following commands:

```
RDEFINE BPX.DAEMON OWNER(SECADM) UACC(NONE)
PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Important: If the BPX.DAEMON FACILITY class is not defined, your system has UNIX-level security as described in 3.18.2, “UNIX-level security” on page 153. In this case, the system is less secure.

In order for daemon processes to be able to invoke **setuid()** for superusers, define a superuser with a user ID of BPXROOT on all systems. To define the BPXROOT user ID, issue:

```
ADDUSER BPXROOT DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/')
                 PROGRAM('/bin/sh')  NOPASSWORD
```

Exactly how requests are passed to a daemon depends on the type of daemon. For TCP/IP-based daemons like the telnet or rlogin daemons, the user request is passed over a socket connection from the IP network. For the cron daemon, the request is passed via a parameter area and a cross memory post to the daemon.

Important: It can easily be seen that daemon authority is far-reaching and offers possibilities for compromising the security and integrity of the system. Therefore, READ access to profile BPX.DAEMON should be given to as few user IDs in the system as possible. Some examples of daemons that need READ access to BPX.DAEMON are: cron, uucpd, rlogind, and rshd.

3.19.2 RACF program control protection

The purpose of protecting load modules is to provide installations with the ability to control who can execute what programs and to treat those programs as assets. Any program loaded into an address space with daemon authority must be a controlled program.

You protect individual load modules (programs) by creating a profile for the program in the PROGRAM general resource class. A program protected a profile in the PROGRAM class is called a *controlled* program.

The name of the profile can be complete, in which case the profile protects only one program, or the name of the profile can end with an asterisk (*), in which case the profile can protect more than one program. For example, a profile named ABC* protects all programs that begin with ABC, unless a more specific profile exists. In this way you can find which programs are causing the environment (such as PADS checking) to not work.

```
RDEFINE  PROGRAM   *  UACC(READ)  ADDMEM +
                ('SYS1.LINKLIB'/'******'/NOPADCHK +
                'CEE.SCEERUN'/RLTPAK/NOPADCHK +
                'SYS1.SEZALOAD'//NOPADCHK )
SETROPTS  WHEN(PROGRAM)  REFRESH
```

The profile for a controlled program must also include the name of the program library that contains the program and the volume serial number of the volume containing the program library. The profile can also contain standard access list of users and groups and their associated access authorities.

Program access to data sets (PADS)

PADS allows an authorized user or group of users to access specified data sets with the user's authority to execute a certain program. That is, some users can access specified data sets at a specified access level only while executing a certain program (and the program access is restricted to controlled programs).

To set up program access to data sets, create a conditional access list for the data set profile protecting the data sets. To do this, specify WHEN(PROGRAM(program-name)) with the ID and ACCESS operands on the PERMIT command. Specifying the WHEN(PROGRAM) operand requires that the user or group specified must be running the specified program to receive the specified access.

Choosing between the PADCHK and NOPADCHK operands: With the ADDMEM operand of the RDEFINE and RALTER commands, you can also specify PADCHK or NOPADCHK as follows:

NOPADCHK	NOPADCHK means that RACF does not perform the program-accessed data checks for the program. The program is loaded and has access to any currently opened program-accessed data sets, even though the user ID/program combination is not in the conditional access list. NOPADCHK allows an installation to define entire libraries of modules (such as the PL/I transient routines or ISPF) as controlled programs without having to give each of these modules explicit access to many program-accessed data sets Use NOPADCHK if you trust the programs to access only data they should.
PADCHK	PADCHK (the default) means that RACF checks for program-accessed data sets that are already open before executing the program. If there are any open program-accessed data sets, RACF ensures, before it allows this program to be loaded, that this user ID/program combination is in the conditional access list of each data set.

Program control considerations for daemons

Program control can be used to protect both MVS data sets and files that are in HFS or zFS file systems. In most cases, programs loaded into an address space that requires daemon authority must be controlled programs, as follows:

- All HFS programs must be program-controlled.

  You define profile BPX.FILEATTR.PROGCTL in the FACILITY class and authorize your z/OS UNIX administrators to have READ access to allow them to set the program control extended attribute for an HFS program.

Authority to profile BPX.SUPERUSER will not be enough to perform this task. The most obvious candidates for authorization to BPX.FILEATTR.PROGCTL are members of administrative groups or individuals. Issue the following RACF commands:

```
RDEF FACILITY BPX.FILEATTR.PROGCTL OWNER(SECADM) UACC(NONE)
PERMIT BPX.FILEATTR.PROGCTL CL(FACILITY) ID(HARRY) ACCESS(READ)
```

- Programs loaded from MVS libraries do not have to be program-controlled if the BPX.DAEMON.HFSCTL in the FACILITY class profile has been defined. This FACILITY class profile was introduced in z/OS V1R2.

When defining BPX.DAEMON.HFSCTL, this can override some of the program control rules for daemons and servers that definition of BPX.DAEMON and BPX.SERVER normally require. BPX.DAEMON and BPX.SERVER normally restrict the daemon or server environment to the following situations:

- Executing only those MVS programs defined to RACF in the PROGRAM class
- Executing z/OS UNIX programs defined to with the extattr +p (setting program control)

  By defining BPX.DAEMON.HFSCTL and permitting the daemon or server to access that profile, you allow it to execute MVS programs that are not defined in the PROGRAM class. This requires that any z/OS UNIX program that it executes must be defined with extattr +p.

  Note: Because defining and allowing access to BPX.DAEMON.HFSCTL slightly weakens security in a daemon or server environment, you should carefully consider and restrict its use to those cases where you cannot run a certain function without it.

Security checks for daemon processing

When a daemon address space starts processing, the following security processing takes place:

- If a service that changes a caller's z/OS user identity, such as **`setuid()`**, is used, the kernel checks to see if BPX.DAEMON has been defined.
- If it has, then the kernel checks whether all programs loaded into the address space have been defined to program control.
- If an uncontrolled program has been loaded, the address space is marked *dirty*.
- If marked dirty, the controlled program cannot do any of the controlled functions, such as **`setuid()`**.
- An error return code and reason code are issued. All BPX.DAEMON privileges are revoked, including the right to check passwords.

Program control programs

The following programs are considered to be or should be program-controlled:

- All modules loaded from LPA
- Daemons that are shipped by z/OS reside in the HFS and are controlled programs, so you do not need to define them to program control.
- RTLS libraries must be defined to program control

  If you are using RTLS, you must set up FACILITY profiles as documented in the CSVRTLxx description in *z/OS MVS Initialization and Tuning Reference,* SA22-7592.

Programs can be defined to program control in the following ways:

- MVS load modules can be loaded from a load library, where all modules in the library can be defined to program control, or specific modules in the library can be defined to program control.
- The module can reside in the HFS with the sticky bit on. This causes the system to search with MVS search order and the rules for program control apply as above.
- The module can reside in the HFS with the external attribute set for program control.

3.19.3 Enhanced program security mode z/OS V1R4

Program control authorizes users to programs via PROGRAM class profiles. With program control, programs can be protected. Program access to data sets (PADS) authorizes users to data sets while running a particular program via DATASET profiles. With PADS, data sets can be protected by restricting access to specified users only when running particular programs.

Prior to z/OS V1R4, when specifying a program name in the conditional access list, the name of the program that actually did the loading needed to be known. Situations where the user invokes one program, which actually opens another data set, required you to know both program names rather than just the high level program name. With this enhancement in z/OS V1R4, you only have to know the high level name.

New enhanced security mode for PADS

In the RACF profile IRR.PGMSECURITY, in the RACF FACILITY class, a new enhanced program security mode can be specified that provides improved usability and increased security when using PADS.

Using IRR.PGMSECURITY, the APPLDATA specifies whether RACF will operate in basic, enhanced, or enhanced-warning PGMSECURITY mode, as specified by using the APPLDATA keyword. The new modes for the APPLDATA( ) keyword are:

(BASIC)	If the APPLDATA is exactly (BASIC), then RACF will run in basic PGMSECURITY mode.
(ENHANCED)	If the APPLDATA is exactly {ENHANCED), then RACF will run in enhanced PGMSECURITY mode.
( )	If the APPLDATA is empty or contains any other value, such as (ENHWARN), RACF will run in enhanced PGMSECURITY mode, which is a warning mode, rather than failure mode.

Recommendation: Use the ENHANCED-WARNING program security mode as part of your implementation of ENHANCED program security mode.

ENHANCED-WARNING mode

With ENHANCED-WARNING mode, RACF ensures that programs accessing data sets through PADS, or running execute-controlled programs, meet the added restrictions of ENHANCED mode. However, if they do not meet the added restrictions, RACF still allows the access if it would have worked in BASIC mode. This allows you to test your setup to make sure it is suitable for ENHANCED mode, while continuing to operate like BASIC mode while you adjust your profiles.

When you migrate to the new mode, you will have some profiles defined in the PROGRAM class but probably none of them specify APPLDATA(MAIN) or APPLDATA(BASIC), as those specifications do not mean anything in BASIC program security mode. Therefore, specify the IRR.PGMSECURITY profile defined in the FACILITY class and use the APPLDATA to specify your desired mode.

For example, to use the new ENHANCED-WARNING mode, do the following:

1. Use the RDEFINE command to define the IRR.PGMSECURITY profile in the FACILITY class, and specify an APPLDATA value other than ENHANCED or BASIC; for example:

   ```
   RDEFINE FACILITY IRR.PGMSECURITY APPLDATA('ENHWARN')
   ```

2. Issue the SETROPTS REFRESH command to change modes:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

Attention: You should remain in warning mode until you have done at least one IPL, to ensure that you have tested with all your daemons.

To ease migration from BASIC to ENHANCED program security mode, the mode switch does not affect systems running any release earlier than z/OS V1R4. It also does not affect jobs, started tasks, or TSO sessions that are already running. For this reason, you should IPL the system at least once while in ENHANCED-WARNING mode to ensure that you test any jobs, started tasks, and TSO users that started before you migrate from BASIC to ENHANCED program security mode.

While running in ENHANCED-WARNING mode, you may receive messages ICH427I or ICH430I to indicate the need for further necessary changes. After receiving the messages, making the relevant changes, and allowing a sufficient test period of running in ENHANCED-WARNING mode without getting further messages, you can switch to ENHANCED program security mode.

For additional information on this new enhancement, see *z/OS V1R4.0 Security Server RACF Security Administrator's Guide,* SA22-7683.

The mode becomes effective at SETR WHEN(PROGRAM) or SETR WHEN(PROGRAM) REFRESH. The default mode is BASIC.

Program control and PADS will function as before if the FACILITY IRR.PGMSECURITY profile or the FACILITY class are not activated.

RDEFINE PROGRAM defines each program control. You can add APPLDATA to specific PROGRAM class profiles. ADDMEM is still needed for library data. Following is the definition of specific program control:

```
RDEFINE PROGRAM pgmname APPLDATA('value')
```

The APPLDATA values are as follows:

MAIN	Trusted enhanced mode program
BASIC	Program exempted from enhanced PGMSECURITY, and it overrides ENHANCED mode.
anything else	Not trusted in enhanced mode

SPECIFIC profiles are only valid. First program must be specified as MAIN or BASIC for authorization. MAIN applies only to first program in // EXEC PGM=program or TSOEXEC program. BASIC applies to first program of any TCB and to all daughter TCBs. BASIC allows use of old security programs with ENHANCED mode. You should realize that BASIC weakens security in ENHANCED mode.

For new PADS, you should specify the first program for any mother TCB rather than the OPENing program, for example the first program described in // EXEC PGM=program or TSOEXEC program.

Let us consider some example of this support. When you use // EXEC PGM=A and program A links to program B, which does OPEN, prior to z/OS V1R4, you had to specify B in a conditional access list, and maybe specify A unless defined as NOPADCHK. Now you can specify either A or B. If running in ENHANCED mode, program A must be specified as MAIN or BASIC. You still need to specify the other program unless it is defined as NOPADCHK.

When you use // EXEC PGM=A and module A ATTACHes module B, which does an OPEN, prior to z/OS V1R4, you had to specify module B in a conditional access list, and maybe specify module A unless it was defined as NOPADCHK. Now you can specify either module A or module B. If running with ENHANCED mode, module A must be MAIN or BASIC, or B must be BASIC. You still need to specify the other module unless it is defined as NOPADCHK.

BPX.DAEMON and BPX.MAINCHECK defined

If you enable enhanced program security, and you have any daemons or servers that run execute-controlled programs (MVS programs defined to RACF in the PROGRAM class using EXECUTE authority, or loaded from libraries using EXECUTE authority), then you must define the initial program executed by your daemon or server as a *trusted* program to RACF via the PROGRAM class by specifying APPLDATA(MAIN) for the profile. If this initial program resides in the z/OS UNIX file system, rather than in an MVS library, you will need to move it to an MVS library.

Additionally, you can choose whether to extend the enhanced program security protection to your UNIX daemons and servers that do not make use of RACF execute-controlled programs. You would enable this function by defining the profile BPX.MAINCHECK to RACF in the FACILITY class. Again, you would need to ensure that the initial program executed by your daemon or server resides in an MVS library and you would need to define it to class PROGRAM with APPLDATA(MAIN).

Setting up enhanced security mode

When using RACF as your security product and z/OS V1R4 is installed, do the following:

- Enable RACF enhanced program security selecting a mode:

  ```
  RDEFINE FACILITY IRR.PGMSECURITY APPLDATA('ENHWARN')
  ```

- Enable BPX.MAINCHECK:

  ```
  RDEFINE FACILITY BPX.MAINCHECK UACC(NONE)
  ```

- Determine which privileged HFS programs you run that are affected by setting up RACF enhanced program security. The RACF programs that would be affected are the main jobstep programs of one of the following types of privileged applications:

  – z/OS UNIX applications that require a program-controlled environment. This includes applications that require permission to BPX.DAEMON, BPX.SERVER or BPX.SRV.userid or those that use a privileged function like __passwd(). Examples of applications that would be affected by this are **`rlogin`**, **`telnet`**, and **`su`**.

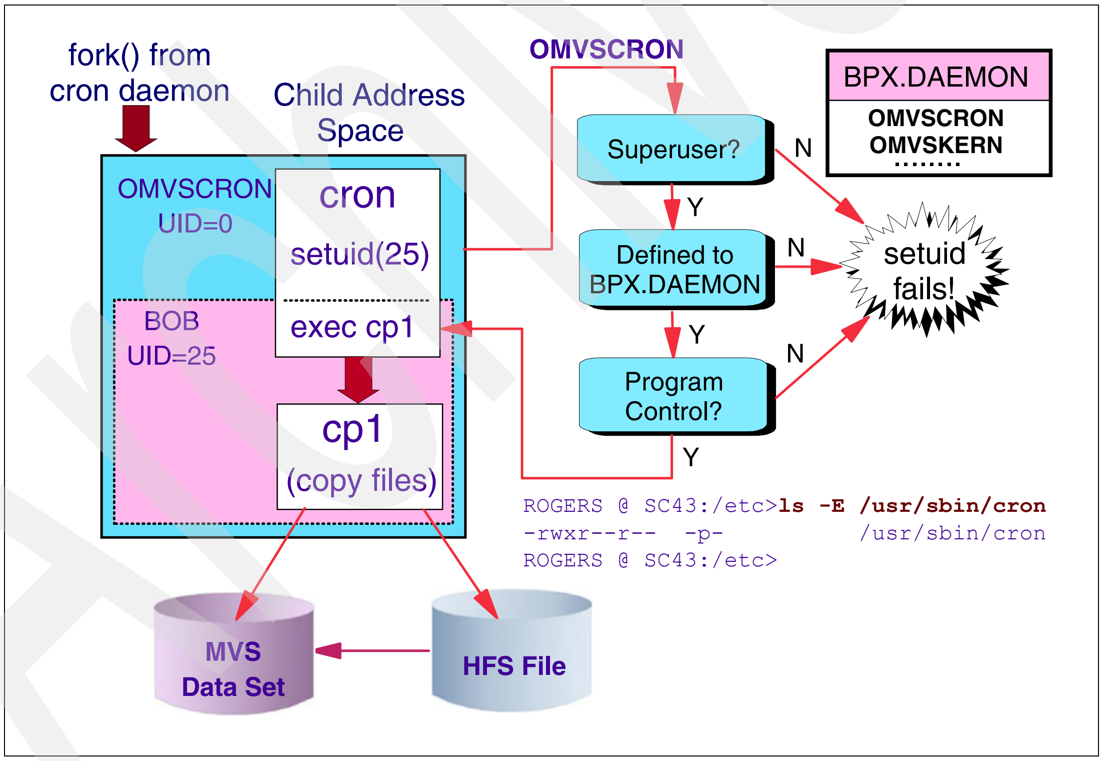
– Applications that gain access to MVS data sets by using RACF program access to data sets via entries in a DATASET class profile's conditional access list.

3.19.4 z/OS UNIX highest level of security example

Cron is a clock daemon that runs commands at specified dates and times. Figure 3-56 on page 161 shows the cron daemon running a shell script for user ID BOB (UID=25). The script will copy HFS files to an MVS data set. Before cron can run the script, it forks a new process and sets the identity of this process to UID=25 and the MVS identity to BOB. This ensures that the script can be run successfully with BOB's shell environment and BOB's access to his MVS data sets. When the job is done, the cron child process ends, and cron will not have any access to BOB's MVS data sets.

When setuid() SYSCALL is issued, the caller (daemon) program, and all other programs currently loaded in the address space, must have been loaded from a z/OS data set with the RACF Program Control activated—they must be controlled programs. Since it is the cloned child daemon program that issues the request, it inherits the contents of its address space from the parent daemon via fork.

This solution enables an installation to have some superusers that have authority to perform system maintenance, for example, to manage the hierarchical file system, while other special superusers (daemon user IDs) are allowed to change the identity of a process.



```
ROGERS @ SC43:/etc>ls -E /usr/sbin/cron
-rwxr--r--   -p-          /usr/sbin/cron
ROGERS @ SC43:/etc>
```

Figure 3-56 z/OS UNIX level of security

Note: The command to start the cron daemon is shown in Figure 3-56.

3.19.5 Defining daemon security

After understanding the security requirements with BPX.DAEMON and program control, the following steps describe how to define security for a daemon.

1. Define a user ID for the daemon that is a superuser with UID=0, for example OMVSCRON:

   ```
   ADDUSER OMVSCRON DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
   ```

2. Define the BPX.DAEMON FACILITY class in RACF. The name BPX.DAEMON must be used. No substitutions for this name are allowed. UACC(NONE) is recommended. If this is the first RACF FACILITY class defined in RACF, the SETROPTS command must be used to activate the class.

   ```
   RDEFINE FACILITY BPX.DAEMON UACC(NONE)
   ```

   Activate the class if this is the first RACF FACILITY class:

   ```
   SETROPTS CLASSACT(FACILITY) GENERIC(FACILITY) AUDIT(FACILITY)
   SETROPTS RACLIST(FACILITY) REFRESH
   ```

3. Permit the daemon user ID to the BPX.DAEMON class with access READ:

   ```
   PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSCRON) ACCESS(READ)
   ```

4. Protect the program libraries that need to be protected from unauthorized updates. The ADDSD command creates data set profiles for the data sets. You should protect against unauthorized updates so that nobody can replace a daemon program with a fake daemon program. If these profiles are already defined, this step can be skipped.

   ```
   ADDSD 'SYS1.LINKLIB' UACC(READ)
   ADDSD 'SYS1.SCEERUN' UACC(READ)
   ADDSD 'SYS1.SEZALOAD' UACC(READ)
   ADDSD 'SYS1.SEZATCP' UACC(READ)
   ```

   Mark the data sets as controlled libraries. An installation has a choice of either protecting all programs in a program library, or as individual programs. To protect all members in a data set, specify PROGRAM *.

   ```
   RDEFINE PROGRAM * ADDMEM('SYS1.LINKLIB'//NOPADCHK +
                            'SYS1.SCEERUN'//NOPADCHK +
                            'SYS1.SEZALOAD'//NOPADCHK +
                            'SYS1.SEZATCP'//NOPADCHK) UACC(READ)
   ```

   Or, mark the cron daemon program as controlled instead of the whole library:

   ```
   RDEFINE PROGRAM CRON ADDMEM('SYS1.LINKLIB'//NOPADCHK)
   UACC(READ) AUDIT(ALL)
   ```

5. Activate RACF program control.

   Place the PROGRAM profile in storage:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

3.20 File security packet extattr bits

The extended attribute bits in the FSP, as shown in Figure 3-57 on page 163, give special authorities to the files. Four extended attributes are defined:

APF-authorized programs	The behavior of these programs is the same as other programs that are loaded from APF-authorized libraries.
Program control	All programs that are loaded into an address space that requires daemon authority need to be marked as controlled.

Shared AS	The program shares its address space with other programs.
Shared library	Programs using shared libraries contain references to the library routines that are resolved by the loader at run time.

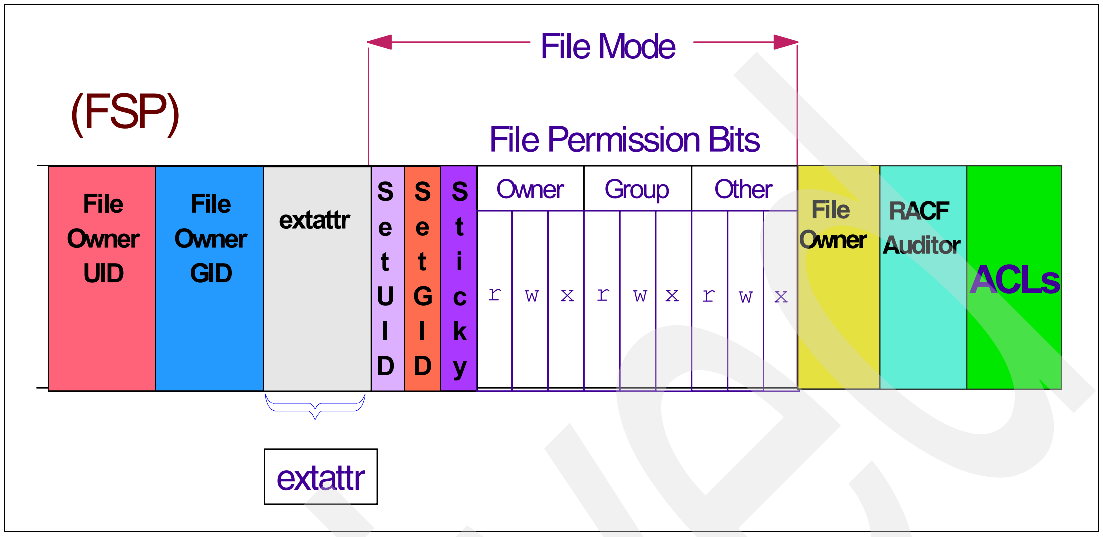


Figure 3-57 FSP showing the extattr bits

3.20.1 External attribute bits (extattr bits)

The **extattr** command is used to set, reset, and display extended attributes for files to allow executable files to be marked so they run APF-authorized, as a program-controlled executable, or not in a shared address space. The extattr bits in the FSP are mapped as follows:

a The program runs APF-authorized if linked AC=1. To turn on the APF-authorized bit:

```
extattr +a /user/sbin/progd
```

p The program is considered program-controlled. To turn on the program-controlled bit:

```
extattr +p /user/sbin/progc
```

s The program runs in a shared address space. To turn on the shared AS bit:

```
extattr +s /user/sbin/progb
```

l The program is loaded from the shared library region. To set the shared library attribute, issue the **extattr** command with the +l option.

```
extattr +l  /user/sbin/proga
```

To display or set the extended attribute bits, the following commands can be used:

```
ls -E
extattr
```

APF-authorized programs

The APF rules for programs that reside in the HFS are similar to those for programs that reside in MVS-authorized libraries. Setting the APF-authorized extended attribute bit should be thought of as putting that program into an authorized library. If you try to run a program from an authorized library that is not linked AC=1, it will not run APF-authorized, but that same program could be fetched by another that is running APF-authorized and executed in the authorization state in which it is called, or even have its state changed.

Although APF-authorization is not required for programs stored in the HFS to achieve program control, a program will run APF-authorized if the following requirements are met:

- The program must have been linked with the AC=1 attribute.
- The program must be loaded from an APF-authorized library.
- The program must be the initial program (that is, it must be the job step task program), or it was invoked by a caller that is running APF-authorized.

  If the specified program is going to be invoked as a job step program, you must linkedit it with AC=1. For example:

  ```
  c89 -Wl, AC=1
  ```

  In order to avoid possible integrity problems, do not set AC=1 if the program will be run in an APF-authorized environment but not as the job step program (such as DLL).

To find out whether the APF-authorized extended attribute of the HFS file has been set, use the **`ls -E`** command.

Setting APF-authorization

Defining BPX.FILEATTR.APF in the FACILITY class profile controls who can set APF authorization for HFS programs. Issue the commands:

```
RDEFINE FACILITY BPX.FILEATTR.APF OWNER(SECADM) UACC(NONE)
PERMIT BPX.FILEATTR.APF CL(FACILITY) ID(HARRY) ACCESS(READ)
```

Another profile, related to BPX.FILEATTR.APF, is BPX.DEBUG in the FACILITY class. Users with READ access to this profile can use **`ptrace`** (via dbx) to debug programs that run with APF authority or with BPX.SERVER authority.

```
RDEFINE FACILITY BPX.DEBUG OWNER(SECADM) UACC(NONE)
PERMIT BPX.DEBUG CLASS(FACILITY) ID(HARRY) ACCESS(READ)
```

Defining shared library programs

Although the shared library attribute is not required to achieve program control, a program is loaded as a system shared library program if the HFS program has the shared library extended attribute set.

To find out if the shared library extended attribute has been set, use the **`ls -E`** command.

Profile BPX.FILEATTR.SHARELIB in the FACILITY class controls who can set the shared library extended attribute. You need to have at least READ access before you can set the shared library extended attribute. Issue the commands:

```
RDEFINE FACILITY BPX.FILEATTR.SHARELIB OWNER(SECADM) UACC(NONE)
PERMIT BPX.FILEATTR.SHARELIB CL(FACILITY) ID(HARRY) ACCESS(READ)
```

Program control for HFS programs only

All programs loaded into an address space that requires daemon authority need to be marked as controlled. This means that user programs and any runtime library modules that are loaded must be marked as controlled by setting up profile BPX.FILEATTR.PROGCTL in the FACILITY class.

Profile BPX.DAEMON must be defined before issuing the commands:

```
RDEF FACILITY BPX.FILEATTR.PROGCTL OWNER(SECADM) UACC(NONE)
PERMIT BPX.FILEATTR.PROGCTL CLASS(FACILITY) ID(HARRY) ACCESS(READ)
```

After a file is marked program-controlled, any activity that can change its contents results in the extended attribute being turned off. If this occurs, a system programmer with the appropriate privilege will have to verify that the file is still correct and reissue the extattr command to mark the file as program-controlled.

All modules loaded from LPA are considered to be controlled. RTLS libraries must be defined to RACF for program-controlled support.

3.21 Using sanction lists

You can compile a list to contain the lists of path names and program names that are sanctioned by the installation for use by APF-authorized or program-controlled calling programs. This file contains properly constructed path names and program names as defined in *z/OS UNIX System Services User's Guide,* SA22-7801.

Sanction lists contain three separate lists delineated by three keywords:

:authprogram_path	This keyword is the start of a list of directories that is only used in the execution of an hfsload (or C dlload), exec, spawn, or attach_exec from an authorized program.
:programcontrol_path	This keyword is the start of a list of directories that is only used in the execution of an hfsload (or C dlload), exec, spawn, or attach_exec from an executable that is running program controlled.
afprogram_name	This keyword is the start of a list of program names that are allowed to get control of APF-authorized programs as a result of an exec or spawn. These names are MVS program names.

3.21.1 Creating a sanction list

Decide what directories and what programs are to be set into the sanction list file. You can partially construct this file and add path names and program names as you go along. A partially complete file can be activated and when additional entries are known, this file can be updated. A background task automatically checks this file every 15 minutes for updates and then incorporates them.

You also need to be aware that only one sanction list check is done for each program invocation. Although links in directories are supported, sanction list processing only performs one check. This check uses the path name or program name that was specified by the user.

BPXPRMxx PARMLIB member

Use the AUTHPGMLIST statement in BPXPRMxx to define the path name for the sanction list; to activate it use the SETOMVS command. Specify the path name as follows:

```
AUTHPGMLIST('/etc/authfile')
```

Activate the sanction list as follows:

```
SETOMVS AUTHPGMLIST='/etc/authfile'
```

```
/****************************************************************/
/* */
/* Name: Sample authorized program list */
/* */
/* Description: Contains lists of approved directories and */
/* program names from which privileged programs */
/* may be invoked */
/* */
/*****************************************************************/
/*****************************************************************/
/* Authorized program directories */
/*****************************************************************/
:authprogram_path
/bin/test
/bin/test/beta
/*****************************************************************/
/* Program control directories */
/*****************************************************************/
:programcontrol_path
/in/test/specials
/*****************************************************************/
/* APF authorized programs */
/*****************************************************************/
:apfprogram_name
PAYOUT
```

Figure 3-58 Sample authorized program list

You can turn off sanction list checking with the SETOMVS command:

```
SETOMVS AUTHPGMLIST=NONE
```

3.22 Security for servers

This chapter describes security for your server applications. It uses the word “server” to mean “server application,” which is an application that provides a service for clients. This server could be part of a software product that will run on any company’s z/OS computing environment, or it might be written by your application programmers for your own company’s use.

Appropriate decisions need to be made regarding server security. In the past, applications had to run as APF-authorized to be able to call RACF to build task-level security. z/OS UNIX provides services for servers written in C to create task-level security without being APF-authorized. This chapter describes how a server can create thread-level security environment and how to control which servers have the ability to do so. It also describes the procedures for preparing a z/OS system for a server that uses thread-level security for its clients. (Note that a thread on UNIX systems corresponds to a task on MVS; so, thread-level security is the same as task-level security.)

z/OS UNIX supports two fundamental types of application servers: multithreaded servers and single-threaded servers.

- A multithreaded server has multiple sequential flows of control. In this family of applications, the server can process more than one unit of work at a time.
- A single-threaded server has one sequential flow of control. In this family of applications, the server processes one unit of work at a time

3.22.1 The pthread_security_np() callable service

z/OS UNIX provides the pthread_security_np() callable service and support through the C run-time library. It enables *unauthorized* multithreaded servers to create and delete a RACF security environment in a way that is mediated and controlled by the kernel and RACF. Multithreaded servers can customize the security environment of a thread, thus allowing it to be executed under a different RACF identity than that of the server. You must authorize the server to use that service.

The term *unauthorized* refers to applications that are not APF-authorized and do not run in supervisor state or in a system storage protection key.

A server that uses the pthread_security_np() service can customize the RACF identity of a thread. Such server initiates a thread that processes the client's request. If the server customizes the thread initiated for the client with the client's RACF identity, any resource access decisions to RACF-protected resources are made using the client's RACF identity and authorizations.

Depending on the trust you place in a server, you have the option of enforcing whether to use both the server's RACF identity and the RACF identity of the client in resource access control decisions on z/OS.

You can choose one of the following:

- Only the RACF user ID of the client is used in local resource access control decisions made by RACF on z/OS.
- Both the RACF user ID of the server and the RACF user ID of the client are used in local resource access control decisions on z/OS.

The use of the pthread_security_np service is in part protected by profile BPX.SERVER in class FACILITY.

3.22.2 Establishing the correct level of security for servers

The choice of security level is a decision more likely made by management than by security administrators. That decision depends on answers to the questions "How secure does our company's information need to be?" and "How much do we trust our employees?" Regardless of who makes the decision, it is important that both application developers and security administrators understand the two levels of security supported by z/OS, and the differences between them. The two levels are: UNIX level and z/OS UNIX level.

Note: The discussion about the two levels of security (UNIX level and z/OS UNIX) for servers follows exactly the same line as the one regarding daemons in chapter 2.8 of this book. Daemons run software without client involvement and they are permitted to profile BPX.DAEMON, servers involve clients and are permitted to profile BPX.SERVER.The difference comes in the level of access to BPX.SERVER you may choose for your servers.

UNIX level: BPX.SERVER is not defined

If the BPX.SERVER FACILITY class is not defined, your system has UNIX-level security. In this case, the system is less secure. Server programs that run with superuser authority can issue **`pthread_security_np()`** function to change the MVS identity of a thread.

To establish UNIX-level security, assign a UID of 0 to your superusers and assign a UID of 0 to the user ID used for running server programs; for example, SERVSTU.

z/OS UNIX level: BPX.SERVER is defined

There are two z/OS UNIX levels:

- Profile BPX.SERVER defined in the FACILITY class.
- Profiles BPX.SERVER and BPX.MAINCHECK are defined (profiles in class PROGRAM providing *enhanced* program security). You can use enhanced program security for any privileged z/OS UNIX application that requires a program-controlled environment. An example is the __passwd() service, which is used by applications such as telnet and rlogin.

Issue the following command to establish the first level of z/OS UNIX security for servers:

```
RDEF FACILITY BPX.SERVER OWNER(SECADM) UACC(NONE)
```

3.22.3 Two levels of z/OS UNIX security for servers

We now examine the two levels of z/OS UNIX security:

- BPX.SERVER defined
- BPX.SERVER and BPX.MAINCHECK defined

BPX.SERVER defined

If BPX.SERVER FACILITY class is defined, your system has z/OS UNIX-level security. In this case, the system is more secure than a traditional UNIX system. If this profile is defined, then the RACF user ID that is associated with the server needs at least READ authority to use the **`pthread_security_np()`** service.

This profile is also used to restrict the use of the BPX1ACK service, which determines access authority to z/OS resources.

Servers with authority to BPX.SERVER must run in a clean program-controlled environment. z/OS UNIX verifies that the address space has not loaded any executables that are uncontrolled before it allows any of the following services that are controlled by z/OS UNIX to succeed:

- seteuid
- setuid
- setreuid
- pthread_security_np()
- auth_check_resource_np()
- _login()
- _spawn() with userid change
- _password()

You can also use the BPX.SERVER profile to set the scope of z/OS resources that the server can access when acting as a surrogate for its clients. There are two levels of authority that can be granted to the server using thread-level security services:

UPDATE access Lets the server establish a thread-level (task-level) security environment for clients connecting to the server. When the RACF identity of the server has been granted UPDATE authority to BPX.SERVER in the RACF FACILITY class, the server is capable of acting as a *surrogate* for the client. This means that the identity of the thread associated with the request from the server's client runs with the z/OS user ID of the server's client. Access control decisions to z/OS resources (such as data sets) and to z/OS UNIX resources (such as HFS files) that are accessed by the client's thread in the server, are made using the RACF identity of the client.

READ access The user ID of the server and the user ID of the client must be authorized to the resources which the server will be accessing. A thread-level security context in which both the client's and server's identity is used in the access control decision and a password was not supplied by the client is called an *unauthenticated client* security context.

Depending on the design and implementation of the client/server application, a client may have to supply an authenticator to the server. For example, the client may be prompted to supply a password or a password substitute, such as a RACF PassTicket to the server to prove its identity. If a RACF password or PassTicket is specified as a parameter on the pthread_security_np() service, and the *password or PassTicket is valid* for the client user ID, even if the server's user ID has been granted READ access to the profile BPX.SERVER in the RACF FACILITY class, the task level security environment is only used in access control decisions. That is, only the RACF user ID of the client is used in making access control decisions. This task level security environment created by a server is called an *authenticated client* security context. Since the client has trusted the server sufficiently to supply a RACF password (or PassTicket) to the server, the server is granted the capability of acting as a *surrogate* for that client (user).

This capability enables you to determine:

– On behalf of which user IDs the server can act
– What resources the server can access when acting on behalf of one of its clients

Potentially, for additional security checking, two audit records can be produced to audit:

- The client accessing the resource
- The server accessing the resource on behalf of the client

If you choose to implement this additional security checking, you might need to authorize the server's user ID to the resource profiles that protect the resources accessed by the server on behalf of its clients.

BPX.SERVER and BPX.MAINCHECK defined

The steps to establish the second level of z/OS UNIX security for servers are exactly the same as for daemons. Refer to 3.19, "z/OS UNIX level security for daemons" on page 154 for how to set up enhanced program security.

3.23 Checking authority to use protected resources

Application developers might want a server to check the authority of a user to access profiles defined to RACF general resource classes. The resources include printers and tapes, but not HFS files and directories and MVS data sets. Through z/OS UNIX, the auth_check_resource_np (BPX1ACK) callable service enables application servers to invoke RACF authorization services. This service is also supported by the C run-time library through the __check_resource_auth_np() function call.

The server must have read access to the BPX.SERVER FACILITY class profile or have UID(0); in addition, all server modules must be defined to RACF.

For more information on the auth_check_resource_np callable service, see *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803.

3.23.1 Limitations of RACF client ACEE support

If both the server's RACF identity and the client's RACF identity are used to make access decisions, you should be aware of limitations of the RACF client ACEE support.

- RACROUTE REQUEST=FASTAUTH processing does not check both the server and client RACF identities automatically. Unauthorized servers cannot use the RACROUTE REQUEST=LIST instruction to build in-storage profiles for RACF defined resources. Profiles must reside in storage before RACROUTE REQUEST=FASTAUTH can verify a user's access to a resource.
- The client/server relationship is not propagated from the server.

If your server controls access to resources by checking and authenticating both the server's RACF identity and client's RACF identity, treat servers you do not trust as *end points* on z/OS. These servers should not be allowed to submit batch jobs or use the services of other servers that run exclusively under the identity of the client. You must ensure that servers that do not meet this criteria are not authorized to the profile BPX.SERVER in the RACF FACILITY class.

3.23.2 Defining servers to use thread-level security

This section shows how to set up servers. The following steps are for a sample server called SERVER1. As you add servers, you will need to follow similar procedures.

1. All programs that are loaded into an address space requiring server authority (including the server program and any run-time library modules) need to be marked as controlled.

   To make all programs controlled see "Program control programs" on page 157.

2. Assign a user ID to the server:

   ```
   AU SERVSTU DFLTGRP(STG) OWNER(STG) OMVS(AUTOUID HOME('/') PROG('/bin/sh')) +
   NOPASSWORD
   ```

3. Create a cataloged procedure:

   ```
   //SERVER1 PROC
   //DATASRVR EXEC PGM=SERVER1,REGION=0M,TIME=NOLIMIT,
   // PARM=POSIX(ON) ALL31(ON)/ serverparms
   //SYSPRINT DD SYSOUT=*
   ```

4. In order for this SERVER1 cataloged procedure to obtain control with the desired user identity, add it to the RACF STARTED class:

   ```
   RDEF STARTED SEVER1.** OWNER(SECADM) STDATA(USER(SERVSTU) GROUP(STG))
   ```

5. The next decision that must be made is the level of authority to be granted to the server using thread-level security services. The BPX.SERVER FACILITY class profile controls the server's access to the **`pthread_security_np()`** service.

   There are two choices when setting the server's authority:

   – UPDATE access allows the server to establish a thread-level (task-level) security environment for clients connecting to the server. Decisions about access control for z/OS resources (such as data sets) and to z/OS UNIX resources (such as HFS files) that are accessed by the client's thread in the server are made using only the RACF identity of the client.

     To give UPDATE access in the BPX.SERVER profile in class FACILITY to user ID SERVSTU:

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(SERVSTU) ACCESS(UPDATE)
```

– READ access allows the server to establish a thread-level security environment for the clients that it services. However, unless the server has specified a valid RACF password or PassTicket on the pthread_security_np() service invocation, the user ID of the server and the user ID of the client are used in resource access control decisions. Following is the PERMIT command to give SERVSTU server authority for unauthenticated clients:

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(DATASRVR) ACCESS(READ)
```

6. If you are installing a product that uses thread-level security services, check the documentation that is supplied with the product to determine if the server requires READ or UPDATE access to the BPX.SERVER profile.

   If you grant READ access to the BPX.SERVER profile in the FACILITY class, and the server does not request a password or PassTicket for its clients, both the server's user ID and the client's user ID are used in decisions about resource access control. Additional security administration will have to be performed to ensure that both the server's user ID and the client's user ID were appropriately authorized to the resources that are accessed by the server.

7. To start SERVER1, issue the following command from the MVS console:

```
S SERVER1
```

3.23.3 Defining servers to process users without passwords

Depending on the design and implementation of a client/server application, a client may not supply an authenticator to the server. For example, some servers process user requests that come from generic user IDs representing anonymous users, or use a method of authentication other than a user ID and password combination.

In this case, in which the RACF password or password substitute (such as the RACF PassTicket) is not specified on the **pthread_security_np()** service invocation, an additional check is made to ensure that the server is authorized to act as the client. z/OS UNIX uses profiles defined to the RACF SURROGAT class to authorize the server to act as a surrogate of a client. Profiles defined to the SURROGAT class are of the form:

BPX.SRV.<userid>

where <userid> is the MVS user ID of the user that the server will act as a surrogate of.

Some servers have the requirement to process user requests that come from generic user IDs representing anonymous users. In order for servers to process requests for thread-level security without passwords, follow the steps shown below.

The following steps are for a sample server called SERVER1 (run by user ID SERVSTU) that can support user ID PUBLIC without a password. As you add more servers, you will need to follow similar procedures.

To create the SURROGAT class profile for user PUBLIC, issue:

```
RDEFINE SURROGAT BPX.SRV.PUBLIC UACC(NONE)
```

A similar SURROGAT profile is required for each user ID that a server must support without a password.

To permit server SERVER1 to create a thread-level security environment for user PUBLIC, issue the PERMIT command:

```
PERMIT BPX.SRV.PUBLIC CL(SURROGAT) ID(SERVSTU) ACCESS(READ)
```

3.24 Security for operations in z/OS UNIX

The SUPERUSER.FILESYS.VREGISTER resource only lets a server like NFS initialize. Users that are connected as clients through facilities such as NFS do not get special privileges based on this resource or other resources in the UNIXPRIV class.

Authorization to the BPX.DEBUG resource is also required to trace processes that run with APF authority or BPX.SERVER authority. For example, a user debugging a daemon would want to use the SUPERUSER.PROCESS.GETPSENT, SUPERUSER.PROCESS.KILL, and SUPERUSER.PROCESS.PTRACE privileges.

SUPERUSER.FILESYS.MOUNT

This profile allows a user to issue the TSO/E MOUNT command or the mount shell command with the nosetuid option. Also allows users to unmount a file system with the TSO/E UNMOUNT command or the **unmount** shell command mounted with the nosetuid option.

Users permitted to this profile can use the **chmount** shell command to change the mount attributes of a specified file system.

READD Allows a user to issue the TSO/E MOUNT command or the mount shell command with the setuid option. Also allows user to issue the TSO/E UNMOUNT command or the **unmount** shell command with the setuid option.

Users permitted to this profile can issue the **chmount** shell command on a file system that is mounted with the setuid option.

UPDATE Allows the user to issue the TSO/E MOUNT command or the mount shell command with the setuid option. Also allows user to issue the TSO/E UNMOUNT command or the **unmount** shell command with the setuid option.

Users permitted to this profile can issue the **chmount** shell command on a file system that is mounted with the setuid option.

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.MOUNT UACC(NONE)
PERMIT SUPERUSER.FILESYS.MOUNT CLASS(UNIXPRIV) ID(ROGERS) ACCESS(UPDATE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

SUPERUSER.FILESYS.QUIESCE

This profile allows a user to issue **quiesce** and **unquiesce** commands for a file system mounted with the nosetuid option.

READ Allows a user to issue **quiesce** and **unquiesce** commands for a file system mounted with the nosetuid option.

UPDATE Allows a user to issue **quiesce** and **unquiesce** commands for a file system mounted with the setuid option.

SUPERUSER.FILESYS.PFSCTL

Allows a user to use the pfsctl() callable service. Only READ access is required.

zFS with z/OS V1R3 supports the SUPERUSER.FILESYS.PFSCTL profile of the UNIXPRIV class. This makes it possible for a zFS administrator to have just READ authority to this UNIXPRIV profile resource, SUPERUSER.FILESYS.PFSCTL, rather than requiring a UID of 0 for **zfsadm** commands that modify zFS file systems or aggregates. The same is true for the other zFS commands and utilities. So zFS administrators do not need a UID of 0.

In order to allow the zFS administrator to mount and unmount file systems, permit update access to another profile, SUPERUSER.FILESYS.MOUNT, in class UNIXPRIV.

Note: UPDATE access is needed if the user needs to **`mount`**, **`chmount`**, or **`unmount`** file systems with the setuid option; otherwise, READ access is sufficient.

UNIXPRIV authorization is invoked by creating the needed resources in the UNIXPRIV class and then giving users READ authority to it, as follows:

```
SETROPTS CLASSACT(UNIXPRIV)
SETROPTS RACLIST(UNIXPRIV)
RDEFINE UNIXPRIV SUPERUSER.FILESYS.PFSCTL UACC(NONE)
PERMIT SUPERUSER.FILESYS.PFSCTL CLASS(UNIXPRIV) ID(ROGERS) ACCESS(READ)
```

SUPERUSER.FILESYS.VREGISTER

This profile allows a server to use the **`vreg()`** callable service to register as a VFS file server. Only READ access is required.

SUPERUSER.IPC.RMID

This profile allows a user to issue the **`ipcrm`** command to release IPC resources. Only READ access is required.

SUPERUSER.PROCESS.GETPSENT

This profile allows a user to use the **`w_getpsent()`** callable service to receive data for any process. Only READ access is required.

SUPERUSER.PROCESS.KILL

This profile allows a user to use the **`kill()`** callable service to send signals to any process. Only READ access is required.

SUPERUSER.PROCESS.PTRACE

This profile allows a user to use the **`ptrace()`** function through the dbx debugger to trace any process.

It also allows users of the **`ps`** command to output information on all processes. This is the default behavior of **`ps`** on most UNIX platforms. Only READ access is required.

SUPERUSER.SETPRIORITY

This profile allows a user to increase his own priority. Only READ access is required.

3.24.1 BPX.SAFFASTPATH

Enables faster security checks for file system and IPC constructs.

When the BPX.SAFFASTPATH FACILITY class profile is defined, the security product is not called if z/OS UNIX can quickly determine that file access will be successful. When the security product is bypassed, better performance is achieved, but successful file accesses cannot be audited. If the security product is called, it is still possible that access will be successful, and that audit records will be created; for example, when the permission bits do not grant access, but UNIXPRIV authority, or an access control list, does. Be aware that auditing successful accesses can generate enormous amounts of audit records, particularly for directory searches. Use this profile as follows:

- If the BPX.SAFFASTPATH FACILITY class profile is defined when the system is IPLed, the SAF fastpath support is enabled.

- If it is defined after the system is IPLed, you must issue the SETOMVS or SET OMVS operator command to activate the fastpath support.

You can also start the refresh by issuing the following command, where xx represents a BPXPRMxx member that is empty:

```
SET OMVS=xx
```

Note: Users do not need to be permitted to the BPX.SAFFASTPATH profile.

To define the BPX.SAFFASTPATH profile, issue the following RACF command:

```
RDEFINE FACILITY BPX.SAFFASTPATH UACC(NONE)
```

Tip: If your installation uses the IRRSXT00 exit to control HFS access, do not define the BPX.SAFFASTPATH profile.

3.24.2 BPX.JOBNAME

To control which users are allowed to set their own job names, use the _BPX_JOBNAME environment variable or the inheritance structure on spawn.

Users with READ or higher permissions to this profile can define their own job names.

3.24.3 BPX.STOR.SWAP

This profile in the FACILITY class controls which users can make address spaces nonswappable. Users permitted with at least READ access to BPX.STOR.SWAP can invoke the __mlockall() function to make their address space either nonswappable or swappable.

When an application makes an address space nonswappable, it may cause additional real storage in the system to be converted to preferred storage. Because preferred storage cannot be configured offline, using this service can reduce the installation's ability to reconfigure storage in the future. Any application using this service should warn the customer about this side effect in their installation documentation.

3.24.4 BPX.WLMSERVER

This profile in the FACILITY class controls access to the WLM server functions _server_init() and _server_pwu(). It also controls access to these C language WLM interfaces:

- QuerySchEnv()
- CheckSchEnv()
- DisconnectServer()
- DeleteWorkUnit()
- JoinWorkUnit()
- LeaveWorkUnit()
- ConnectWorkMgr()
- CreateWorkUnit()
- ContinueWorkUnit()

A server application with READ permission to this FACILITY class profile can use the server functions, as well as the WLM C language functions, to create and manage work requests.

3.24.5 Security for ServerPac and CBPDO install

Security requirements for ServerPac and CBPDO installation are necessary before you can do the ServerPac or CBPDO installation or install maintenance, as follows:

- The user ID must be UID=0 or permitted to the BPX.SUPERUSER resource in the RACF FACILITY class, and be connected to a group that has a GID.
- Define the following user ID and group IDs in your security data base. Even though they are lowercase in the example, these names should be defined in uppercase for ease of use and manageability.
  - Group IDs are as follows:
    - uucpg
    - TTY
  - User IDs are as follows:
    - uucp

Rules for the user ID and group IDs

The GID and UID values assigned to these IDs cannot be used by any other IDs. They must be unique. If you assign the same GID to multiple groups, control at an individual group level is lost, because the GID is used in z/OS UNIX security checks. Because RACF groups that have the same GID assignment are treated as a single group during the z/OS UNIX security checks, the sharing of resources between groups might happen unintentionally. Likewise, the sharing of UIDs allows each user access to all of the resources associated with the other users of that shared UID. The shared access includes not only z/OS UNIX resources such as files, but also includes the possibility that one user could access z/OS UNIX resources of the other user that are normally considered to be outside the scope of z/OS UNIX.

You must duplicate the required user ID and group names in each security database, including the same UID and GID values in the OMVS segment. This makes it easier to transport the HFS data sets from test systems to production systems. For example, the group name TTY on System 1 must have the same GID value on System 2 and System 3. If it is not possible to synchronize your databases you will need to continue running the FOMISCHO job against each system after z/OS UNIX is installed.

RACF definitions

The following describes how to define these IDs to RACF. (If you are using an equivalent security product, refer to that product's documentation.) All the RACF commands are issued by a TSO/E user ID with RACF SPECIAL authority. Three procedures are described:

- If you use uppercase group and user IDs
- If you use mixed-case group and user IDs
- If you have problems with names such as UUCP, UUCPG, and TTY

If you use uppercase group and user IDs

If you use only uppercase group and user IDs on your system, RACF users can use the BPX1SEC1 sample in SAMPLIB or the following commands to define the group IDs and user IDs:

- To define the TTY group:

  ```
  ADDGROUP TTY (OMVS(GID(2))
  ```

  Where 2 is an example of a unique group ID on your system. Do not connect users to this group. This is the same group that is specified on the TTYGROUP statement in the BPXPRMxx PARMLIB member on your target system.

Certain shell commands, such as **`mesg`**, **`talk`**, and **`write`** require pseudoterminals to have a group name of TTY. When a user logs in, or issues the OMVS command from TSO/E, the group name associated with these terminals is changed to TTY. As part of installation, you had to define the group TTY or use the group alias support.

Note: Give this group a unique GID and do not connect users to this group.

Recommendation: To make it easier to transport the data sets from test systems to production systems, be sure that this entry is duplicated in all of your security data bases, including the same UID and GID values in the OMVS segment

- To define the UUCPG group:

  ```
  ADDGROUP UUCPG OMVS(GID(8765))
  ```

  Where `8765` is an example of a unique group ID on your system.
- To define the UUCP user ID, issue:

  ```
  ADDUSER UUCP DFLTGRP(UUCPG) PASSWORD(xxxxxxx)
  OMVS(UID(396) HOME('/usr/spool/uucppublic') PROGRAM('/bin/sh'))
  ```

  Where `123456` is an example of a unique account number and `396` is an example of a unique OMVS UID. Do not use UID(0).

If you use mixed-case group and user IDs

If you already use mixed-case group and user IDs on your system and the user (uucp) and group (uupcg) do not conflict with existing names, perform the steps for uppercase IDs as in "If you use uppercase group and user IDs" on page 175.

It is not necessary to add the lowercase or mixed-case names to your alias table, mapping them to uppercase. Using the alias table degrades performance and increases systems management and complexity. When lowercase or mixed-case names are not found in the alias table, or there is no table active, they are folded to uppercase.

If you have problems with names such as uucp, uucpg, and TTY

If names such as uucp, uucpg, and TTY are not allowed on your system (or if they conflict with existing names), these are the RACF commands to define the group ID and user IDs:

- To define a group ID instead of the TTY group, issue:

  ```
  ADDGROUP XXTTY OMVS(GID)2))
  ```

  Where `2` is an example of a unique group ID on your system, and `XXTTY` is replaced by a 1- to 8-character group ID of your choice. Do not connect users to this group. This would be the same group name to be specified in the TTYGROUP statement in the BPXPRMxx PARMLIB member on your target system.
- To define a group ID instead of the uucpg group, issue:

  ```
  ADDGROUP xxuucpg OMVS(GID(8765))
  ```

  Where `8765` is an example of a unique group ID on your system, and `xxuucpg` is replaced by a 1- to 8-character group ID of your choice.
- To define a uucp user ID, issue:

  ```
  ADDUSER xxuucp DFLTGRP(UUCPG) PASSWORD(xxxxxxx)
  OMVS(UID(396) HOME('/usr/spool/uucppublic') PROGRAM('/bin/sh'))
  ```

  Where `396` is an example of a unique UID (do not use a UID of 0) and `xxuucp` is replaced by a user ID of your choice. This is a normal user ID that owns all the UUCP files and directories. Use this user ID when editing configuration files or performing other administrative tasks.

- Set up a user ID alias table.

  Using the alias table causes poorer performance and increases systems management costs and complexity.

  If you do not have a user ID alias table defined, you need to create one. This must be done first on your driving system and then on any system image using this product. This fits in with the IBM strategy to place all customized data in the /etc directory. This table is specified by the USERIDALIASTABLE keyword in the BPXPRMxx PARMLIB member. Because the user ID name alias table must be protected from update by nonprivileged users, only users with superuser authority should be given update access to it. All users should have read access to the file.

  Your userid alias table will need to contain your MVS chosen names and the associated required names. Your chosen MVS user ID and group names must be located in columns 1-8 and the associated aliases must be located on the same line in columns 10-17.

- Activate the userid alias table. If you are already using the userid alias table, new database queries will yield the new alias if the user ID performing the query has read/execute access to the userid/group name alias table. The table is checked every 15 minutes and refreshed if it has been changed. If a change needs to be activated sooner, you can use the SETOMVS or SET OMVS operator commands. If you are not using the userid alias table, you can use the SET OMVS operator command to activate it now. For example:

  ```
  SET OMVS USERIDALIASTABLE=/etc/tablename
  ```

  Where /etc/tablename is the name of your userid alias table. You can also use the SETOMVS operator command. See z/OS MVS System Commands, SA22-7627 for a complete description of the SET OMVS and SETOMVS commands.

- Update your BPXPRMxx PARMLIB member, specifying the USERIDALIASTABLE to make this change permanent for your next IPL.
- Perform these tasks on all of your driving, test, and production system images.

3.25 Auditing for z/OS UNIX

The auditing for z/OS UNIX System Services comprises the following steps: creating an audit policy (which files/directories to audit, level of access for violations and successes, format, and frequency of reports), setting up audit controls, collecting SMF records, producing reports, follow-up, and corrective actions; see Figure 3-59 on page 178.

RACF provides utilities for unloading SMF data (IRRADU00) and data from the RACF database (IRRDBU00) that can be used as input in audit reports.

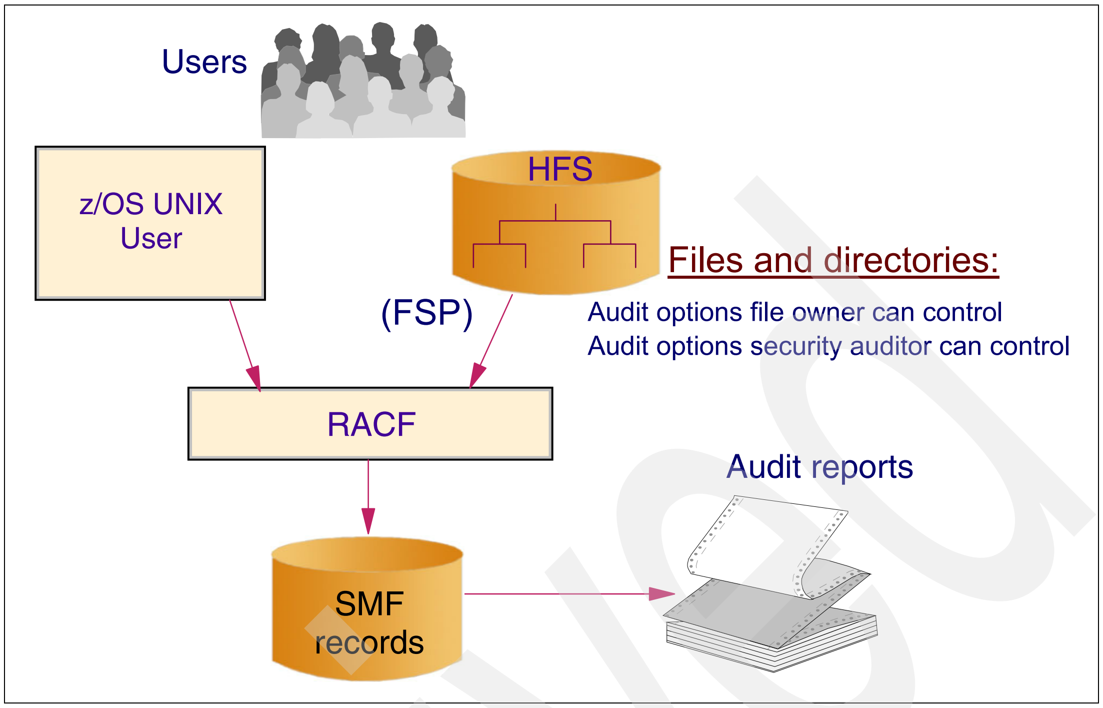

Users
z/OS UNIX User
HFS
Files and directories:
Audit options file owner can control
Audit options security auditor can control
(FSP)
RACF
Audit reports
SMF records

Figure 3-59 Auditing options for z/OS UNIX

Every file and directory has security information in the FSP as indicated in Figure 3-60 on page 179, which consists of:

- File access permissions
- UID and GID of the file
- Audit options that the file owner can control
- Audit options that the security auditor can control

The security auditor uses reports formatted from RACF system management facilities (SMF) records to check successful and failing accesses to z/OS UNIX resources. An SMF record can be written at each point where the system makes security decisions.

Six classes are used to control auditing of z/OS UNIX security events. These classes have no profiles. They do not have to be active to control auditing.

The security administrator or the file owners can also specify auditing at the file level in the file system.

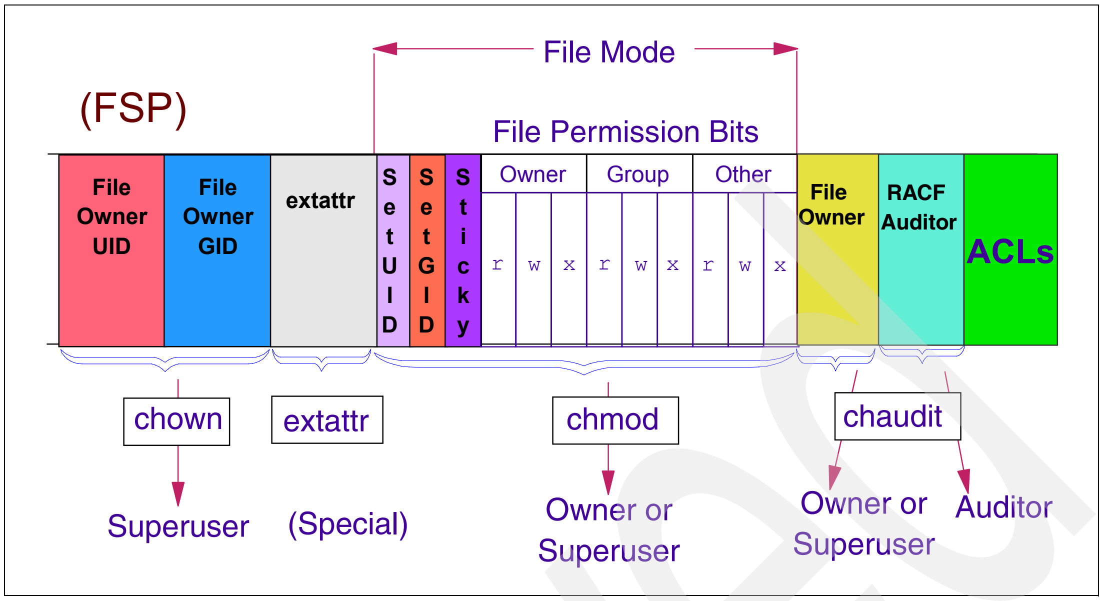


Figure 3-60 Auditing fields in the FSP

3.25.1 Setting up audit controls

Auditing for file access is specified in the file security packet (FSP) with the **chaudit** command. Only a file owner or a security auditor can specify if auditing is turned on or off, and when audit records should be written for a directory or file. There are two separate sets of auditing flags:

- Auditing set by the file owner (and superuser)
- Auditing set by the RACF AUDITOR

Audit records are written based on the combined owner and auditor settings. Auditing is set for read, write, and execute (search for directories) for the following kinds of accesses:

- Successful accesses
- Failures, that is, access violations
- All, which is both successes and failures
- None

When a file or a directory is created, default audit options are assigned. Different defaults are set for users and auditors. The same audit option is used no matter what kind of access is attempted (read, write, or execute). If auditing is not specified for a file, the defaults are:

- For owner auditing—audit failed accesses
- For RACF AUDITOR auditing—no auditing
- The user-requested-audit flags are set to audit failed attempts to read, write, or execute. Only the file owner or a superuser can specify user audit options.
- The auditor-requested-audit flags are set off (no auditing). To specify auditor audit options, one must have security auditor authority.

When a file is created, these are the default audit options:

- User audit options: for all access types, audit_access_failed
- Auditor audit options: for all access types, don't_audit

3.25.2 Auditing access to files and directories

The security auditor uses reports formatted from RACF system management facilities (SMF) records to check successful and failing accesses to kernel resources. An SMF record can be written at each point where the system makes security decisions.

Seven classes are used to control auditing of security events. These classes have no profiles. They do not have to be active to control auditing. Use the SETROPTS command to specify the auditing options for the classes. For a list of the classes used for auditing and an explanation of how to specify the audit options, see z/OS Security Server RACF Auditor's Guide.

You can also specify auditing at the file level in the file system. Activate this option by:

- Specifying DEFAULT in the class LOGOPTIONS on the SETROPTS command.

  The auditing levels for LOGOPTIONS are:

ALWAYS	All access attempts to resources protected by the class are audited.
NEVER	No access attempts to resources protected by the class are audited (all auditing is suppressed).
SUCCESSES	All successful access attempts to resources protected by the class are audited.
FAILURES	All failed access attempts to resources protected by the class are audited.
DEFAULT	Auditing is controlled by the auditing bits in the FSP for z/OS UNIX files and directories.

- Using the **`chaudit`** command to specify audit options for individual files and directories. If you activate auditing for additional levels of file system access, you may generate excessive amounts of SMF Type 80 records.

You can also specify, in a RACF user profile, that all actions taken by the user be audited. Actions taken by superusers can be audited or not, determined by RACF commands. If you are using RACF profiles in the UNIXPRIV class to control certain superuser functions, you can use those same profiles to audit those superuser functions.

Auditing can be controlled by using the commands SETROPTS LOGOPTIONS and SETROPTS AUDIT, as follows:

- SETROPTS LOGOPTIONS(auditing_level(class_name)) audits access attempts to the resources in the specified class according to auditing level specified and can be used for all classes.
- SETROPTS AUDIT(class_name) specifies the names of the classes that RACF should audit. The AUDIT option can be used for the classes FSOBJ, IPCOBJ, and PROCESS.
- SETROPTS LOGOPTIONS(DEFAULT) indicates you want no class auditing and only file level auditing (use **`chaudit`** to specify).

Audit records are always written when:

- A user who is not defined as a z/OS UNIX user tries to dub a process.
- A user who is not a superuser tries to mount or unmount a file system.
- A user tries to change a home directory.
- A user tries to remove a file, hard link, or directory.
- A user tries to rename a file, hard link, symbolic link, or directory.
- A user creates a hard link.

Note: There is no option to turn off these audit records.

3.25.3 Specifying file audit options

For violations occurring in the UNIX System Services environment, the user's effective UID and effective GID are displayed in the message. These IDs were used to determine the user's privilege for the intended operation. Note that they may not always match the IDs defined in the relevant RACF USER and GROUP profiles, since UNIX System Services provides methods by which another identity can be assumed.

Specify file audit options using the ISPF shell, or a **`chaudit`** command. The command can be used to specify either *user* audit options or *auditor* audit options.

To specify user audit options, you must be a superuser or the owner of the file. To specify auditor audit options, you must have RACF AUDITOR authority.

If you have AUDITOR authority, you do not need access in the permission bits to:

- Search and read any directory in the file system
- Use the **`chaudit`** command to change the auditor audit options for any file in the file system

If both user and auditor audit options are set, RACF merges the options and audits all the set options.

Classes that control auditing for z/OS UNIX System Services

RACF writes audit records for the z/OS UNIX System Services auditable events in SMF type 80 records. The following classes are defined to control auditing:

DIRSRCH	Controls auditing of directory searches.
DIRACC	Controls auditing of access checks for read/write access to directories.
FSOBJ	Controls auditing of all access checks for file system objects except directory searches via SETROPTS LOGOPTIONS and controls auditing of creation and deletion of file system objects via SETROPTS AUDIT.
FSSEC	Controls auditing of changes to the security data (FSP) for file system objects.
PROCESS	Controls auditing of changes to the UIDs and GIDs of processes and changing of the thread limit via the SETROPTS LOGOPTIONS, and controls auditing of dubbing, undubbing, and server registration of processes via SETROPTS AUDIT.
PROCAT	Controls auditing of functions that look at data from or affect other processes.
IPCOBJ	Specifies auditing options for IPC accesses and access checks for objects and changes to UIDs, GIDs, and modes. For access control and for z/OS UNIX user identifier (UID), z/OS UNIX group identifier (GID), and mode changes, use SETROPTS LOGOPTIONS. For object create and delete, use SETROPTS AUDIT.

The classes are in the class descriptor table (ICHRRCDX). No profiles can be defined in these classes. They are for audit purposes only. These classes do not need to be active to be used to control z/OS UNIX System Services auditing.

Activating the classes has no effect on auditing or authorization checking, except for the FSSEC class, which enables the use of ACLs in authorization checking. You can use profiles in the UNIXPRIV class to audit certain superuser functions.

Each of the classes controls auditing for z/OS UNIX System Services in a particular way. The descriptions that follow define the type of auditing each class controls and include:

- The audit event types that it controls
- The RACF callable services that write the audit record
- The z/OS UNIX services that can cause the event

Auditable events

RACF writes audit records for the z/OS UNIX System Services auditable events in SMF type 80 records. File owners and auditors can establish separate sets of auditing rules, and can also specify auditing for each file and directory. For more information on these event codes, see *z/OS Security Server RACF Macros and Interfaces,* SA22-7682.

3.25.4 Commands to activate auditing

You can control auditing by using the existing SETROPTS LOGOPTIONS and SETROPTS AUDIT to activate the classes shown in "Classes that control auditing for z/OS UNIX System Services" on page 181.

Here is an example of controlling the RACF classes DIRSRCH and DIRACC:

```
SETROPTS LOGOPTIONS(FAILURES(DIRSRCH,DIRACC))
```

In addition, you can use the SETROPTS AUDIT option to control auditing for the FSOBJ, IPCOBJ, and the PROCESS classes, as follows:

```
SETROPTS AUDIT(FSOBJ,PROCESS)
```

3.25.5 Using the chaudit command

The **chaudit** command changes the audit attributes of the specified files or directories. Audit attributes determine whether or not accesses to a file are audited by the system authorization facility (SAF) interface.

Note: **chaudit** can be used only by the file owner or a superuser for non-auditor-requested audit attributes. It takes a user with auditor authority to change the auditor-requested audit attributes.

Figure 3-61 on page 183 shows examples of **chaudit** command usage by the file owner (or superuser). The default audit settings are shown in the upper right-hand corner of the figure, as follows:

- The command **chaudit w+s prog1** adds (+) auditing for successful accesses (s) for write accesses (w).
- The next command, **chaudit rwx=sf prog1** specifies that all (a) accesses, that is both successes (s) and failures (f), are to be audited for reads, writes, and executes.
- The next command **chaudit r-s,x-sf prog1** says to stop (-) auditing successes (s) for read (r), and stop (-) auditing both successes (s) and failures (f) for execute (x) access. The same effect can be achieved with the command **chaudit r=f,x= prog1**.

Examples of **chaudit** command usage by the RACF Auditor. The default audit settings are shown in the middle of this visual, as follows:

- The command **chaudit -a r+f,w+s,x+f prog1** adds auditing of successes (s) and failures (f) for write access, and specifies to write an audit record whenever an access failure (f) occurs for read or execute accesses.
- The command **chaudit -a r-f,x-f prog1** turns off (-) auditing for failures (f) for read and execute accesses.
- The last command **chaudit -a rwx=f prog1** turns on auditing for unsuccessful (f) read, write, and execute accesses.

The auditor includes the option **-a** when issuing the **chaudit** command. The auditor can only set the audit flags in the auditor's section of the FSP.

The audit condition part of a symbolic mode is any combination of the following:

s	Audit on successful access if the audit attribute is on
f	Audit on failed access if the audit attribute is on

The following command changes the file prog1 so that all successful and unsuccessful file accesses are audited:

```
chaudit rwx=sf prog1
```

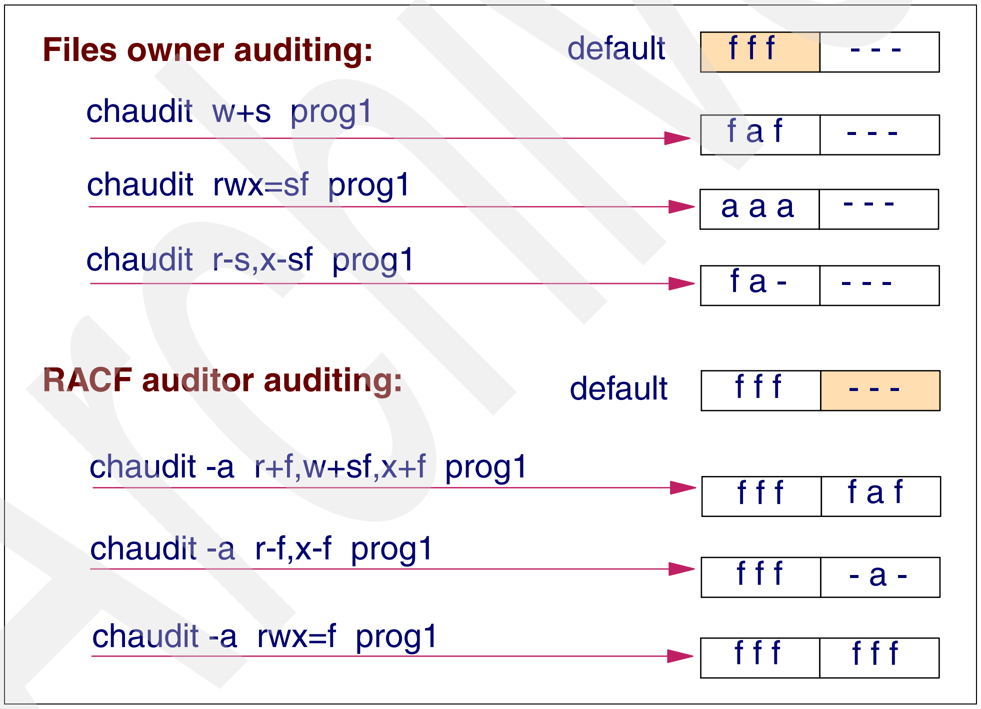


Figure 3-61 Using the chaudit command to change auditing for files and directories

3.25.6 Auditing for superuser authority in the UNIXPRIV class

If you use profiles in the UNIXPRIV class to control superuser authorities, you can use the same profiles for auditing.

RACF logs successful attempts to use superuser authorities. If you want to check the use of superuser authority for specific resources, you can audit successful uses of the UNIXPRIV profiles. RACF logs failed attempts to use SHARED.IDS in the UNIXPRIV class. For other

UNIXPRIV resources, no audit record is written to show authorization failures in the UNIXPRIV class.

For example, to audit the successful uses of the **kill()** function, granted by the SUPERUSER.PROCESS.KILL profile, set the audit options as follows:

```
RALTER UNIXPRIV SUPERUSER.PROCESS.KILL AUDIT(SUCCESS(READ))
```

LOG=NOFAIL is specified on all authorization checks in the UNIXPRIV class, except for SHARED.IDS. Therefore, RACF does not log failures, even when you specify AUDIT(FAILURES) or AUDIT(ALL) in the profile. RACF also ignores any SETROPTS LOGOPTIONS settings in the UNIXPRIV class because the RACROUTE REQUEST=FASTAUTH request performs all authorization checks in that class.

It is possible to see multiple audit records for the same operation, as described in the following example:

1. You are auditing successful uses of the SUPERUSER.PROCESS.KILL profile.
2. You also issued the SETROPTS LOGOPTIONS(SUCCESSES(PROCACT)) command to audit success in the PROCACT class.

   Note: This is not recommended because of the large number of audit records it could produce.

3. User Al has UID 40 and READ access to the SUPERUSER.PROCESS.KILL profile in the UNIXPRIV class.
4. User Al issued the **kill()** function for another user's process.

The **kill()** function succeeds and RACF writes two audit records as a result of:

- Auditing for the PROCACT class
- A RACROUTE REQUEST=FASTAUTH call in the UNIXPRIV class

4

Overview and customization of TCP/IP for z/OS UNIX

The Internet grew from fewer than 6000 at the end of 1986 to more than 15 million networks today. Networks have grown so quickly because they provide an important service. It is the nature of computers to generate and process information, but this information is useless unless it can be shared with the people who need it. The common thread that ties the enormous Internet together is TCP/IP network software. TCP/IP is a set of communication protocols that define how different types of computers talk to each other.

This unit explains the basic concepts of TCP/IP and offers information on how to customize TCP/IP for z/OS UNIX.

The following topics are covered:

- Overview of TCP/IP
- Customizing and starting TCP/IP
- Customizing for inetd and rlogind daemons
- Syslogd daemon
- Otelnetd daemon
- REXECD and RSHD servers
- SMTP server
- FTPD daemon
- Customizing and starting NFS

4.1 Overview of TCP/IP

The TCP/IP protocol suite is named for two of its most important protocols: Transmission Control Protocol (TCP) and Internet Protocol (IP). Another name for it is the Internet Protocol Suite, and this is the phrase used in official Internet standards documents. The more common term TCP/IP is used to refer to the entire protocol suite.

The first design goal of TCP/IP was to build an interconnection of networks that provided universal communication services: an internetwork, or Internet. Each physical network has its own technology-dependent communication interface, in the form of a programming interface that provides basic communication functions (primitives). Communication services are provided by software that runs between the physical network and the user applications and that provides a common interface for these applications, independent of the underlying physical network. The architecture of the physical networks is hidden from the user.

The second aim is to interconnect different physical networks to form what appears to the user to be one large network. Such a set of interconnected networks is called an internetwork or an Internet.

To be able to interconnect two networks, we need a computer that is attached to both networks and that can forward packets from one network to the other; such a machine is called a router. The term IP router is also used because the routing function is part of the IP layer of the TCP/IP protocol suite.

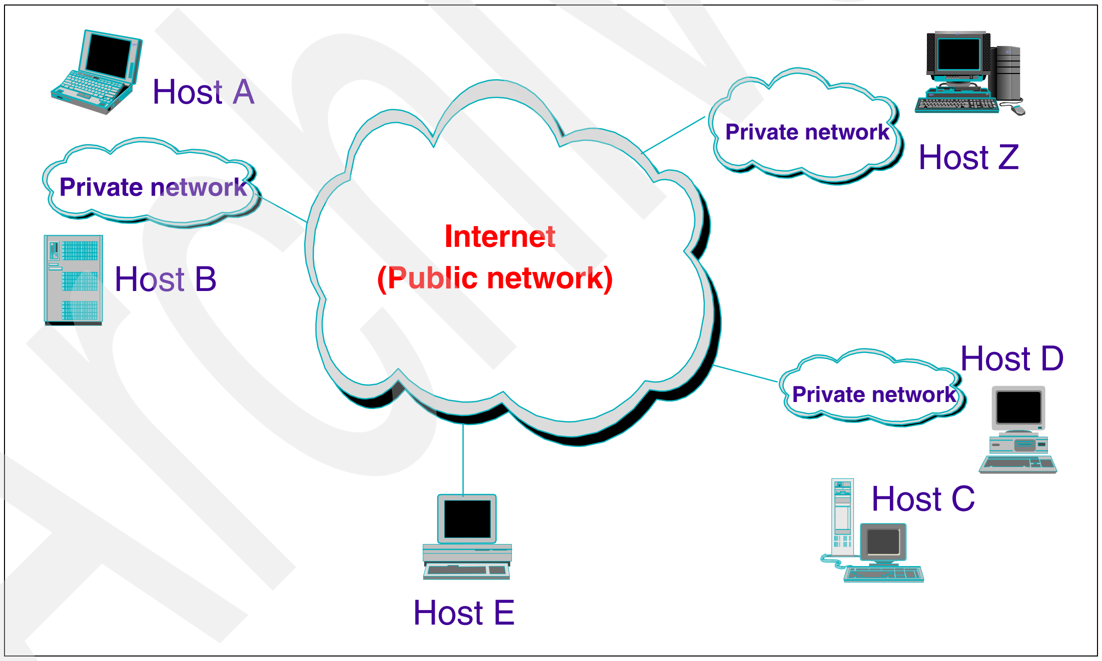


Figure 4-1 Interconnection of networks

- Host A wants to send an e-mail to Host B.
- Host A wants to transfer data to Host D.
- Host C wants to get a Web page from Host E.
- Host Z also wants access to Host E.

This is possible because a public network is a network established and operated by a telecommunication administration or by a Recognized Private Operating Agency (RPOA) for the specific purpose of providing circuit-switched, packet-switched, and leased-circuit services to the public.

Terminology

The following terminology is commonly used to describe a TCP/IP environment:

Host	In the Internet suite of protocols, this is an end system. The end system can be any workstation; it does not have to be a mainframe.
Gateway	A functional unit that interconnects two computer networks with different network architectures. A gateway connects networks or systems of different architectures. A bridge interconnects networks or systems with the same or similar architectures. In TCP/IP, this is a synonym for router.
Port	Each process that wants to communicate with another process identifies itself to the TCP/IP protocol suite by one or more ports. A port is a 16-bit number, used by the host-to-host protocol to identify to which higher level protocol or application program (process) it must deliver incoming messages. There are two types of port:
Well-known	Well-known ports belong to standard servers, for example Telnet uses port 23. Well-known port numbers range between 1 and 1023 (prior to 1992, the range between 256 and 1023 was used for UNIX-specific servers). Well-known port numbers are typically odd, because early systems using the port concept required an odd/even pair of ports for duplex operations. Most servers require only a single port. The well-known ports are controlled and assigned by the Internet central authority (IANA) and on most systems can only be used by system processes or by programs executed by privileged users. The reason for well-known ports is to allow clients to be able to find servers without configuration information.
Ephemeral	Clients do not need well-known port numbers because they initiate communication with servers and the port number they are using is contained in the UDP datagrams sent to the server. Each client process is allocated a port number as long as it needs it by the host it is running on. Ephemeral port numbers have values greater than 1023, normally in the range 1024 to 65535.
Socket	An endpoint for communication between processes or application programs. A synonym for port.
Socket address	The address of an application program that uses the socket interface on the network. In Internet format, it consists of the IP address of the socket's host and the port number of the socket. The application program is usually not aware of the structure of the address.
Socket interface	A Berkeley Software Distribution (BSD) application programming interface (API) that allows users to easily write their own programs.
Router	A router interconnects networks at the internetwork layer level and routes packets between them. The router must understand the addressing structure associated with the networking protocols it supports and take decisions on whether, or how, to forward packets. Routers are able to select the best transmission paths and optimal packet sizes. The basic routing function is implemented in the IP layer of the TCP/IP protocol stack, so any host or workstation running TCP/IP over more than one interface could, in theory and also with most of today's TCP/IP implementations, forward IP datagrams. However, dedicated routers provide much more sophisticated routing than the minimum functions implemented by IP.

IP addressing

To be able to identify a host on the Internet, each host is assigned an address, called the IP address, or Internet address. When the host is attached to more than one network, it is called multi-homed and it has one IP address for each network interface.

IP addresses are represented by a 32-bit unsigned binary value which is usually expressed in a dotted decimal format, where each byte is represented by its decimal form, like 9.12.1.43. The numeric form is used by the IP software. The mapping between the IP address and an easier-to-read symbolic name, for example myhost.ibm.com, is done by a Domain Name System. We first look at the numeric form, which is called the IP address. The Internet Protocol uses IP addresses to specify source and target hosts on the Internet.

Each host must have a unique Internet address to communicate with other hosts on the Internet. The network address part of the IP address is centrally administered by the Internet Network Information Center (the InterNIC) and is unique throughout the Internet. Each IP address is made up of two logical addresses:

```
IP address = <network address> <host address>
```

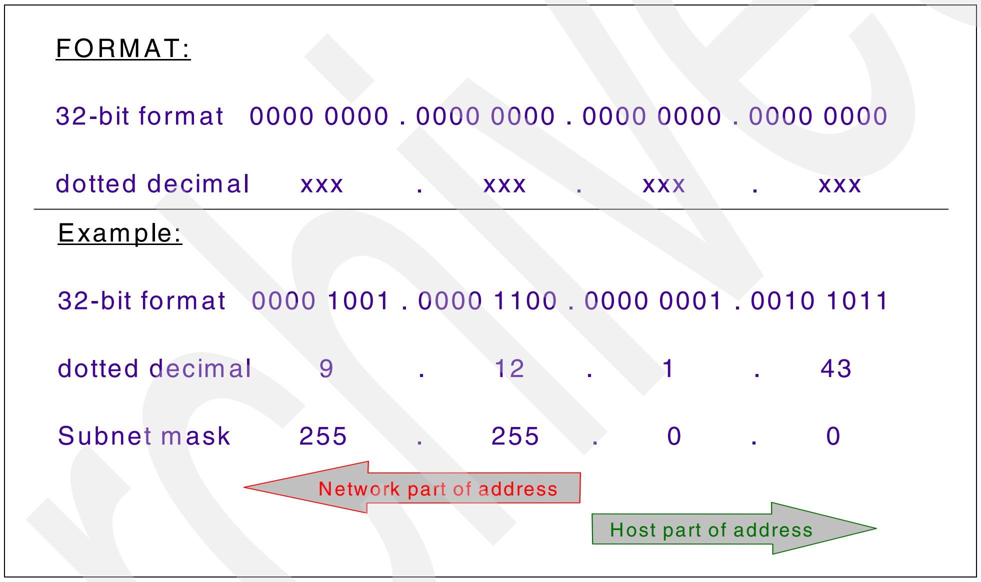

FORMAT:
32-bit format 0000 0000 . 0000 0000 . 0000 0000 . 0000 0000
dotted decimal xxx . xxx . xxx . xxx
Example:
32-bit format 0000 1001 . 0000 1100 . 0000 0001 . 0010 1011
dotted decimal 9 . 12 . 1 . 43
Subnet mask 255 . 255 . 0 . 0
Network part of address
Host part of address

Figure 4-2 The format of an IP address

The first bits of the IP address specify how the rest of the address should be separated into its network and host part.

The Internet Protocol moves data between hosts in the form of datagrams. Each datagram is delivered to the address contained in the Destination Address of the datagram's header.

network address	Represents a specific physical network within the Internet.
host address	Specifies an individual host within the physical network identified by the network address.
subnet mask	Is used to differentiate the network address and host address.

In this example, 9.12.1.43 is an IP address, with 9.12 being the network address and 1.43 being the host address.

4.2 Customizing and starting TCP/IP

Since z/OS V2R5 was shipped, the z/OS Communications Server (CS) was included as a base element. The TCP/IP z/OS component, called z/OS CS, is a reconstructed stack and is able to support both UNIX and non-UNIX socket APIs. It is often called the converged IP stack.

All the TCP/IP socket APIs that supported HPNS are now transparently redirected by run-time support to call the UNIX kernel LFS. The REXX socket API has also been directed to call the kernel. HPNS support is no longer required. The Pascal API, however, still requires the VMCF/IUCV address space to be started, which links the API to the new stack. VMCF or TNF do not respond to commands. This is probably because one or both of the on-restartable versions of VMCF or TNF are still active. To get them to respond to commands, stop all VMCF/TNF users, FORCE ARM VMCF and TNF, then use the EZAZSSI procedure to restart.

The SNA and IP networking stacks have been integrated to a considerable extent. Both stacks use common Data Link Control (DLC) routines to access network hardware, and both types of protocols can flow over the same hardware link. Also, common service routines such as Communications Storage Manager (CSM) exploit use of buffers in common storage for both IP and SNA performance.

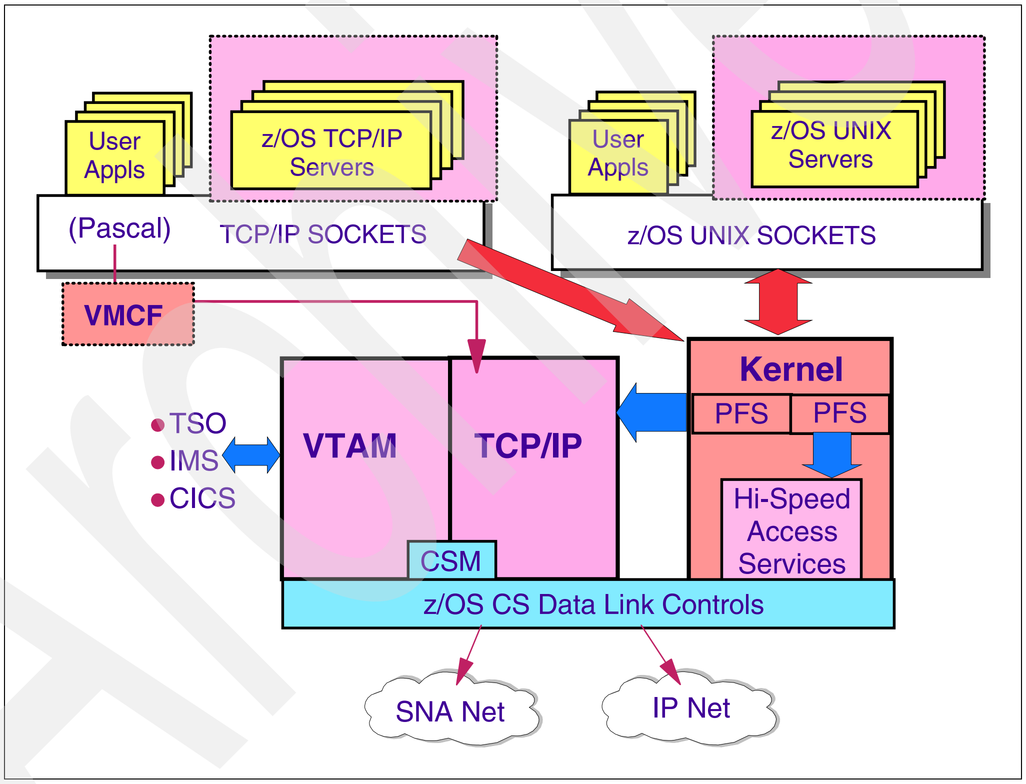


Figure 4-3 z/OS Communication Server

Communication Server for z/OS provides networking support for z/OS UNIX. It exceeds the scope of this book to detail TCP/IP base installation. The customization of TCP/IP for z/OS is documented in product documentation and in *z/OS Communications Server IP Configuration Guide,* SC31-8775. The description of TCP/IP concepts and protocols can be found in *TCP/IP Tutorial and Technical Overview,* GG24-3376.

4.2.1 Using the z/OS TCP/IP configuration wizard on the Web

We recommend using "z/OS UNIX Configuration Wizard", a Web-based tool, to help you set up z/OS UNIX in full function mode. This wizard begins with a series of interviews in which you will answer questions about your application environment and intentions regarding use of z/OS and TCP/IP. After you finish answering all of the interview questions, you ask the wizard to build the output. Then the wizard produces a checklist of steps for you to follow, as well as customized jobs and other data sets for you to use. Specifically, it builds two BPXPRMxx members and two HFS files and some RACF ALTUSER commands. The checklist of follow-on actions includes links to sections of *z/OS UNIX System Services Planning,* GA22-7800 and the *z/OS Communications Server IP Configuration Guide,* SC31-8775, thus eliminating the need to reference multiple documents.

Use this wizard to configure z/OS UNIX for the first time or to check and verify some of your configuration settings. To use the wizard, go to:

http://www-1.ibm.com/servers/eserver/zseries/zos/wizards/

4.2.2 TCP/IP data sets and configuration files

The following files, data sets, and parmlib members to be customized are:

- Configuration files used by TCP/IP
  - PROFILE.TCPIP is used only for the configuration of the TCPIP stack. During initialization of the TCPIP stack, also referred to as the TCPIP address space, system operation and configuration parameters for the TCPIP stack are read from the configuration file PROFILE.TCP.
  - TCPIP.DATA is used during configuration of both the TCPIP stack and applications. This data set, TCPIP.DATA, is used to specify configuration information required by TCP/IP client programs.

Customize the TCP/IP PROFILE data set

TCP/IP reads the parameters from the TCP/IP PROFILE data set.

The IP address of the HOME statement for this host as well as the GATEWAY values such as subnet mask, subnet, and DEFAULTNET (or default gateway) can be obtained from your network administrator.

- The parameters that need to be changed are:

AUTOLOG	Uncomment FTPD or any other daemons that need to be activated.
DEVICE	Provide the z/OS address network interface. It could be the OSA address, CTC, IBM 2216 router or any other supported network device.
LINK	Provide the description of network interface.
HOME	Specify the IP address of the z/OS system.
Begin route	Specify the IP address the net and subnet to which this host belongs.
Route default	Specify the default gateway IP address. Usually, this is the IP address of the network router to which this host is attached.

Note: A sample of the PROFILE data sets is provided in hlq.SEZAINST(SAMPPROF), which you can copy to SYS1.TCPPARMS(PROFILE).

Customize the TCPIP.DATA data set

The TCPIP.DATA configuration data set is the anchor configuration data set for the TCP/IP stack and all TCP/IP servers and clients running on that stack. In z/OS Communication Server, you may define the TCPIP.DATA parameters in an HFS file or in an MVS data set. The TCPIP.DATA configuration data set is read during initialization of all TCP/IP server and client functions.

The SYSTCPD DD explicitly identifies which data set is to be used to obtain the parameters defined by TCPIP.DATA.

The SYSTCPD DD statement should be placed in the TSO/E logon procedure or in the JCL of any client or server executed as a background task. The data set can be any sequential data set or a member of a partitioned data set (PDS).

```
//SYSTCPD  DD DSN=SYS1.TCPPARMS(TCPDATA),DISP=SHR
```

Parameters that need to be changed for the TCPDATA file are:

- TCPIPJOBNAME specifies the TCPIP started task job name.
- HOSTNAME specifies the host name (SYSNAME on IEASYSxx) or IEASYMxx.
- DATASETPREFIX specifies the hlq you have selected before.

Other optional parameters are:

- DOMAINORIGIN specifies your domain (yourdomain.com).
- NSINTERADDR specifies your name server IP address.

The values for the DOMAINORIGIN and NSINTERADDR statements can be obtained from your network administrator.

Note: The syntax for the parameters in the TCPIP DATA file can be found in *z/OS Communication Server IP Configuration Reference, z/OS Communications Server IP Configuration Reference,* SC31-8776

A sample TCPIP.DATA config file is provided in hlq.SEZAINST(TCPDATA).

z/OS IP search order profile

During the initialization of the TCP/IP stack, system configuration and configuration parameters for the TCP/IP stack are read from the configuration profile PROFILE.TCPIP.

The search order used by TCP/IP to find PROFILE.TCPIP data sets involves both explicit and dynamic data set allocation; see Figure 4-4 on page 192.

When TCP/IP starts, it looks for the PROFILE data set in the following order:

1. //PROFILE DD explicitly specified in the PROFILE DD statement of the TCP/IP started task procedure
2. jobname.nodename.TCPIP data set
3. hlq.nodename.TCPIP data set
4. jobname.PROFILE.TCPIP data set
5. hlq.PROFILE.TCPIP data set

Important: The search stops if one of these data sets is found.

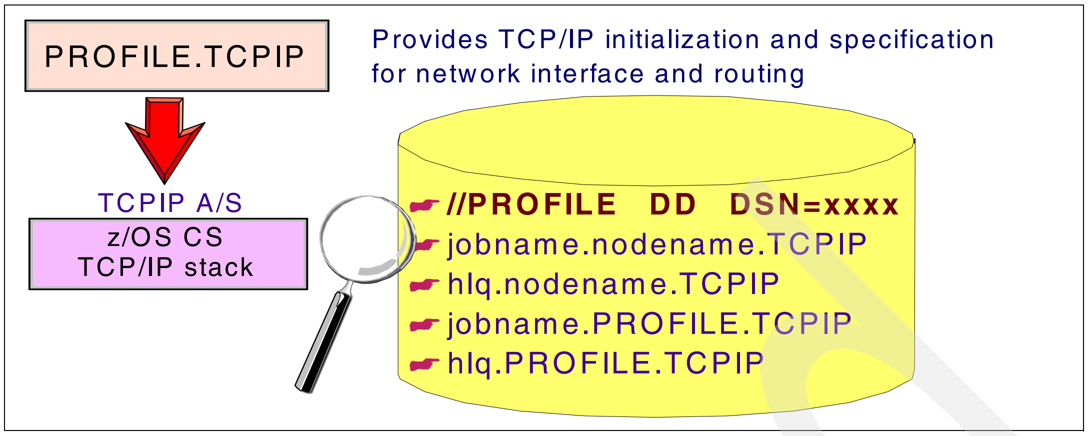


Figure 4-4 zOS IP search order profile

z/OS IP search order

The z/OS CS environment consists of the z/OS CS stack, z/OS CS applications (z/OS UNIX Services applications such as rtelnetd, FTPD, etc.) and the z/OS TCP/IP native z/OS applications; see Figure 4-5.

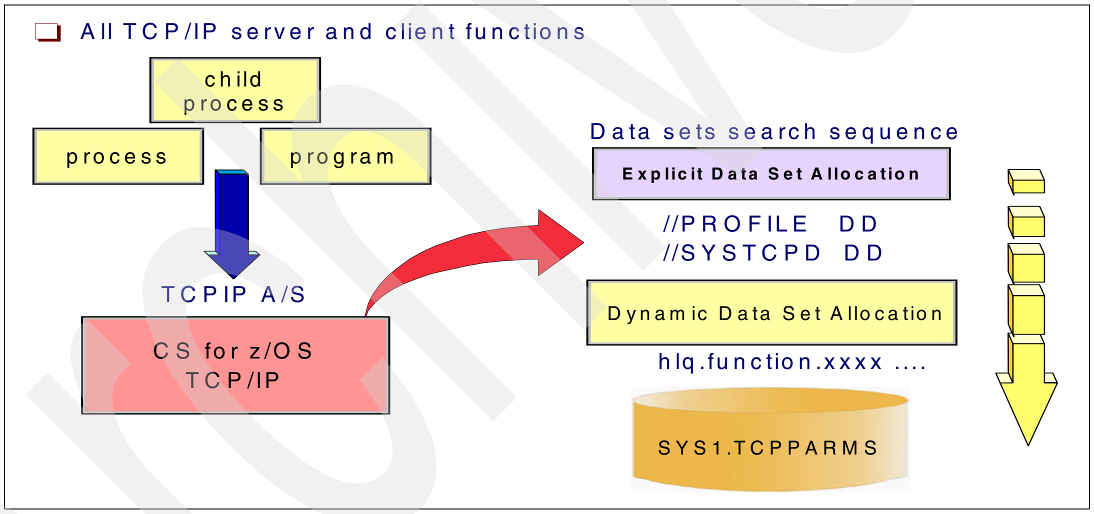

All TCP/IP server and client functions
child process
process
program
TCPIP A/S
CS for z/OS TCP/IP
Data sets search sequence
Explicit Data Set Allocation
//PROFILE DD
//SYSTCPD DD
Dynamic Data Set Allocation
hlq.function.xxxx ....
SYS1.TCPPARMS

Figure 4-5 z/OS IP search order

The TCP/IP stack and set of applications have some common configuration files, but they also use configuration files that are different.

Different configuration files may be used for a TCP/IP stack where there is a need to understand the search order for each z/OS UNIX application.

The search order is applied to any configuration file, and the search ends with the first file found, as follows:

- Explicit Data Set Allocation consists of those data sets that you specify through the use of DD statements in JCL procedures.
- Dynamic Data Set Allocation consists of multiple versions of a data set, each having a different high-level qualifier or middle-level qualifier, and some data sets that can only be dynamically allocated by TCP/IP (they cannot be allocated using DD statements in JCL).

There is a naming convention for dynamically allocated data sets.

TCPDATA search order

TCPIP.DATA is used during configuration of both the TCP/IP stack and applications. The search order to find the TCPIP.DATA data set is the same for both the TCP/IP stack and applications. The search order used by TCP/IP and applications is as follows:

1. MVS data set or HFS file specified by the environment variable RESOLVER_CONFIG
2. /etc/resolv.conf
3. //SYSTCPD DD
4. jobname.TCPIP.DATA data set
5. SYS1.TCPPARMS(TCPDATA)
6. hlq.TCPIP.DATA data set

Important: The search stops if one of these data sets is found.

A name resolver converts a TCP/IP host name to an IP address, or vice versa. There are several name resolvers in TCP/IP, one in Language Environment, and one in CICS, and multiple resolver libraries exist. There are MVS native and Language Environment resolver APIs. The search order for TCPIP.DATA varies across resolver libraries. This can lead to an inconsistent name resolution process. It makes it difficult to provide resolver enhancements in a consistent and timely manner.

TCP/IP resolver address space

A new TCP/IP resolver address space introduced in z/OS V1R2 supports the consolidation of the many ways to resolve host names or IP addresses. It provides for both global and local user settings to be configured.

LFS is responsible for starting the new TCP/IP resolver address space. The new address space is started during IPL. In order for LFS to start this new resolver address space, the system must be configured with an AF_INET socket file system domain name in BPXPRMxx.

Setting up a resolver address space

There are two ways in which to start the resolver address space:

- z/OS UNIX initialization will attempt to start the resolver unless explicitly instructed not to. Using z/OS UNIX is the recommended method since it will ensure that the resolver is available before any applications can make a resolution request.

  A BPXPRMxx statement, RESOLVER_PROC, is used to specify the procedure name, if any, to be used to start the resolver address space. If the RESOLVER_PROC statement is not in the BPXPRMxx parmlib member or is specified with a procedure name of DEFAULT, z/OS UNIX will start a resolver address space with the assigned name of RESOLVER. The resolver uses the applicable search order for finding TCPIP.DATA statements but without a GLOBALTCPIPDATA specification. If the address space cannot be started, z/OS UNIX initialization continues.

  When z/OS UNIX starts the resolver, it is done so that the resolver does not require JES (that is, SUB=MSTR is used). For SUB=MSTR considerations, refer to *z/OS MVS JCL Reference,* SA22-7597.

  If the RESOLVER_PROC statement has been used to specify a start procedure name, then:

– To find the procedure, it must reside in a data set that is specified by the MSTJCLxx parmlib member's IEFPDSI DD card specification. For MSTJCL considerations, refer to *z/OS MVS Initialization and Tuning Reference,* SA22-7592.

– The procedure must not contain any DD cards that specify `SYSOUT=*`.

Since z/OS UNIX does not receive any error indication when it tries to start the address space, it issues an informational message containing the name of the procedure it has started. The message is:

```
BPXF224I THE RESOLVER_PROC, procname, IS BEING STARTED.
```

Note: If the RESOLVER_PROC statement is not present or is specified with a procedure name of DEFAULT, procname will be RESOLVER even though no start procedure was used. If you want to use the procedure name RESOLVER, a RESOLVER_PROC(RESOLVER) statement must be added to your BPXPRMxx parmlib member.

If the start procedure is not found or has a JCL error in it, the usual z/OS error messages will be issued.

For more detailed information refer to *z/OS UNIX System Services Planning,* GA22-7800.

- An installation can use its automation tools to start the resolver with the MVS START operator command. If this approach to starting the resolver is used, care should be taken to ensure that no applications that need resolver services (for example, INETD) are started before the resolver address space is initialized. This may mean removing the starting of INETD from the z/OS UNIX /etc/rc file and starting INETD with automation after the resolver has initialized.

Note: For more detailed information about resolver configuration, refer to *z/OS Communications Server IP Configuration Guide,* SC31-8775.

Customize the TCP/IP procedure

Create the following PROCLIB members, as follows:

- TCPIP started task procedure; a sample is provided in hlq.SEZAINST(TCPIPROC).
- EZAZSSI procedure to start the TCP subsystem interface
- Add procedure EZAZSSI to your system PROCLIB. A sample of this procedure is located in the data set hlq.SEZAINST (where hlq is the high-level qualifier for the TCP/IP product data sets in your installation).

```
//EZAZSSI    PROC   P=''
//STARTVT    EXEC   PGM=EZAZSSI,PARM=&P
//STEPLIB    DD     DSN=hlq.SEZALINK,DISP=SHR
//           DD     DSN=hlq.SEZATCP,DISP=SHR
```

You may remove the STEPLIB DD if these data sets are defined on LNKLSTxx.

Modify your TSO/E logon procedure:

- To include hlq.SEZAHELP in //SYSHELP DD.
- To include hlq.SEZAMENU in //ISPMLIB DD.
- To include hlq.SEZAPENU in //ISPTLIB DD and //ISPPLIB DD.

- Optionally add a //SYSTCPD DD to point to the TCPDATA data set in order to use TCP/IP client functions and some administrative functions such as OBEYFILE under TSO/E.

  SYSTCPD explicitly identifies the data set used to obtain parameters defined by TCPIP.DATA.
- The SYSTCPD statement should be placed in the TSO/E logon procedure or in the JCL of any client or server executed as a background task. The data set can be any sequential data set or a member of a partitioned data set (PDS). TSO client functions can be directed against any of a number of TCP/IP stacks. Obviously, the client function must be able to find the TCPIP.DATA appropriate to the stack of interest at any one time. Two methods are available for finding the relevant TCPIP.DATA:
  - Add a SYSTCPD DD statement to your TSO logon procedure. The issue with this approach is that a separate TSO logon procedure per stack is required, and users have to log off TSO and log on again using another TSO logon procedure in order to switch from one stack to another.
  - Use one common TSO logon procedure without a SYSTCPD DD statement. Before a TSO user starts any TCP/IP client programs, the user has to issue a TSO ALLOC command wherein the user allocates a TCPIP.DATA data set to DDname SYSTCPD. To switch from one stack to another, the user simply has to de-allocate the current SYSTCPD allocation and allocate another TCPIP.DATA data set.

Combine the first and second methods. Use one logon procedure to specify a SYSTCPD DD for a default stack. To switch stacks, issue TSO ALLOC to allocate a new SYSTCPD. To switch back, issue TSO ALLOC again with the name that was on the SYSTCPD DD in the logon procedure. The disadvantage of this approach is that the name that was on the SYSTCPD DD is “hidden” in the logon procedure and needs to be retrieved or remembered.

Customizing TCP/IP in parmlib

Choose a High-Level Qualifier (hlq). TCP/IP is distributed with default high-level qualifier (HLQ) of TCPIP. This HLQ is a hard-coded character string within z/OS UNIX. TCP/IP uses dynamic allocation for several parameter data sets, and TCP/IP uses the default HLQ for allocating the data set. You can accept the default HLQ or override it. To override the default HLQ used by dynamic data set allocation, specify the DATASETPREFIX statement in the PROFILE.TCPIP or TCPIP.DATA configuration files. But remember, the DATASETPREFIX value is used as the last step in the search order for most configuration files.

Note: Another way is to put the parameter in the /etc directory, which resides in the HFS. For more information on HLQ and dynamic data set allocation, see *z/OS Communications Server IP Configuration Guide,* SC31-8775.

We recommend that you use the `DATASETPREFIX` items in the TCP/IP profile data set and the TCP/IP client parameter (data) file to specify an HLQ for your installation. We used this method and specified TCPIP as the HLQ.

Update the following parmlib members:

- Update IEAAPFxx or PROGxx parmlib member to authorize the following TCP/IP libraries:
  - hlq.SEZATCP
  - hlq.SEZADSIL
  - hlq.SEZALOAD
  - hlq.SEZALNK2
  - hlq.SEZALPA
  - hlq.SEZAMIG

- Update the LNKLSTxx parmlib member to contain hlq.SEZALOAD.
- Update the LPALSTxx parmlib member to contain hlq.SEZALPA and hlq.SEZATCP.
- Update the IECIOSxx parmlib member to disable the MIH processing for communication devices used by TCP/IP:

  ```
  Set MIH TIME=00:00,DEV=(cuu-cuu)
  ```

  Where `cuu` is the range of devices used by TCP/IP.
- Update the IEFSSNxx parmlib member to use restartable VMCF and TNF by adding the following two statements:

  ```
  SUBSYS SUBNAME(TNF)
  SUBSYS SUBNAME(VMCF)
  ```
- Update the COMMNDxx parmlib member to start SSI during IPL by adding the following:

  ```
  COM='S EZAZSSI,P=sys_name'
  ```

  Where `sys_name` is the SYSNAME in IEASYSxx or specified in IEASYMxx using the SYSDEF statement.

  The label of the HOSTNAME statements in the hlq.TCPIP.DATA refers to sys_name. This node name can be set to the MVS NJE node of this system or the SYSID.
- Update the IFAPRDxx parmlib member to ensure the state of the TCP/IP BASE feature is enabled with the following:

  ```
  NAME(z/OS) ID(5647-A01)
  ```
- Customize the z/OS UNIX BPXPRMxx parmlib member. Copy the statements in Figure 4-6 on page 196 into your BPXPRMxx member to activate TCP/IP support for a single transport provider.

```
FILESYTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)
   NETWORK DOMAINNAME(AF_INET)
           DOMAINNUMBER(2)
           MAXSOCKETSR(10000)
           TYPE(INET)
```

Figure 4-6 BPXPRMxx entries for a single TCP/IP transport provider

Customize TCP/IP in RACF

You need to define a RACF user ID with an OMVS segment for TCPIP, PORTMAP, NFS, and FTPD, as shown:

```
PROCNAME     RACF user ID   UID     RACF Group GID  Trusted
TCPIP        TCPIP           0      OMVSGRP     1    No
PORTMAP      TCPIP           0      OMVSGRP     1    No
NFS          TCPIP           0      OMVSGRP     1    No
FTPD         TCPIP           0      OMVSGRP     1    No
INETD        OMVSKERN        0      OMVSGRP     1    No
```

You also need to define the following data sets to program control:

- SYS1.LINKIB
- hlq.SEZALOAD
- hlq.SEZATCP
- CEE.SCEERUN (or Language Environment Run-time Modules)

In a z/OS UNIX environment, there are additional security concerns related to the Hierarchical File System (HFS) and the loading of programs that are considered trusted.

Program control facilities in RACF and z/OS provide a mechanism for ensuring that the z/OS program loading process has the same security features that APF authorization provides in the native MVS environment.

It is recommended that you enable program control in your installation. If you define the BPX.DAEMON Facility Class, then you must enable program control for certain z/OS Communication Server for z/OS load libraries. Review the section on Program Control in *z/OS UNIX System Services Planning,* GA22-7800 to decide whether program control is appropriate for your installation.

When you use program control, make sure that all load modules that are loaded into an address space come from controlled libraries. If the MVS contents supervisor loads a module from a non controlled library, the address space becomes dirty and loses its authorization. To prevent this from happening, define all the libraries from which load modules can be loaded as program controlled. At a minimum, this should include the C runtime library, the TCP/IP for MVS SEZALOAD and SEZATCP libraries, and SYS1.LINKLIB.

Customize TCP/IP

Customize SERVICES and RPC data sets, as follows:

Copy hlq.SEZAINST(SERVICES) to hlq.ETC.SERVICES. This file specifies the combination of port and services (UPD or TCP) used by TCP/IP.

To establish a relationship between the servers defined in the /etc/inetd.conf file and specific port numbers in the z/OS UNIX environment, insure that statements have been added to ETC.SERVICES for each of these servers. See the sample ETC.SERVICES installed in the /usr/lpp/tcpip/samples/services directory for how to specify ETC.SERVICE statements for these servers.

An HFS file, /etc/services, could also be created instead of this file:

Copy hlq.SEZAINST(ETCRPC) to hlq.ETC.RPC. This file specifies the port mapper, which used to be called portmap daemon.

Note: See *z/OS UNIX System Services Planning,* GA22-7800 for more information on TCP/IP for z/OS data set name rules with UNIX System Services.

Starting and testing TCP/IP

If you start TCP/IP when OMVS is up, you should get messages like the following:

```
EZZ4202I EZZ4202I OPENEDITION-TCP/IP CONNECTION ESTABLISHED FOR TCPIP
```

Figure 4-7 z/OS console output after TCP/IP and z/OS UNIX connect

You may use the TSO PING, TRACERTE, NETSTAT, and NSLOOKUP commands from the z/OS UNIX environment.

The z/OS UNIX **`ping`** command sends an echo request to a foreign node (remote node) to determine whether the computer is accessible.

When a response to a **`ping`** command is received, the elapsed time is displayed. The time does not include the time spent communicating between the user and the TCP/IP address space.

Use the **`ping`** command to determine the accessibility of the foreign node.

Note: **`Ping`** is a synonym for the **`oping`** command in the z/OS UNIX shell. The **`ping`** command syntax is the same as that for the **`oping`** command.

The z/OS UNIX **`nslookup`** command enables you to query any name server to perform the following tasks from the z/OS UNIX environment:

- Identify the location of name servers.
- Examine the contents of a name server database.
- Establish the accessibility of name servers.

```
MVS TCP/IP NETSTAT CS  V1R4       TCPIP NAME: TCPIP       17:18:26
User Id  Conn  Local Socket       Foreign Socket          State
-------  ----  ------------       --------------          -----
BPXOINIT 00013 0.0.0.0..10007     0.0.0.0..0              Listen
TCPIP    00011 127.0.0.1..1025    127.0.0.1..1026         Establs
TCPIP    00012 0.0.0.0..23        0.0.0.0..0              Listen
TCPIP    0000B 0.0.0.0..1025      0.0.0.0..0              Listen
TCPIP    00010 127.0.0.1..1026    127.0.0.1..1025         Establs
TCPIP    00015 0.0.0.0..1028       *..*                     UDP
```

Figure 4-8 Example using NETSTAT TCP TCPIP

With the preceding customization steps, you should be able to use the TSO client and to start up the TCP/IP and OMVS started tasks.

4.2.3 Implementing the sample system

To implement the multiple transport driver (stack), perform the following steps:

- Naming conventions

  Plan your naming conventions according to your site conventions.

- Started tasks

  Define the stack started task. We used `TCPIP`.

- BPXPRMxx

  Customize the z/OS UNIX BPXPRMxx parmlib member.

The started task name of the transport provider for z/OS UNIX has to be reflected in the NAME option:

```
SUBFILESYSTYPE NAME(TCPIP)
TYPE(CINET)
ENTRYPOINT(EZBPFINI)
DEFAULT
```

Copy the following (see Figure 4-9 on page 199) to your BPXPRMxx member to activate TCP/IP support for multiple transport providers.

```
/* Parameter for Common Internet Socket support   */

FILESYSTEM  TYPE(CINET)  ENTRYPOINT(BPXTCINIT)
SUBFILESYSTYPE NAME(TCPIP)
               TYPE(CINET)
               ENTRYPOINT(EZBPFINI)
               DEFAULT

NETWORK DOMAINNAME(AF_INET)
        DOMAINUMBER(2)
        MAXSOCKETS(10000)
        TYPE(CINET)
        INADDRANYPORT(4901)
        INADDRANYCOUNT(100)
```

Figure 4-9   BPXPRMxx entries for a multiple TCP/IP transport provider

RACF

Define the required RACF profiles for the started tasks.

Table 4-1   RSCF profiles for the started tasks

Procname	RACF user ID	UID	RACF Group	GID	Trusted
TCPIP	TCPIP	0	OMVSGRP	1	Yes

Name resolution environment

Customize your name resolution environment:

- If you use a name server for name resolution, reflect your transport providers in the name server database.
- If you use host site tables for name resolution, reflect the names and IP addresses of foreign IP hosts in your hlq.HOSTS.LOCAL data set. Use the TCP/IP MAKESITE command to compile the hlq.HOSTS.LOCAL data set into the hlq.HOSTS.SITEINFO and hlq.HOSTS.ADDRINFO data sets. Make sure that the names and IP addresses of your transport providers are also entered in the name tables of other hosts.

TSO logon procedure

Add a `TCPDATA DD` statement in your TSO logon procedure. In our environment, this statement was as follows:

```
//SYSTCPD  DD  DSN=TCPIPMVS.TCPIP.DATA,DISP=SHR
```

Note: The `//SYSTCPD DD` statement points to the TCP/IP MVS client parameter file. The default DSN is TCPIP.TCPIP.DATA or SYS1.TCPPARMS(TCPDATA). If you do not use one of these defaults, you have to add a `SYSTCPD DD` statement in the TSO logon procedure in order to use TCP/IP client functions and some administrative functions such as (OBEYFILE) under TSO. TCPIPMVS.TCPIP.DATA was the parameter data set that we used. Therefore, we had to add the `//SYSTCPD DD` statement to our logon procedure.

If you start your TCP/IP transport providers when OMVS is up, you should get a message like the one shown in Figure 4-10 for each transport provider.

```
EZZ4202I OPENEDITION-TCP/IP CONNECTION ESTABLISHED FOR TCPIP
BPXF206I ROUTING INFORMATION FOR TRANSPORT DRIVER TCPIP HAS BEEN
INITIALIZED OR UPDATED
```

Figure 4-10 z/OS console output after TCP/IP transport provider startup

Start and test your transport providers

Start your transport providers and do some connectivity tests using TSO client commands.

The following commands are samples you could use to test that the steps you did up to now were successful:

- PING <hostname>
- HOMETEST
- NETSTAT DEVLINKS TCP <tcpipjobname>
- TESTSITE if you use host tables
- NSLOOKUP <hostname> if you use a nameserver

If you issue the TSO NETSTAT command now, you will still not see a socket connected to the transport provider. This is because the z/OS UNIX inetd daemon is not started. See: 4.3, “Customizing for inetd and rlogind daemons” on page 201 for information on how to customize the inetd daemon.

Now we have two TCP/IP stacks running. To be able to start another server to relate to a specific stack, you can use the _BPXK_SETIBMOPT_TRANSPORT z/OS UNIX shell environment variable.

For example, if your TCP/IP stack is configured under C_INET and you want to start an FTPD server instance that was associated with it, you could use the FTPD procedure as shown in Figure 4-11.

```
//FTPD  PROC PARMS='TRACE'
//*
//*  z/OS UNIX shell zOS FTP Server main process
//*  Resulting address space name will be FTPD1, when
//*  we use this method to start FTPD
//*
//FTPD EXEC PGM=FTPD,
//        REGION=40M,TIME=NOLIMIT
//        PARM=('POSIX(ON),ALL31(ON)',
//        'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIP")',
//        '/&PARMS')
//CEEDUMP DD SYSOUT=*
//        .
//        .
//        .
```

Figure 4-11 FTPD procedure

4.3 Customizing for inetd and rlogind daemons

Follow the following steps to customize your inetd and rlogin environment daemons.

Using inetd, master of daemons

The inetd daemon is a master of other daemons that execute in z/OS UNIX. The function of inetd is to listen on certain well-known network ports for a request to run one of a number of daemons. When a request is received, inetd creates a new socket for remote connection, and then fork()s a new address space and uses exec() to start the requested daemon program.

The daemon started by inetd relates to the port where the request arrived. The correlation between port number and daemon is stored in configuration file /etc/inetd.conf.

The daemons started by inetd include:

- The rlogin daemon starts a shell session for a user rlogin request.
- The telnet daemon starts a shell session for a user telnet request.
- The rexec daemon executes a single command on z/OS UNIX requested by a remote user entering a rexec command.
- The rsh daemon starts a shell session and runs a script generated by a remote user entering a rsh command.

Customization is needed to enable inetd to run on your system. You must decide how to start it, and what RACF ID it will execute under. If you have implemented enhanced daemon security with BPX.DAEMON, you must define inetd to BPX.DAEMON and implement program control. Finally, you have to configure the relationship between the ports that inetd listens on and the daemons to be started.

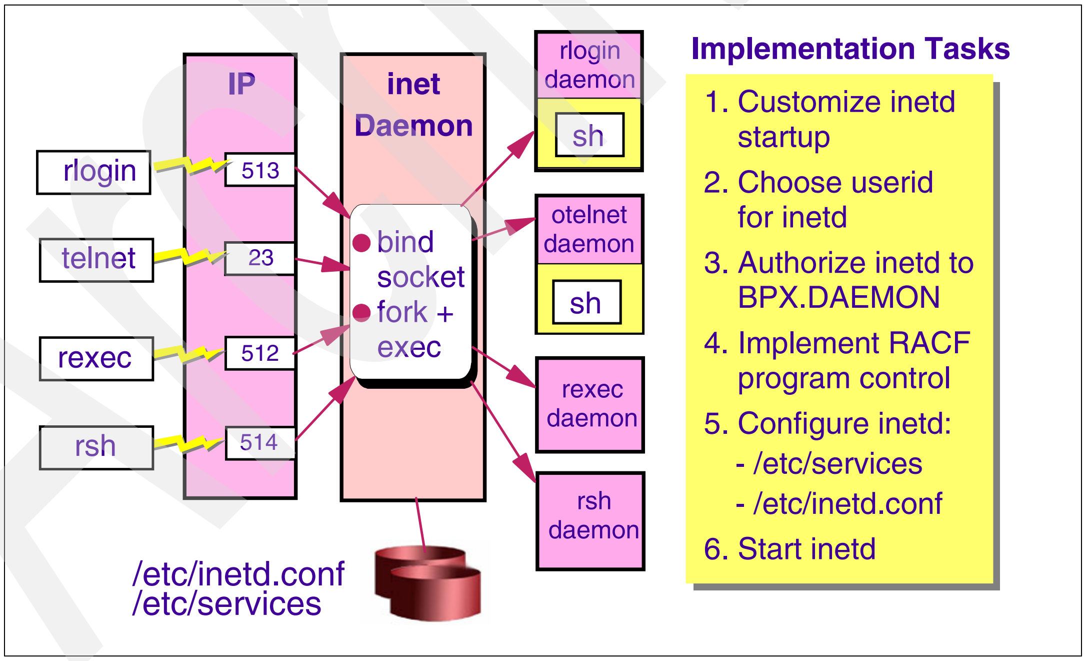


Figure 4-12 Inetd, master of daemons

4.3.1 Customize inetd

Start inetd

The inetd daemon program can be found in two places. In the HFS, the program file is /usr/sbin/inetd, but IBM has set the sticky bit on. A copy of this program is found in SYS1.LINKLIB(INETD), so this is the program that is used. Start from a line in the initialization script /etc/rc. In this case, use a command similar to the line shown:

```
# Start the INET daemon for remote login activity
_BPX_JOBNAME='INETD' /usr/sbin/inetd  /etc/inetd.conf &
```

Establish the inetd userid

The next step is to decide which user ID to associate with inetd. It needs to be a superuser (UID=0), and have minimum access to MVS data sets. How you do this depends on start mode:

- When started from /etc/rc, inetd inherits user ID OMVSKERN, which is a superuser. Starting up via /etc/rc you are effectively locked into using the user ID under which the /etc/rc script is running, as inetd is forked from that script. The user ID for /etc/rc is the kernel ID OMVSKERN.

  ```
  RDEFINE STARTED INET.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(NO))
  ```

- If you have activated the RACF BPX.DAEMON facility, then the inetd user ID must be authorized to this facility.

  ```
  PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSKERN) ACCEDD(READ)
  ```

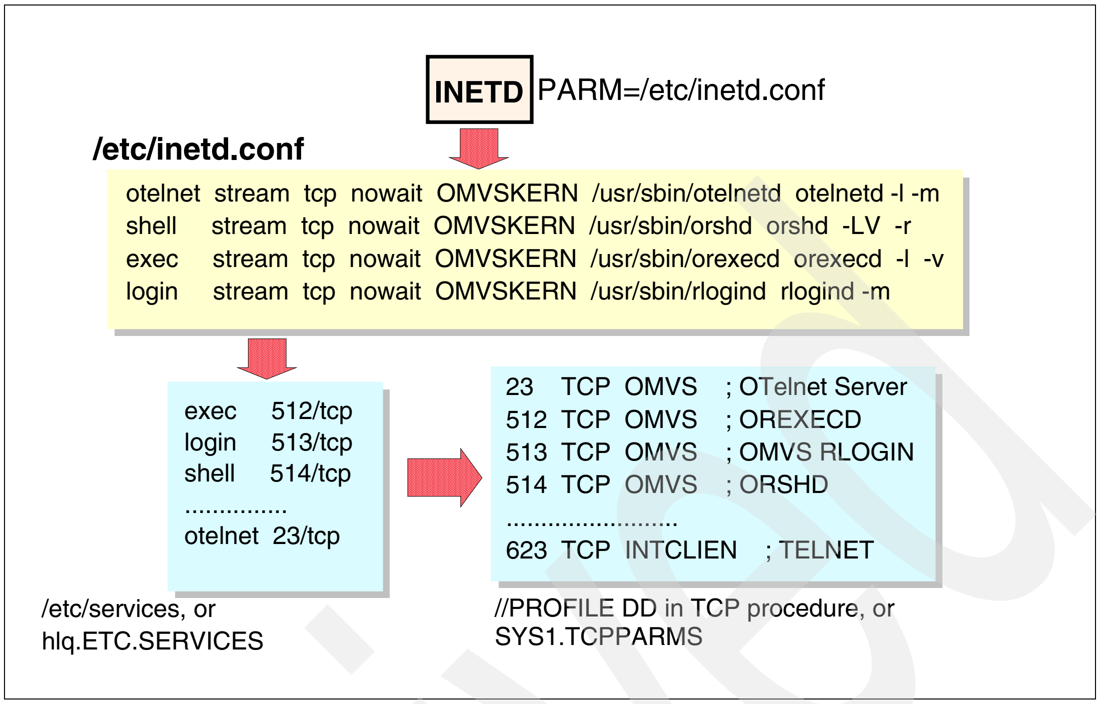
Switch on program control

If you have set up the BPX.DAEMON, then you need to make sure that all programs are loaded into the inetd address space. At a minimum, you should protect the following programs:

- SYS1.LINKLIB(INETD)
- CEE.SCEERUN - LE/MVS run-time, whole library

Inetd configuration files

Figure 4-13 shows three configuration files that have to be updated for inetd support.



```
otelnet  stream  tcp  nowait  OMVSKERN  /usr/sbin/otelnetd  otelnetd -l -m
shell    stream  tcp  nowait  OMVSKERN  /usr/sbin/orshd  orshd  -LV  -r
exec     stream  tcp  nowait  OMVSKERN  /usr/sbin/orexecd  orexecd  -l  -v
login    stream  tcp  nowait  OMVSKERN  /usr/sbin/rlogind  rlogind -m
```

```
exec     512/tcp
login    513/tcp
shell    514/tcp
...............
otelnet  23/tcp
```

```
23   TCP  OMVS    ; OTelnet Server
512  TCP  OMVS    ; OREXECD
513  TCP  OMVS    ; OMVS RLOGIN
514  TCP  OMVS    ; ORSHD
.........................
623  TCP  INTCLIEN    ; TELNET
```

Figure 4-13 Inetd configuration files

The primary file is /etc/inetd.conf, which is the inetd configuration file. There is one entry (line) in this file for each daemon controlled by inetd. The fields are interpreted as follows:

- Field (1) - Service name - match daemon entry in /etc/services file
- Field (2) - Daemon socket type - stream or dgram
- Field (3) - Daemon socket protocol - TCP or UDP
- Field (4) - Wait_flag - can be wait (single thread server - one request at a time) or nowait (multiple requests queued)
- Field (5) - Login_name - RACF user ID under which daemon will run
- Field (6) - Server_program - name of daemon program in HFS
- Field (7) - Server-arguments - first string is job name for daemon address space, and the rest is the parm string to pass to daemon

There is a corresponding entry in /etc/services for each daemon in inetd.conf. The entry lists the port where inetd listens for daemon requests.

The TCPIP PROFILE configuration must list the same ports in the PORT section. This entry identifies the job name authorized to open the socket to this port and the type of socket allowed.

The two TCP/IP files usually exist already—you must make sure that inetd.conf corresponds with the values listed. You may want to change the port number for a daemon.

4.3.2 Customizing the rlogind daemon

Rlogin to z/OS UNIX services

This is a similar flow through an rlogin request made through TCP/IP to z/OS UNIX. Assume that the user has already done a login to the local host as *rob*. The user issues **`rlogin`** from the shell session. The format depends on the local host. z/OS UNIX accepts an rlogin under the current ID (rob) or the new ID (jane).

The AIX/UNIX rlogin client sends the request to port 513 on the host, monitored by the inetd daemon. inetd forks a new address space and initializes the rlogind server.

The rlogind server uses a z/OS UNIX socket, created by inetd and passed via fork, to communicate with the rlogin client. The server then proceeds to validate the rlogin request, as follows:

- It reads the RACF user profile for the rlogin user ID passed (the current or new user ID). It also reads the contents of the RACF OMVS segment.
- It prompts the remote client for the correct (RACF) password. Note that z/OS UNIX does not support the use of either /etc/equiv.hosts or $HOME/.hosts files defined in HFS to bypass authentication.

If authentication is good, rlogind allocates a standard z/OS UNIX pseudo tty terminal pair, and then initiates the client shell in one of two ways:

- It creates a child shell process using local_spawn() and the validated user ID.
- It forks a copy of itself in a new address space, uses **`setuid()`** and **`seteuid()`** commands to set RACF security to a valid user ID, and then runs an exec() shell program.

Figure 4-14 illustrates how this works.

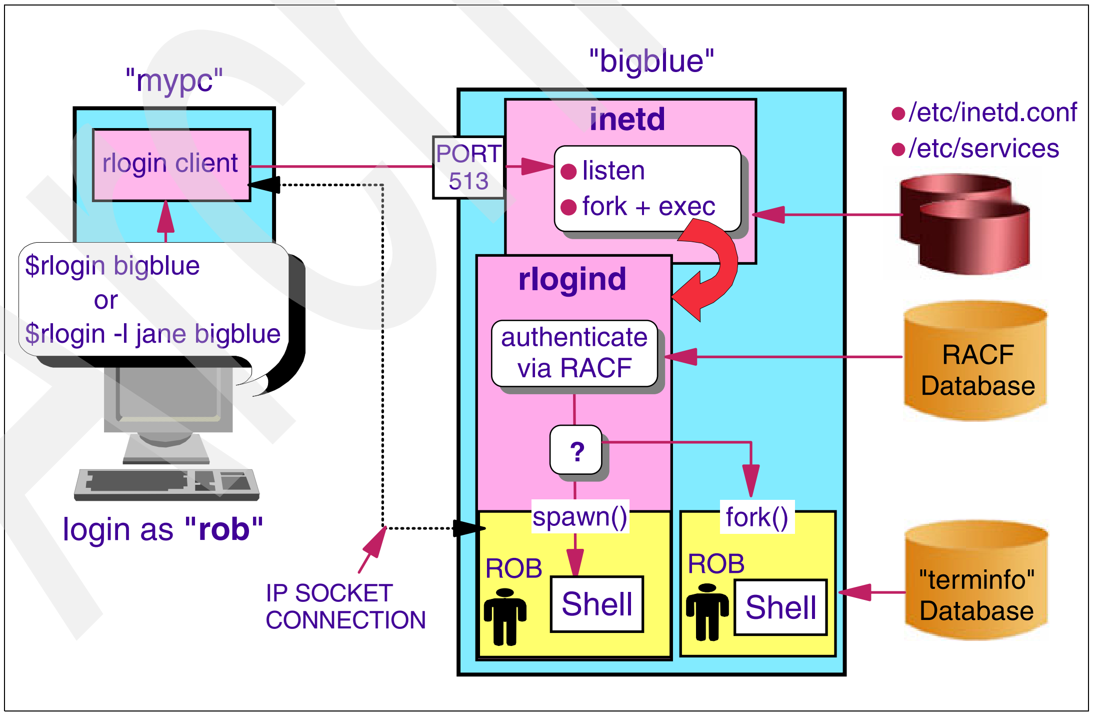


Figure 4-14 Inetd daemon

Activating the z/OS UNIX rlogin daemon

Figure 4-15 on page 206 describes the steps to customize z/OS UNIX for the rlogin daemon, as follows:

1. Go through the steps to customize the z/OS UNIX inetd daemon, and test that the daemon is able to start.

   Identify the user ID under which rlogind (the login daemon) will run. The rlogind program as a daemon needs to be a superuser (UID=0), and authorized to access the BPX.DAEMON RACF facility, if used. The kernel user ID is typically used.

2. Configure parms for starting rlogind as follows:

   – Ensure that the TCPIP.ETC.SERVICES file has active entry as shown in Figure 4-15. This assigns port 513 to the rlogin daemon.
   – Update the inetd configuration file, /etc/inetd.conf, to include the entry for the rlogin daemon.

     • login - The ID of the entry for rlogin; must match TCPIP.ETC.SERVICES.
     • stream tcp - Identifies the daemon socket protocol (this is required).
     • nowait - INETD accepts multiple current connections on behalf of rlogind.
     • OMVSKERN - The user ID under which the rlogin daemon runs.
     • /usr/sbin/rlogind - Pathname of the rlogin daemon program. Sticky bit `on` means that the system actually fetches SYS1.LINKLIB(RLOGIND).
     • Remaining string = parameters for rlogin daemon (see below).

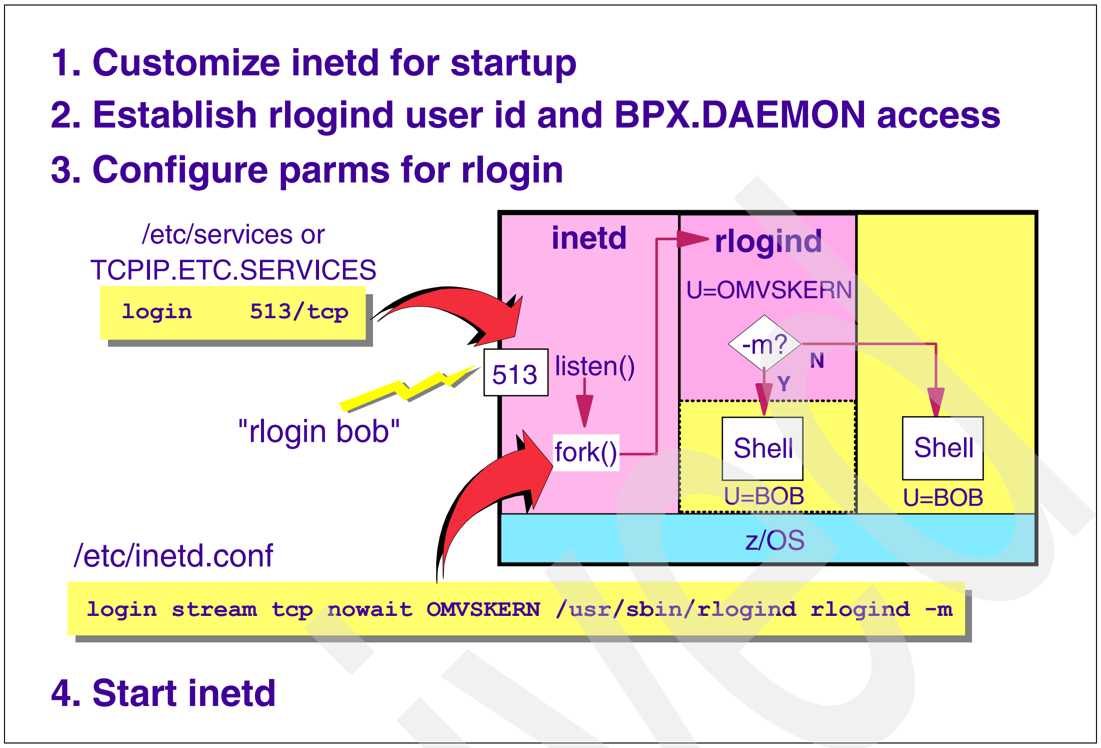
   – Parameters in the rlogind parameter string can include:

     • rlogind - Job name of server process.
     • -m - If specified, the shell process shares address space with the rlogin daemon.
     • -d - Switches on debug - extra messages are written to the system log.

3. To start rlogind support, you need to start the inetd daemon.

4. Let us walk through the process of doing an rlogin:

   – First the inetd daemon starts up, either when the z/OS UNIX kernel is started from the /etc/rc script, or via a **`start`** command and procedure.
   – The inetd daemon reads the configuration file and discovers that it must listen on TCP/IP port 513 for incoming requests for the rlogind daemon (entry login).
   – When an incoming request is received on port 513, inetd BINDS a new socket for the request and then forks an inetd copy in a new address space.
   – The inetd copy sets the job name for the new address space to RLOGIND (from inet.conf parm 7), does **`setuid`** to the user ID for rlogon (OMVSKERN), and then does **`exec ()`** to call the rlogind program. It passes the rest of the argument string from inetd.conf as a parameter.
   – The rlogind daemon uses the supplied socket to contact the client and validate the incoming login request. If the client gives a valid ID, rlogind reads the contents of the OMVS segment for the user ID and allocates a PTY/TTY virtual terminal pair for the session.
   – Then rlogind tests for the -m parameter. If this is supplied, it runs the shell as a child process in the rlogind address space. Otherwise, rlogind forks a new address space and execs the shell in that address space. In either case, the shell runs under the client user ID.



1. Customize inetd for startup
2. Establish rlogind user id and BPX.DAEMON access
3. Configure parms for rlogin

/etc/services or TCPIP.ETC.SERVICES

```
login     513/tcp
```

/etc/inetd.conf

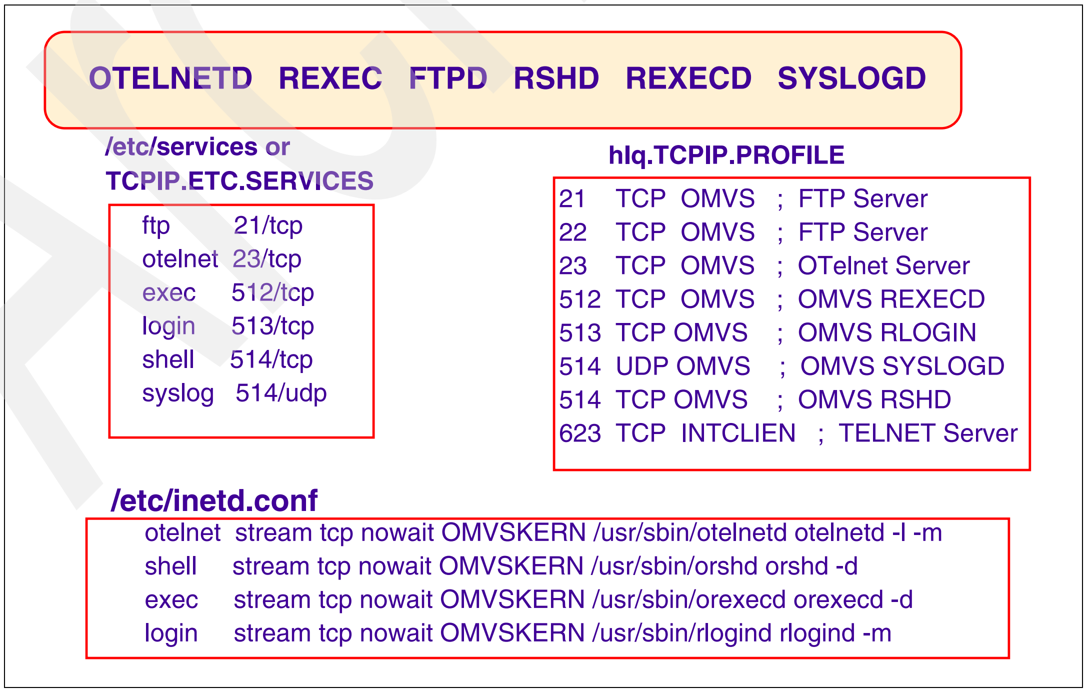
```
login stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
```

4. Start inetd

Figure 4-15 Steps to customize rlogind

4.4 Define TCP/IP daemons

The TCP/IP z/OS UNIX Application feature provides several other TCP/IP functions that you might want to configure, as shown in Figure 4-16.



OTELNETD REXEC FTPD RSHD REXECD SYSLOGD

/etc/services or TCPIP.ETC.SERVICES

```
ftp       21/tcp
otelnet   23/tcp
exec      512/tcp
login     513/tcp
shell     514/tcp
syslog    514/udp
```

hlq.TCPIP.PROFILE

```
21   TCP  OMVS    ; FTP Server
22   TCP  OMVS    ; FTP Server
23   TCP  OMVS    ; OTelnet Server
512  TCP  OMVS    ; OMVS REXECD
513  TCP OMVS     ; OMVS RLOGIN
514  UDP OMVS     ; OMVS SYSLOGD
514  TCP OMVS     ; OMVS RSHD
623  TCP  INTCLIEN  ; TELNET Server
```

/etc/inetd.conf

```
otelnet  stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l -m
shell    stream tcp nowait OMVSKERN /usr/sbin/orshd orshd -d
exec     stream tcp nowait OMVSKERN /usr/sbin/orexecd orexecd -d
login    stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
```

Figure 4-16 TCP/IP daemons

TELNET	Allows remote users to log in to z/OS using a telnet client. The z/OS UNIX telnet server is started for each user by the INETD listener program.
REXEC	Remote execution client and server support the sending and receiving of a command.
FTP	File transfer program supports transfer into and out of the Hierarchical File System.
RSH	Provides remote execution facilities with authentication based on privileged port numbers, user IDs and passwords.
SYSLOGD	Supplies the logging functions for programs that execute in the z/OS UNIX environment.

Ports need to be assigned to the functions that you choose to configure. The hlq.TCPIP.PROFILE data set has an entry for each function and its port and protocol. If you will be configuring both the z/OS UNIX version and the standard TCP/IP version, you will need to decide which one will use the well-known port assignment.

The TCP/IP resolver function also needs to have the port assignments. These can reside in either the TCPIP.ETC.SERVICES data set or the /etc/services file.

Each daemon then has its own configuration information. The inetd program comes with z/OS UNIX and is the listener program for several of the TCP/IP daemons. The commands inetd will use to initiate each program are put in the /etc/inetd.conf file.

The SYSLOG and FTP daemons have their own configuration files, /etc/syslog.conf and /etc/ftpd.data respectively, and each requires a startup procedure.

4.4.1 Syslogd daemon

The syslog daemon (syslogd) is a server process that must be started as one the first processes in your z/OS UNIX environment. z/OS Comminations Server applications and components use syslogd for logging purposes and can also send trace information to syslogd. Servers on the local system use AF_UNIX sockets to communicate with syslogd, while remote servers use the AF_INET socket.

The syslogd daemon reads and logs system messages to the MVS console, log files, other machines, or users, as specified by the configuration file /etc/syslog.conf. A sample is provided in /usr/lpp/tcpip/samples/syslog.conf.

If the syslog daemon is not started, the application log may appear on the MVS console.

The syslog daemon must have a user ID; for example, SYSLOGD defined in RACF with UID=0. The syslogd daemon uses the following files:

/dev/console	Operator console
/etc/syslog.pid	Location of the process ID
/etc/syslog.conf	Default configuration file
/dev/log	Default log path for z/OS UNIX datagram socket
/usr/sbin/syslog	Syslog server

If you want syslogd to receive log data from or send log data to remote syslogd servers, reserve UDP port 514 for the syslogd job in your PROFILE.TCP/IP data set and enter the syslog service for UDP port 514 in the services file or data set (for example, /etc/services).

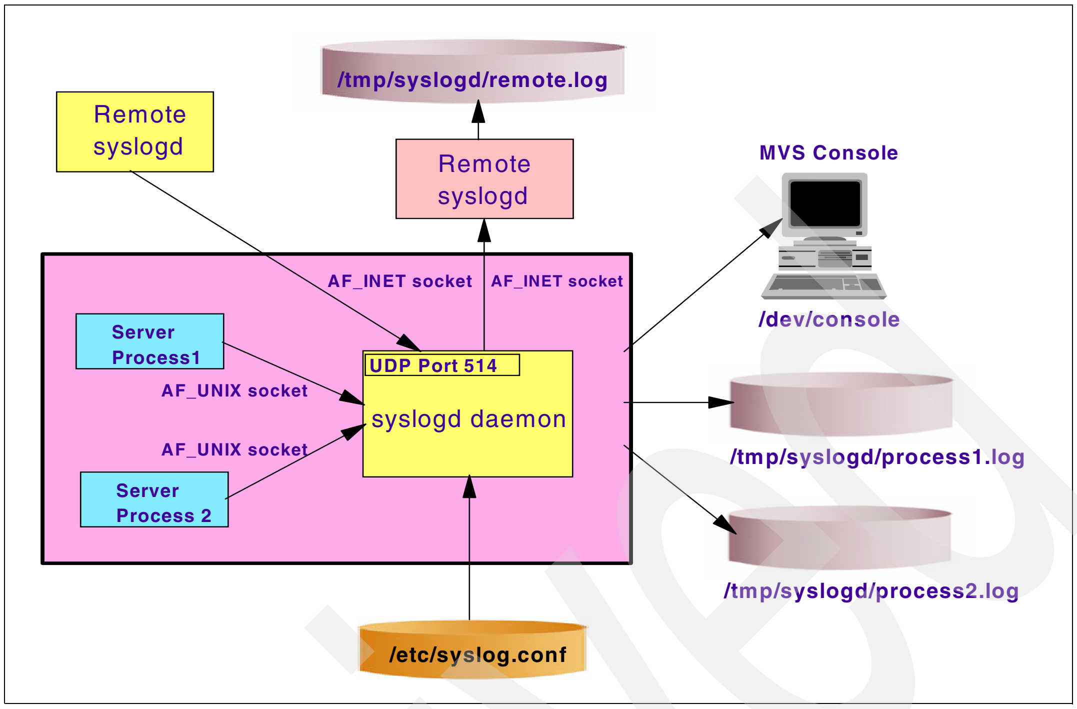


Figure 4-17 Syslogd daemon

Note: Syslogd can only be started by a superuser and can be terminated using the SIGTERM signal.

4.4.2 Otelnetd daemon

The telnet server is used to enable remote telnet clients to log on to the z/OS UNIX shell environment in either raw mode (also called character mode) or line mode.

Setting up otelnetd

Figure 4-18 on page 209 illustrates the steps required to set up otelnetd.

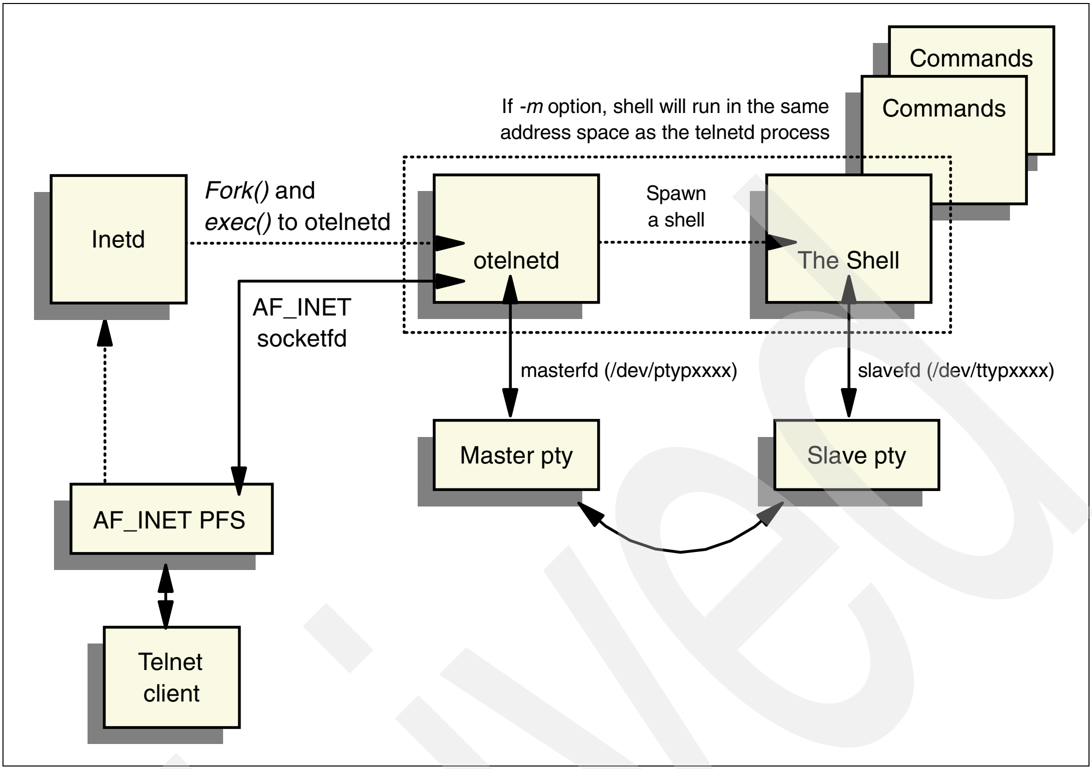


Figure 4-18 FTPD overview

Customize the number of pseudoterminal files

The pseudoterminal files are created in the HFS in the /dev directory. The file names are ptypnnnn and ttypnnnn, where nnnn is a number from 0000 to 9999. These files are used in pairs by the telnetd server and the rlogind server.

You can allocate the pseudoterminal files during z/OS UNIX Application Service installation by running BPXISMKD from the SYS1.SAMPLIB. The parameter MAXPTYS determines the maximum number of pseudoterminal files to be allocated. You may modify the EXEC to allocate extra pseudoterminals afterward (see Figure 4-19). The EXEC should be executed by a superuser.

```
/* REXX */
 MAXPTYS = 255         /* new maximun number */
 PTYSTART = 51         /* first new ptyp and ttyp */
 call syscalls('On")

 Do count = PTYSTART to MAXPTYS
  ptyscnt = $root"dev/ptyp"Right(count,4,0)"'"
  ttyscnt = $root"dev/ttyp"Right(count,4,0)"'"
  call syscallm mknod ptyscnt "666 1" count
  call syscallm mknod ttyscnt "666 2" count
 End
```

Figure 4-19 Sample REXX exec to increase the number of pseudoterminals

After you have increased the number of pseudoterminal files, you should update the `MAXPTYS` statement in the BPXPRMxx parmlib member:

```
MAXPTYS(256)
```

The value of MAXPTYS should be one more than the highest pseudoterminal number you created, because the terminal number starts from 0000.

Starting the z/OS UNIX telnet server

The telnetd server is started via inetd.

The default port number for the telnetd server is 23. This is a well-known port number, and you can reserve the port to z/OS Communication Server in the PROFILE data set:

```
23 TCP OMVS ;  UNIX System Services Telnet Server
```

If the default port number 23 is used, a client has to know only the name or IP address of the server to establish a connection and can use a command such as:

```
telnet 9.24.104.43
```

It is also possible to reserve a different port for the z/OS UNIX telnet server in the PROFILE data set of z/OS Communication Server, for example:

```
2023 TCP OMVS ;  UNIX System Services Telnet Server
```

In this case, a client has to specify both the host name (or IP address) and the port number of the server with the **`telnet`** login command, as follows:

```
telnet 9.24.104.43 2023
```

If you assign an alternate port number to your z/OS UNIX telnet server, you also need to update your /etc/services configuration file with the chosen port number in order for inetd to listen for telnet client requests on the chosen port:

```
telnet        2023/tcp
```

Since inetd is the listener for telnetd, you have to customize the /etc/inetd.conf file so that it can fork() the telnetd upon request. You should uncomment the following statement:

```
#======================================================================
# service | socket | protocol | wait/ | user | server  | server program
# name    | type   |          | nowait|      | program |   arguments
#======================================================================
#
   :
otelnet  stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l -m
   :
```

If your configuration has more than one z/OS Communication Server for a z/OS IP stack running on one z/OS image, all of these stacks must have identical port reservations for the z/OS UNIX telnet server. The chosen port number is a system-wide value in the z/OS UNIX environment. For more information on running multiple stacks, see *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 1: Base and TN3270 Configuration,* SG24-5227.

A telnetd server process is forked from inetd whenever a telnet client connects to z/OS UNIX System Services. See Figure 4-18 on page 209 for an overview of how telnetd operates in the z/OS UNIX environment.

4.4.3 REXECD and RSHD servers

Both the REXECD server and the RSHD server are used to execute z/OS UNIX shell commands from remote users. Figure 4-20 shows an overview of how both servers are implemented in the z/OS UNIX environment.

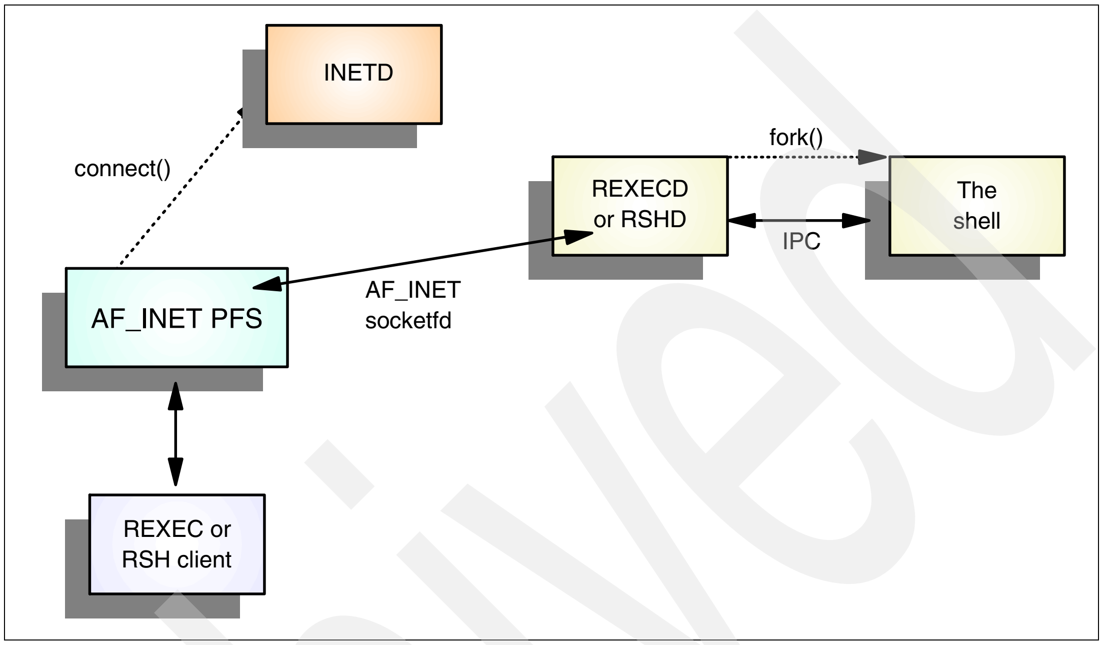


Figure 4-20 z/OS REXECD and RSHD implementation overview

For each remote request, inetd forks a new process with either REXECD or RSHD. The REXECD or RSHD server in turn forks a shell process with which it communicates via pipes.

The default port numbers for the two servers are 512 and 514, and both must be reserved in the PROFILE data sets of the TCP/IP stacks that act as z/OS UNIX AF_INET transport providers:

```
PORT
   512   TCP   OMVS       ;REMOTE EXECUTION SERVER
   514   TCP   OMVS       ;REMOTE SHELL SERVER
```

Customize the /etc/inetd.conf file. Figure 4-21 shows a part of the file in our test system:

```
#
#======================================================================
# service | socket | protocol | wait/ | user | server  | server program
# name    | type   |          | nowait|      | program |   arguments
#======================================================================
shell    stream tcp nowait OMVSKERN /usr/sbin/orshd orshd -LV
exec     stream tcp nowait OMVSKERN /usr/sbin/orexecd orexecd -l -v
#
```

Figure 4-21 Sample /etc/inetd.conf file for REXECD and RSHD servers

There are a few situations where the RSHD server may encounter an error so early in the processing of a command that it has not established a proper EBCDIC-to-ASCII translation yet. In such a situation, the client end user may see “garbage” data returned to his or her terminal. A packet trace will reveal that the response is in fact returned in EBCDIC, which is the reason for the garbage look on an ASCII workstation. We have seen this happen if the z/OS UNIX name resolution was not configured correctly, so the RSHD server, for example, was not able to resolve IP addresses and host names correctly. If your RSH clients encounter such a problem, then go back and check your name resolution setup. If you are using a local hosts table, make sure that the syntax of the entries in your hosts file is correct.

We also saw an authentication error during our tests. The REXEC server, for example, needs to be associated with a user who has READ authority to the BPX.DAEMON facility class. Otherwise your REXEC client's request will fail. For more information on the BPX.DAEMON facility, refer to *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 1: Base and TN3270 Configuration,* SG24-5227.

4.4.4 FTPD daemon

File Transfer Protocol (FTP) is used to transfer files between TCP/IP hosts. The FTP client is the TCP/IP host that initiates the FTP session, while the FTP server is the TCP/IP host to which the client connects; see Figure 4-22.

The FTP server uses two different ports and manages two TCP connections as follows:

- Port 21 is used to control the connection (user ID and password).
- Port 20 is used for actual data transfer based on the FTP client's requests.

The FTP server in z/OS IP consists of the daemon (the listener) or ftpd and server address space (or processes). The daemon performs initialization, listens for new connections and starts a separate server address space for each connection.

When a new client FTP-connects to the FTPD daemon process, ftpd forks an FTP server process; thus, a new job name is generated by z/OS UNIX.

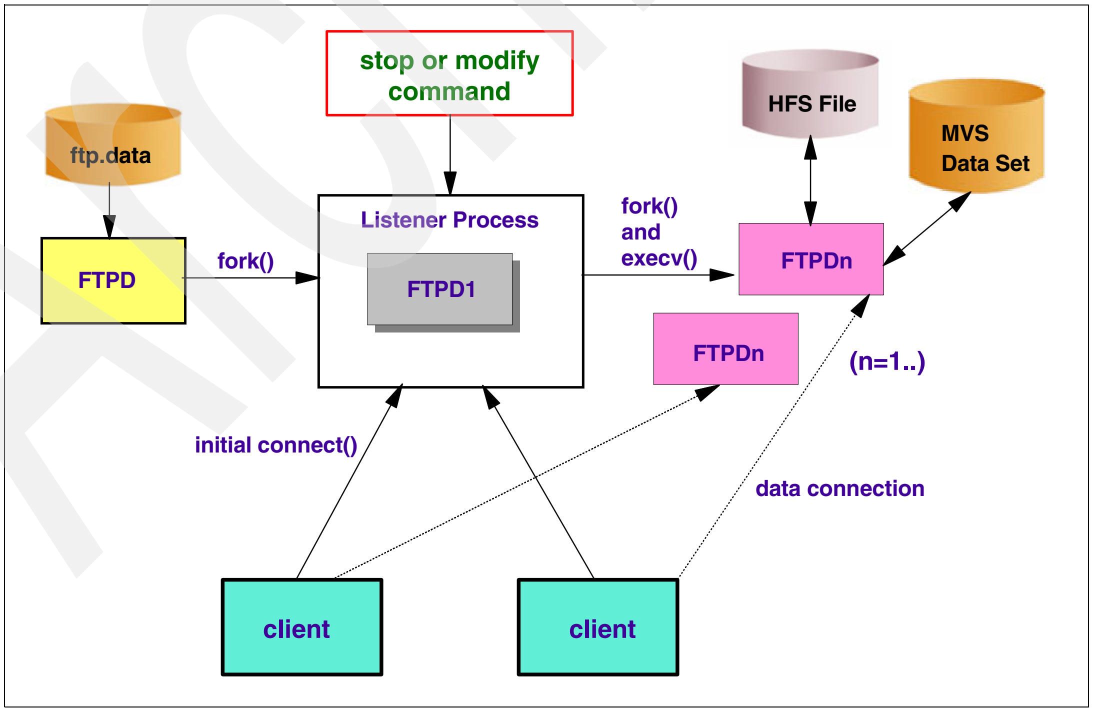


Figure 4-22 Process flow of the z/OS UNIX FTP server

z/OS IP search order - FTP

FTP.DATA is used to override the default FTP client and server parameters for the FTP server.

You may not need to specify the FTP.DATA data set if the default parameters are used.

A sample is provided in hlq.SEZAINST(FTPDATA) for the client and hlq.SEZAINST(FTPSDATA) for the server.

When an FTPD daemon or started task is started, it searches the FTP.DATA file in the following order (see Figure 4-23):

1. //SYSFTPD DD in FTPD started task procedure
2. userid/jobname.FTP.DATA
3. `/etc/ftp.data`
4. SYS1.TCPPARMS(FTPDATA)
5. hlq.FTP.DATA

Note: The search stops if one of these data sets is found.

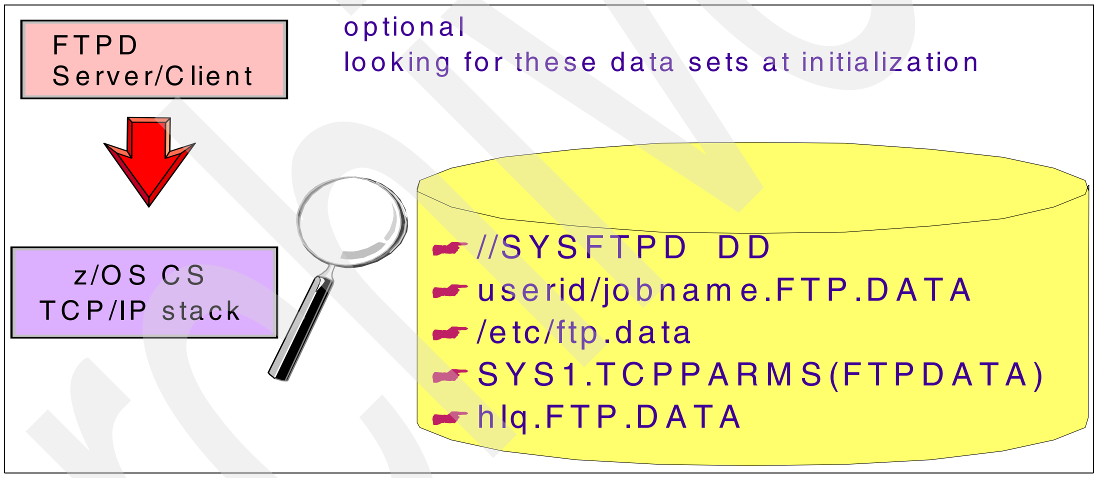


Figure 4-23 z/OS IP search order - FTP

z/OS IP search order - services

The ETC.SERVICES data set is used to establish port numbers for UNIX application servers using TCP and UDP. This file or data set is required for any daemon or application that needs the use of a specific port.

Standard applications, like telnet or FTP, are assigned port numbers inside the well-known port number range. You can assign port numbers to your own server applications by adding entries to the /etc/services file.

For example, rlogind listens on 513/TCP and telnetd listens on port 23/TCP, while syslogd listens on port 514/UDP. This specification is provided in the ETC.SERVICES data set.

When TCP/IP and the daemons start, they look for the ETC.SERVICE file or data set in the following order (see Figure 4-24 on page 214):

- /etc/services (HFS file)
- userid/jobname.ETC.SERVICES

- hlq.ETC.SERVICES

The search stops if one of these data sets is found.

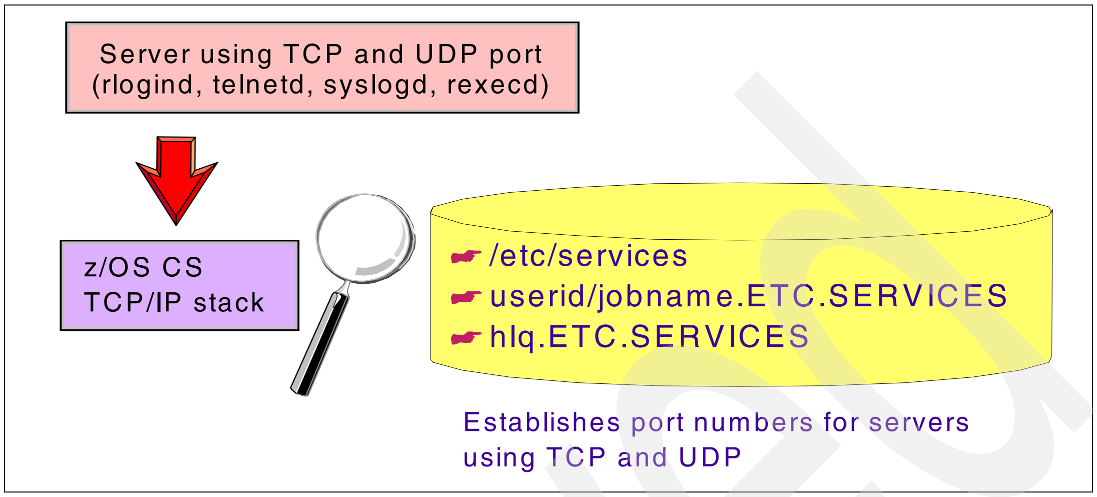

Server using TCP and UDP port
(rlogind, telnetd, syslogd, rexecd)
z/OS CS
TCP/IP stack
/etc/services
userid/jobname.ETC.SERVICES
hlq.ETC.SERVICES
Establishes port numbers for servers
using TCP and UDP

Figure 4-24 z/OS search order - services

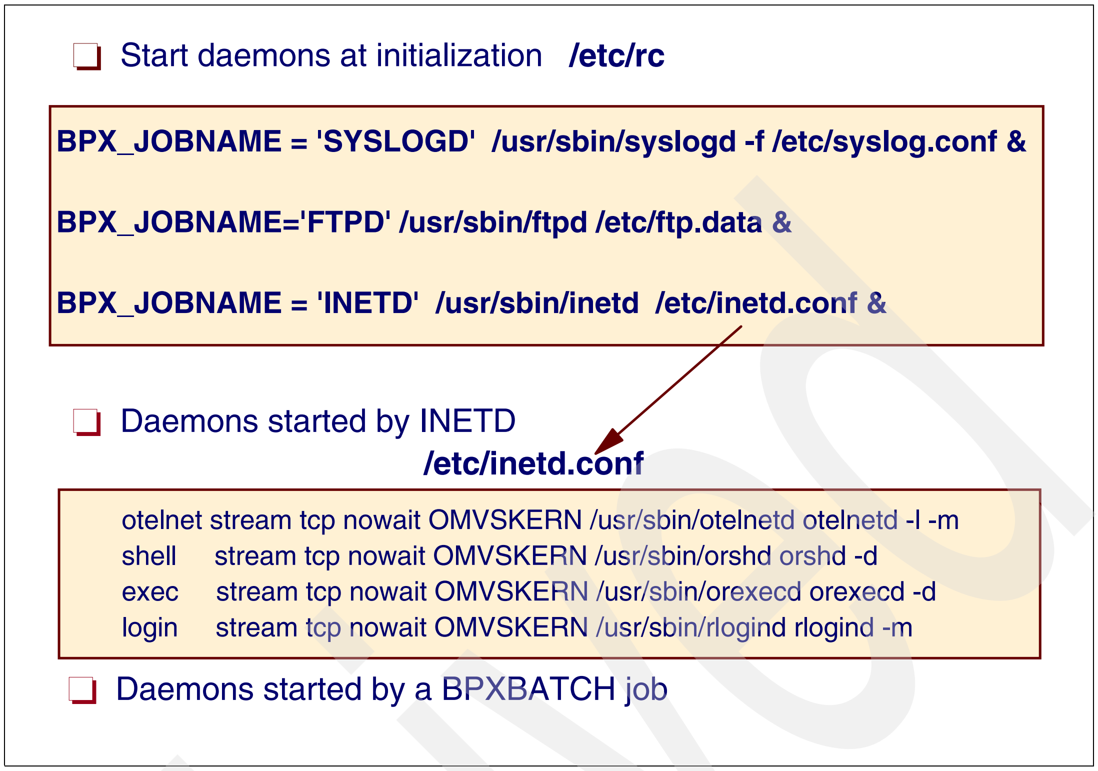
4.4.5 Start daemons

After the configuration files have been completed, the daemons need to be started before any remote requests can be processed.

The /etc/rc script is a good place to put the start command, as Figure 4-25 on page 215 shows. In this case, the daemons will be started during the initialization processing for z/OS UNIX. The _BPX_JOBNAME environment variable will give the daemon an MVS job name.

Since inetd is responsible for starting the other daemons (telnet, rlogin, remote shell, and remote execution), start commands for them are in inetd's configuration file.

In case any of these daemons fail, you should have other procedures created to restart them since /etc/rc is only used at z/OS UNIX initialization. You could use shell scripts or MVS procedures for this.


❑ Start daemons at initialization /etc/rc

```
BPX_JOBNAME = 'SYSLOGD'  /usr/sbin/syslogd -f /etc/syslog.conf &

BPX_JOBNAME='FTPD' /usr/sbin/ftpd /etc/ftp.data &

BPX_JOBNAME = 'INETD'  /usr/sbin/inetd  /etc/inetd.conf &
```

❑ Daemons started by INETD

/etc/inetd.conf

```
otelnet stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l -m
shell    stream tcp nowait OMVSKERN /usr/sbin/orshd orshd -d
exec     stream tcp nowait OMVSKERN /usr/sbin/orexecd orexecd -d
login    stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
```

❑ Daemons started by a BPXBATCH job

Figure 4-25 Start the syslogd, ftpd, and inetd daemons

4.5 SMTP server

The Simple Mail Transfer Protocol (SMTP) is a TCP/IP application that is used to transport electronic mail. Electronic mail enables you to send notes, messages, letters, or correspondence to others on the network. It is similar to sending a letter through the post office. You compose the message just as you would an ordinary letter, address the letter to one or more people and possibly carbon copy others. You enclose copies of the letter in envelopes, address them to the recipients, and give them to the delivery system. You expect the mail to be delivered to the correct address available for pickup when the recipient is ready. And you want any undeliverable mail returned to you. You can even keep a log of the mail you send and receive. The following commands are available to let you send and receive mail:

The simple mail architecture defines Mail User Agent (MUA), Mail Transfer Agent (MTA), and Mail Delivery Agent (MDA).

- The MUA is any of various offered programs like the original UNIX mail program (/bin/mail) or the Berkeley MAIL program or the Netscape Communicator, etc. which a user runs to compose, dispose, read and reply to e-mail notes.
- The MTA is software that sends the prepared note by the MUA to a remote MTA responsible for the recipient using an SMTP connection.

  The *sendmail* program is an MTA on the sending and on the receiving side.
- Finally, the SMTP server uses a local mailer program (for example /bin/mail) to deliver the note to a mail spool file by appending the note to this file. SMTP server now has finished its work.
- The user (MUA) may now retrieve his mail from the spool file.

- Another approach, more common today, is that a separate server, a popper server running the POP3 protocol, is used to retrieve notes from the mail spool file. This can be done for example through a Netscape Communicator working as a POP3 client invoking the popper server. A prerequisite, however, is that the MUA supports the POP3 protocol.

  The *popper* is an MDA using the POP3 protocol for the transport between MDA and MUA.

  Figure 4-26 shows how it works.

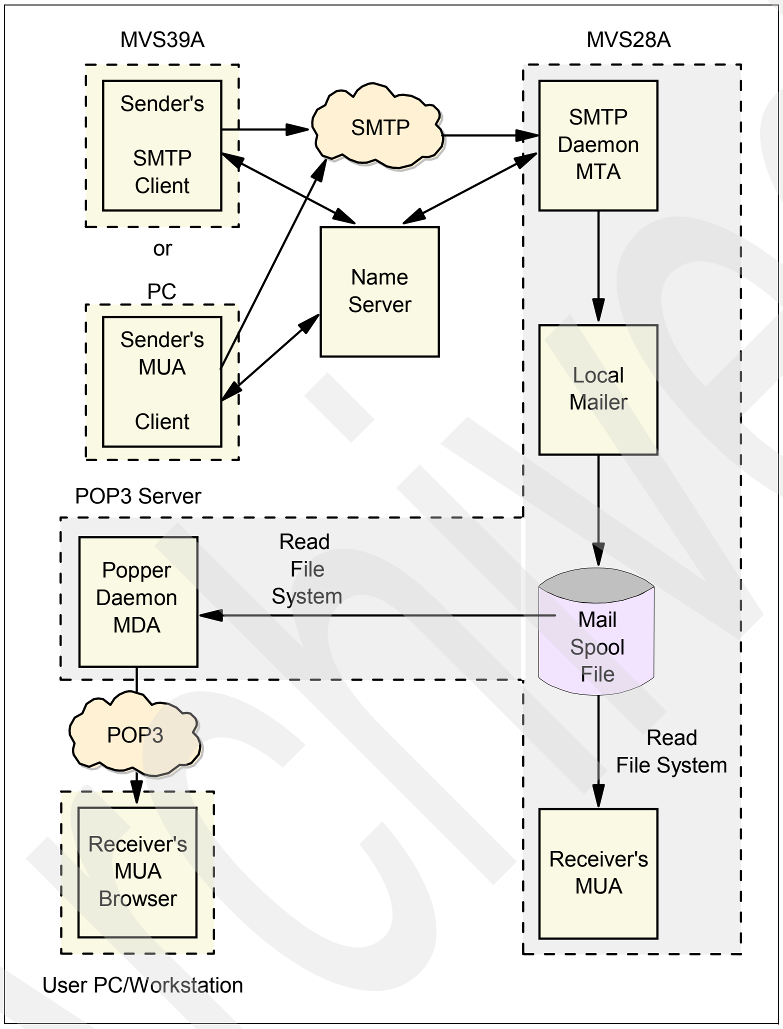


Figure 4-26 Relationship between MUA, MTA, and MDA

For implementation and customization of an SMTP server, see *z/OS Communications Server IP Configuration Guide,* SC31-8775.

4.6 Sending e-mail using SMTP commands

SMTP mail can be sent and received interactively over a TCP/IP network. Mail from TCP/IP network sites destined for local MVS users (or users on an NJE network attached to the local MVS system) arrives over this interface. All commands and data received and transmitted through this interface use ASCII characters.

Interface from the JES spool, including any connected NJE nodes. SMTP commands can be written into a SYSOUT data set, with an external writer name of the SMTP address space.

SMTP processes each of the commands in the data set in sequence, exactly as if it had been transmitted over a TCP/IP connection. This is how mail is sent from local MVS users to recipients on the TCP network. Batch SMTP data sets must contain commands and data in EBCDIC characters.

For a description of batch SMTP in TSO utilities, see *z/OS Communications Server IP Configuration Guide,* SC31-8775.

To send mail to a TCP network recipient, see the batch SMTP commands as inline input for SYSUT1 and SYSUT2, create the following JCL using the IEBGENER utility on the TSO/ISPF application shown in the Figure 4-27.

```
//BATSMTP  JOB (userid,nn),MSGCLASS=B,PRTY=12,MSGLEVEL=(2,1)
//IEBGENER EXEC PGM=IEBGENER
//SYSIN DD DUMMY
//SYSUT1 DD *
HELO YOURMVS
MAIL FROM:<LIVIO@YOURMVS>
RCPT TO:<msgs@rsch.our.edu>
RCPT TO:<alice@ai.our.edu>
DATA
Date: Thur, 26 Jul 03 21:48:57 EST
From: Livio <LIVIO@YOURMVS>
To:   <msgs@rsch.your.edu>
Cc:   <alice@ai.your.edu>
Subject: update

Mike: Cindy stubbed her toe.  Bobby went to
baseball camp.  Rebecca made the cheerleading team.
Jan got glasses.  Peter has an identity crisis.
Greg made dates with 3 girls and couldn't
remember their names.
.
QUIT
/*
//SYSUT2 DD SYSOUT=(B,smtp)
//*                |   v
//*                v  SMTP address space name for external writer
//*               SYSOUT class
//SYSOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
```

Figure 4-27 Sample of sending mail from an NJE network host

4.7 Customizing and starting NFS

Client systems in a TCP/IP network that support the NFS client protocol can use traditional MVS data sets and z/OS UNIX HFS files as part of their file system. The z/OS NFS server uses OE-Sockets and supports both HFS and traditional MVS data sets.

4.7.1 Configuring the z/OS NFS client

This section describes how to configure the z/OS NFS client.

During z/OS UNIX file system initialization, the z/OS NFS client is started and run in the logical file system (LFS) colony address space. The filesystype parmlib statement for the z/OS NFS client must be present in the SYS1.PARMLIB(BPXPRMxx) parmlib member in order to start the z/OS NFS client. For more information on z/OS UNIX file system reference, see *z/OS UNIX System Services File System Interface Reference,* SA22-7808.

Updating MVS system data sets for the client

Start TCP/IP and portmap. Wait until this message appears:

```
MVPOED0001I OPENEDITION-TCP/IP connection established
```

To accommodate the z/OS NFS client you must update MVS system data sets parmlib and proclib, and the DD statement.

Parmlib updates

Add the data set defined in the GFSCPROC STEPLIB containing the z/OS NFS client library to the system's APF authorization list (IEAAPFxx). A sample cataloged procedure named GFSCPROC is provided as a member of the sample library NFSSAMP; see Figure 4-28.

```
//MVSNFSC PROC SYSNFS=SYS1,SYSLE=SYS1,NFSPRFX=MVSCLNT,TCPIP=TCPIP
//MVSCLNT EXEC PGM=BPXVCLNY,
// REGION=0M,
// TIME=1440
//SYSTCPD DD DISP=SHR,DSN=&TCPIP..TCPIP.DATA
//STEPLIB DD DISP=SHR,DSN=&SYSNFS..NFSLIB
// DD DISP=SHR,DSN=&SYSLE..SCEERUN
//*
//SYSPRINT DD SYSOUT=*
//SYSMDUMP DD DISP=SHR,DSN=&NFSPRFX..SYSMDUMP
//OUTPUT DD SYSOUT=*
//SYSERR DD SYSOUT=*
//NFSCMSG1 DD DISP=SHR,DSN=&NFSPRFX..LOG1
//NFSCMSG2 DD DISP=SHR,DSN=&NFSPRFX..LOG2
```

Figure 4-28 Sample z/OS NFS client startup procedute

Add the filesystype parmlib statement shown in Figure 4-29 to be the z/OS parmlib member (BPXPRMxx:)

```
FILESYSTYPE
   TYPE(NFS)
   ENTRYPOINT(GFSCINIT)
   PARM('installation parms')
   ASNAME(proc_name)
```

Figure 4-29 Sample filesystype parmlib statement

Note: The `proc_name` is also used for the name of the address space.

For data integrity and data isolation among different PFSs, the z/OS NFS client is required to start in a separate and standalone colony address space. To do that, a unique proc_name must be used.

For information about BSAM, QSAM and VSAM ESDS access to remote files, application access to HFS or remote files and their restrictions, see *Network File System Customization and Operation,* SC26-7417 and *z/OS DFSMS Using Data Sets,* SC26-7410.

Mounting remote file systems

z/OS UNIX does not support z/OS NFS mounts in the SYS1.PARMLIB member statement. You can use the z/OS UNIX **`automount`** facility (/etc/rc shell script support) or the TSO MOUNT command to make a connection between a mount point on your local MVS HFS file system and one or more files on a remote MVS, AIX®, UNIX, OS/390, z/OS, Linux, or other file system. The remote file system can be mounted using the TSO MOUNT command, which can only be used by an MVS superuser. For additional information about the TSO MOUNT command, when used with a z/OS NFS client, see *DFSMS/MVS Network File System Users Guide,* SC26-7419.

The remote file system must be mounted on the z/OS UNIX file system prior to any reference to the remote data. Once mounted, the remote file system can be treated as an extension of the local z/OS UNIX file system.

4.7.2 Configuring the z/OS NFS server

This section describes how to configure the z/OS NFS server.

Enable portmap

NFS uses the TCP/IP RCP protocol for client-server communication. This is why the portmapper function has to be enabled. If not already done, copy the sample PORTMAP procedure from hlq.SEZAINST(PORTPROC) to your procedure library. PORTMAP needs read access to the hlq.ETC.RPC file. Copy it from the hlq.SEZAINST(ETCRPC) file.

Add the entries in the hlq.ETC.RPC file for the service provided by the NFS server; see Figure 4-30.

```
Service       Number       Description
-------       ------       ------------
nfs           100003       NFS daemon
mountd        100005       Mount daemon
mvsmount      100044       daemon for mvslogin and mvslogout
showattrd     100059       showattr daemon
pcnfsd        150001       pcnfs deamon
nlm           100021       network lock manager
nsm           100024       network status monitor
```

Figure 4-30 Modifying the hlq.ETC.RPC file

Allocating the attributes, checklist, and exports data sets

Copy the members from SYS1.NFSSAMP and rename them. Update the EXPORT member with the available data sets and directories in NFS. After that, add these members into your start procedure for the NFS server; see Figure 4-31 on page 220.

```
//COPY1   EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//INDS    DD DSN=SYS1.NFSSAMP,DISP=SHR
//OUTDS   DD DSN=SAMPLE.NFS.CONTROL,DISP=(,CATLG),
//           SPACE=(TRK,(2,2,10)),UNIT=3390,
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=27920,DSORG=PO)
//SYSIN   DD *
 COPY OUTDD=OUTDS,INDD=INDS
  SELECT MEMBER=((GFSAPATT,NFSATTR))
  SELECT MEMBER=((GFSAPCHK,CHKLIST))
  SELECT MEMBER=((GFSAPEXP,EXPORTS))
/*
```

Figure 4-31 Sample JCL to allocate NFSSAMP data sets

Updating the exports data set

The exports data set contains entries for directories that can be exported to clients. It is used by the server to determine which data sets' high-level qualifiers or HFS directories can be mounted by a client in a read or write mode.

Modify the sample exports data set to suit your installation as shows Figure 4-32. For security considerations and for the assignment of UIDs and GIDs in your NFS network, see 4.7.3, "Security settings for the z/OS NFS environment" on page 223. We used EXPORTS security for z/OS UNIX HFS files.

```
  mvsnfs             -ro                       # give read-only access
                                               # to all clients
  robert.mixds -rw=fsrs001:fslab004:fslab007 #
                                               # give read/write access
                                               # to the clients named
                                               # fsrs001, fslab004 and
                                               # fslab007, and give
                                               # read-only access to
                                               # all other clients
```

Figure 4-32 Sample of exports data set

Note: You cannot specify exporting a "parent directory" or a subdirectory of an exported directory. For example, if you specify `DIR1` in the exports data set, `DIR1` and all its subdirectories are exported. You cannot specify any subdirectories under `DIR1` in the exports data set.

Allocating the mount handle data sets

The mount handle data sets are used to record active mounts during server operation and allow clients to stay mounted when the server is shut down and restarted. The Network File System alternates between two data sets to record this information; only one data set is used at a time, and it is switched at either shutdown or at resource cleanup timeout.

Allocate the mount handle data sets. You can find sample JCL in hlq.NFSSAMP(GFSAMHDJ). We provide a sample for you in Figure 4-33 on page 221.

```
//DEF1  EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSIN   DD *
 DEFINE CL (NAME(SAMPLE.FHDBASE) -
  CYL(1,1) -
  VOL(SBOX77) -
  INDEXED  -
  REUSE    -
  KEYS(16 0) -
  SHAREOPTIONS(3 3) -
  RECSZ(400 1600))
 LISTC ENT(SAMPLE.FHDBASE) ALL
/*
//DEF2  EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSIN   DD *
 DEFINE CL (NAME(SAMPLE.FHDBASE2) -
  CYL(1,1) -
  VOL(SBOX77) -
  INDEXED  -
  REUSE    -
  KEYS(16 0) -
  SHAREOPTIONS(3 3) -
  RECSZ(400 1600))
 LISTC ENT(SAMPLE.FHDBASE2) ALL
/*
```

Figure 4-33 Sample JCL to create a mount handle data set

Note: Delete and allocate the mount handle data sets before running any new versions of the NFS. If an old mount handle data set is used, the server issues a message and shuts down.

Customizing the translation table for NFS

For text processing mode, data is converted between EBCDIC and ASCII. No double-byte character set (DBCS) or multiple-byte character set (MBCS) forms of data are converted.

You can customize the translation table for the Network File System using the processing attribute xlat(member_name). The parameter (member_name) is the member name of a PDS or PDSE containing the customized translation table. This attribute can be specified either in the mount operation or in the attribute file. It can be specified on a file operation but is ignored. Only the mount or xlat values take effect. We have provided a sample in Figure 4-34 on page 222.

```
//STEP1    EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 CONVXLAT 'TCPIP.SEZATCPX(STANDARD)' 'SAMPLE.NFS.TEST'
/*
//STEP2   EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1  DD DISP=(OLD,DELETE),DSN=SAMPLE.NFS.TEST
//SYSUT2  DD DISP=(NEW,CATLG),DSN=SAMPLE.NFS.XLAT.NEW,
//            DCB=(RECFM=F,LRECL=256,BLKSIZE=256),
//            SPACE=(TRK,(1,2,5)),UNIT=3390
//SYSIN   DD *
 GENERATE MAXNAME=1
 MEMBER NAME=STANDARD
/*
```

Figure 4-34 Sample JCL to customize a translation table

Note: See “Using Translation Tables” in *z/OS Communications Server IP Configuration Reference*, *z/OS Communications Server IP Configuration Reference,* SC31-8776, for more information about creating and customizing your own translation tables.

Customize the MVSNFS procedure

Copy the sample MVSNFS procedure from hlq.NFSSAMP(GFSAPROC) to your procedure library and customize it. Assign a user ID to the NFS procedure (we used the same user ID as TCPIP) with an OMVS segment.

Update the following MVS system data sets to accommodate the z/OS NFS server.

Add the DD statements:

- EXPORTS as the DD for the exports data set
- NFSATTR as the DD for the attributes data set
- FHDBASE and FHDBASE2 as the DD for the mount handle data set
- NFSXLAT as the DD to enable the xlat processing attribute
- NFSLOG1 as the DD for the primary log data set
- NFSLOG2 as the DD for the secondary log data set
- SYSxDUMP as the DD for the SYSxDUMP data set ('x' = U or M)

We have provided sample z/OS NFS server startup procedures in Figure 4-35 on page 223.

```
//NFSMVS  PROC MODULE=GFSAMAIN,
//           SYSNFS=SYS1,
//           NFSPRFX=OS390NFS,
//           TCPIP=TCP,
//           TCPDATA=TCPDATA
//GFSAMAIN EXEC PGM=&MODULE,
//             REGION=0M,
//             TIME=1440,
//         PARM=(,
//         'ENVAR("_BPXK_SETIBMOPT_TRANSPORT=TCPIP")/')
//SYSTCPD  DD   DISP=SHR,DSN=&TCPIP..&SYSNAME..TCPPARMS(&TCPDATA.)
//STEPLIB  DD   DISP=SHR,DSN=&SYSNFS..NFSLIB
//SYSPRINT DD   SYSOUT=*
//OUTPUT   DD   SYSOUT=*
//SYSERR   DD   SYSOUT=*
//NFSATTR  DD   DISP=SHR,DSN=&NFSPRFX..&SYSNAME.MVS.PARMS(ATTRIB)
//NFSLOG1  DD   DISP=SHR,DSN=&NFSPRFX..&SYSNAME.MVS.SERVER.LOG1
//NFSLOG2  DD   DISP=SHR,DSN=&NFSPRFX..&SYSNAME.MVS.SERVER.LOG2
//FHDBASE  DD   DISP=SHR,DSN=&NFSPRFX..&SYSNAME.MVS.FHDBASE1
//FHDBASE2 DD   DISP=SHR,DSN=&NFSPRFX..&SYSNAME.MVS.FHDBASE2
//NFSXLAT  DD   DISP=SHR,DSN=&NFSPRFX..&SYSNAME.MVS.XLAT
```

Figure 4-35 Sample NFS server startup procedure

MVSLOGIN and MVSLOGOUT

If you specified security(SAF) or security(SAFEXP) in your NFS attributes file, you must download the **mvslogin** and **mvslogout** client commands to the NFS client system. See *Network File System Customization and Operation,* SC26-7417 for further information.

Access a z/OS UNIX HFS

You should now be able to access a z/OS UNIX HFS from an NFS client. Figure 4-36 shows the command sequence to be used to mount a z/OS UNIX HFS directory from a Linux system. We used EXPORTS security, so we did not use the **mvslogin** command.

```
# showmount -e 9.12.6.9
Export list for 9.12.6.9:
/ (everyone)
clinuxa:~ # mount 9.12.6.9:/ /mnt/livio
clinuxa:~ # df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/dasda1           2.2G  2.1G   23M  99% /
/dev/dasdb1           2.3G  2.1G   31M  99% /mnt/dasdb
9.12.6.9:/            117M   59M   59M  50% /mnt/livio
```

Figure 4-36 Command sequence to mount an HFS directory from a Linux system

As previously mentioned, we used EXPORTS security, so we did not use the **mvslogin** command. For more information on NFS, see *DFSMS/MVS Network File System Users Guide,* SC26-7419.

4.7.3 Security settings for the z/OS NFS environment

Mapping of UNIX security and MVS security is always cumbersome, especially if NFS is involved. This section provides customization hints to do this more easily and efficiently.

MVS NFS security levels

Four different security levels can be selected in the MVS NFS attributes data set:

NONE	No security checking is performed.
EXPORTS	The EXPORTS file is used to check security.
SAF	The System Authorization Facility checking is performed. RACF provides the security information.
SAFEXP	Both SAF and EXPORTS file checks are performed.

For most MVS installations, selecting security(NONE) is unacceptable, so security checking has to be done by RACF using the MVS SAF support, or by MVS NFS using the EXPORTS file, or via both mechanisms.

EXPORTS security implements the way security checking is usually done by UNIX systems with NFS implementations. A parameter file (on MVS, known by the DD name EXPORTS, and on UNIX systems, in most cases known as the /etc/exports file), contains the names of directories that are available for NFS requests. This parameter file also contains the host names of those NFS client systems that have permission to mount these directories for read and/or write purposes. The parameter file does not contain any user ID associated with the NFS mount, but it does contain the IP host name of the NFS client.

SAF security implements a more MVS-like approach. It requires the NFS client to provide a valid RACF user ID and password to z/OS NFS. The access to MVS data sets and HFS files is checked by RACF.

SAFEXP security requires that both security checks (SAF and EXPORTS) grant access to the files.

Security information exchange between NFS client and server

This section describes how an IP host name and a RACF user ID are provided to MVS NFS.

- How is the client IP host name resolved by the NFS server?

  If an NFS client connects to the NFS server, only the IP address is transmitted over the network. On the NFS server side, the client IP address is resolved into a client host name by using either local hosts tables or a query to a local or remote name server. It should be noted that in most implementations it is fairly easy to use a fake IP address on the client side.

- How is a RACF user ID provided to NFS?

  – For single user PC systems, this is usually not a simple task.

    Most PC systems support an extension to the NFS protocol called PC-NFS client. MVS/ESA has, like many NFS implementations, a PC-NFS server. PC-NFS support is enabled by specifying the pcnfsd verb in the NFS attributes data set. If PC-NFS support is enabled on the MVS/ESA NFS server, after a `mount` command is received from the NFS client, the server prompts the client asking for a user ID and password. Both are checked by RACF.

  – The problem arises with UNIX systems, which usually do not provide a PC-NFS client, because they are by design multi-user systems.

    In most cases on a UNIX system, users must have superuser authority (usually called root authority) to issue a `mount` command. Because standard mount processing does not provide user ID and password checking, NFS running on MVS/ESA has implemented a protocol extension to mount called mvsmount.

The mvsmount protocol extension is implemented by the commands **mvslogin** and **mvslogout**, which allow the entry of a RACF user ID and password on several UNIX platforms.

Access to the HFS

HFS security is based on permission bits associated with an HFS file, UID and GID values associated with the file, and the requesting RACF user ID.

A UID associated with a user is a number specified in the OMVS segment of a RACF user ID. A GID associated with a user is a number specified in the OMVS segment of the default RACF group to which the user belongs.

Permission bits specify whether read, write, or execute permission is granted to the file owner, the group to which the file owner belongs, or to everyone. When a file is created, it is automatically associated with the UID of the user that creates the file (the file owner) and the GID of the directory it is in (the parent directory).

If a UNIX System Services user tries to access a HFS file, the UID and GID are compared with the UID and GID associated with the file. Depending on whether the values are equal, UNIX System Services grants the access rights of the file owner, the owner's group, or the rights that are granted to everyone.

If NFS is used to access HFS files, we must take into account which UIDs and GIDs are in effect on the NFS client system, and which security scheme is used with MVS NFS.

Because the UID and GID number is associated with an HFS file (and not related to a user name or group name), the `ls -l` command on an NFS client system will return different file owning user names and file owning group names than what is on the NFS server system, if the assignment of UIDs to user names and GIDs to group names is not consistent within the NFS network.

Also we have to consider which UIDs and GIDs are assigned to the NFS client users when they access an HFS file. This is dependent on the NFS security scheme that is in use:

- If EXPORTS security is used, there is no identification to RACF. For this reason, the UID and GID values associated with the user ID acting on the NFS client system are used for UNIX System Services security checking. An exception to this rule is UID=0 (superuser); it is mapped to -2.
- If SAF security is used, the NFS client user has to identify itself to RACF by either a PC-NFS mount or the **mvslogin** command. RACF associates UID and GID values with the NFS client user. These values are used in further processing on the MVS NFS side. The problem is that additional security checking is done on the NFS client side. This makes things complicated when the UID/GID values are not consistent in your network.

For example, we have an NFS connection between a UNIX NFS client and an MVS NFS server. There are four user IDs involved:

1. A user ID defined under UNIX System Services to be used with **mvslogin**. Let us call it MVSUSER with UID=200, GID=200.
2. The NFS client system user ID issuing the mount. Let us call it ROOT (ROOT has a UID=0).
3. Two other NFS client user IDs accessing the HFS file. Let us call them LUCKY with UID=200, GID=200, and UNLUCKY with UID=100, GID=100.

Independent of the security scheme used, root issues the **mount** command and the mount is successful. (On UNIX machines, you need root authority to issue NFS mounts.)

If EXPORTS security is used, no **mvslogin** command is needed. LUCKY will get access to MVSUSER's files as if it were MVSUSER because they have the same UID and GID values (200); UNLUCKY, which has a UID=100 and GID=100, gets world access.

If SAF security is used, the **mount** command will return OK, but as long as no **mvslogin** command is issued, all NFS client users will get security violation messages if they try to access an HFS file. Both LUCKY and UNLUCKY issue the **mvslogin** command with the user ID MVSUSER and the corresponding password. Now MVS NFS will grant to both users the rights of MVSUSER to HFS files. LUCKY will be happy after that, but UNLUCKY will not. Why?

MVS NFS sends the UID and GID associated with the mounted files to the NFS client system. The NFS client system will show LUCKY as owner of the files that MVSUSER owns in HFS (UID and GID both equal 200). Because the NFS client system also does security checking, UNLUCKY will be stopped by the local security system and again only get world access rights. In addition, if by chance there is a user JOE with UID=100 and GID=100 on UNIX System Services, the NFS client system will show UNLUCKY to be the owner of JOE's files, but if UNLUCKY tries to access the files, he will be stopped by SAF security because SAF security already granted MVSUSER's rights to LUCKY when LUCKY used the **mvslogin** command earlier.

Recommendations for using NFS

Following are recommendations for using NFS with z/OS UNIX:

1. Regardless of the security scheme you use, assign consistent UID and GID values in your NFS network. Each user should have the same UID and GID on every system he/she works.
2. Enable the PC-NFS support in MVS/ESA NFS and use PC-NFS where possible.
3. Use EXPORTS security when you can trust your UNIX system administration.
4. Use SAF security when your environment has additional security requirements.

   Follow this sequence of commands when:

   – Mounting to z/OS NFS:

     i. Log in to user ID root on the NFS client (if using a UNIX workstation).

     ii. Issue the **mvslogin** command. This is not required to mount, but it is useful to check whether everything is okay.

     iii. Issue the **mount** command.

     iv. Check the access to the HFS directory by using the **df** command on UNIX.

   – Using HFS:

     i. Log in to the user ID you want to work with in your UNIX environment.

     ii. Issue the **mvslogin** command.

     iii. You should now be able to access the mounted file system as permitted.

     iv. If you logged out from UNIX, issue the **mvslogin** command after the next UNIX login.

   – Unmounting from MVS NFS:

     i. Log in to user ID root on the NFS client (if using a UNIX workstation).

     ii. Be sure that no other user needs the mounted directory any more.

     iii. Issue the **umount** command.

     iv. Issue the **mvslogout** command.

- Tell the UNIX users which directories are mounted from z/OS NFS, and that they may have different access rights for HFS files than for local files if UID and GID values do not match.
- Because the EXPORTS file is not used, the NFS clients **`showmount`** command will reply:

  ```
  no exported file systems
  ```

  The information would be useless anyway.
- Be aware of the fact that the user ID used for **`mvslogin`** could also be used for other services such as rlogin, ftp, and telnet.

SAFEXP security combines EXPORTS and SAF security, making it the most secure NFS security level. However, because of its complexity (checking UID and GID values, along with user ID and passwords), this level of security may also cause additional confusion.

- Use SAFEXP security only if you want to hide parts of the HFS from the outside world.
- Be aware that faking the IP-host address is not a difficult task, especially on PC systems in an office environment.
- Keep the EXPORTS file as simple as possible. You have more flexibility if you assign access rights to HFS files by using RACF user IDs with different UID and GID values for security checking.
- With SAFEXP security, the **`showmount`** command will give a response, but this just reflects the EXPORTS file.
- All other security exposures mentioned for SAF security apply also for SAFEXP security.

5

z/OS Distributed File Service zSeries File System (zFS)

The z/OS Distributed File Service zSeries File System (zFS) is a z/OS UNIX System Services (z/OS UNIX) file system that can be used in addition or to the hierarchical file system (HFS). zFS file systems contain files and directories that can be accessed with z/OS UNIX application programming interfaces (APIs). These file systems can support access control lists (ACLs). zFS file systems can be mounted into the z/OS UNIX hierarchy along with other local (or remote) file system types (for example, HFS, TFS, AUTOMNT and NFS).

This chapter introduces zFS file systems, including the following topics:

- Application programming interfaces
- zFS physical file system
- zFS colony address space
- zFS file system aggregates
- Metadata cache
- zFS file system clones
- zFS logs
- zFS recovery

5.1 zFS introduction

The z/OS Distributed File Service (DFS) zSeries File System (zFS) is a z/OS UNIX file system that can be used in addition to the Hierarchical File System (HFS). zFS provides significant performance gains in accessing files approaching 8K in size that are frequently accessed and updated. The access performance of smaller files is equivalent to that of HFS.

zFS provides reduced exposure to loss of updates by writing data blocks asynchronously and not waiting for a sync interval.

zFS is a *journaling* file system. It logs metadata updates, then if a system failure occurs, zFS replays the log when it comes back up to ensure that the file system is consistent.

zFS is a Physical File System (PFS) that is started by UNIX System Services (USS) during IPL. A physical file system is the part of the operating system that handles the actual storage and manipulation of data on a storage medium.

5.2 Application programming interfaces (APIs)

zFS file systems contain files and directories that can be accessed with the z/OS hierarchical file system application programming interfaces on the z/OS operating system as follows:

- An application interface composed of C interfaces, some of which are managed within the C Run-Time Library (RTL), while others access kernel interfaces to perform authorized system functions on behalf of the unauthorized caller
- An interactive z/OS shell interface used by shell users

The PFS interface is a set of protocols and calling interfaces between the logical file system (LFS) and the PFSs that are installed on z/OS UNIX, as shown in Figure 5-1. In a USS environment, UNIX programs and UNIX users access their files through these interfaces. PFSs mount and unmount file systems and perform other file operations.

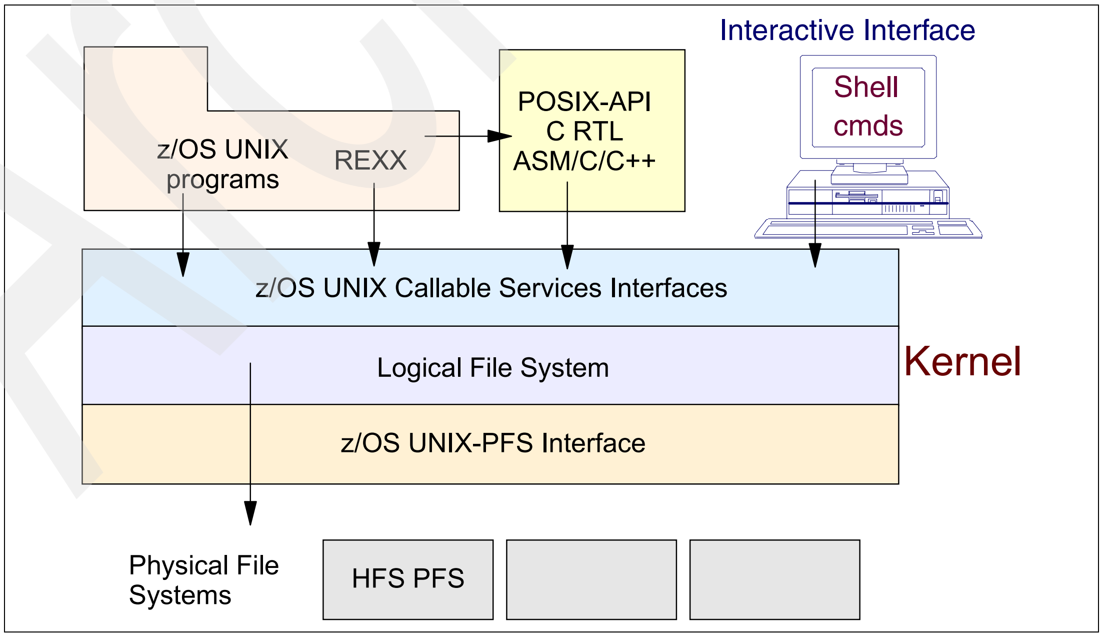


Figure 5-1 UNIX System Services (USS)

5.3 zFS physical file system

z/OS UNIX System Services (z/OS UNIX) allows you to install virtual file system servers (VFS servers) and PFSs.

A VFS server makes requests for file system services on behalf of a client. It is similar to a POSIX program that reads and writes files, except that it uses the lower-level VFS callable services API instead of the POSIX C-language API.

A PFS controls access to data. PFSs receive and act upon requests to read and write files that they control. The format of these requests is defined by the PFS interface.

In a UNIX System Services environment, the physical file systems are defined in the BPXPRMxx PARMLIB member. zFS, as a physical file system, is also to be defined in the PARMLIB member. Figure 5-2 shows all the physical file systems that can be defined in a USS environment.

The logical file system (LFS) is called by POSIX programs, non-POSIX z/OS UNIX programs, and VFS servers.

The PFS interface is a set of protocols and calling interfaces between the LFS and the PFSs that are installed on z/OS UNIX. PFSs mount and unmount file systems and perform other file operations.

There are two types of PFSs, those that manage files and those that manage sockets:

- File management PFSs, such as HFS and zFS, deal with objects that have path names and that generally follow the semantics of POSIX files.
- Socket PFSs deal with objects that are created by the socket() and accept() functions and that follow socket semantics.

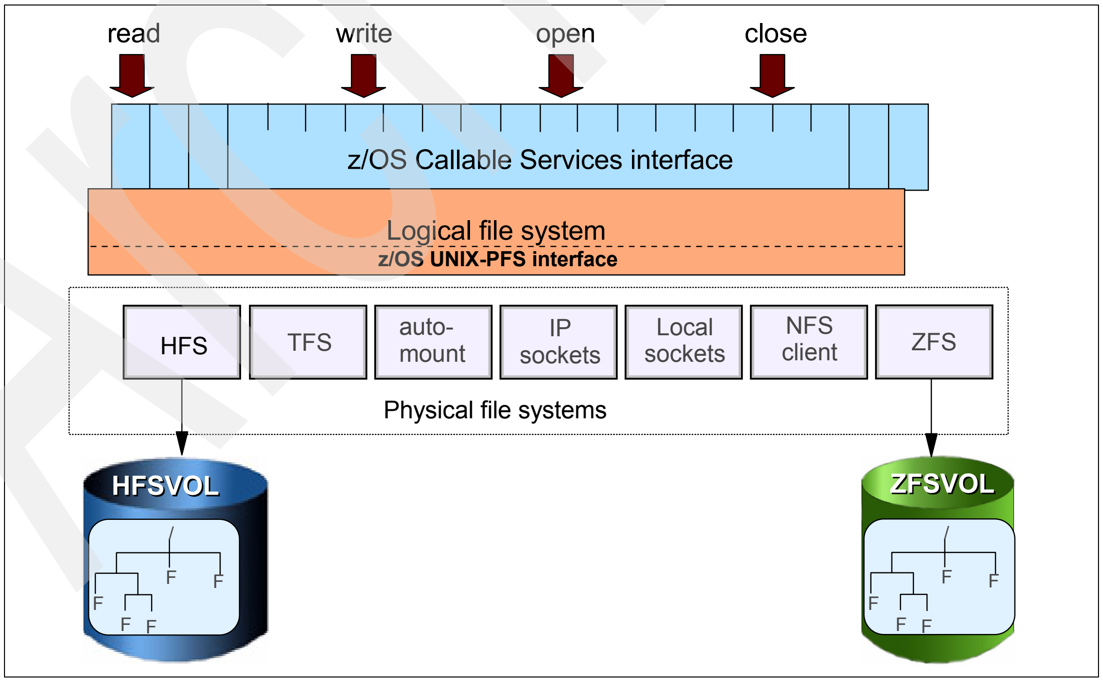


Figure 5-2 UNIX System Services physical file systems

zFS does not replace HFS; it can be considered to be complimentary to HFS.

5.4 zFS colony address space

zFS runs in a UNIX System Services (USS) colony address space. A *colony address space* is an address space that is separate from the USS address space. HFS runs inside the USS address space and zFS runs in its own address space, as shown in Figure 5-3.

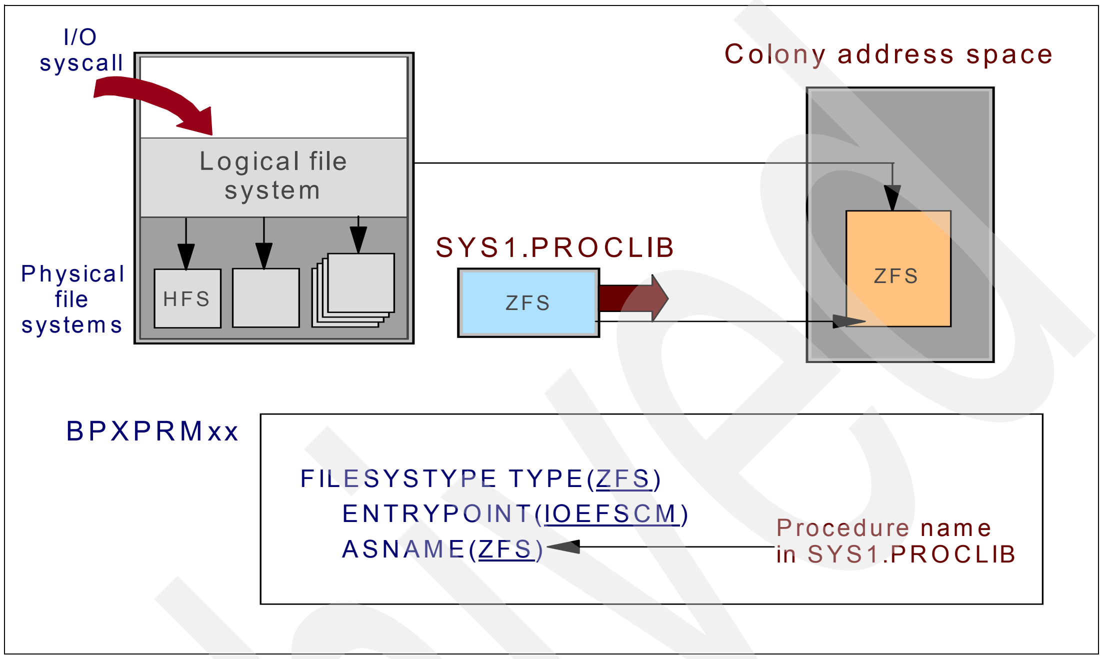


Figure 5-3 zFS executes in a colony address space

5.5 zFS supports z/OS UNIX ACLs

In order to provide better granularity of access control for z/OS UNIX files and directories, access control lists were introduced with z/OS V1R3. You can use access control lists (ACLs) to control access to files and directories by individual UIDs and GIDs. This provides the means to allow specific users and groups to have access to a file or directory.

To manage an ACL for a file, you must have one of the following security accesses:

- Be the file owner
- Have superuser authority (UID=0)
- Have READ access to SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class

Beginning with z/OS V1R3, ACLs are supported by the HFS and zFS file systems. You must also know whether your security product supports ACLs and what rules are used when determining file access.

ACL support works similar to the way access to MVS data sets is permitted, although the implementation is different. The ACL is a part of the File Security Packet (FSP), which is maintained by the PFS.

5.6 zFS file system aggregates

A zFS aggregate is a data set that contains zFS file systems. The aggregate is a VSAM Linear Data Set (VSAM LDS) and is a container that can contain one or more zFS file systems. An aggregate can only have one VSAM LDS, but it can contain an unlimited number of file systems. The name of the aggregate is the same as the VSAM LDS name.

Sufficient space must be available on the volume or volumes, as multiple volumes may be specified on the DEFINE of the VSAM LDS. DFSMS decides when to allocate on these volumes during any extension of a primary allocation. VSAM LDSs greater than 4 GB may be specified by using the extended format and extended addressability capability in the data class of the data set.

After the aggregate is created, formatting of the aggregate is necessary before any file systems can exist in it. A zFS file system is a named entity that resides in a zFS aggregate. It contains a root directory and can be mounted into the USS hierarchy. While the term file system is not a new term, a zFS file system resides in a zFS aggregate, which is different from an HFS file system.

zFS aggregates come in two types:

- Compatibility mode aggregates
- Multi-file system aggregates

5.6.1 Compatibility mode aggregates

A compatibility mode aggregate can contain only one zFS file system, making this type of aggregate more like an HFS file system. This is flagged in the aggregate when it is created. The name of the file system is the same as the name of the aggregate, which is the same as the VSAM LDS cluster name. The file system size (called a quota) in a compatibility mode aggregate is set to the size of the aggregate. Compatibility mode aggregates are more like an HFS data set, except that they are VSAM linear data sets instead of HFS data sets. We recommend that you start using compatibility mode aggregates first, since they are more like the familiar HFS data sets. Figure 5-4 shows a compatibility mode aggregate.

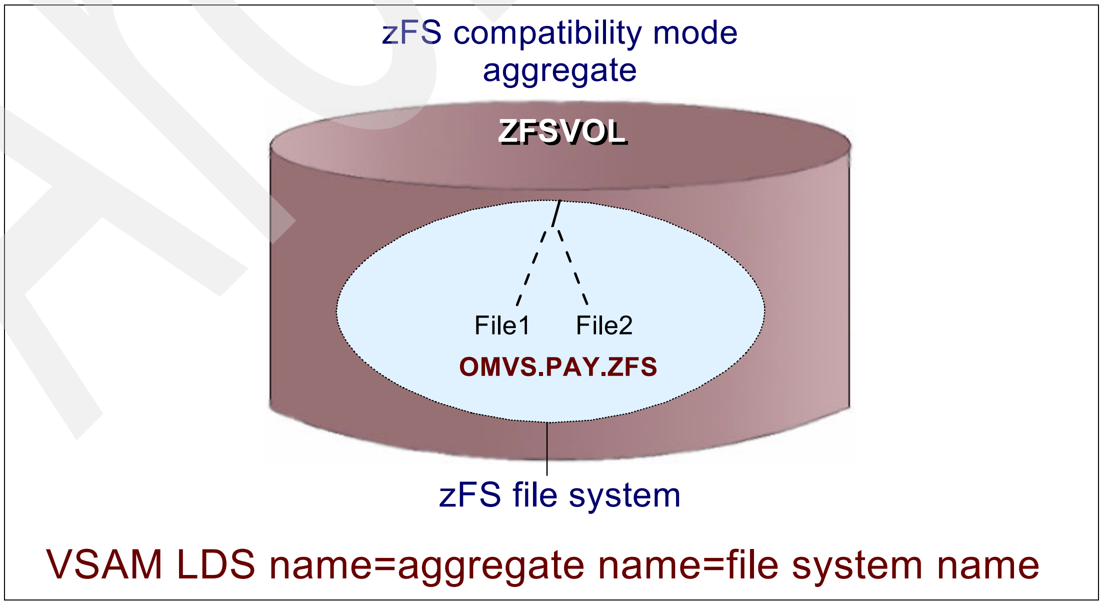

zFS compatibility mode aggregate
ZFSVOL
/
File1
File2
OMVS.PAY.ZFS
zFS file system
VSAM LDS name=aggregate name=file system name

Figure 5-4 Compatibility mode aggregate

5.6.2 Multi-file system aggregates

A multi-file system aggregate allows the administrator to define multiple zFS file systems in a single aggregate. This allows *space sharing*.

Space sharing

Space sharing means that if you have multiple file systems in a single data set, and files are removed from one of the file systems—which frees DASD space—another file system can use that space when new files are created. This new type of file system is called a multi-file system aggregate.

The multiple file system aggregate OMVS.MUL02.ZFS, shown in Figure 5-5, can contain multiple zFS file systems. This makes it possible to do space sharing between the zFS file systems within the aggregate.

The multiple file system aggregate has its own name. This name is assigned when the aggregate is created. It is always the same as the VSAM LDS cluster name. Each zFS file system in the aggregate has its own file system name. This name is assigned when the particular file system in the aggregate is created. Each zFS file system also has a predefined maximum size, called the *quota*.

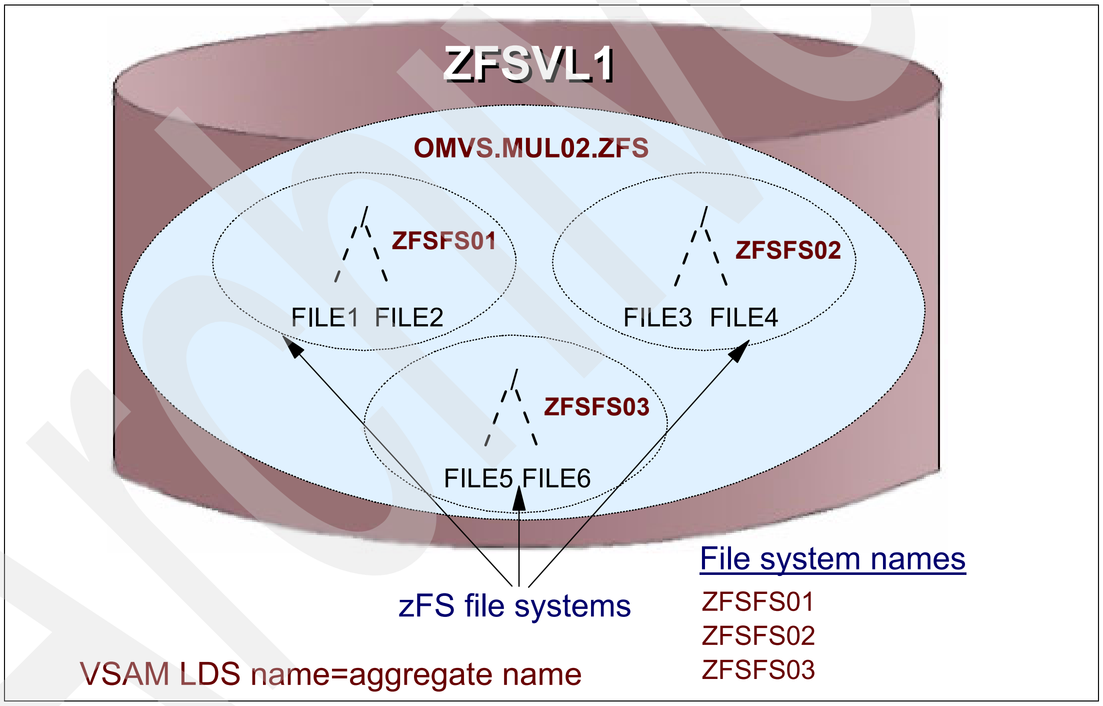


Figure 5-5 Multi-file system aggregate

Attention: In a future release, IBM plans to withdraw support for zFS multi-file system aggregates. When this support is withdrawn, only zFS compatibility mode aggregates will be supported.

5.7 Metadata cache

The zFS file system has a cache for file system metadata, which includes directory contents and the data of files that are smaller than the aggregate block size. The setting of this cache size is important to performance because zFS references the file system metadata frequently. Synchronous reads of metadata increase I/O rates to disk and server response times. Metadata consists of things like owner, permission bit settings, and data block pointers.

The metadata cache is stored in the zFS primary address space; its default size is 32 MB. Since the metadata cache only contains metadata and small files, it normally does not need to be nearly as large as the user file cache.

Note: With z/OS V1R4, a new backing cache for metadata provides an extension to the meta cache and resides in a data space. Specify the following in the IOEFSPRM file:

```
metaback_cache_size=64M,fixed
```

The values allowed are 1 MB to 2048 MB. It is used as a *paging* area for metadata and allows a larger meta cache for workloads that need large amounts of metadata. This cache is only needed if the meta cache is constrained

5.8 zFS file system clones

zFS allows an administrator to make a read-only clone of a file system in the same aggregate. This clone file system can be made available to users to provide a read-only point-in-time copy of a file system. The clone operation happens relatively quickly and does not take up too much additional space because only the metadata is copied.

When a file system is cloned, a copy of it is created in the same aggregate, as shown in Figure 5-6. There must be physical space available in the aggregate for the clone to be successful. For the clone to be used, it must be mounted.

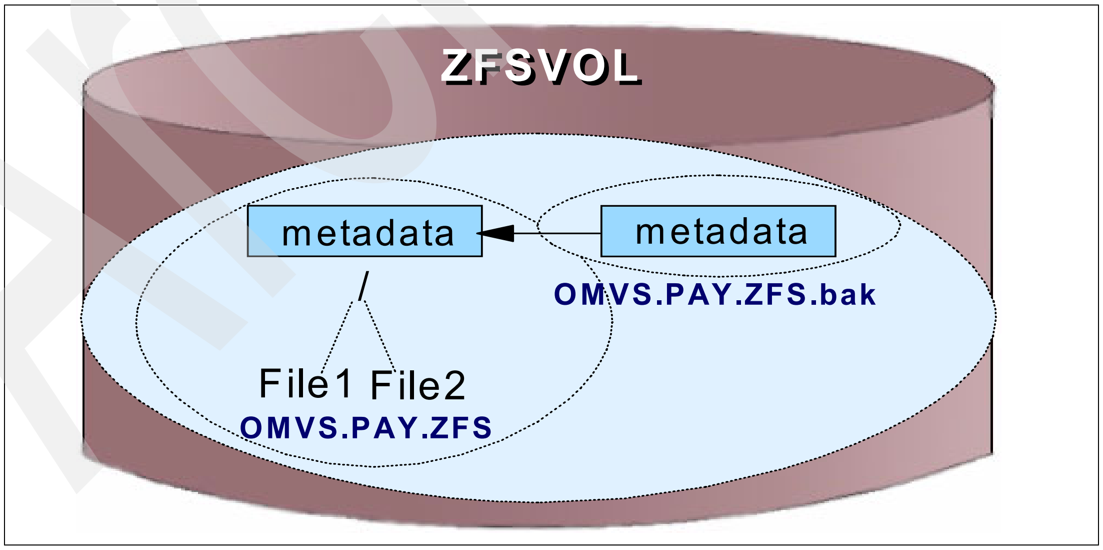


Figure 5-6 zFS file system clone

5.8.1 Backup file system

The zFS file system that is the result of the clone operation is called the *backup file system*. The backup file system is a read-only file system and can only be mounted as read-only.

5.9 zFS log files

Every zFS aggregate contains a log file that is created when the aggregate is formatted. This log is used to record transactions describing changes to the file system structure. The default for the log file is 1% of the aggregate size, but it can be changed during the format of the aggregate. Usually, 1% is sufficient for most aggregates. However, very large aggregates might need less than 1% while very small aggregates might need more than 1% if a high degree of parallel update activity occurs for the aggregate.

5.10 zFS recovery

zFS provides a recovery mechanism that uses a zFS file system log to verify or correct the structure of an aggregate. This recovery mechanism is invoked by an operator command, **`ioeagslv`**.

When you do a system restart, a recovery program called the *salvager* uses the zFS file system log to return consistency to a file system by running recovery on the aggregate on which the file system resides. Recovery consists of reading the log that contains all the changes made to metadata as a result of the operations done to the aggregate, such as file creation and deletion. If problems are detected in the basic structure of the aggregate, if the log mechanism is damaged, or if the storage medium of the aggregate is damaged, the **`ioeagslv`** command must be used to verify or repair the structure of the aggregate.

5.11 Additional information

For more information and details on zFS and migration from HFS to zFS see the zFS Redbook *z/OS Distributed File Service zSeries File System Implementation*, SG24-6580.

USS sysplex sharing

This chapter provides information on how z/OS UNIX System Services has been designed and how it works. The topics described are:

- ► USS sysplex sharing design
- ► USS file system structures
- ► USS file system sharing
- ► AUTOMOVE system list
- ► USS file sharing structures for a system
- ► USS file system sharing implementation
- ► Effects of USS sysplex sharing
- ► Mount table limit monitoring
- ► Mount table limiting
- ► BRLM in a shared environment
- ► Considerations for the version root
- ► Replacing the root without IPL
- ► Licensed products and the root file system
- ► System specific data and the version root

6.1 USS sysplex sharing design

At the time USS sysplex sharing was introduced, there were requirements from customers and also IBM's ideas, of course, on what functions should be provided and how they should be implemented. The latter had to focus on a reasonable way to send out the basic file structures and to provide the new functions without too much complexity—and as soon as possible.

Following we list some important design objectives, with brief explanations:

- One root file system structure for both sysplex and non-sysplex environments

  This makes it possible to ship the base USS file structure as one single object.

- Allow for "rolling IPLs" to introduce new systems

  It is necessary, of course, to IPL new systems while others are already running.

- Allow for backout of systems from sysplex
- Multiple releases of the root file system in the sysplex

  Not all the systems run at the same release or version of z/OS, and there are often several service levels used for systems running at the same release.

- Preserve structures and philosophies IBM recommended in the past—and customers used!
- Accessing all file systems from any system in a sysplex

  This is one of the main requests to:

  – Make it easier to do maintenance.
  – Allow users to access their data independent of the system they log on to.

- Sharing USS file systems across a sysplex in R/W mode

  This is another very important request. There would still be no system independence without this.

- Same structure and view on all systems

  It is essential for users to have the same clear view of the USS file structure with all the data that is important (for a specific system) as before. Figure 6-1 on page 239 shows this structure.

- Same behavior in both sysplex and none-sysplex environments

  Both environments must be implemented as similarly as possible to allow easy switching to and from sysplex sharing.

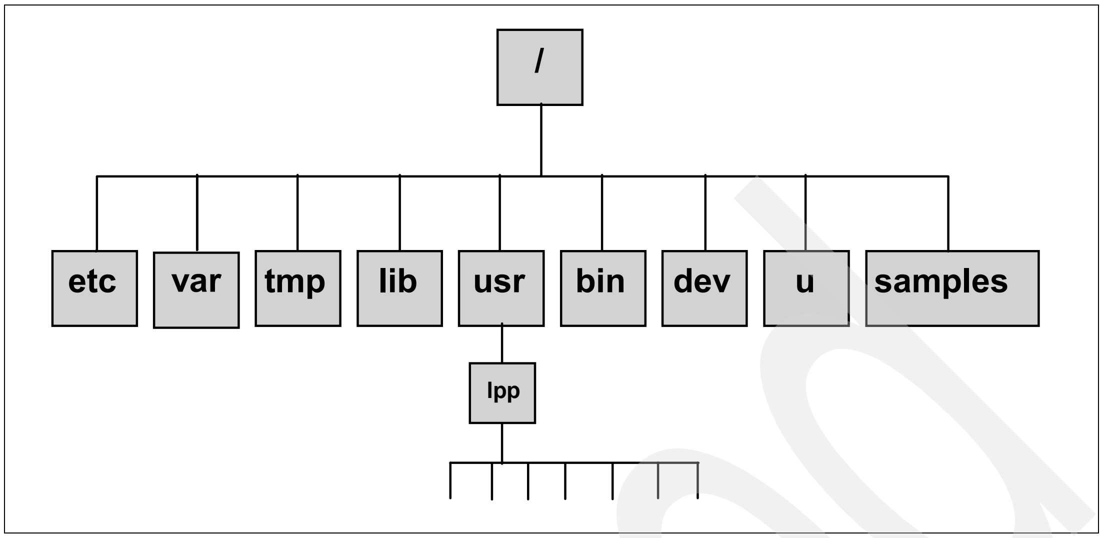


Figure 6-1 Logical view (transparent USS file structure)

6.2 USS file system structures

Using USS file system sysplex support, the administrator can do the following:

- Write to file systems from all systems in the sysplex
- Have greater availability of data in the event of a system outage
- Have a common file system hierarchy on all systems
- Better manage file system placement

The advantages of the shared environment are:

- Greater user mobility
- Flexibility with file system balancing
- Consolidation of data
- One common BPXPRMxx for all systems

6.2.1 Symbolic links

Using symbolic links is one of the key techniques used to implement USS sysplex sharing, make it work as needed, and necessitate only very few changes when working with the UNIX file structure. Understanding how the links are used is essential for understanding the data structures introduced with USS sysplex sharing.

Note: Using symbolic links is just a technique to work with the USS sharing file system structures and does not provide the means to share UNIX file systems. This is shown later in 6.3, “USS file system sharing” on page 251.

Before availability of USS sysplex sharing, symbolic links had been used almost exclusively to provide an alias name to a UNIX file. In the /bin directory there are a huge number of symbolic links pointing to executables down the /usr/lpp structure where USS file systems for program products are located. This makes it possible to run these programs just by entering the name of the symbolic link instead of having to provide the full path name.

What is new in USS sharing is the fact that now symbolic links are used intensively to point to directories. Figure 6-2 shows a sample for a symbolic link named /etc that points to a subdirectory etc under another directory named /SY1.

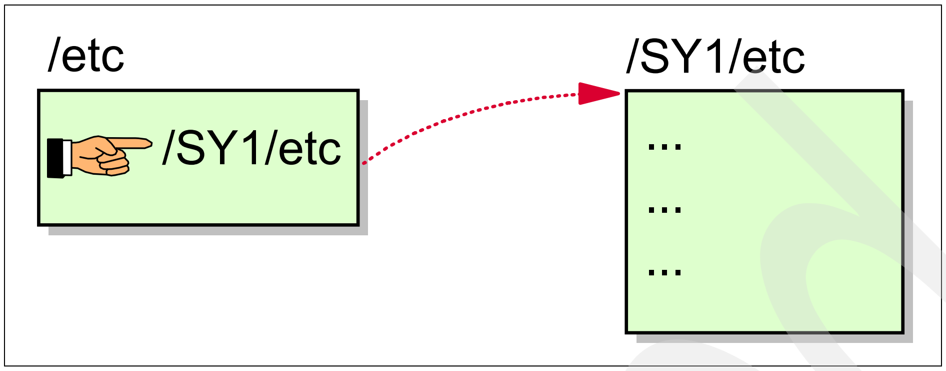


Figure 6-2 Symbolic link pointing to a directory

This means that you can simply use “/etc” to address “/SY1/etc”. However, you need to be careful when using UNIX commands or working in the ISHELL to really address the target directory and not the symbolic link itself.

In Figure 6-3 two **find** commands are run to list the file /etc/rc. If etc is a directory and not a symbolic link, you would get the same output for both situations. It is not in the first command (the symbolic link is addressed), but in the second it is the target directory and we get the desired output.

```
$> find /etc -name rc
$> find /etc/ -name rc
/etc/rc
```

Figure 6-3 Sample commands addressing the symbolic link and the target directory

For this reason there is the following simple rule to avoid problems when addressing directories.

Attention: If you want to address a directory and do not know whether the name is a symbolic link or really the directory, always append a slash (/). This always assures that the (target) directory is addressed and you never get into trouble.

And this small restriction, compared to the old way to work with the UNIX file structure, is the biggest change that you must pay attention to when starting to work in an OS/390 or z/OS system beyond OS/390 V2R8.

6.2.2 Single system image OS/390 V2R9 or later

There is a second fact that makes the symbolic links a little bit more complex. It is not enough to just have fixed symbolic links. All the new important symlinks are composed of a variable part and a fixed part. This is needed to have the flexibility to point to target directories depending on the following:

- Whether running in sysplex sharing mode or not
- The system they are used in (if USS sharing is active)
- A definition setting in the BPXPRMxx parmlib member (if USS sharing is active)

Figure 6-4 shows the structure of the new root file system as it is shipped by IBM. As before, there are still directories bin, usr, lib, opt, samples and u in the top level directory /, but etc, dev, tmp and var are now symbolic links whose contents start with a variable named $SYSNAME. In case of a single system image it is resolved to /SYSTEM.

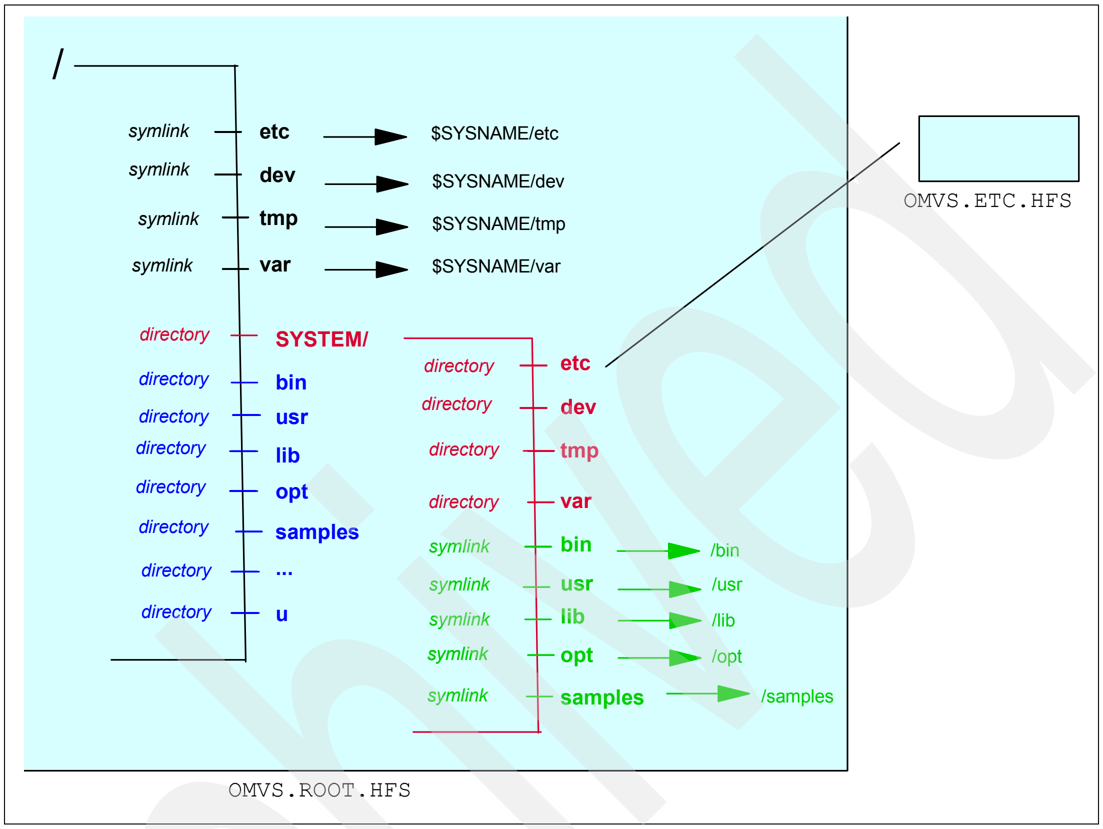


Figure 6-4 Single system image

And SYSTEM is a new subdirectory that contains the real directories etc, dev, tmp, and var that are used as mount points for the corresponding file systems containing these structures.

For convenience in this second level directory, symbolic links are provided pointing back to the real first-level structures /bin, /usr, /lib, /opt, and /samples.

Note: The user still sees the same transparent view of the UNIX file structure as before. This is true for the first level (/) and the second level (/SYSTEM).

6.2.3 USS enhancements in case of sysplex sharing

When exploiting the USS sysplex sharing function we need to deal with the following new objects or changes:

- USS file system structures that exist in a sysplex
- New BPXPRMxx parmlib statements and options
- New OMVS couple data

We first describe the file system structures and the new BPXPRMxx settings. We have a look at the couple data set later.

6.2.4 USS file system structures in a sysplex

To reflect the situation in a sysplex we need the following new file system structures and hierarchy.

Sysplex root file system

The sysplex root is an HFS data set that is used as the sysplex-wide root. This HFS data set must be mounted read-write and designated AUTOMOVE. Only one sysplex root is allowed for all systems participating in shared HFS. The sysplex root is created by invoking the BPXISYSR sample job in SYS1.SAMPLIB.

Note: No files or code reside in the sysplex root data set. It consists of directories and symbolic links only, and it is a small data set.

The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequently as possible.

Version file system

The version HFS is the IBM-supplied root HFS data set. To avoid confusion with the sysplex root HFS data set, "root HFS" has been renamed to "version HFS". You can use one version HFS for each set of systems participating in shared HFS and that are at the same release level.

System-specific file system

Directories in the system-specific HFS data set are used as mount points, specifically for /etc, /var, /tmp, and /dev. To create the system-specific HFS, run the BPXISYSS sample job in SYS1.SAMPLIB on each participating system (in other words, you must run the sample job separately for each system that will participate in shared HFS). IBM recommends that the name of the system-specific data set contain the system name as one of the qualifiers. This allows you to use the &SYSNAME. symbolic (defined in IEASYMxx) in BPXPRMxx.

6.2.5 New or changed BPXPRMxx statements

The following BPXPRMxx statements are changed or have new parameters:

- SYSPLEX(YES/NO)
- VERSION('nnnn')
- MOUNT statement

SYSPLEX(YES/NO)

Based on this statement, a system started will stay as single system (specifying NO) or will become a member of the USS sysplex sharing group and participate in HFS or zFS data sharing. To do that, the systems must be at the OS/390 V2R9 level or later. Systems that specify SYSPLEX(YES) make up the participating group for the sysplex.

Note: For compatibility reasons, the system default is SYSPLEX(NO).

VERSION('nnnn')

VERSION('nnnn') allows multiple releases and service levels of the binaries to coexist and participate in HFS sharing. `nnnn` is a qualifier to represent a level of the version HFS. The most appropriate values for `nnnn` are the name of the target zone, &SYSR1., or another qualifier meaningful to the system programmer. A directory with the value `nnnn` specified on VERSION will be dynamically created at system initialization under the sysplex root and will be used as a mount point for the version HFS.

MOUNT statement

The new parameters AUTOMOVE, NOAUTOMOVE, and UNMOUNT on ROOT and MOUNT indicate what happens to the file system if the system that owns that file system goes down or fails, as follows:

- AUTOMOVE - specifies that ownership of the file system is automatically moved to another system. It is the default.
- NOAUTOMOVE - specifies that the file system will not be moved if the owning system goes down and the file system is not accessible.
- UNMOUNT specifies that the file system will be unmounted when the system leaves the sysplex. This option is new with z/OS V1R3.

Note: This option is not yet available to be specified for automounted file systems. They are set automovable by default.

Previous to z/OS V1R3, file systems could be mounted as AUTOMOVE and NOAUTOMOVE. If AUTOMOVE is specified, the file system is moved to another system in the event that the owning system is taken down. If NOAUTOMOVE is specified, the file system remains mounted when the owning system goes down, but it now has an unknown owner, as shown in Figure 6-5.

```
HFS           445 UNOWNED                    RDWR
  NAME=WTSCPLX2.SC64.SYSTEM.HFS
  PATH=/SC64
  OWNER=          AUTOMOVE=N CLIENT=Y
```

Figure 6-5 Display of file system showing unknown owner

When the failed system reinitializes, the file system recovers and becomes active again.

A new UNMOUNT option has been added in order to unmount file systems associated with a failed system. This allows for file systems that are required or desirable to not move to another system to be unmounted. This avoids either recovering or converting them to “unowned” status.

For systems previous to z/OS V1R3, the following system command can be used to avoid this “unowned” status, too:

```
F BPXOINIT,SHUTDOWN=FILESYS
```

If run, probably during normal shutdown processing, this command assures that all file systems that are not mounted as automovable and are owned (see 6.3, “USS file system sharing” on page 251 for an explanation of file ownership) by this system, together with all automounted file systems owned by this system, are unmounted automatically.

6.2.6 The UNIX sysplex sharing structures

We now provide more details about all the UNIX file and data structures mentioned previously, especially the symbolic links used.

Sysplex root (before system startup)

Figure 6-6 shows the sysplex root structure as it has been defined by JCL BPXISYSR and before the first system is IPLed. It contains directories and symlinks to redirect USS file structures correctly across the sysplex. It does not contain any files or code.

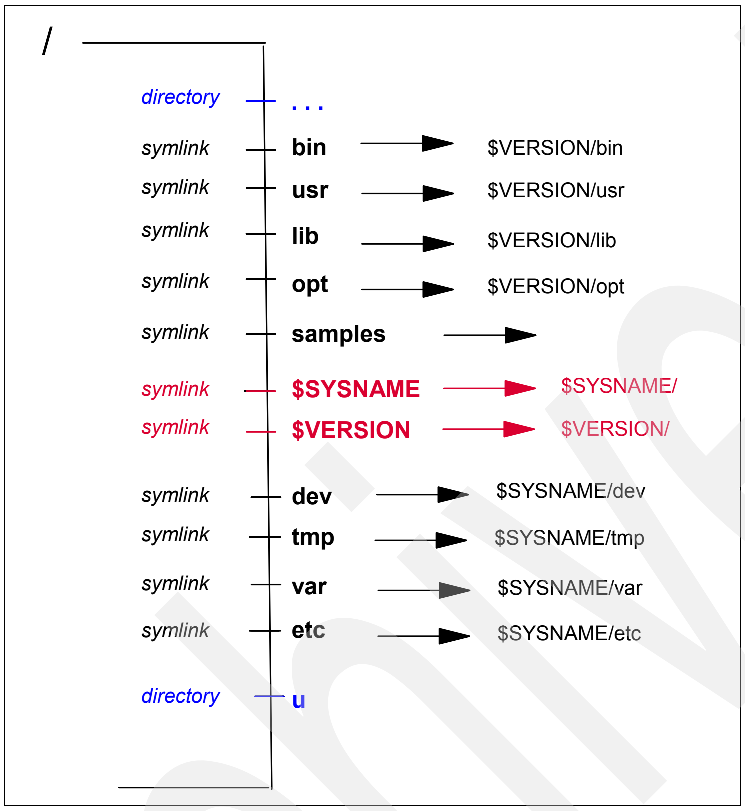


Figure 6-6 Sysplex root file system before first system is IPLed

The symbolic links dev, tmp, var, and etc have the same contents as in the root file system used in a single system. However, in sysplex sharing the variable $SYSNAME is resolved through system symbol &SYSNAME.

The directories bin, usr, lib, opt, and samples are replaced by symbolic links starting with another variable, $VERSION, which is resolved using the value specified on the VERSION statement in a BPXPRMxx parmlib member as soon as a system is IPLed.

For convenience, two other symbolic links, $SYSNAME and $VERSION, are provided to allow listing the version or the system-specific structure easily.

Sysplex root (after system startup)

Figure 6-7 shows the sysplex root file system after the first system participating in USS sysplex sharing is IPLed. Two new dynamically created directories are created, the mount point directories for both, the system specific file system, and the version file system used by this system.

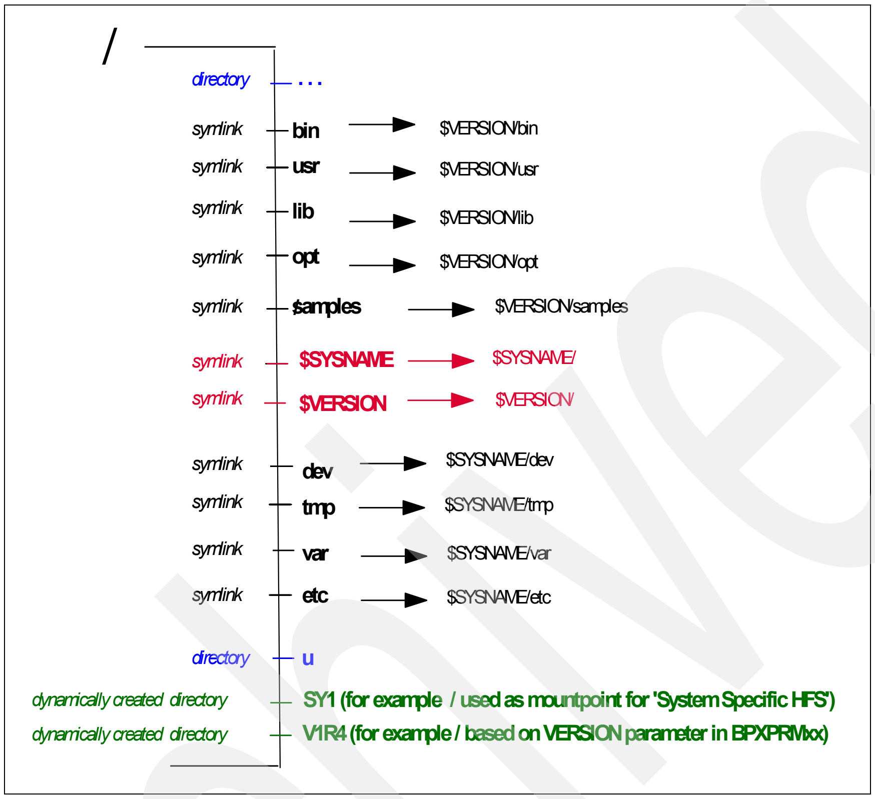

/
directory ...
symlink bin $VERSION/bin
symlink usr $VERSION/usr
symlink lib $VERSION/lib
symlink opt $VERSION/opt
symlink samples $VERSION/samples
symlink $SYSNAME $SYSNAME/
symlink $VERSION $VERSION/
symlink dev $SYSNAME/dev
symlink tmp $SYSNAME/tmp
symlink var $SYSNAME/var
symlink etc $SYSNAME/etc
directory u
dynamically created directory SY1 (for example / used as mountpoint for 'System Specific HFS')
dynamically created directory V1R4 (for example / based on VERSION parameter in BPXPRMxx)

Figure 6-7 Sysplex root file system after the first system is IPLed

You can list these structures by selecting the real mount point directory, if you know the name, or you can use the two symlinks $SYSNAME and $VERSION, whichever is easier.

System-specific file system

Figure 6-8 on page 246 shows the system-specific file system structure as it has been created by JCL BPXISYSS. Similarly, as the sysplex root, this file system contains only directory mount points and symlinks. The structure is identical with that of subdirectory SYSTEM in the root file system, now called version file system. And it is used exactly the same way. Because every system in the sysplex needs its own system-specific data, there is one for each of the systems running in the sysplex sharing environment.

The symbolic links bin, usr, lib, opt, and samples provided again allow a global view of the whole UNIX file structure for the system. The resolution is a two-step process. They are pointing back to the symlinks in the sysroot structure and then getting to the final directories in the version root file system, described next.

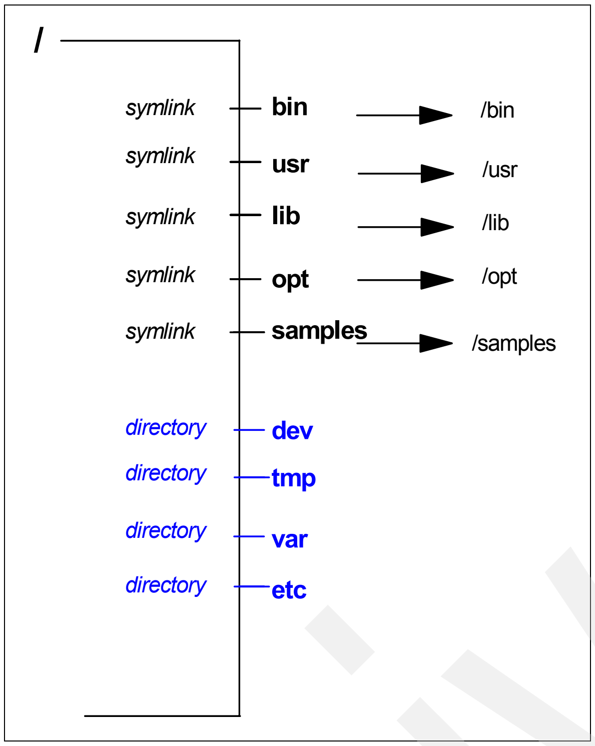


Figure 6-8 USS system-specific file system

Version file system

Figure 6-9 on page 247 shows the version file system structure. Just like magic, the same symbolic links that are working in a single system without file sharing active, are working. The symlinks are now resolved to point to the system-specific file structure.

Note that again we have the convenience of a global view with the version root level because all the important structures are directly accessible.

Important: Subdirectory SYSTEM and mount point directories like ... and u in the version file system are still there, but with sysplex sharing active they are not used.

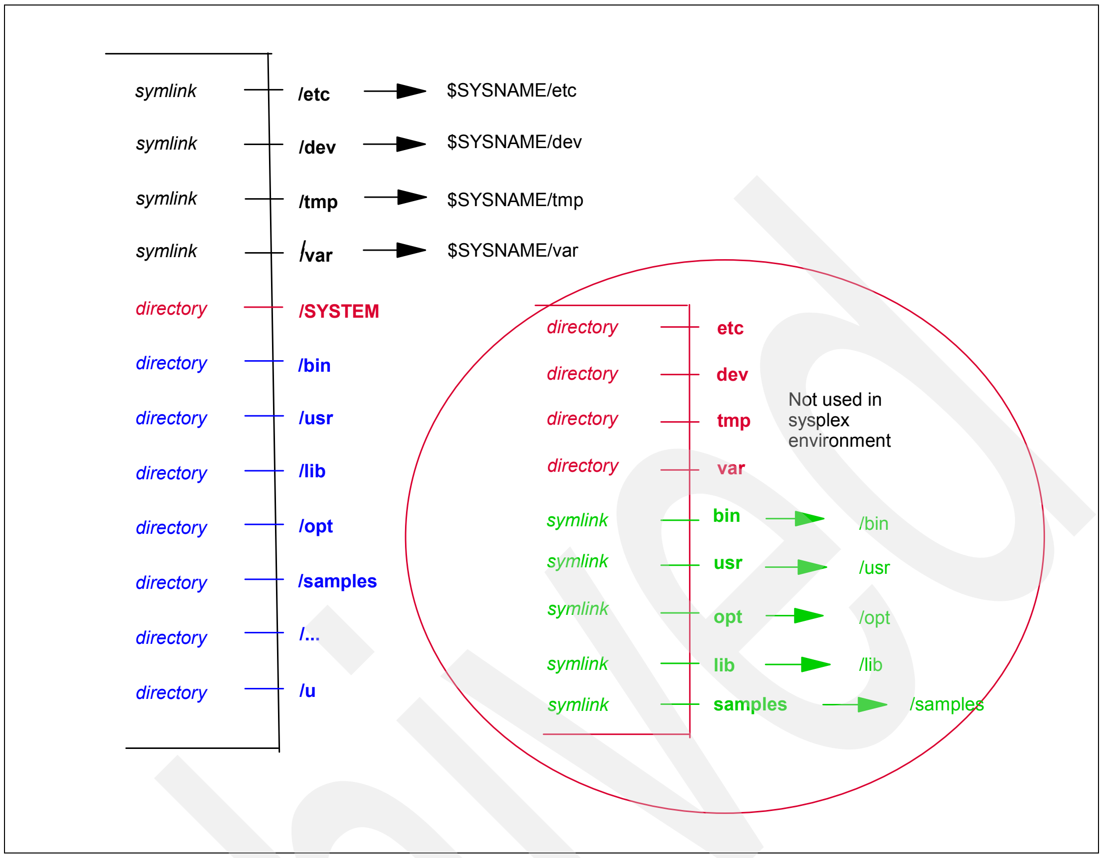


Figure 6-9 USS version file system

First system in a sysplex

Figure 6-10 on page 248 shows the whole picture for the first system IPLed in the sysplex sharing environment.

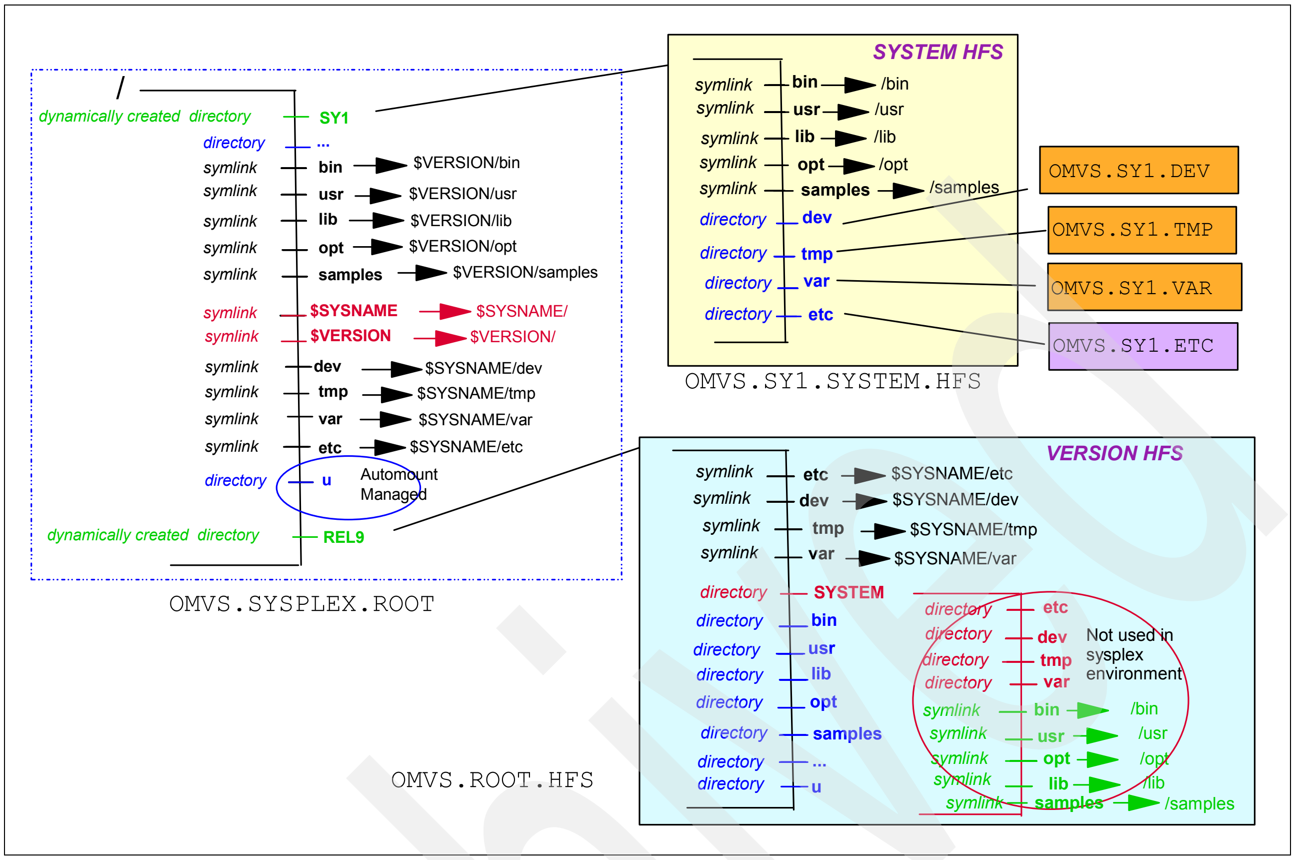


Figure 6-10 First system in sysplex

BPXPRMxx parmlib member

In Figure 6-11 on page 249 we show the root and the mount statements from the BPXPRMxx member.

```
VERSION('V1R4')
SYSPLEX(YES)

ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE (HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE(HFS) MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..ETC')
TYPE(HFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./etc')
....
```

Figure 6-11 BPXPRMxx parmlib member

Sysplex sharing enables you to use one BPXPRMxx member to define all the file systems in the sysplex. This means that each participating system may have its own BPXPRMxx member to define system limits, but shares a common BPXPRMxx member to define the file systems for the sysplex. This is done through the use of system symbols.

Note: In order to run with one general BPXPRMxx member, the version file system to be mounted would also need to contain a system global. This is not in our sample in order to keep it a little bit simpler. A typical symbol to use is &SYSR1, for example.

You can also have multiple BPXPRMxx members defining the file systems for individual systems in the sysplex.

You should define your version and sysplex root HFS data as AUTOMOVE, and define your system-specific file systems as UNMOUNT. Do not define a file system as NOAUTOMOVE or UNMOUNT and a file system underneath it as AUTOMOVE. If you do, the file system defined as AUTOMOVE will not be recovered after a system failure until that failing system has been restarted.

Multiple systems in a sysplex

Figure 6-12 on page 250 shows a sample of two systems in sysplex sharing that use the same version file system.

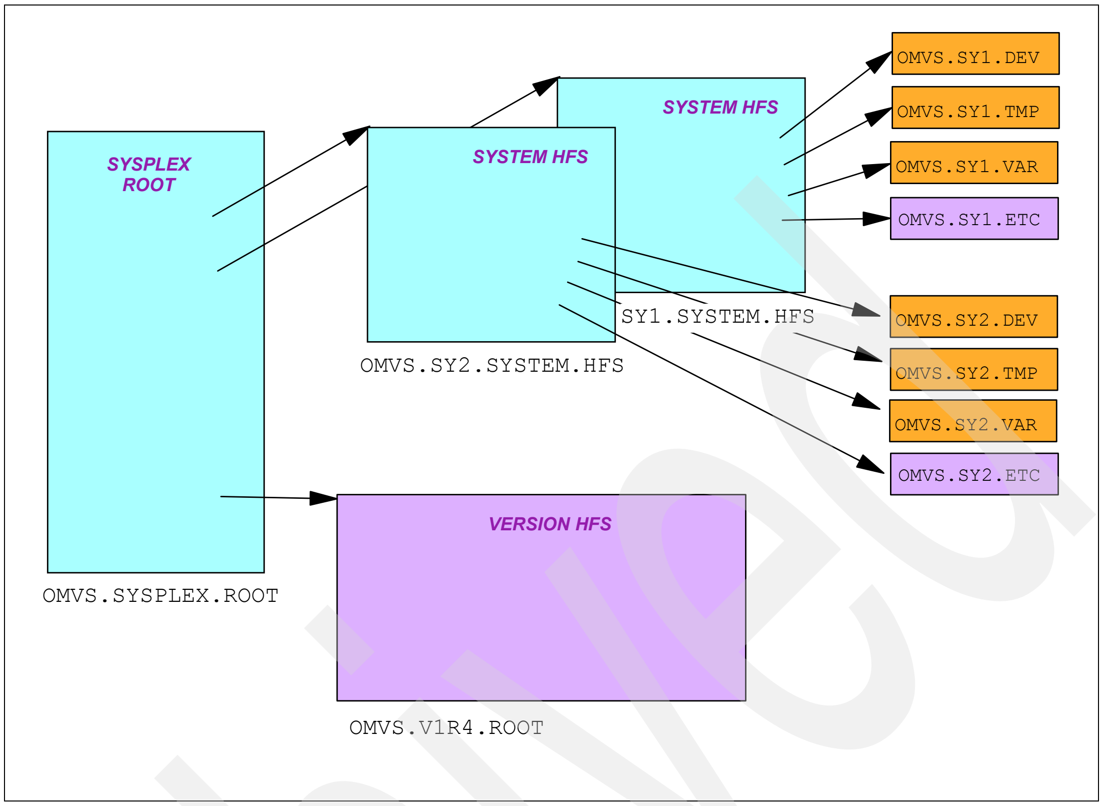


Figure 6-12 Two systems in a sysplex

Figure 6-13 shows that the users still see the same global file system structure and do not need to know about naming conventions in the sysplex. Just the new directories are seen.

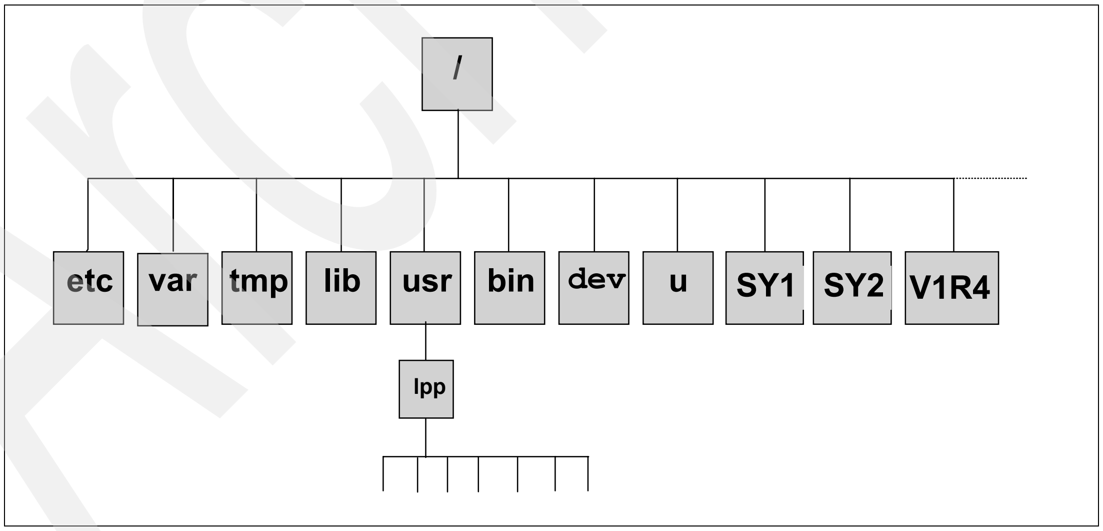


Figure 6-13 Transparent file system structure (multi-system)

Figure 6-14 on page 251 finally shows two systems at two different release levels. Therefore, we also have two different version file systems.

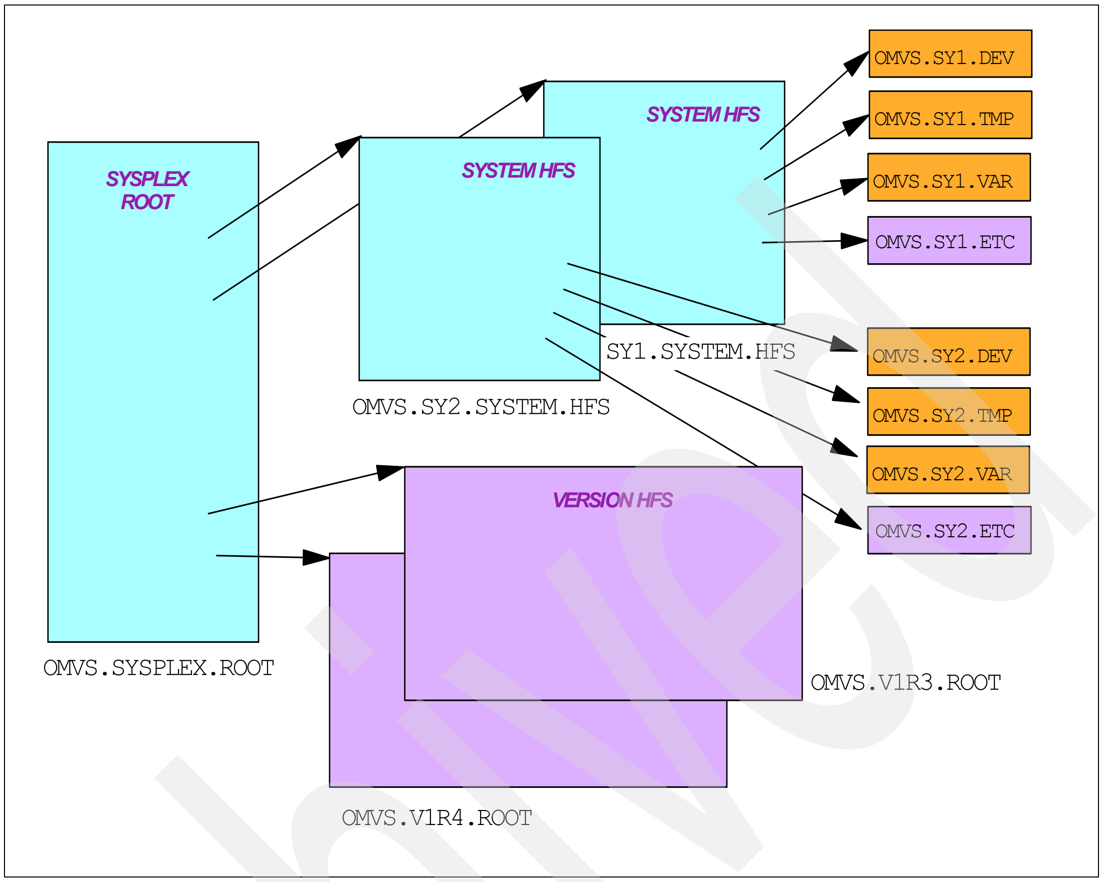


Figure 6-14 Multiple systems and versions in a sysplex

6.3 USS file system sharing

The sharing support allows access to file systems in R/W mode. The implementation of USS data sharing in a sysplex is a logical implementation. There is no physical sharing implemented. This means the old rules are still valid.

- You can have many systems sharing a file system in read-only mode.
- If one system has read-write access to a file system, no other system can have direct access to this file system.

Figure 6-15 shows three systems in a sysplex sharing the environment. System 2 has R/W access to the file system. It is called the owner of the file system. The other systems do not have physical access. They send their requests to System 2 and get back data using XCF.

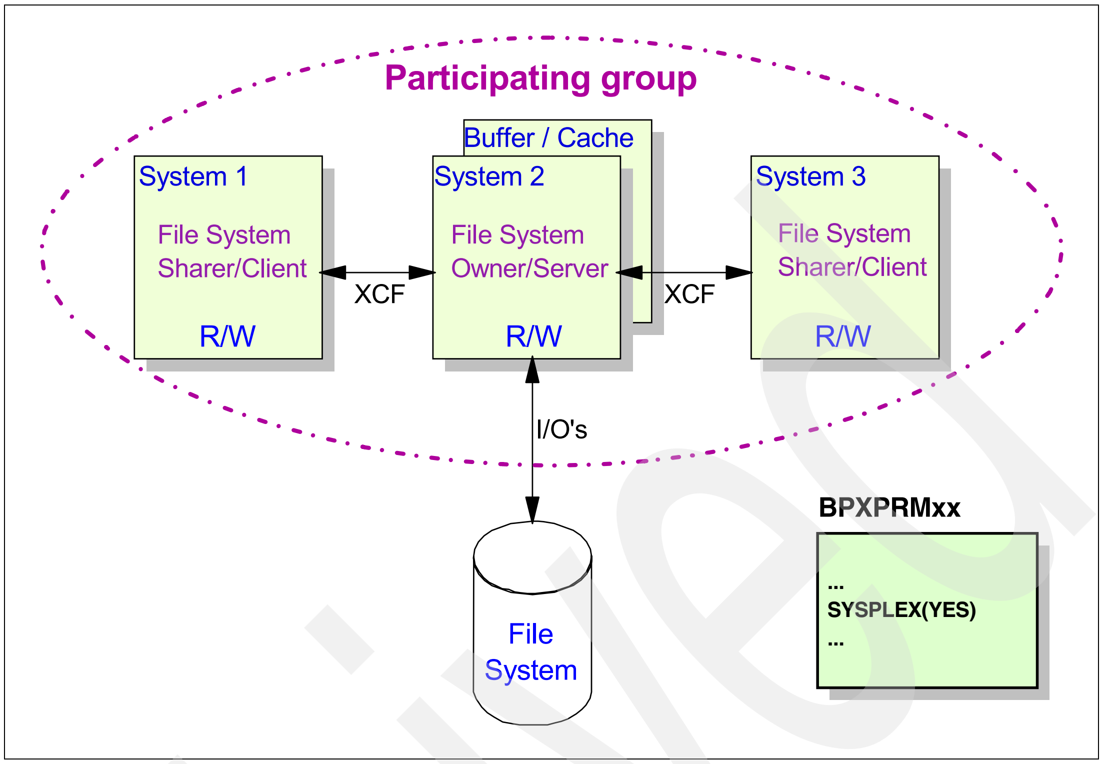


Figure 6-15 Sharing in R/W mode

6.3.1 Logical and physical file system relationship

Figure 6-16 on page 253 shows that only the owner of a file system deals with the physical file system. All the other systems just access the data within the logical file system layer.

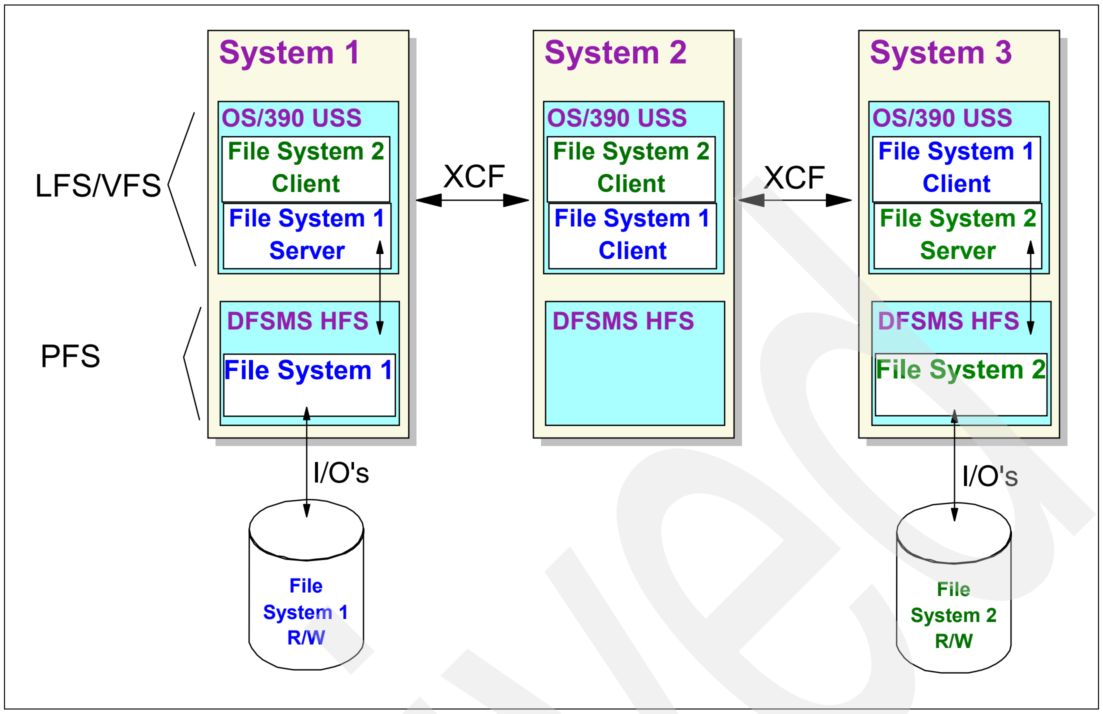


Figure 6-16 LFS and PFS relationship in R/W mode

Figure 6-17 demonstrates that there is no change for file systems shared in R/O mode. In this situation all the systems have direct access to File System 1.

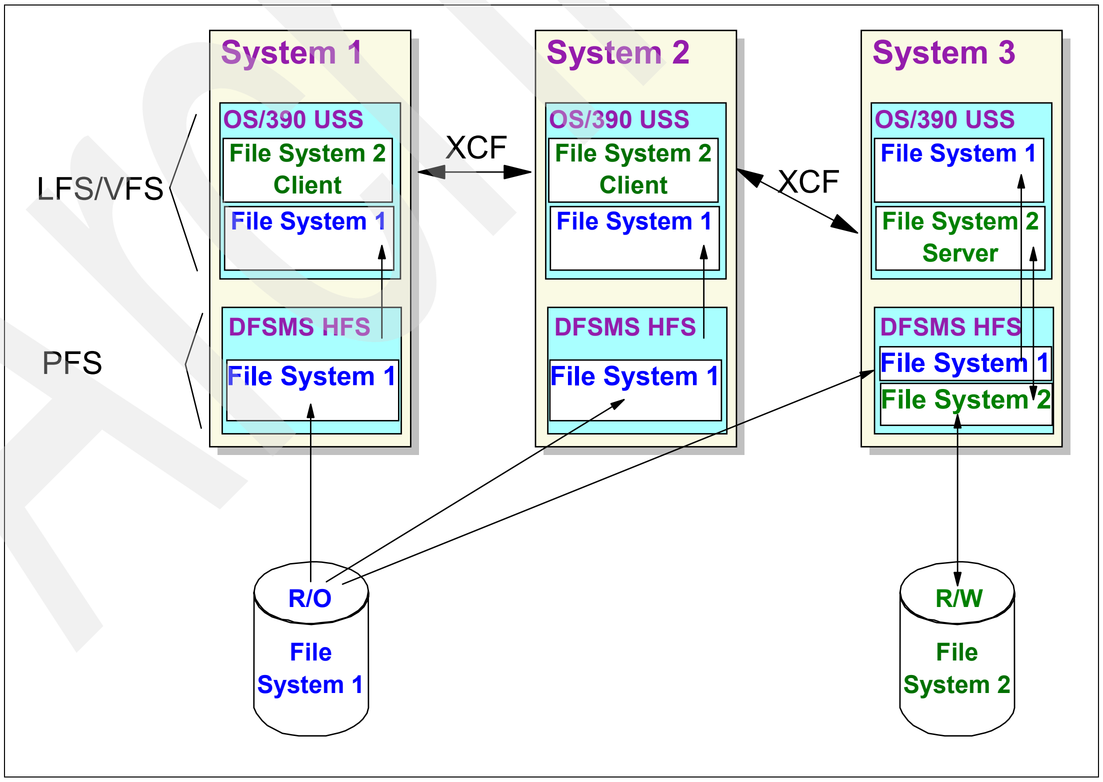


Figure 6-17 LFS and PFS relationship in R/O mode

6.3.2 Shared USS couple data set

The couple data set (CDS) contains the sysplex-wide mount table and information about all participating systems, and all mounted file systems in the sysplex. To allocate and format a CDS, customize and invoke the BPXISCDS sample job in SYS1.SAMPLIB. The job will create two CDSs: one is the primary and the other is a backup that is referred to as the alternate. In BPXISCDS, you also specify the number of mount records that are supported by the CDS.

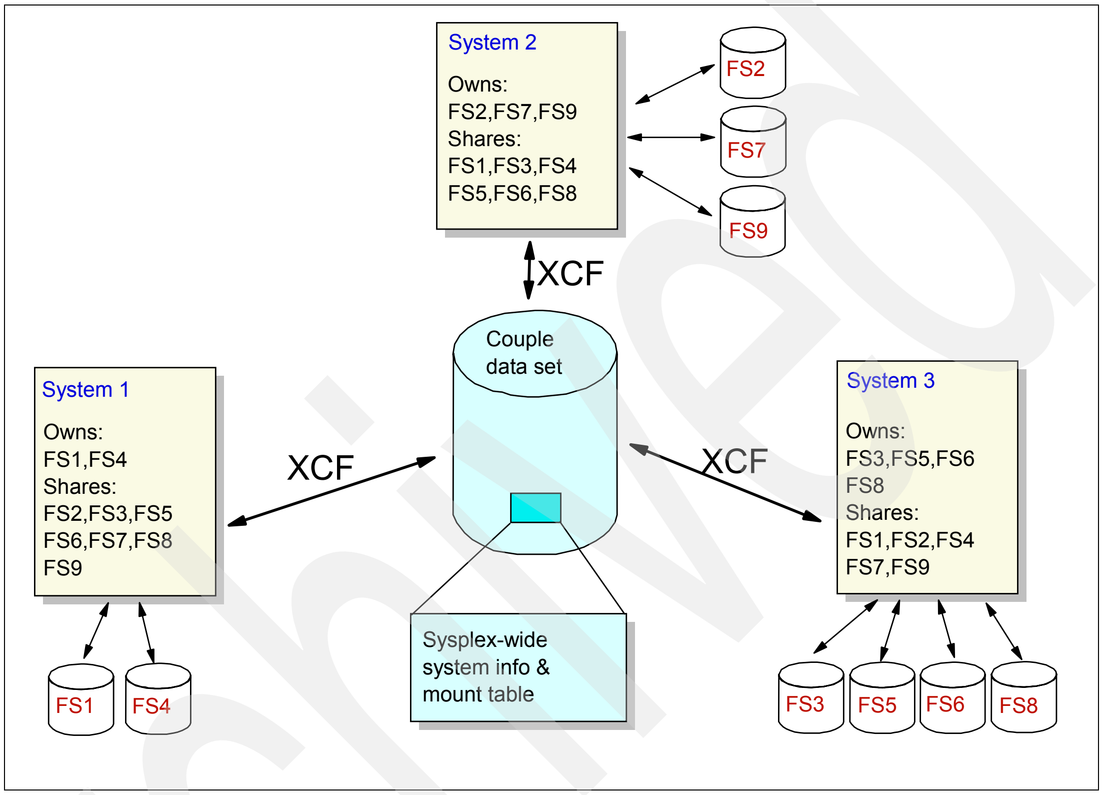


Figure 6-18 Shared USS couple data set

New format level of the BPXMCDS couple data set

The type BPCMCDS couple data set has changed in z/OS V1.4 to hold additional data to be used for enhanced shared USS algorithms. The new format level (version 2) is created when the OMVS couple data set is formatted using a version of IXCL1DSU at z/OS V1R4 or higher.

The initial or base z/OS UNIX couple data set format level (version 1) is created when the OMVS couple data set is formatted using a version of IXCL1DSU prior to z/OS V1R4.

Systems running z/OS V1.4 and systems of previous releases of z/OS or OS/390 can coexist in a sysplex with a version 2 format BPXMCDS couple data set.

For more information about the BPXMCDS couple data set and the changes in z/OS V1.4, see *z/OS UNIX System Services Planning,* GA22-7800 and *z/OS MVS Setting Up a Sysplex,* SA22-7625.

6.3.3 USS file system recovery

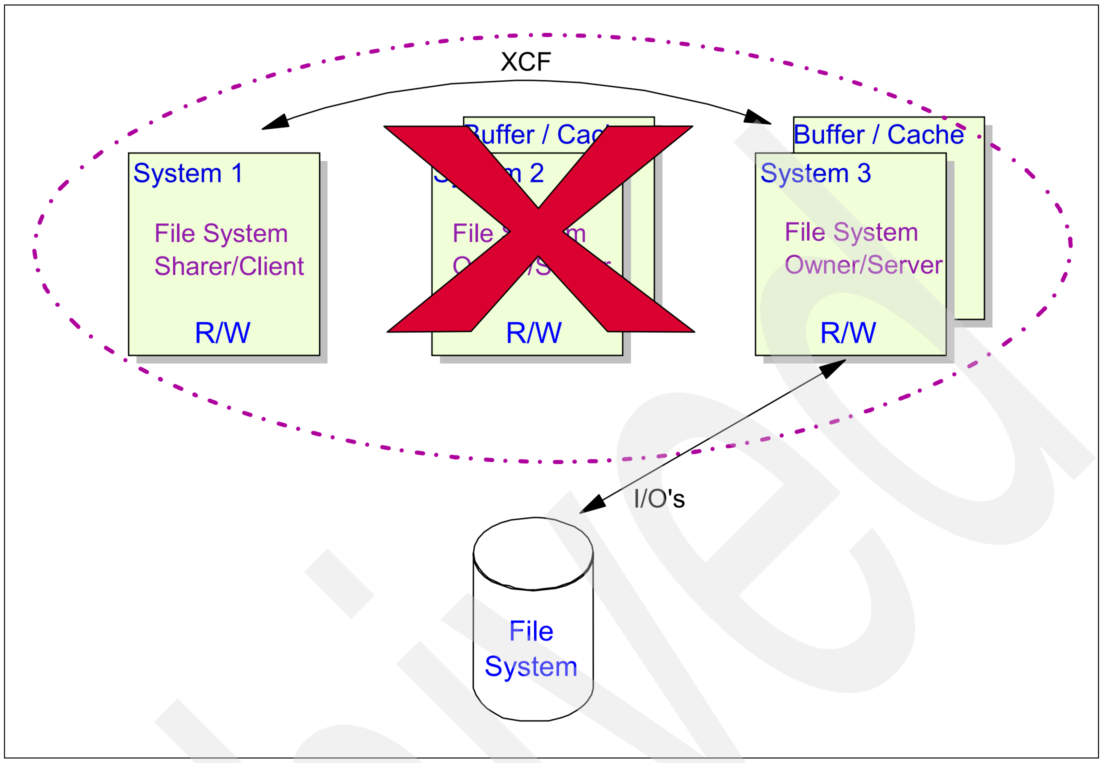


Figure 6-19 USS file system recovery

File system recovery in a shared HFS environment takes into consideration file system specifications such as AUTOMOVE, NOAUTOMOVE, UNMOUNT, and whether or not the file system is mounted read-only or read-write.

Generally, when an owning system fails, ownership over its automove-mounted file system is moved to another system and the file is usable. However, if a file system is mounted read-write and the owning system fails, then all file system operations for files in that file system fail. This happens because data integrity is lost when the file system owner fails. All files should be closed (BPX1CLO) and reopened (BPX1OPN) when the file system is recovered.

For file systems that are mounted read-only, specific I/O operations that were in progress at the time the file system owner failed may need to be started again.

In some situations, even though a file system is mounted AUTOMOVE, ownership of the file system may not be immediately moved to another system. This may occur, for example, when a physical I/O path from another system to the volume where the file system resides is not available. As a result, the file system becomes unowned; if this happens, you will see message BPXF213E.

This is true if the file system is mounted either read-write or read-only. The file system still exists in the file system hierarchy so that any dependent file systems that re owned by another system are still usable. However, all file operations for the unowned file system will fail until a new owner is established. The shared HFS support continues to attempt recovery of AUTOMOVE file systems on all systems in the sysplex that are enabled for shared HFS. Should a subsequent recovery attempt succeed, the file system transitions from the unowned to the active state.

File systems that are mounted NOAUTOMOVE or UNMOUNT become unowned when the file system owner exits the sysplex. The file system remains unowned until the original owning system restarts or until the unowned file system is unmounted. Because the file system still exists in the file system hierarchy, the file system mount point is still in use.

An unowned file system is a mounted file system that does not have an owner. The file system still exists in the file system hierarchy. As such, you can recover or unmount an unowned file system.

6.4 Automove system list

z/OS V1R4 includes the capability to specify a prioritized automove system list to indicate which system will become the new owner for a file system in a shared file system environment, in the event of a loss of the owning system. In the shared sysplex environment shown in Figure 6-20, system SC63 is the owning system of the zFS file system OMVS.CMP01.ZFS.

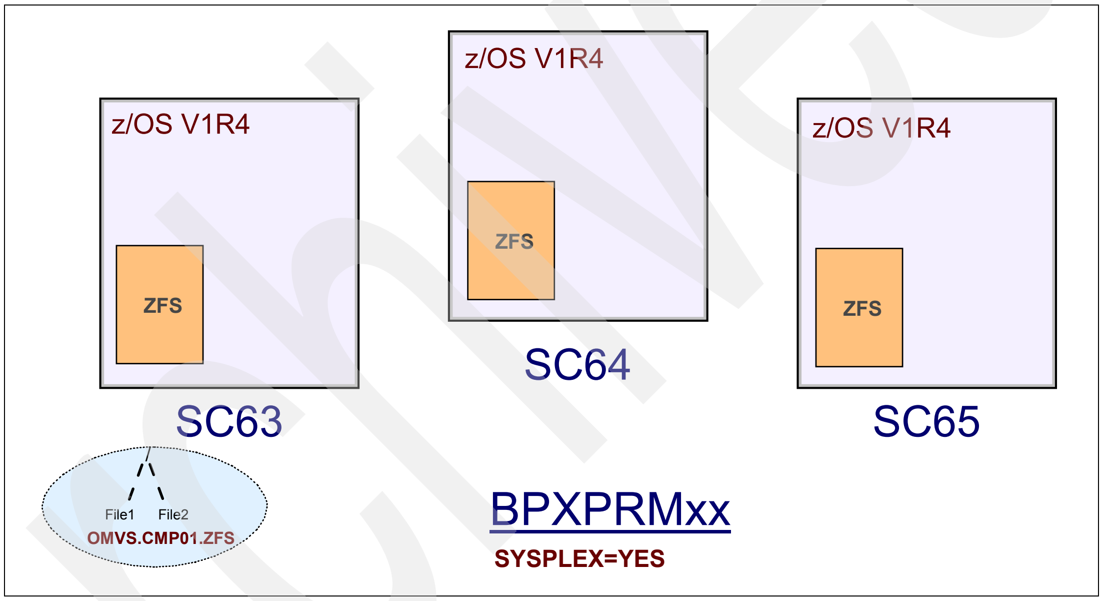


Figure 6-20 Shared file system sysplex environment

The automove system list is defined using the AUTOMOVE parameter in any one of the following methods of mounting a file system:

- BPXPRMXX parmlib member MOUNT statement or TSO/E MOUNT command
- Shell command
- ISHELL panels
- C program, assembler program, or REXX program

The system list can be changed for a file system after it has been mounted and can also be displayed.

6.4.1 Automove system list specification

The automove system list can be specified in many different ways to automove a file system. The list begins with an indicator to either include or exclude, followed by a list of system names. The indicator can be abbreviated as “i” or “e”.

Specify the indicator as follows:

i Use with a system list to provide a prioritized list of systems to which the file system may be moved if the owning system goes down. The list of systems is in priority order and if none of the systems specified in the list can take over as the new owner, the file system will be unmounted.

e Use with a system list to provide a list of systems to which the file system may not be moved.

Note: It is not possible to define an include and exclude list at the same time for the same file system. One of the options will override the previously defined option.

BPXPRMxx parmlib specification

A new operand has been added to the AUTOMOVE keyword on the MOUNT statement, as shown in Figure 6-21.

```
AUTOMOVE(indicator,name1,name2,...,nameN)
```

```
mount filesystem(omvs.test1.hfs) mountpoint('/tmp/test1') type(hfs) mode(rdwr)
automove(i,sc64,sc65)
```

Figure 6-21 Mount statement showing the new AUTOMOVE options

Note: The automove system list is optional. If not specified, the system that will become the new server is randomly chosen.

Mount shell command

The **mount** command issued from an OMVS shell session has been modified to include an automove system list specification, as follows:

```
mount [-t fstype] [-rv] [-a yes|include,sysname1,... sysnameN |exclude,sysname1,...
sysnameN |no|unmount] [-o fsoptions] [-d destsys] [-s nosecurity|nosetuid] -f fsname
pathname
```

An example of the new option:

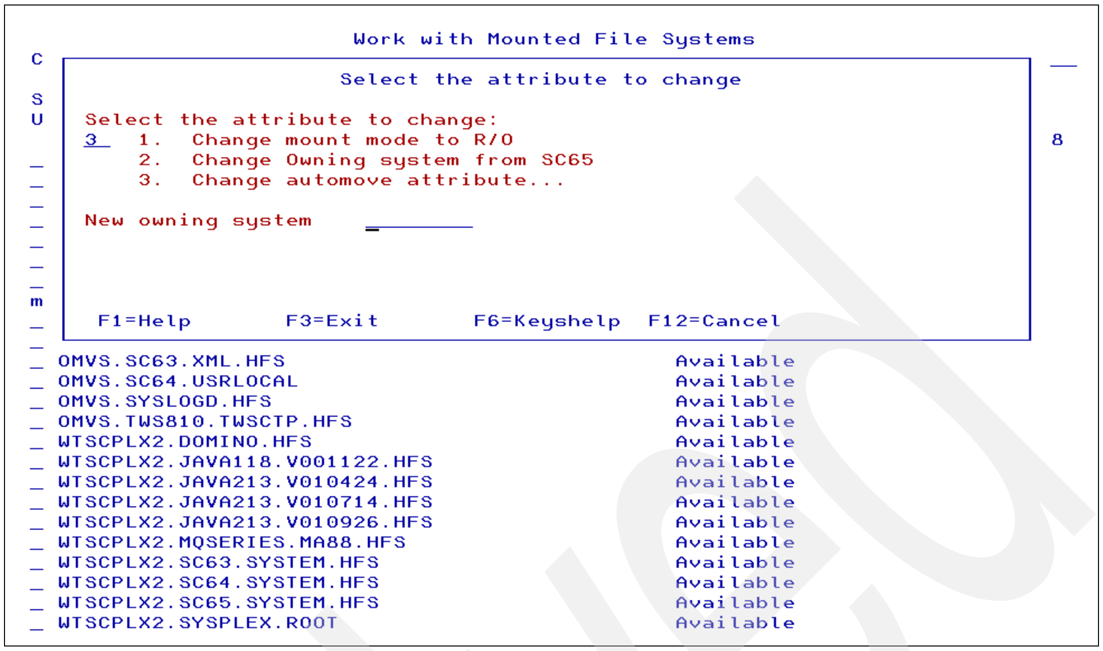
```
mount -a i,SC64,SC65 -f OMVS.CMP01.ZFS /tmp/test
```

Note: In z/OS V1R6 the mount utility is enhanced with the -v verbose option. If -v is specified on the mount command and the mount fails, the file system name that had the mount failure will be included in the failure information.

ISHELL panels

The ISHELL panel for mounts allows a selection to set the automove attribute. If selected, it displays a new panel to choose the automove type and specify a list of up to 32 systems (refer to Figure 6-23 on page 258). This panel is accessed from the Main ISHELL Panel; select **File_systems** → **Option 1 - Mount Table** → **Modify**.

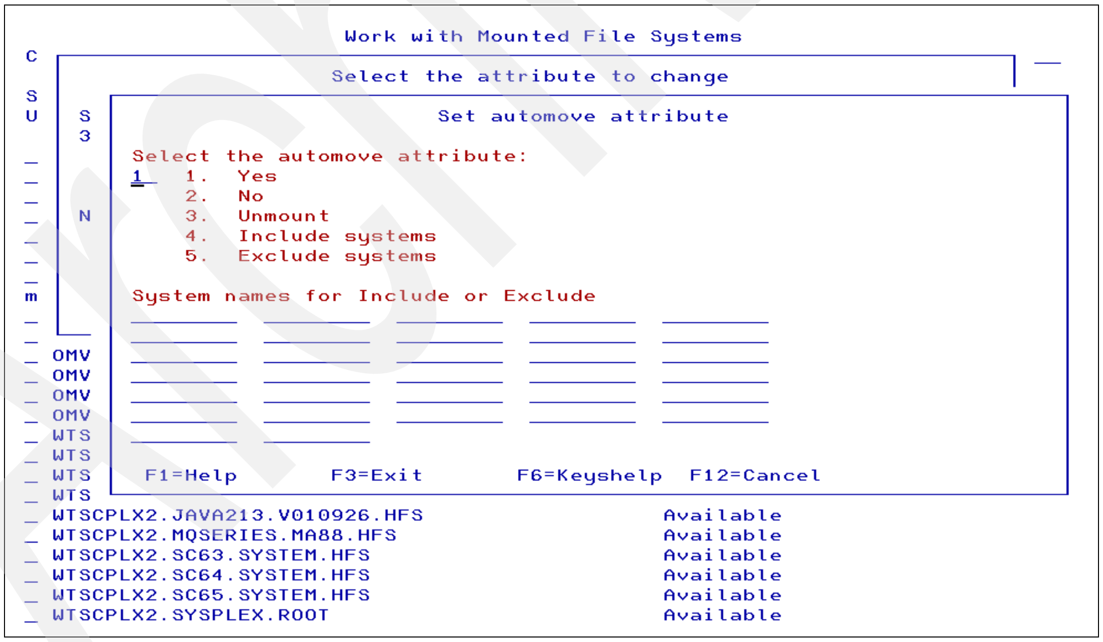
When you place an `M` for modify next to a mounted file system, the “Select the attribute to change” window is displayed; see Figure 6-22. This window is modified in z/OS V1R4, with Option 3 being new and replacing Option 3 and 4 from the previous releases.



```
                        Work with Mounted File Systems
C  ┌─────────────────────────────────────────────────────────────┐ __
   │                 Select the attribute to change              │
S  │                                                             │
U  │ Select the attribute to change:                             │
   │ 3_   1.  Change mount mode to R/O                           │ 8
_  │      2.  Change Owning system from SC65                     │
_  │      3.  Change automove attribute...                       │
_  │                                                             │
_  │ New owning system      _________                            │
_  │                                                             │
_  │                                                             │
_  │                                                             │
m  │                                                             │
_  │  F1=Help       F3=Exit       F6=Keyshelp   F12=Cancel       │
_  └─────────────────────────────────────────────────────────────┘
_ OMVS.SC63.XML.HFS                                  Available
_ OMVS.SC64.USRLOCAL                                 Available
_ OMVS.SYSLOGD.HFS                                   Available
_ OMVS.TWS810.TWSCTP.HFS                             Available
_ WTSCPLX2.DOMINO.HFS                                Available
_ WTSCPLX2.JAVA118.V001122.HFS                       Available
_ WTSCPLX2.JAVA213.V010424.HFS                       Available
_ WTSCPLX2.JAVA213.V010714.HFS                       Available
_ WTSCPLX2.JAVA213.V010926.HFS                       Available
_ WTSCPLX2.MQSERIES.MA88.HFS                         Available
_ WTSCPLX2.SC63.SYSTEM.HFS                           Available
_ WTSCPLX2.SC64.SYSTEM.HFS                           Available
_ WTSCPLX2.SC65.SYSTEM.HFS                           Available
_ WTSCPLX2.SYSPLEX.ROOT                              Available
```

Figure 6-22 Select the attribute to change window

Selecting Option 3 displays the new window shown in Figure 6-23, which allows you to create or modify an automove system list. Select option 4 or 5 to either include or exclude the systems you specify by system names.



```
                        Work with Mounted File Systems
C  ┌──────────────────────────────────────────────────────────────┐ __
   │                 Select the attribute to change               │
S  │   ┌──────────────────────────────────────────────────────────────┐
U  │ S │                    Set automove attribute                    │
   │ 3 │                                                              │
_  │   │ Select the automove attribute:                               │
_  │   │ 1_   1.  Yes                                                 │
_  │   │      2.  No                                                  │
_  │ N │      3.  Unmount                                             │
_  │   │      4.  Include systems                                     │
_  │   │      5.  Exclude systems                                     │
_  │   │                                                              │
m  │   │ System names for Include or Exclude                          │
_  │   │ ________  ________  ________  ________  ________             │
_  └── │ ________  ________  ________  ________  ________             │
_ OMV  │ ________  ________  ________  ________  ________             │
_ OMV  │ ________  ________  ________  ________  ________             │
_ OMV  │ ________  ________  ________  ________  ________             │
_ OMV  │ ________  ________  ________  ________  ________             │
_ WTS  │ ________  ________                                           │
_ WTS  │                                                              │
_ WTS  │  F1=Help       F3=Exit       F6=Keyshelp   F12=Cancel         │
_ WTS  └──────────────────────────────────────────────────────────────┘
_ WTSCPLX2.JAVA213.V010926.HFS                       Available
_ WTSCPLX2.MQSERIES.MA88.HFS                         Available
_ WTSCPLX2.SC63.SYSTEM.HFS                           Available
_ WTSCPLX2.SC64.SYSTEM.HFS                           Available
_ WTSCPLX2.SC65.SYSTEM.HFS                           Available
_ WTSCPLX2.SYSPLEX.ROOT                              Available
```

Figure 6-23 Set the automove attribute window

C program

Using the callable service _mount(BPX2MNT), the syslist and indicator may be specified in the MNTE as input.

6.4.2 Changing an automove system list

After a file system is mounted, the AUTOMOVE attribute can be changed by using one of the following commands:

- The **setomvs** command

  ```
  SETOMVS FILESYS,FILESYSTEM=filesystem,AUTOMOVE=YES | NO| UNMOUNT |
  indicator(sysname1,sysname2,...,sysnameN)
  ```

  ```
  setomvs filesys,filesystem='omvs.cmp01.zfs',automove=e(sc64)
  ```

- The **chmount** shell command

  ```
  chmount [-R][-r][-w][-D | -d destsys] [-a yes | no | unmount |
  indicator,sysname1,..., sysnameN] pathname
  ```

  ```
  chmount -a i,SC64,SC65 /tmp/test
  ```

Commands to display the automove system list

Console and shell display commands have been modified in order to give information about the automove system list.

- The **d omvs,file** command displays the system list and the indicator, if the system list has been defined; see Figure 6-24.

```
d omvs,file
HFS           126 ACTIVE                      RDWR
  NAME=OMVS.CMP01.ZFS
  PATH=/SC63/tmp/test1
  OWNER=SC63 AUTOMOVE=I CLIENT=N
  INCLUDE SYSTEM LIST:  SC64     SC65
```

Figure 6-24 z/OS command to display the automove system list

- The **df -v** command provides system list information and the indicator, if the system list has been defined, see Figure 6-25.

```
@ SC65:/>df -v /tmp/test1
Mounted on      Filesystem                Avail/Total    Files      Status
/SC65/tmp/test1 (OMVS.TEST1.HFS)           14224/14400    4294967294 Available
HFS, Read/Write, Device:126, ACLS=Y
File System Owner : SC65        Automove=I      Client=N
System List (Include) : SC64       SC63
Filetag : T=off   codeset=0
```

Figure 6-25 OMVS shell command to display the automove system list

- F BPXOINIT,FILESYS=DISPLAY,ALL has been modified to display syslist information; see Figure 6-26 on page 260.

```
OMVS.TEST1.HFS                                        126  RDWR
  PATH=/SC63/tmp/test1
  STATUS=ACTIVE                  LOCAL STATUS=ACTIVE
  OWNER=SC63        RECOVERY OWNER=SC63     AUTOMOVE=I PFSMOVE=Y
  TYPENAME=HFS      MOUNTPOINT DEVICE=      72
  MOUNTPOINT FILESYSTEM=/SC63/TMP
  ENTRY FLAGS=91000000  FLAGS=40000000  LFSFLAGS=00000000
  LOCAL FLAGS=40000000  LOCAL LFSFLAGS=20000000
  SYSLIST STS=00000000  SYSLIST VALID=00000000
  INCLUDE SYSTEM LIST (2 SYSTEM(S) IN LIST):
    SC64     SC65
```

Figure 6-26   z/OS command to display the automove system list

6.5 Showing all USS file sharing structures for a system

Figure 6-27 shows, for reference, all the USS file structures that are used in a USS sysplex sharing environment.

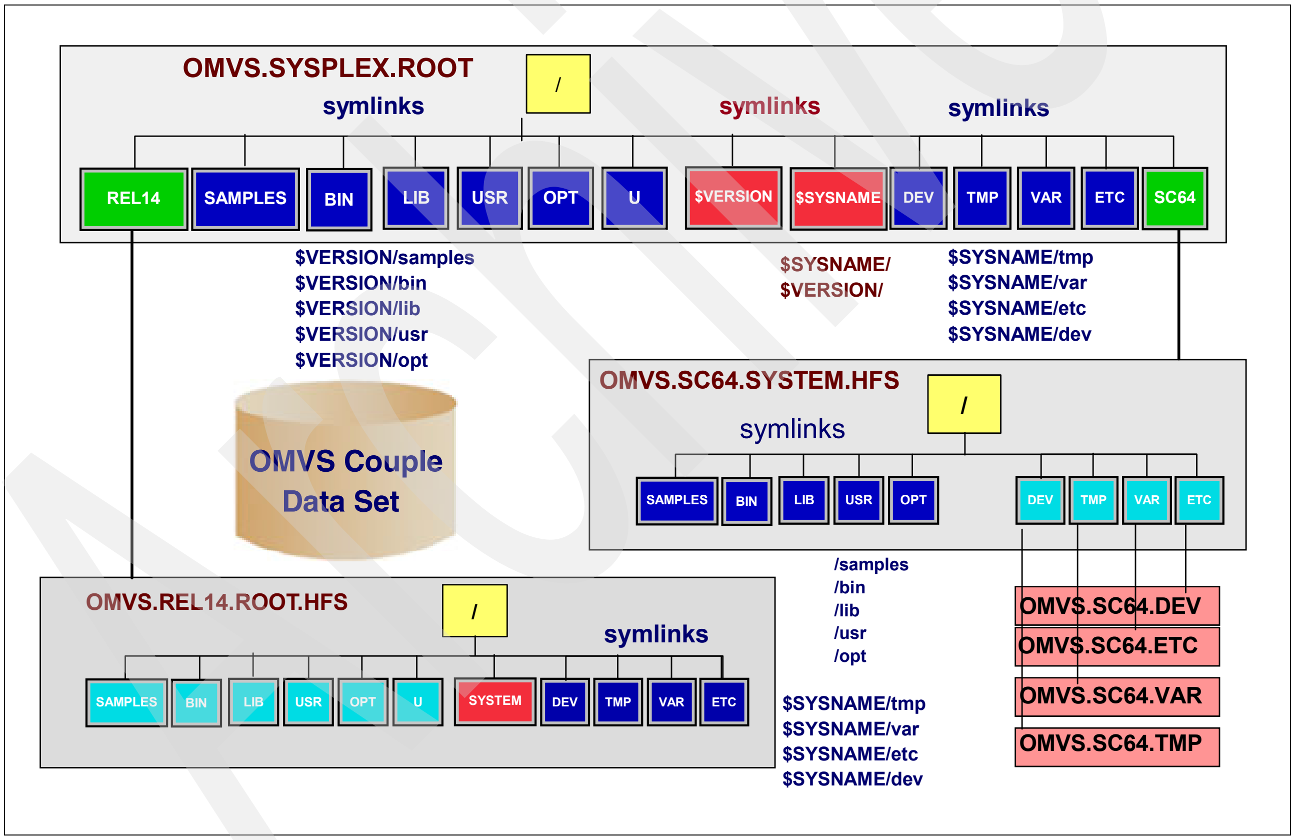


Figure 6-27   All the USS file sharing structures for a system

6.6 USS file system sharing implementation

Here we provide information on what needs to be done and list the sample jobs and commands that you can adapt for your system environment. If you need to see all the single steps together with the results, see the redbook *Hierarchical File System Usage Guide,* SG24-5482.

6.6.1 Creating and defining the USS couple data sets

In this step you define the data structure named BPXMCDS. This is used by XCF to maintain all the information to support USS file system sharing across the sysplex. Figure 6-28 shows the BPXISCDS sample job in SYS1.SAMPLIB.

```
//BPXISCDS JOB
...
//STEP10   EXEC PGM=IXCL1DSU
//STEPLIB  DD   DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
  /* Begin definition for OMVS couple data set (1)                */
     DEFINEDS SYSPLEX(PLEX1)           /* Name of the sysplex in
                                          which the OMVS couple data
                                          set is to be used.         */
              DSN(SYS1.OMVS.CDS01) VOLSER(3390X1) /* The name and
...
              MAXSYSTEM(8)             /* Number of systems in the
                                          sysplex to be supported by
...
              NOCATALOG                /* Default is not to CATALOG.
                                                             §01C*/
          DATA TYPE(BPXMCDS)           /* The type of data in the
...
               ITEM NAME(MOUNTS) NUMBER(500) /* Specifies the number of
                                          MOUNTS that can be supported
                                          by OMVS.
...
               ITEM NAME(AMTRULES) NUMBER(50) /* Specifies the number
                                          of automount rules that can
                                          be supported by OMVS.
...
  /* Begin definition for OMVS couple data set (2)                */
     DEFINEDS SYSPLEX(PLEX1)           /* Name of the sysplex in
...
              DSN(SYS1.OMVS.CDS02) VOLSER(3390X2) /* The name and
...
              MAXSYSTEM(8)             /* Number of systems in the
...
              NOCATALOG                /* Default is not to CATALOG.
                                                             §01C*/
          DATA TYPE(BPXMCDS)           /* The type of data in the
...
               ITEM NAME(MOUNTS) NUMBER(500) /* Specifies the number of
...
               ITEM NAME(AMTRULES) NUMBER(50) /* Specifies the number
...
/*
```

Figure 6-28 Sample job BPXISCDS

The job creates two CDS data sets; one is the primary and the other is a backup that is referred to as the *alternate*. You need to specify the maximum number of mounts and automount rules that are allowed in this USS sharing environment. Furthermore, the maximum number of systems must be defined.

Then you need to update the active COUPLExx parmlib member to make the names of the primary and secondary couple data sets known to XCF, as shown in Figure 6-29.

```
/* For all systems in any combination, up to an eightway  */
COUPLE INTERVAL(60)                    /* 1 minute */
   OPNOTIFY(60)                        /* 1 minute */
   SYSPLEX(PLEX1)                      /* SYSPLEX NAME*/
   PCOUPLE(SYS1.PCOUPLE,CPLPKP)        /* COUPLE DS */
   ACOUPLE(SYS1.ACOUPLE,CPLPKA)        /* ALTERNATE DS*/
   MAXMSG(750)
   RETRY(10)
DATA TYPE(CFRM)
   PCOUPLE(SYS1.PFUNCT.CTTEST,FDSPKP)
   ACOUPLE(SYS1.AFUNCT.CTTEST,FDSPKA)
DATA TYPE(BPXMCDS)
   PCOUPLE(SYS1.OMVS.CDS01,3390x1)
   ACOUPLE(SYS1.OMVS.CDS02,3390x2)
/* CTC DEFINITIONS: ALL SYSTEMS  */
PATHOUT DEVICE(8E0)
PATHIN DEVICE(CEF)
```

Figure 6-29 COUPLExx parmlib member

You can use the following commands to dynamically add the OMVS couple data sets to XCF:

```
SETXCF COUPLE,TYPE=BPMCDS,PCOUPLE=SYS1.OMVS.CDS01
SETXCF COUPLE,TYPE=BPMCDS,ACOUPLE=SYS1.OMVS.CDS02
```

You may check whether they are now known by each system:

```
D XCF,COUPLE,TYPE=BPXMCDS
```

6.6.2 Creating the USS sysplex root file system

The sysplex root is created by invoking the BPXISYSR sample job in SYS1.SAMPLIB. This file is shown in Figure 6-30.

```
//BPXISYSR JOB <JOB CARD PARAMETERS>
...
//IKJEFT1A  EXEC PGM=IKJEFT1A,PARM='BPXISYS1'
//*
//ROOTSYSP  DD DSNAME=OMVS.SYSPLEX.ROOT,
//          DISP=(,CATLG),
//          DSNTYPE=HFS,
//          SPACE=(CYL,(2,0,1)),
//          STORCLAS=storclas
//*         UNIT=uuuu,VOL=SER=vvvvvv
//*
//SYSEXEC   DD DSN=SYS1.SAMPLIB,DISP=SHR,
//          UNIT=SYSALLDA,VOL=SER=tvol2
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DUMMY
//*
```

Figure 6-30 Sample job BPXISYSR

You can customize the JCL and run it in superuser mode. REXX BPXISYS1 is called to create the necessary directories and symbolic links in the root structure.

6.6.3 Creating the USS system-specific file system

To create the system-specific file systems, run the sample job BPXISYSS in SYS1.SAMPLIB separately on each system that will participate in USS sysplex sharing.

```
//BPXISYSS JOB <JOB CARD PARAMETERS>
...
//IKJEFT1A  EXEC PGM=IKJEFT1A,PARM='BPXISYS2'
//*
//HFSSYSTS  DD DSNAME=OMVS.sysname.HFS,
//          DISP=(,CATLG),
//          DSNTYPE=HFS,
//          SPACE=(CYL,(2,0,1)),
//          STORCLAS=storclas
//*         UNIT=uuuu,VOL=SER=vvvvvv
//*
//SYSEXEC   DD DSN=SYS1.SAMPLIB,DISP=SHR,
//          UNIT=SYSALLDA,VOL=SER=tvol2
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DUMMY
//*
```

Figure 6-31 Sample job BPXISYSS

Again, you can customize the JCL and run it in superuser mode. The job runs REXX BPXISYS2 to create directories and symbolic links.

6.7 Additional notes and comments

In order to avoid unpredictable results or confusing system behavior we recommend that you use consistent automount policies on all systems participating in the USS sysplex sharing environment.

Note that the implementation of USS sysplex sharing does not use CFs, but only CDS and XCF.

In general we recommend to use the same filesystype statements in all systems. Otherwise you may not have the necessary consistent file system structure view on all systems. In particular, there will be a hole (missing structures) in a system if the filesystype of one of the file systems mounted in the sysplex is not active in that system.

6.7.1 Using TFS filesystype in a colony address space

However, we know one situation where it may be useful not to follow this rule. APAR OW43826 describes a problem in the UNIX logical file system code for file system types living in a colony address space. A colony address space is just an address space outside of OMVS. This has the advantage that this file system type can be stopped without impact to OMVS.

If the filesystype is deactivated in one system in a USS sysplex sharing environment, this causes all file systems of that type to be unmounted in that system, of course. The big trouble is that this unmount processing is done for all file systems of that type in the whole sysplex. This is, of course, undesirable.

This problem has been solved in z/OS V1.2, but we never got a PTF for OS/390 or z/OS V1.1. This means that you still suffer this problem as long as you have at least one system active in your sharing environment that is older than z/OS V1.2.

A TFS file system can be set up to be located in the OMVS address space or a colony address space. The latter is the better choice. TFS is intended to be used for temporary data like the /tmp structure.

If you run with USS sysplex sharing and do not have all the systems at level z/OS V1.2, at least you can circumvent the problem by using different filesystype names on different systems. The fact that all these TFS file systems are seen only locally does not hurt because they are used in the system-specific structure and are not needed on any other system.

Figure 6-32 lists an excerpt of parmlib member BPXPRMxx.

```
...
FILESYSTYPE TYPE(&SYSNAME.TS) ENTRYPOINT(BPXTFS) ASNAME(&SYSNAME.TS)
...
MOUNT
FILESYSTEM('/&SYSNAME./TMP')
TYPE(&SYSNAME.TS) MODE(RDWR) NOAUTOMOVE SYSNAME(&SYSNAME)
 PARM('-s 512')
 MOUNTPOINT('/&SYSNAME./tmp')
...
```

Figure 6-32 Using different filesystypes for TFS on different systems

The filesystype and mount statements used meet these requirements and are suitable if the parmlib member is used by all the systems of the sysplex. Use the **chmod** command shown in Figure 6-33 to make the temp structure available to all users in a reasonable manner. The lines listed can be put to /etc/rc, for example.

```
# Allow only file owner to remove files from /tmp/
chmod 1777 /tmp/
```

Figure 6-33 Setting permission bits for TFS /tmp/ structure

Note: Using /tmp/ is a valid reference to address directory /&SYSNAME./tmp without the need to include the system string into the command.

6.8 Effects of USS sysplex sharing

USS file system sharing affects file I/O, but there is no I/O performance reduction for USS file systems mounted read only (R/O) because in this situation every system continues to physically access the data set.

If a file system is mounted for read and write (R/W) on system SYS2 and a user on SYS1 works with this file system structure, he may suffer the following effects:

- Additional path length (locally, data is often available from cache)

- Latency caused by involvement of the XCF messaging function

Tip: File systems should be mounted on the system where they are most heavily used.

You may see increased XCF message traffic. Therefore, we recommend that you:

- Monitor the number and size of message buffers and the number and performance of XCF message paths in the sysplex.
- Add additional XCF paths and XCF message buffers if necessary.
- Look to the USS sharing XCF traffic recorded in XCF via group name SYSBPX.

6.8.1 How USS sysplex sharing affects mount times

Additional overhead is incurred through I/O to the USS-specific couple data set (CDS). The mount time increases as a function of the following three parameters: the number of mounts, the number of systems participating in the USS sysplex sharing, and the size of your CDS.

Following we list how and why these parameters influence the mount time:

- Number of mounts
  - When a system joins a sysplex, mount information is written to CDS.
  - Mounts are performed serially, one at a time.
  - Already active mounts must be read first and performed when a new system IPLs.
  - New mounts must be processed by the other systems in the plex, too.
  - The next mount is possible when the last one is completely processed on all systems.
- Number of members in the sysplex
  - The more systems that are active in the sysplex, the more mounts need to be recorded in the USS CDS.
  - There is competition among the systems to read and perform the new mounts written to the CDS.
  - There is the possibility of increased mount response time on a system writing new mounts to the CDS.
- The size of your CDS
  - A system reading the CDS for mount information must read the entire CDS.
  - All systems in the sysplex must do that and read through the entire CDS.

Attention: It is important to specify an appropriate number of mounts when formatting the USS CDS.

6.9 Shared HFS unmount option

Previous to z/OS V1R3, file systems could be mounted as AUTOMOVE and NOAUTOMOVE. If AUTOMOVE is specified, the file system is moved to another system in the event that the owning system is taken down. If NOAUTOMOVE is specified, the file system remains mounted when the owning system goes down, but the file system now has an unknown owner, as shown in Figure 6-34.

```
HFS           445 UNOWNED                     RDWR
  NAME=WTSCPLX2.SC64.SYSTEM.HFS
  PATH=/SC64
  OWNER=         AUTOMOVE=N CLIENT=Y
```

Figure 6-34 Display of file system showing unknown owner

When the failed system reinitializes, the file system recovers and becomes active again.

6.9.1 UNMOUNT option

In z/OS V1R4 a new UNMOUNT option was added in order to unmount file systems associated with a failed system. This allows for file systems that are required or desirable to not move to another system to be unmounted. This avoids either recovering or converting them to “unowned” status. Therefore, the options now are:

```
AUTOMOVE|NOAUTOMOVE|UNMOUNT
```

AUTOMOVE	Specifies that ownership of the file system is automatically moved to another system. It is the default.
NOAUTOMOVE	Specifies that the file system will not be moved if the owning system goes down and the file system is not accessible.
UNMOUNT	Specifies that the file system will be unmounted when the system leaves the sysplex. This option is not available for automounted file systems.

AUTOMOVE

The AUTOMOVE parameter is only applicable to sysplex environments and can be used in a number of ways. Its main purpose is to move the file system to another system when the current owner goes down.

In z/OS V1R4 an INCLUDE indicator was provided to control a list of systems that would be used if an owner system should leave the sysplex.

```
AUTOMOVE(I,sysname1,sysname2,...,sysnameN)
```

To achieve the opposite, the EXCLUDE parameter can be used. This excludes one or more systems from being used as candidates to move file systems to.

```
AUTOMOVE(E,sysname1,sysname2,...,sysnameN)
```

When all systems in the sysplex are at the level of z/OS V1R6, there is an additional wildcard option available if you want to use AUTOMOVE system lists. It is permitted to be the last item of the INCLUDE system list.

```
AUTOMOVE(INCLUDE,SC64,*)
```

It provides you with the ability to prioritize the list of system names, then the rest of the systems are included by adding the wildcard character at the end. It means less work to type in all the system names, and also reduces the number of typing errors. The wildcard support is especially helpful when you have a large number of systems participating in a sysplex.

UNMOUNT

The UNMOUNT option is supported in the following ways:

- BPXPRMxx parmlib MOUNT statement

The AUTOMOVE | NOAUTOMOVE | UNMOUNT parameters on the ROOT and MOUNT statements indicate what happens to the file system if the system that owns that file system goes down.

The TSO/E MOUNT command

```
MOUNT filesystem(OMVS.HFS1.HFS) mountpoint('/u/vivarhfs') type(HFS) mode(rdwr)
UNMOUNT
```

- The shell **`mount`** command

```
mount [-t fstype][-rv][-a yes|no|unmount][-o fsoptions][-d destsys][-s
nosecurity|nosetuid] -f fsname pathname
```

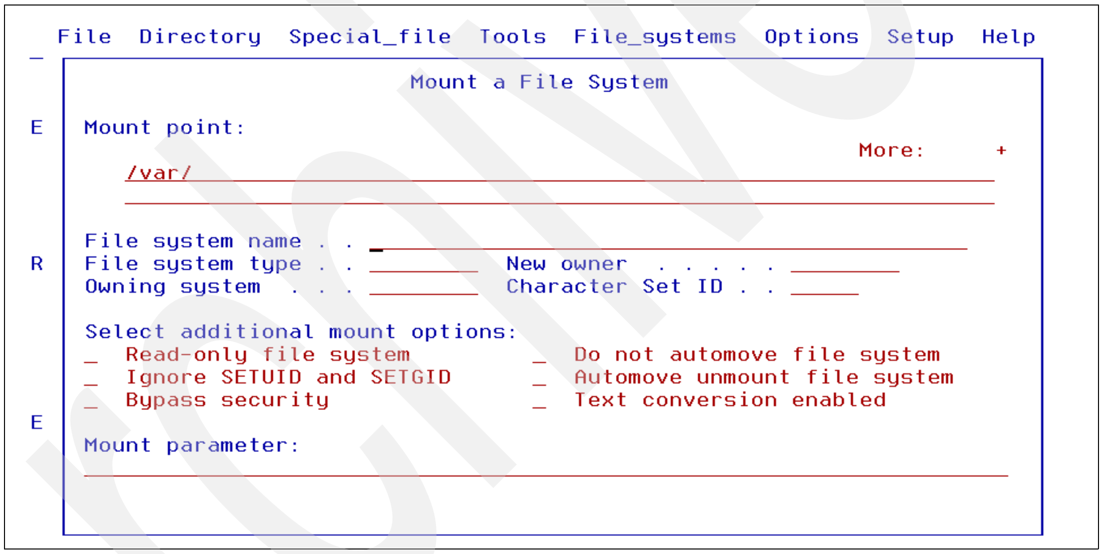
- The SETOMVS command

```
SETOMVS FILESYS,FILESYS=filesystem,AUTOMOVE=YES|NO|UNMOUNT
```

- The shell **`chmount`** command

```
chmount [-R [-D |-d destsys][-a yes|no|unmount]pathname...
```

- The ISHELL mount interface in the Mount File System panel, as shown in Figure 6-35 on page 267, is accessed by **ISHELL panel** → **File System pulldown Menu** → **Option 3 - Mount)**. The new mount option is **Automove unmount file system**.



```
   File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
 _
                          Mount a File System

 E    Mount point:
                                                              More:     +
         /var/


      File system name  . .  _
 R    File system type  . .  ________  New owner  . . . . .  ________
      Owning system   . . .  ________  Character Set ID . .  _____

      Select additional mount options:
      _  Read-only file system           _  Do not automove file system
      _  Ignore SETUID and SETGID        _  Automove unmount file system
      _  Bypass security                 _  Text conversion enabled
 E
      Mount parameter:
```

Figure 6-35 Mount file system panel

Display command changes

The following display commands have been modified to include information about the UNMOUNT option:

- D OMVS,F includes a “U” character in the AUTOMOVE field for UNMOUNT mounted file systems, as shown in Figure 6-36.

```
HFS            422 ACTIVE                          RDWR
  NAME=OMVS.TESTCD.HFS
  PATH=/TESTC/TESTCD
  OWNER=SC64      AUTOMOVE=U CLIENT=N
```

Figure 6-36 Display to show the UNMOUNT option for an HFS data set

- The shell **`df -v`** command

```
Mounted on      Filesystem                Avail/Total    Files      Status
/TESTC          (OMVS.TESTC.HFS)          14192/14400    4294967291 Available
HFS, Read/Write, Device:478, ACLS=Y
File System Owner : SC64        Automove=U      Client=N
Filetag : T=off   codeset=0
```

Figure 6-37   df -v command output to show the unmount option

- F BPXOINIT,FILESYS=D,ALL and F BPXOINIT,FILESYS=D,FILESYSTEM=filesystem commands

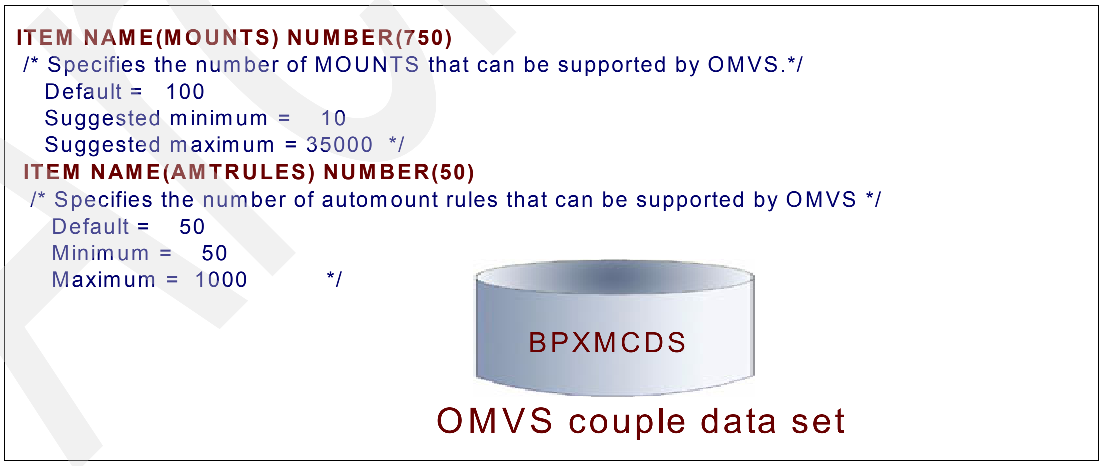
```
OMVS.TESTC.HFS                                         478   RDWR
  PATH=/TESTC
  STATUS=ACTIVE                 LOCAL STATUS=ACTIVE
  OWNER=SC64        RECOVERY OWNER=SC64      AUTOMOVE=U PFSMOVE=Y
  TYPENAME=HFS      MOUNTPOINT DEVICE=       1
  MOUNTPOINT FILESYSTEM=WTSCPLX2.SYSPLEX.ROOT
  ENTRY FLAGS=90000000  FLAGS=40000018  LFSFLAGS=00000000
  LOCAL FLAGS=40000018  LOCAL LFSFLAGS=20000000
```

Figure 6-38   BPXOINIT commands to display the UNMOUNT option

6.10 Mount table limit monitoring

In previous releases, users needed the capability to determine when the number of file system mounts in a shared HFS was approaching the configured limit. Before z/OS V1R3, there was no way to easily determine when the mount limit, specified in the BPXMCDS CDS shown in Figure 6-39, was being approached.

z/OS V1R3 introduces the possibility to monitor the shared HFS mount limits, specified in the CDS, by issuing a console message when the limit has almost been reached.



```
ITEM NAME(MOUNTS) NUMBER(750)
 /* Specifies the number of MOUNTS that can be supported by OMVS.*/
   Default =   100
   Suggested minimum =    10
   Suggested maximum = 35000  */
 ITEM NAME(AMTRULES) NUMBER(50)
  /* Specifies the number of automount rules that can be supported by OMVS */
    Default =    50
    Minimum =    50
    Maximum =  1000         */
```

Figure 6-39   Mount and automount entries for shared sysplex support

BPXMCDS couple data set

Shared HFS support uses a type BPXMCDS couple data set (CDS) to maintain data about mounted file systems in the sysplex configuration. The primary and alternate CDSs are formatted, using the IXCL1DSU utility, with a maximum number of mount entries as specified in the NUMBER value that specifies the number of mounts, as shown in Figure 6-40 on page 269.

```
//STEP10   EXEC PGM=IXCL1DSU,REGION=OM
//STEPLIB  DD   DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
  /* Begin definition for OMVS couple data set (1)                   */
     DEFINEDS SYSPLEX(SANDBOX)         /* Name of the sysplex in
                                          which the OMVS couple data
                                          set is to be used.         */
              DSN(SYS1.XCF.OMVS05) VOLSER(SBOX63) /* The name and
                                          volume for the OMVS
                                          couple data set. The
                                          utility will allocate a
                                          new data set by the name
                                          specified on the volume
                                          specified.                 */
              MAXSYSTEM(8)             /* Number of systems in the
                                          sysplex to be supported by
                                          this couple data set. Default
                                          value is eight.        @01A*/
CATALOG                  /* Default is not to CATALOG.
                                                                 @01C*/
          DATA TYPE(BPXMCDS)           /* The type of data in the
                                          data set being created is
                                          for OMVS. BPXMCDS is the
                                          TYPE for OMVS.             */
               ITEM NAME(MOUNTS) NUMBER(750) /* Specifies the number of
                                          MOUNTS that can be supported
                                          by OMVS.
                                          Default =   100
                                          Minimum =     1
                                          Maximum = 50000        @D1C*/
               ITEM NAME(AMTRULES) NUMBER(50) /* Specifies the number
                                          of automount rules that can
                                          be supported by OMVS.
                                          Default =    50
                                          Minimum =    50
                                          Maximum =  1000        @D1A*/
```

Figure 6-40 Job that creates the BPXMCDS couple data set

Displaying the mount table limit

Once the mount limit is reached, no more file systems can be mounted in the sysplex until a larger type BPXMCDS CDS is enabled. Mount table limit monitoring allows an installation to detect when a primary CDS is reaching its mount table limit in order to begin corrective actions before denial of service.

You can display the number of mount entries and the number in use by using the F BPXOINIT,FILESYS=DISPLAY,GLOBAL command, as shown in Figure 6-41 on page 270.

```
f bpxoinit,filesys=display,global
BPXF041I 2002/05/09 13.42.25 MODIFY BPXOINIT,FILESYS=DISPLAY,GLOBAL
221
SYSTEM   LFS VERSION ---STATUS--------------------- RECOMMENDED ACTION
SC64       1.  3.  1 VERIFIED                       NONE
SC63       1.  3.  1 VERIFIED                       NONE
SC65       1.  4.  1 VERIFIED                       NONE
CDS VERSION=  1        MIN LFS VERSION=  1.  3.  1
BRLM SERVER=N/A        DEVICE NUMBER OF LAST MOUNT=     706
MAXIMUM MOUNT ENTRIES=     500   MOUNT ENTRIES IN USE=     430
```

Figure 6-41 Display of mount table limits

BPXPRMxx parmlib member

Mount table limit monitoring is enabled by specifying the LIMMSG parameter in the BPXPRMxx parmlib member, or dynamically by using the SETOMVS command, with the values SYSTEM or ALL.

LIMMSG=SYSTEM Console messages are to be displayed for all processes that reach system limits. In addition, messages are to be displayed for each process limit of a process if:

The process limit or limits are defined in the OMVS segment of the owning user ID.

The process limit or limits have been changed with a SETOMVS PID=pid,proces_limit command.

LIMMSG=ALL In this case, console messages are to be displayed for the system limits and for the process limits, regardless of which process reaches a process limit.

For more information about the LIMMSG parameter, see *z/OS MVS System Commands,* SA22-7627.

Display the LIMMSG parameter

Both LIMMSG values were defined in previous z/OS releases, and you can monitor the current value of the LIMMSG parameter option using the D OMVS,LIMITS command, as shown in Figure 6-42 on page 271.

```
D OMVS,LIMITS
BPXO051I 10.43.10 DISPLAY OMVS 747
OMVS     000F ACTIVE          OMVS=(4A)
SYSTEM WIDE LIMITS:         LIMMSG=SYSTEM
                  CURRENT  HIGHWATER     SYSTEM
                    USAGE      USAGE      LIMIT
MAXPROCSYS             42         54        300
MAXUIDS                 1          2         50
MAXPTYS                 0          3        256
MAXMMAPAREA             0          0       4096
MAXSHAREPAGES           0          0   32768000
IPCMSGNIDS             10         10      20000
IPCSEMNIDS              0          0      20000
IPCSHMNIDS              0          0      20000
IPCSHMSPAGES            0          0    2621440
IPCMSGQBYTES          ---         72     262144
IPCMSGQMNUM           ---          6      10000
IPCSHMMPAGES          ---          0      25600
```

Figure 6-42 Display the BPXPRMxx parameter specifications

Mount table limit monitoring messages

Once the LIMMSG parameter is set to SYSTEM or ALL, a BPXI043E console message will be issued when the mount table limit reaches a critical value. The BPXI043E message has the following format:

```
BPXI043E MOUNT TABLE LIMIT HAS REACHED <limit_reached> OF ITS CURRENT CAPACITY OF
<current_limit>
```

Where:

<limit_reached>	Has the value of 85, 90, 95, or 100.
<current_limit>	Indicates the mount NUMBER value in the BPXMCDS CDS.

The message is updated when the percentage, limit_reached field, has changed to a new value (from 85 to 90, from 90 to 95, or from 95 to 100), as shown in Figure 6-43. The message is deleted when the percentage decreases below 85%.

```
*BPXI043E MOUNT TABLE LIMIT HAS REACHED 85% OF ITS CURRENT CAPACITY OF 500
*BPXI043E MOUNT TABLE LIMIT HAS REACHED 90% OF ITS CURRENT CAPACITY OF 500
*BPXI043E MOUNT TABLE LIMIT HAS REACHED 100% OF ITS CURRENT CAPACITY OF 500
BPXI045I THE PRIMARY CDS SUPPORTS A LIMIT OF 700 MOUNTS AND A LIMIT OF 50 AUTOMOUNT
RULES.
BPXI044I RESOURCE SHORTAGE FOR MOUNT TABLE HAS BEEN RELIEVED.
```

Figure 6-43 Mount table limit monitoring messages

Mount table limit procedure

When the BPXI043E message has been issued, you must begin corrective actions before the limit reaches the 100% limit, which will provoke denial of service for new mounts. The corrective actions may consist of the following steps:

- Use the following steps to switch to an existing and enabled alternate CDS, which is presumably defined with more mount entries:
  - Format a new, larger type BPXMCDS by increasing the mount limits.

– Once the CDS is defined, it can be enabled as the alternate CDS with the command:

```
SETXCF COUPLE,TYPE=BPXMCDS,ACOUPLE=(alternate_name,alternate_volume)
```

– Finally, switch the alternate CDS to the primary CDS with the command:

```
SETXCF COUPLE,PSWITCH
```

Once the corrective actions have been made and the new larger primary CDS has been enabled, the following console messages are issued:

```
BPXI045I THE PRIMARY CDS SUPPORTS A LIMIT OF 700 MOUNTS AND A LIMIT OF 50 AUTOMOUNT
RULES.
BPXI044I RESOURCE SHORTAGE FOR MOUNT TABLE HAS BEEN RELIEVED.
```

Figure 6-44 Messages issued when a new BPXMCDS CDS is enabled

The BPXI045I message is issued when a PSWITCH occurs.

6.11 Shared HFS support for the confighfs command

The **`/usr/lpp/dfsms/bin/confighfs`** shell command is used to perform certain functions directly with the HFS physical file system, which include:

- Query HFS limits
- Query HFS global statistics
- Setting HFS virtual and fixed storage pool sizes

Previously, there was a restriction since the OS/390 V2R9 implementation for shared HFS support that the **`confighfs`** command would only provide file system data for file systems that were mounted as RDWR if the command was issued from the owner system. Otherwise, the **`confighfs`** command failed with the following message:

```
Error issuing PFSCTL: RC=0 ERRNO=129(81) REASON=5B360105
HFS is not mounted on this system/LPAR
```

This restriction has been removed in z/OS V1R3 and support has been added so that the **`confighfs`** command can now be issued from any system for any active HFS file system.

Note: This command can now be issued from any system in a sysplex at z/OS V1R3 or later, assuming that the system on which the file system is mounted is also running z/OS V1R3 or later. Otherwise, the command will fail.

6.12 Byte-range locking in a shared HFS environment

With z/OS V1R4, you can lock all or part of a file that you are accessing for read-write purposes by using the byte-range lock manager (BRLM). As a default, the lock manager is initialized on only one system in the sysplex. The first system that enters the sysplex initializes the BRLM and becomes the system that owns the manager. This is called a “centralized BRLM”.

In a sysplex environment, a single BRLM handles all byte-range locking in the shared HFS group. If the BRLM server crashes, or if the system that owns the BRLM is partitioned out of the sysplex, the BRLM is reactivated on another system in the group. All locks that were held under the old BRLM server are lost. An application that accesses a file that was previously locked receives an I/O error, and has to close and reopen the file before continuing.

Distributed BRLM

You can choose to have distributed BRLM initialized on every system in the sysplex. Each BRLM is responsible for handling locking requests for files whose file systems are mounted locally in that system. Use distributed BRLM if you have applications which lock files that are mounted and owned locally.

For distributed BRLM to be activated, the z/OS UNIX couple data set (BPXMCDS) must be updated as shown in Figure 6-45, and the supported code must be installed and running on each system. See APAR OW52293 for more information.

```
ITEM NAME(DISTBRLM) NUMBER(1)
     /*Enables conversion to a distributed BRLM.
       1, distributed BRLM enabled,
       0, distributed BRLM is not enabled during next sysplex IPL
       Default = 0  */
```

Figure 6-45 Update BPXMCDS for BRLM

This support allows you to change to using distributed BRLM (rather than a single, central BRLM) in the sysplex. With distributed BRLM, each system in the sysplex runs a separate BRLM, which is responsible for locking files in the file systems that are owned and mounted on that system. Because most applications (including cron, inetd, and Lotus Domino) lock local files, the dependency on having a remote BRLM up and running is removed. Running with distributed BRLM is optional.

z/OS R1V4 implements the first phase of movable BRLM in a sysplex. Movable BRLM provides the capability of maintaining the byte-range locking history of applications, even when a member of the sysplex dies. This first phase will focus on distributing the locking history across all members of the sysplex. As a result, many applications that lock files that are locally mounted will be unaffected when a remote sysplex member dies. Movement away from a centralized to a distributed BRLM will provide greater flexibility and reliability.

BRLM and callable services

If you use the BPX1FCT or BPX1VLO callable services or the **`fcntl()`** or **`lockf()`** C functions to do byte-range locking in a shared HFS sysplex environment, you should be aware of the recovery scenario that was introduced with the shared HFS support.

Note: The following C functions use byte-range locking internally, and can result in the same recovery scenario: **`endutxent()`**, **`getutxent()`**, **`getutxid()`**, **`getutxline()`**, **`pututxline()`**, **`setutxent()`**, **`__utmpname()`**, and **`__utxtrm()`**.

BRLM coexistence and maintenance

Distributed BRLM support is added in the following PTFs for downlevel systems:

- UW85157 (OS/390 V2R9)

- UW85155 (OS/390 V2R10)
- UW85156 (z/OS V1R2)

Additional support is added in PTFs; when the BRLM server crashes, a default SIGTERM is issued against any process that has used byte-range locking and has an open file that was locked. Users can specify a preferred signal to be used instead of the default SIGTERM.

- UW75787 (V2R9)
- UW75786 (V2R10)

6.13 Deciding whether to keep or to unmount a version root

Let us assume we have a version root named OMVS.OS390A.MVSRS1.ROOT, which is mounted at /MVSRS1 with AUTOMOVE YES.

Now system SYST leaves the sysplex. To see whether this version root is still needed in the sharing environment, run command D OMVS,O on all systems and look for `VERSION=MVSRS1` (as seen for SYST in Figure 6-46).

```
D OMVS,O
...
SHRLIBMAXPAGES  =      4096    VERSION         = MVSRS1
SYSCALL COUNTS  = NO           TTYGROUP        = TTY
SYSPLEX         = YES          BRLM SERVER     = SYST
```

Figure 6-46 Excerpts of output from d omvs,o

If it turns out that the version is still needed on at least one other system, nothing needs to be done. However, if SYST is the last one, look to see whether SYST is the current owner of the file system. To do that, use one of the following two commands:

```
F BPXOINIT,FILESYS=DISPLAY,FILESYSTEM='OMVS.OS390A.MVSRS1.ROOT'
```

```
D OMVS,F
```

And if you find that SYST is not the owner, move the ownership to SYST:

```
SETOMVS FILESYS,FILESYSTEM='OMVS.OS390A.MVSRS1.ROOT',SYSNAME=SYST
```

Furthermore, change the AUTOMOVE setting to NO or UNMOUNT, as follows:

```
SETOMVS FILESYS,FILESYSTEM='OMVS.OS390A.MVSRS1.ROOT',AUTOMOVE=NO
```

```
SETOMVS FILESYS,FILESYSTEM='OMVS.OS390A.MVSRS1.ROOT',AUTOMOVE=UNMOUNT
```

If you do not or cannot use UNMOUNT, use the following command to assure unmounting this file system during shutdown processing:

```
F BPXOINIT,SHUTDOWN=FILESYS
```

Note: This command will assure that all file systems owned by this system that are not mounted AUTOMOVE=YES, and all the automount-managed file systems, are unmounted during shutdown processing.

6.14 Replacing the sysplex root without IPLing any system

The following description can be taken as a cookbook on how to solve problems with important file systems in the USS sysplex sharing environment, especially the sysplex root file system. In our description we use WTSCPLX2.SYSPLEX.ROOT as the sample name for this file system.

- Stop all USS processes or applications that may suffer problems when the file systems are not available for a while.

  Problems are likely seen for applications that have USS files open or even locks set on these files. This is known for Lotus Domino servers and the USS daemons cron and inetd. Problems have sometimes been seen with syslogd in such situations, too.

  There may be more applications that may suffer from losing the file systems. You should identify such applications on all the systems participating in USS sysplex sharing.

  You can test this using the **fuser** command and other utilities to find processes having files open in a specific file system. Figure 6-47 on page 276 shows this for inetd and its PID reference file inetd.pid in directory /etc.

  First processes are listed that have open files in the /etc file system. Then a combination of functions and commands from the USS Tools package (described in 10.1.5, “USSTools” on page 390) and REXX functions (described in 10.2, “REXX functions and interfaces” on page 393) is used to get further information in an interactive mode.

  The information listed proves that the inetd process has regular file /etc/inetd.pid open for writing.

```
#> fuser -c /etc
/etc: 5 18c 16777235
#> rexx
REXX - Enter SH, Syscall, TSO, rexx commands or EXIT:
call procinfo 5, "PROCESS"
REXX - Enter SH, Syscall, TSO, rexx commands or EXIT:
say bpxw_logname bpxw_jobname
STCUSER INETD7
REXX - Enter SH, Syscall, TSO, rexx commands or EXIT:
do i=1 to 14; say bpxw_typecd.i bpxw_type.i bpxw_openf.i bpxw_inode.i bpxw_devno
.i; end
rd 1 0 0 3
cd 1 0 0 3
fd 2 2 9310 3
fd 2 2 9310 3
fd 2 2 9310 3
fd 3 129 358 8
fd 7 3 35 17
fd 3 2 2557 3
fd 7 3 0 12
fd 2 145 9310 3
fd 7 3 36 17
fd 7 3 37 17
fd 7 3 38 17
fd 7 3 39 17
REXX - Enter SH, Syscall, TSO, rexx commands or EXIT:
s "stat /etc st."
OMVS Return Value (retval) = 0
REXX - Enter SH, Syscall, TSO, rexx commands or EXIT:
say st.st_dev
8
REXX - Enter SH, Syscall, TSO, rexx commands or EXIT:
sh "find /etc -inum 358"
/etc/inetd.pid
REXX - Enter SH, Syscall, TSO, rexx commands or EXIT:
say s_isreg
3
REXX - Enter SH, Syscall, TSO, rexx commands or EXIT:
say o_creat+o_wronly
129
REXX - Enter SH, Syscall, TSO, rexx commands or EXIT:
exit
#>
```

Figure 6-47 Listing information about open files of the inetd process

The good news is that TCP/IP-related processes are known to have no problems. And without this fact it would not make sense to speak about this scenario in more detail because IP is so important that if you have to stop TCP/IP, this has a similar effect as doing a full IPL.

- Issue F BPXOINIT,FILESYS=UNMOUNTALL. This will unmount all the USS file systems in the whole sharing environment and just leave the dummy SYSROOT file system.
- Solve the problems that you have with your sysplex root file system.

  As an example, you may simply want to replace the current one with a new one.

  – RENAME WTSCPLX2.SYSPLEX.ROOT to WTSCPLX2.SYSPLEX.ROOT.OLD

– RENAME WTSCPLX2.SYSPLEX.ROOT.NEW to WTSCPLX2.SYSPLEX.ROOT

Attention: If the name of the new or modified sysplex root file system is different from the original name, you need to explicitly mount the file system before performing the next step.

Use the ISHELL interface or the TSO MOUNT command to do that. A sample is shown in Figure 6-48.

```
MOUNT FILESYSTEM(WTSCPLX2.SYSPLEX.NEW.ROOT) MOUNTPOINT('/') TYPE(HFS)
```

Figure 6-48 Mount sysplex root file system with changed name

- Run F BPXOINIT,FILESYS=REINIT to remount file systems of all systems currently active in the sharing environment.

  All the file systems that were mounted by means of statements in BPXPRMxx members at IPL are mounted again. Only file systems that are not already mounted are processed. Also, the mount point directory that is needed must be free and available.

  All file systems that were and are mounted with setting AUTOMOVE=Y will be owned by the system now that runs the **`reinit`** command. Therefore, you should choose the most important system in your sysplex to do so.

- Mount the remaining file systems that are needed.

  All other file systems that are needed and are not mounted automatically when used again need to be mounted now. Normally there will be none, and you probably have a procedure or STC to do so, since this is part of normal IPL processing.

- Restart those applications and daemons again that you had stopped in the beginning.

  In Figure 6-49 an STC sample is shown to restart the USS daemons and reinitialize automount processing. Other files needed are provided in Figure 6-50 and Figure 6-51 on page 278. The STC userid assigned could be the OMVS kernel userid. The userid chosen must have at least read access to BPX.DAEMON to have the necessary authorization for cron and inetd. Read access to profile BPX.JOBNAME in class FACILITY may be desired if it has been defined.

```
//* -------------------------------------------------------------------
//* Restarting of USS daemons and other refresh processing
//* Property of IBM  (C) Copyright IBM Corp. 2003
//* -------------------------------------------------------------------
//BPXREST  PROC
//BPXREST  EXEC PGM=BPXBATCH,REGION=0M,TIME=NOLIMIT,
//           PARM='PGM /bin/sh -c /etc/bpxrest'
//* STDIN defaults to /dev/null
//STDOUT   DD PATH='/dev/console',PATHOPTS=(OWRONLY)
//* STDERR defaults to STDOUT
//STDENV   DD PATH='/etc/bpxrest.env',PATHOPTS=(ORDONLY)
//* -------------------------------------------------------------------
```

Figure 6-49 STC BPXREST for restart processing

```
# Restarting of USS daemons and other refresh processing
# Start the Automount Facility
/usr/sbin/automount
# Start the INET daemon for remote login activity
_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf
# Start the SYSLOG daemon for logging
_BPX_JOBNAME='SYSLOGD' /usr/sbin/syslogd -f /etc/syslog.conf
# Start the CRON daemon
_BPX_JOBNAME='CRON' /usr/sbin/cron
```

Figure 6-50 Shell script /etc/bpxrest with permission bits set to 740

The environment file shown in Figure 6-51 references a setting for TZ that may be used in Central Europe. For EST you can use the following value instead:

```
TZ=EST5EDT
```

i

```
# BPXREST Envvar File
TZ=MEZ-1MESZ,M3.5.0,M10.5.0
```

Figure 6-51 /etc/bpxrest.env (environment variables, permission bits 640)

6.15 USS file systems for licensed program products

We describe some ideas on where to place UNIX file systems related to specific program products that are not shipped with z/OS and are not contained in the root or version root file system.

The original idea for z/OS UNIX is to add new mount point directories in /usr/lpp/ for other products and mount the file systems that come with the products there. In this directory all the directories containing the data for the z/OS standard products are residing, such as dce, dfs, dfsms, tcpip, and so on.

6.15.1 Using a directory mount point in /usr/lpp

We demonstrate first that it is not a good idea in a USS sysplex sharing environment just to create new directories in a version root file and mount the product data there. To do so, a sample file system is used.

Figure 6-52 on page 279 shows the BPXPRMxx parmlib statements involved in our test in the beginning with the system SC65 being IPLed using this parmlib member.

```
VERSION('&SYSR1.')
SYSPLEX(YES)
...
MOUNT FILESYSTEM('WTSCPLX2.&SYSNAME..SYSTEM.HFS')
      MOUNTPOINT('/&SYSNAME.')
      NOAUTOMOVE
      TYPE(HFS)  MODE(RDWR)

MOUNT FILESYSTEM('HFS.ZOSR05.&SYSR1..ROOT')
      MOUNTPOINT('/$VERSION')
      AUTOMOVE
      TYPE(HFS)  MODE(RDWR) */  /* was MODE(READ)

MOUNT FILESYSTEM('OMVS.NEILOC.TEST.HFS')
      MOUNTPOINT('/$VERSION/usr/lpp/neiloc.test')
      AUTOMOVE
      TYPE(HFS)  MODE(RDWR)     /* neiloc test */
...
```

Figure 6-52 Initial BPXPRMxx parmlib statements

Figure 6-53 lists the contents of directory /usr/lpp/neiloc.test, which is contained in OMVS.NEILOC.TEST.HFS.

```
HERING:/u/hering:$> ls -E /usr/lpp/neiloc.test/
total 32
-rwxr-xr-x  --s-  1 HAIMO    SYS1           5 Jul 14 18:23 dominik
-rwxr-xr-x  --s-  1 HAIMO    SYS1           5 Jul 14 18:23 fabian
-rwxr-xr-x  --s-  1 HAIMO    SYS1           5 Jul 14 18:22 jeffery
-rwxr-xr-x  --s-  1 HAIMO    SYS1           5 Jul 14 18:22 philip
HERING:/u/hering:$>
```

Figure 6-53 Listing the contents of OMVS.NEILOC.TEST.HFS

Now we created a new version root file system to be used at the next IPL and also defined a new VERSION variable setting to be used then. This is shown in Figure 6-54 on page 280.

```
/*VERSION('&SYSR1.')*/
VERSION('USSTEST')
SYSPLEX(YES)
...
MOUNT FILESYSTEM('WTSCPLX2.&SYSNAME..SYSTEM.HFS')
      MOUNTPOINT('/&SYSNAME.')
      NOAUTOMOVE
      TYPE(HFS)  MODE(RDWR)

/*MOUNT FILESYSTEM('HFS.ZOSR05.&SYSR1..ROOT')
      MOUNTPOINT('/$VERSION')
      AUTOMOVE
      TYPE(HFS)  MODE(RDWR) */  /* was MODE(READ) */

MOUNT FILESYSTEM('HFS.USSTEST.&SYSR1..ROOT')
      MOUNTPOINT('/$VERSION')
      AUTOMOVE
      TYPE(HFS)  MODE(RDWR)     /* was MODE(READ) */

MOUNT FILESYSTEM('OMVS.NEILOC.TEST.HFS')
      MOUNTPOINT('/$VERSION/usr/lpp/neiloc.test')
      AUTOMOVE
      TYPE(HFS)  MODE(RDWR)     /* neiloc test */
...
```

Figure 6-54 Changed BPXPRMxx parmlib statements

The next step was to shut down system SC65 and IPL it again with the changed BPXPRMxx settings. Figure 6-55 shows some important messages seen during UNIX initialization processing. The message regarding file system OMVS.NEILOC.TEST.HFS looks like a simple informational message, but it is the main problem because this file system could not be mounted according to the BPXPRMxx member.

```
BPXF026I FILE SYSTEM WTSCPLX2.SC65.SYSTEM.HFS 468
WAS ALREADY MOUNTED.
IEF196I IGD103I SMS ALLOCATED TO DDNAME SYS00010
BPXF013I FILE SYSTEM HFS.USSTEST.Z05RB1.ROOT 470
WAS SUCCESSFULLY MOUNTED.
BPXF026I FILE SYSTEM OMVS.NEILOC.TEST.HFS 471
WAS ALREADY MOUNTED.
BPXF026I FILE SYSTEM HFS.SC65.DEV 472
WAS ALREADY MOUNTED.
BPXF026I FILE SYSTEM HFS.SC65.ETC 473
WAS ALREADY MOUNTED.
BPXF026I FILE SYSTEM HFS.SC65.VAR 474
WAS ALREADY MOUNTED.
```

Figure 6-55 UNIX mount messages during a new IPL

Figure 6-56 on page 281 shows information for the original version root that is now owned by system SC64.

```
F BPXOINIT,FILESYS=DISPLAY,FILESYSTEM=HFS.ZOSR05.Z05RB1.ROOT
BPXM027I COMMAND ACCEPTED.
BPXF035I 2003/07/15 12.07.55 MODIFY BPXOINIT,FILESYS=DISPLAY 407
--------------NAME--------------------------        DEVICE  MODE
HFS.ZOSR05.Z05RB1.ROOT                                 140  RDWR
  PATH=/Z05RB1
  STATUS=ACTIVE                LOCAL STATUS=ACTIVE
  OWNER=SC64        RECOVERY OWNER=SC64     AUTOMOVE=Y PFSMOVE=Y
  TYPENAME=HFS      MOUNTPOINT DEVICE=       1
  MOUNTPOINT FILESYSTEM=WTSCPLX2.SYSPLEX.ROOT
  ENTRY FLAGS=90000000  FLAGS=40000000  LFSFLAGS=08000000
  LOCAL FLAGS=40000002  LOCAL LFSFLAGS=20000000
BPXF040I MODIFY BPXOINIT,FILESYS PROCESSING IS COMPLETE.
```

Figure 6-56 Displaying file system information for the original version root

Figure 6-57 shows that OMVS.NEILOC.TEST.HFS is now owned by system SC63 and is still mounted on the old version root mount point directory.

```
F BPXOINIT,FILESYS=DISPLAY,FILESYSTEM=OMVS.NEILOC.TEST.HFS
BPXM027I COMMAND ACCEPTED.
BPXF035I 2003/07/15 12.11.00 MODIFY BPXOINIT,FILESYS=DISPLAY
--------------NAME--------------------------        DEVICE  MODE
OMVS.NEILOC.TEST.HFS                                   164  RDWR
  PATH=/Z05RB1/usr/lpp/neiloc.test
  STATUS=ACTIVE                LOCAL STATUS=ACTIVE
  OWNER=SC63        RECOVERY OWNER=SC63     AUTOMOVE=Y PFSMOVE=Y
  TYPENAME=HFS      MOUNTPOINT DEVICE=     140
  MOUNTPOINT FILESYSTEM=HFS.ZOSR05.Z05RB1.ROOT
  ENTRY FLAGS=90000000  FLAGS=40000000  LFSFLAGS=00000000
  LOCAL FLAGS=40000002  LOCAL LFSFLAGS=20000000
BPXF040I MODIFY BPXOINIT,FILESYS PROCESSING IS COMPLETE.
```

Figure 6-57 Displaying file system information for OMVS.NEILOC.TEST.HFS

And finally, Figure 6-58 provides the information about the new version root file system that is now mounted at /USSTEST.

```
F BPXOINIT,FILESYS=DISPLAY,FILESYSTEM=HFS.USSTEST.Z05RB1.ROOT
BPXM027I COMMAND ACCEPTED.
BPXF035I 2003/07/15 12.17.15 MODIFY BPXOINIT,FILESYS=DISPLAY 421
--------------NAME--------------------------        DEVICE  MODE
HFS.USSTEST.Z05RB1.ROOT                                165  RDWR
  PATH=/USSTEST
  STATUS=ACTIVE                LOCAL STATUS=ACTIVE
  OWNER=SC65        RECOVERY OWNER=SC65     AUTOMOVE=Y PFSMOVE=Y
  TYPENAME=HFS      MOUNTPOINT DEVICE=       1
  MOUNTPOINT FILESYSTEM=WTSCPLX2.SYSPLEX.ROOT
  ENTRY FLAGS=90000000  FLAGS=40000000  LFSFLAGS=08000000
  LOCAL FLAGS=40000000  LOCAL LFSFLAGS=28000000
BPXF040I MODIFY BPXOINIT,FILESYS PROCESSING IS COMPLETE.
```

Figure 6-58 Displaying file system information for the new version root

The final result is that file system OMVS.NEILOC.TEST.HFS cannot be reached when using the directory specification /usr/lpp/neiloc.test because this is resolved to /USSTEST/usr/lpp/neiloc.test/. And the file system is still mounted at the old mount point.This is shown in Figure 6-59.

```
HERING:/u/hering:$> ls -E /usr/lpp/neiloc.test/
total 0
HERING:/u/hering:$> ls -E /Z05RB1/usr/lpp/neiloc.test/
total 32
-rwxr-xr-x  --s-  1 HAIMO    SYS1          5 Jul 14 18:23 dominik
-rwxr-xr-x  --s-  1 HAIMO    SYS1          5 Jul 14 18:23 fabian
-rwxr-xr-x  --s-  1 HAIMO    SYS1          5 Jul 14 18:22 jeffery
-rwxr-xr-x  --s-  1 HAIMO    SYS1          5 Jul 14 18:22 philip
HERING:/u/hering:$>
```

Figure 6-59 Listing again the contents of OMVS.NEILOC.TEST.HFS

6.15.2 Solution 1, using symbolic links in /usr/lpp

Here we use the IMS Connect USS file structure for the explanation. As the name of the file system we take OMVS.IMSICO.HFS. Furthermore, we assume that the version file system that is to be modified has been mounted at /SERVICE.

Using this new technique, you no longer see the problems of the scenario described in 6.15.1, “Using a directory mount point in /usr/lpp” on page 278. There are many further advantages.

For example, if your version file systems are related to a SYSRES volume and you have to maintain copies of PP-related file systems independent of the z/OS (and USS) service level for each SYSRES, this is no longer necessary.

In the version root you currently have a mount point directory for the PP file systems. In this situation it is /usr/lpp/imsico, or if referring to the full version file directory name, /Z04RB1/usr/lpp/imsico.

Perform the following steps to replace a directory entry by a symbolic link in the version structure:

- Create the directory pp at the root level within the sysplex root:

  ```
  mkdir -m 755 /pp
  ```

  Note: Another very good naming convention is PRDS instead of pp. PRDS is used or suggested as a mount point for product-related file systems in a worldwide IBM system delivery process.

- Create a small file system that will contain the mount points for the pp-related file systems.

  Note: We only created one additional entry pp in the sysplex root file system.

- Mount this new file system at /pp and place a MOUNT statement in BPXPRMxx to assure that this file system is mounted when the first system in the sysplex IPLs.
- Create the directory mount point for IMSICO:

  ```
  mkdir -m 755 /pp/imsico
  ```

- Remove the directory in your version root file system and replace it by a symbolic link as shown in Figure 6-60.

```
#> rmdir /SERVICE/usr/lpp/imsico
#> ln -s /pp/imsico /SERVICE/usr/lpp/imsico
```

Figure 6-60 Replacing a directory mount point with a symbolic link

- Now you can use the BPRXPRMxx mount statement shown in Figure 6-61.

```
 MOUNT FILESYSTEM('OMVS.IMSICO.HFS')
       TYPE(HFS) MODE(READ) AUTOMOVE
       MOUNTPOINT('/$VERSION/usr/lpp/imsico')
```

Figure 6-61 IMSICO mount statement in BPXPRMxx

Note: The mount point could be simply specified as /usr/lpp/imsico, but the choice in Figure 6-61 is more precise.

A further, but more complex, possibility is to point from /usr/lpp to another symbolic link in the system-specific structure. This makes it possible to decide on a system level which pp structure is finally chosen by setting the target path name for this second symbolic link.

6.15.3 Solution 2, using a new path structure without referencing /usr/lpp

Instead of using this technique with symbolic links, you can go one step further and change the pp-related path string from /usr/lpp/imsico directly to /pp/imsico.

Note: We do not know of any pp-related path specification that is hard-coded pointing to /usr/lpp. And if there are some, we can still use a mixture of these two strategies.

Using this approach has the advantage that you do not need to modify the version root file system and no symbolic links need to be maintained.

6.15.4 Solution 3, using "Symlink Symbolics" introduced with z/OS V1R5

At the system level we have USS internal variables $VERSION and $SYSNAME, which allow system and service level related resolution of symbolic links. In z/OS V1R5 two new variables allow unique pathname resolution based on the value of system symbols on a particular system.

This is an excellent means for specifying mountpoints that you want to to share between systems in the USS sysplex sharing group. There are two ways of doing this:

- For an absolute resolution, use $SYSSYMA/
- For a relative resolution, use $SYSSYMR/

To illustrate how this can be exploited, let us assume that we have two systems, SC65 and SC70, both at the same z/OS service level and sharing the same version root. SC65 runs DB2 V7 and has set a static system symbol &DB2VER with value DB2V7; SC70 has V8 and a system symbol value of DB2V8. The way that the DB2 USS file systems can be addressed easily is shown in Figure 6-62 on page 284.

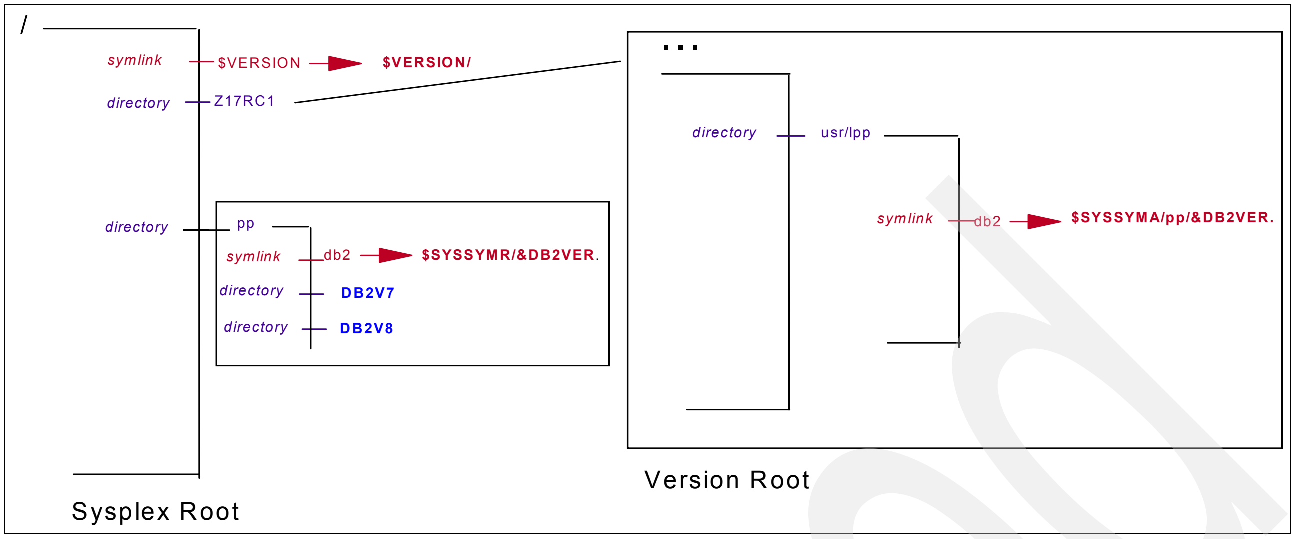


Figure 6-62 Using symlink symbolics for two different DB2 versions

As in solutions 1 and 2, the product-specific file structure is place under directory /pp. The usual DB2 directory is /usr/lpp/db2. This is replaced by a symbolic link pointing to:

```
$SYSSYMA/pp/&DB2VER.
```

This absolute symlink specification is resolved by using the system-specific value for the system symbol &DB2VER. This is /pp/DB2V7 in SC65 and /pp/DB2V8 in SC70.

A second symbolic link named db2 is provided in directory /pp to allow direct access to the right DB2 directory in /pp without the need to know what is the the correct local DB2 version.

In this situation a relative specification is used since the target directories are located at the same directory level in the USS file hierarchy. The content of the link is:

```
$SYSSYMR/&DB2VER.
```

The relative symlink is resolved by using the value for the system symbol &DB2VER again and also adding the directory name that the symlink itself belongs to. Therefore, the final results are once more /pp/DB2V7 in SC65 and /pp/DB2V8 in SC70.

Figure 6-63 provides a sample of how to define the symbolic links mentioned.

```
#> ln -s '$SYSSYMA/pp/&DB2VER.' /usr/lpp/db2
#> ln -s '$SYSSYMR/&DB2VER.' /pp/db2
```

Figure 6-63 Definition of symlinks using the **ln** *command*

Note: Quotes are needed because otherwise $SYSSYMA and $SYSSYMR would be interpreted as UNIX environment variables.

Figure 6-64 on page 285 shows a statement that can be used to mount the correct DB2 file system at the desired directory.

```
MOUNT FILESYSTEM('OMVS.&DB2VER..ZFS')
  MOUNTPOINT('/pp/db2')
  AUTOMOVE TYPE(ZFS) MODE(RDWR)
```

Figure 6-64 BPXPRMxx mount statement

6.16 System-specific data under the version root structure

A similar problem is seen with system-specific data located under the version root structure. If a version root is shared between several systems, you need a means to address a location separately for each system for all directory structures that contain system-specific data.

We use /usr/spool as an example to demonstrate this. Since this UNIX file structure is not shared between systems, you need to take it out of the version file system and put it somewhere under the system-specific structure. Do this for each system sharing that version file system or more generally participating in UNIX sysplex sharing.

- Create a directory called /etc/spool in the system-specific structure in the same way it is suggested for the situation where you have a read-only root file system and are using the cron daemon.
- Now copy the original data to this new directory.

Tip: Because spool directories tend to be used heavily, it is good practice to create a new USS file system and mount it on /etc/spool.

- Then remove the files and directories under and including the /usr/spool directory.
- Finally create a symbolic link for /usr/spool that points to /etc/spool.

Figure 6-65 shows the list of all these commands.

```
#> mkdir /etc/spool
#> chmod 755 /etc/spool
#> cd /usr/spool
#> pax -pe -rw . /etc/spool
#> rm -fR /usr/spool
#> ln -s /etc/spool /usr/spool
```

Figure 6-65 Moving /usr/spool to /etc/spool

Note: A detailed description about this topic is available in informational APAR II12249 (“Customization changes for cron, uucp, and mail utilities for shared hfs”).

6.17 Replacing a version structure dynamically

In the scenario described in 6.15.1, “Using a directory mount point in /usr/lpp” on page 278 we IPLed a system to demonstrate the processing of changed file system mounts in BPXPRMxx. If you simply want to activate a new version file system, this can be done dynamically with the SETOMVS VERSION= command. In our scenario this would have meant the following:

```
SETOMVS VERSION='USSTEST'
```

As long as there are no other file systems mounted on the old and new version file system structures, this is rather a nondisruptive way to modify a version file system. This is true because there normally are no programs that have files open there, especially if many structures have been moved to file systems outside the version file system. The single steps are:

- Create a cloned copy of the old version file system.
- Mount the copy and modify it.
- Mount it at the new version location in the desired mode.
- Issue the SETOMVS VERSION= system command.

6.18 File system remount function for USS sysplex sharing

In single systems the remount function makes it possible to change the mount mode of a file system from RDWR to READ or READ to RDWR. In the more complex sysplex sharing, this function was not supported in the past.

APAR, OA02584, introduced the capability to remount file systems in a shared USS sysplex configuration for z/OS V1.4 systems (PTF UA04906) and is included for z/OS V1R5 and up. There are no PTFs for previous z/OS releases.

Note: All sysplex members must have this support. If one or more members are downlevel, errno EINVAL (79x) and errnojr JrNotSupInSysplex (58804A5x) will still be returned when remount is requested. This reason code simply says that remount is not supported in sysplex.

The syntax for **REMOUNT** in a sysplex is the same as it currently is for non-sysplex; **REMOUNT** is an option on **UNMOUNT**. There are several commands and interfaces that allow to use **REMOUNT**. Following we list some of the possibilities.

The easiest way to do this with a command is probably to use the TSO UNMOUNT command with the REMOUNT parameter:

```
UNMOUNT FILESYSTEM(HFS.ZOSR05.Z05RC1.ROOT) REMOUNT
```

Because no arguments were specified for the **REMOUNT** parameter the mount mode is changed from RDRW to READ or READ to RDRW. If you do specify either argument, the filesystem is remounted in that mode.

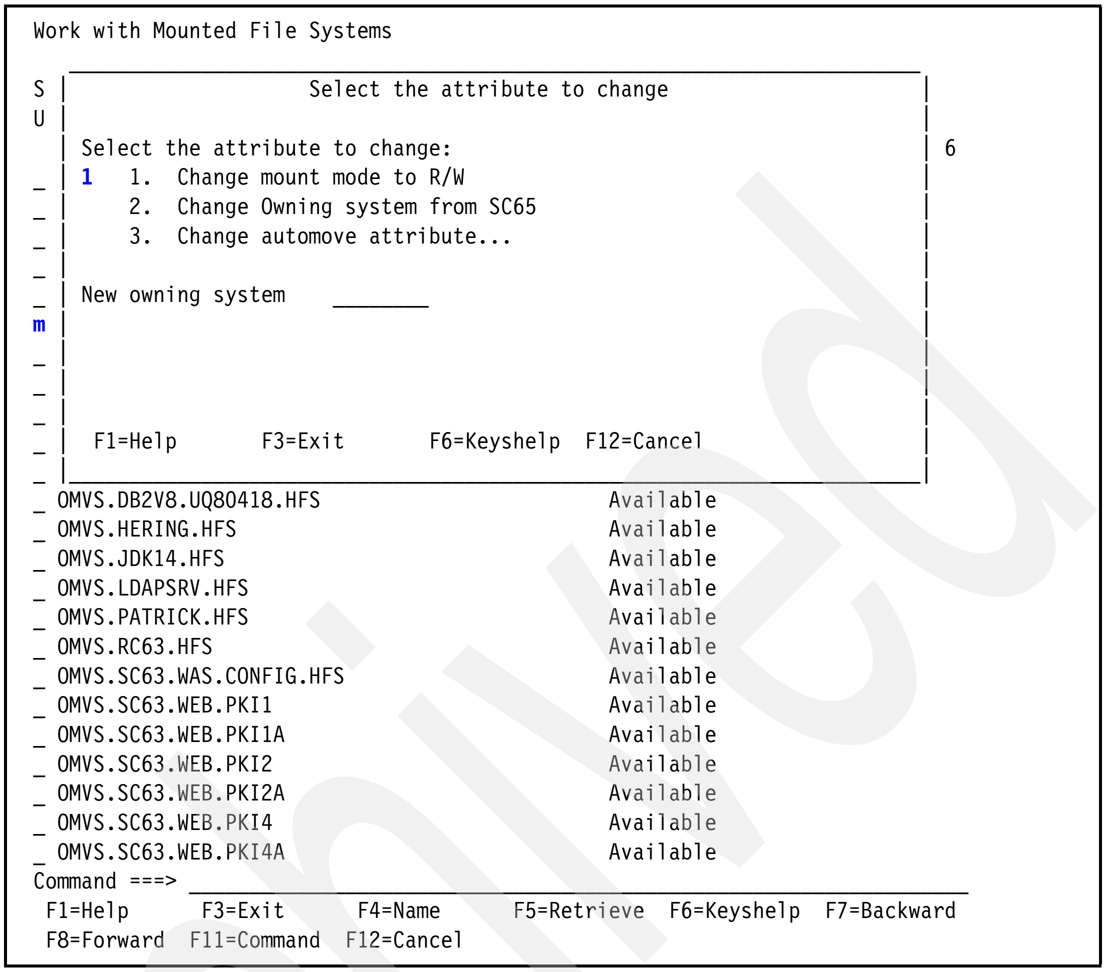
You can also use API functions:

- Use MtmRemount set on in the MtmFlags with an Assembler BPX1UMT call.
- Use the mtm_remount flag with REXX Syscall command **unmount**.

If you do not want to use a command, the ISHELL interface is the best choice.

Note: The ISHELL is using the REXX Syscall interface to perform the remount function.

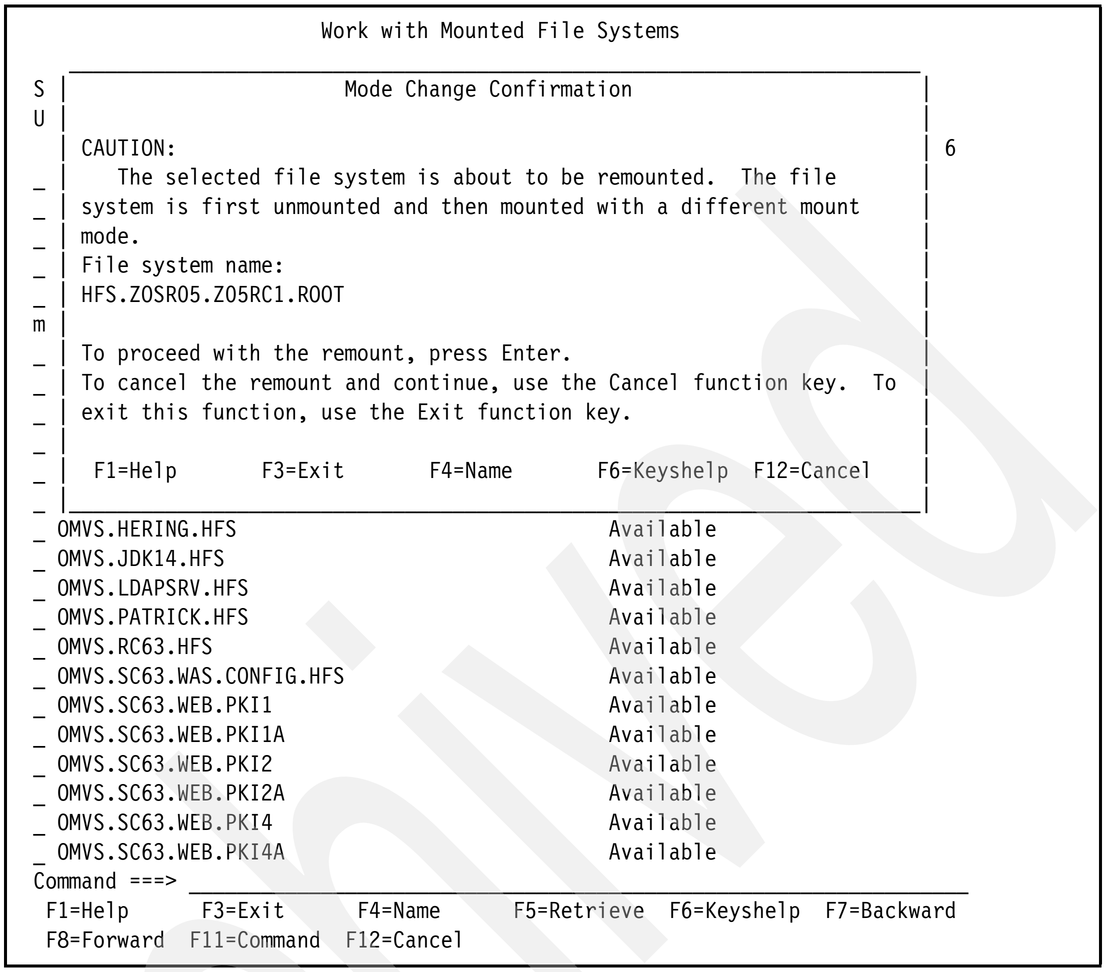
Use the **M** (=Modify) prefix command when displaying the file system mount table. The resulting panel is shown in Figure 6-66.



```
Work with Mounted File Systems
    ________________________________________________________________________
S |                    Select the attribute to change                        |
U |                                                                          |
  | Select the attribute to change:                                          | 6
_ | 1   1.  Change mount mode to R/W                                         |
_ |     2.  Change Owning system from SC65                                   |
_ |     3.  Change automove attribute...                                     |
_ |                                                                          |
_ | New owning system    ________                                            |
m |                                                                          |
_ |                                                                          |
_ |                                                                          |
_ |                                                                          |
_ |  F1=Help       F3=Exit       F6=Keyshelp  F12=Cancel                     |
_ |__________________________________________________________________________|
_ OMVS.DB2V8.UQ80418.HFS                         Available
_ OMVS.HERING.HFS                                Available
_ OMVS.JDK14.HFS                                 Available
_ OMVS.LDAPSRV.HFS                               Available
_ OMVS.PATRICK.HFS                               Available
_ OMVS.RC63.HFS                                  Available
_ OMVS.SC63.WAS.CONFIG.HFS                       Available
_ OMVS.SC63.WEB.PKI1                             Available
_ OMVS.SC63.WEB.PKI1A                            Available
_ OMVS.SC63.WEB.PKI2                             Available
_ OMVS.SC63.WEB.PKI2A                            Available
_ OMVS.SC63.WEB.PKI4                             Available
_ OMVS.SC63.WEB.PKI4A                            Available
Command ===> _________________________________________________________________
 F1=Help      F3=Exit      F4=Name      F5=Retrieve  F6=Keyshelp  F7=Backward
 F8=Forward  F11=Command  F12=Cancel
```

Figure 6-66 Using the M prefix command in the ISHELL mount table

Now select **Change mount mode to R/W** to switch the mount mode. When doing so you get a further display panel to confirm the mode change. This is shown in Figure 6-67 on page 288.



```
                           Work with Mounted File Systems

     ___________________________________________________________________________
 S |                        Mode Change Confirmation                            |
 U |                                                                            |
   | CAUTION:                                                                   | 6
 _ |    The selected file system is about to be remounted.  The file            |
 _ | system is first unmounted and then mounted with a different mount          |
 _ | mode.                                                                      |
 _ | File system name:                                                          |
 _ | HFS.ZOSR05.Z05RC1.ROOT                                                     |
 m |                                                                            |
 _ | To proceed with the remount, press Enter.                                  |
 _ | To cancel the remount and continue, use the Cancel function key.  To       |
 _ | exit this function, use the Exit function key.                             |
 _ |                                                                            |
 _ |  F1=Help       F3=Exit       F4=Name       F6=Keyshelp  F12=Cancel         |
 _ |____________________________________________________________________________|
 _ OMVS.HERING.HFS                              Available
 _ OMVS.JDK14.HFS                               Available
 _ OMVS.LDAPSRV.HFS                             Available
 _ OMVS.PATRICK.HFS                             Available
 _ OMVS.RC63.HFS                                Available
 _ OMVS.SC63.WAS.CONFIG.HFS                     Available
 _ OMVS.SC63.WEB.PKI1                           Available
 _ OMVS.SC63.WEB.PKI1A                          Available
 _ OMVS.SC63.WEB.PKI2                           Available
 _ OMVS.SC63.WEB.PKI2A                          Available
 _ OMVS.SC63.WEB.PKI4                           Available
 _ OMVS.SC63.WEB.PKI4A                          Available
 Command ===> _________________________________________________________________
  F1=Help      F3=Exit      F4=Name      F5=Retrieve  F6=Keyshelp  F7=Backward
  F8=Forward  F11=Command  F12=Cancel
```

Figure 6-67 Selecting "Change mount to ..." in the ISHELL

The remount can be requested from the server or from any of the client systems. When remount successfully completes, the mode of the file system is changed on all systems in the sysplex.

Comments on zFS aggregates

For zFS HFS-compatible aggregates, if both clone and primary are mounted, sysplex remount will be rejected with errno EINVAL and errnojr JrAggregateErr. The suggested action is to unmount the clone and retry the remount. This is because the unmount phase of remount will only unmount the primary, leaving the clone mounted, and leaving the aggregate in an attached state. This will cause the mount phase of remount to fail when remounting from Read-only to Read-Write, since the aggregate will remain attached Read-only.

Exploiting the remount function

You'll have to watch out for applications that have files open within the file system that is about to be remounted. This is normally no problem if you need to modify a R/O file system, for example a version structure, since it will not contain such open critical files. Especially TCP/IP will not get into trouble (even if you take away file systems for awhile), as mentioned in 6.14, "Replacing the sysplex root without IPLing any system" on page 275.

So in the case of a R/O version file system, the temporary switch to R/W, even if other file systems are mounted below, seems to be a very good means to make some corrections or modifications.

Regarding the sysplex root file system, this function may have even more importance. Losing this file system has a big impact on your whole sysplex since it is the top of the UNIX file structure.

The suggested way to run the sysplex root is R/W. This is to allow dynamic creation of new mount points if a new system joins the sysplex sharing environment. Nevertheless, many installations mount the sysplex root R/O with all mount points predefined. This avoids the ugly problems that may occur if this structure gets filled up accidentally, but good planning is needed.

Now, if this new support is available in sysplex sharing, it is very easy to switch from R/O to R/W, IPL a new system, or add other needed mount points, and finally switch back to the save R/O mode.

7

Defining users with z/OS UNIX

This chapter describes how to define general users to get access to z/OS UNIX, and how key started tasks are defined in the z/OS UNIX environment.

In this chapter we discuss how to get access to z/OS UNIX with an interactive shell. The following need to be done to give users access to the z/OS UNIX shell:

- Creating an OMVS segment for the user in the RACF database
- Giving access to the z/OS UNIX environment
- Creating a user file system to store user files
- Creating a mount point for the user file system
- Modifying the BPXPRMxx member to define the automount facility

7.1 Setting up a general user

To access the z/OS UNIX environment it is important that the user have a specific UID and GID. The UID and GID are stored in the RACF database as follows:

- Adding a GID to the RACF group profile for an existing or new RACF group of the user, or the user's default group
- Adding a UID to the RACF user profile for an existing or new TSO/E user and connecting each user to a RACF group that has a GID

The OMVS segment contains information that is listed in Table 7-1.

Table 7-1 OMVS segment description

Value	OMVS segment Description
UID	User's z/OS UNIX user identifier
HOME	User's z/OS UNIX initial directory path name
PROGRAM	User's z/OS UNIX program path name, a default shell program
CPUTIMEMAX	User's z/OS UNIX RLIMIT_CPU (maximum CPU time)
ASSIZEMAX	User's z/OS UNIX RLIMIT_AS (maximum address space size)
FILEPROCMAX	User's z/OS UNIX maximum number of files per process
PROCUSERMAX	User's z/OS UNIX maximum number of processes per UID
THREADSMAX	User's z/OS UNIX maximum number of threads per process
MMAPAREAMAX	User's z/OS UNIX maximum memory map size

To define or change information in the OMVS segment of a user profile, including your own, you must have the RACF SPECIAL attribute or at least UPDATE authority to the segment through field-level access checking. To allow authorization to the entire OMVS segment of a user profile, the user would need authority to the USER.OMVS.* profile in the FIELD class. Individual fields in the OMVS segment can be defined such as USER.OMVS.UID. You can allow users to change their own HOME or PROGRAM values by creating USER.OMVS.HOME and USER.OMVS.PROGRAM in the FIELD class and permitting &RACUID to the profiles.

7.1.1 Defining an OMVS segment

You can assign a user identifier (UID) to a RACF user by specifying a UID value in the OMVS segment of the RACF user profile. When assigning a UID to a user, make sure that the user's default group has an assigned GID. If the user specifies a group during logon or on a batch job, this current connect group must also have an assigned GID. A user with a UID and a default group (and current connect group, if applicable) with a GID can use z/OS UNIX functions and access z/OS UNIX files based on the assigned UID and GID values. If a UID and GID are not available as described, the user cannot use z/OS UNIX functions.

Note: Although you can assign the same UID to multiple users, it is not recommended. However, it may be necessary for some cases, such as superusers. If you assign the same UID to multiple users, control at an individual user level is lost because the UID is used in z/OS UNIX security checks. Users with the same UID assignment are treated as a single user during z/OS UNIX security checks.

To define a minimum z/OS UNIX OMVS segment you can use the following RACF ALTUSER command that gives an existing RACF user the segment:

```
ALU LUTZ OMVS((HOME('/u/lutz') PROGRAM('/bin/sh') UID( &UID ))
```

The UID assignment in the command, shown as `&UID`, depends on the method the installation has chosen for assigning of UIDs, which could be as follows for the ALU command:

- Specify a specific UID number.
- Specify the AUTOUID operand for automatic assignment of a UID; see 3.10.1, "Automatic UID and GID assignment" on page 115.

RACF can automatically generate a unique ID value in the OMVS segment upon your request. This is done by defining a profile called BPX.NEXT.USER in the FACILITY class which contains the UID and GID number range, and then specifying:

- The AUTOUID operand of the ADDUSER and ALTUSER commands
- The AUTOGID operand of the ADDGROUP and ALTGROUP commands

Displaying the OMVS segment

The RACF command LU (list user) displays the defined z/OS UNIX OMVS segment in the RACF database, as follows:

```
LU LUTZ OMVS NORACF
```

The output from the command is shown in Figure 7-1.

```
USER=LUTZ

OMVS INFORMATION
----------------
UID= 0000068215
HOME= /
PROGRAM= /bin/sh
CPUTIMEMAX= NONE
ASSIZEMAX= NONE
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE
```

Figure 7-1 Output from LISTUSER command with a valid z/OS UNIX OMVS segment

If you received the message shown in Figure 7-2, the user does not have a z/OS UNIX OMVS segment defined and has no access to z/OS UNIX.

```
USER=LUTZ

NO OMVS INFORMATION
```

Figure 7-2 Output from LISTUSER without a z/OS UNIX OMVS segment

You can use the following command to define a z/OS UNIX OMVS segment with automatic UID assignment to define the z/OS UNIX UID in the RACF database:

```
ALU LUTZ OMVS((HOME('/u/lutz') PROGRAM('/bin/sh') AUTOUID))
```

Upon successful command completion, informational message IRR52177I is issued to indicate the assigned value as follows:

```
IRR52177I User LUTZ was assigned an OMVS UID value of 4711.
```

7.1.2 Creating user file systems

In traditional MVS environments, general users are given their own profile and the ability to create data sets under a certain high-level qualifier. In z/OS UNIX this is accomplished by allocating separate HFS or zFS data sets for each z/OS UNIX user and mounting these separate data sets onto the directory that you define in the RACF database. These separate data sets can be used to store data unique to this z/OS UNIX user.

This practice allows each z/OS UNIX user to use their own data sets without impacting any other z/OS UNIX shell users. It is also a way of isolating each z/OS UNIX user's data for systems management purposes.

Attention: We recommend that you only use the new zFS file system for user file systems. Nevertheless, we also describe the HFS user file system.

7.1.3 Creating zFS file systems

zFS can only create zFS file systems if your system is running the zFS physical file system. A zFS aggregate is created by defining a VSAM linear data set and then formatting that VSAM LDS as an aggregate. This is done once for each zFS aggregate. A multiple file system aggregate can contain one or more zFS file systems. A zFS file system is equivalent to an HFS file system.

The VSAM LDS is allocated with the VSAM utility program IDCAMS. The **`zfsadm define`** command can also define a VSAM LDS. The command creates a DEFINE CLUSTER command string for a VSAM LDS with SHAREOPTIONS(2) and passes it to the IDCAMS utility. If a failure occurs, the **`zfsadm define`** command may display additional messages from IDCAMS indicating the reason for the failure.

Note: The issuer of the **`zfsadm define`** command requires sufficient authority to create the VSAM LDS.

Steps required to use a zFS file system

The following actions are required to allocate a zFS data set in a multiple file system aggregate:

- Allocate a VSAM linear data set using IDCAMS.
- Format the aggregate using the IOEAGFMT utility.
- Attach the zFS file system to the managing address space.
- Define a zFS file system inside the aggregate.
- Mount the zFS file system.

7.1.4 Allocate and format the VSAM linear data set

The first step is to allocate a home data set for the user. After the aggregate is created, formatting of the aggregate is necessary before any file systems can exist in it. Figure 7-3 shows a job for allocating and formatting a zFS aggregate using a batch job.

```
//ZFSDEF   JOB 'zFS allocate',CLASS=C,MSGCLASS=X,
//             NOTIFY=&SYSUID,MSGLEVEL=(1,1)
/*JOBPARM  ROOM=F21,S=SC65
//*------------------------------------------------------------------*
//* define and format zFS aggreagte file *
//*------------------------------------------------------------------*
//DEFZFS   EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DEFINE CLUSTER -
   (NAME( LUTZ.SC65.HOME.ZFS ) -
   VOLUME(MHLS2A) LINEAR CYL(5 5) SHAREOPTIONS(2))
/*
//FMTZFS   EXEC PGM=IOEAGFMT,
//  PARM='-aggregate LUTZ.SC65.HOME.ZFS -compat
'
//SYSPRINT DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//
```

Figure 7-3 job for allocating and formatting a zFS aggregate

You can then use the **zfsadm define** command, shown in Figure 7-4, or you can use the ISHELL panel to define the aggregate. The **zfsadm format** command formats the aggregate.

```
zfsadm define -a LUTZ.SC65.HOME.ZFS -volume O37CAT -cylinders 5 5
IOEZ00248E VSAM linear dataset LUTZ.SC65.HOME.ZFS successfully created.

zfsadm format -aggregate LUTZ.SC65.HOME.ZFS -compat
IOEZ00077I HFS-compatibility aggregate LUTZ.SC65.HOME.ZFS has been successfully created
```

Figure 7-4 zFS commands allocating and formatting with zfsadm

7.1.5 Attach the aggregate to the zFS address space

For availability it is necessary that you attach the aggregate to the colony address space that manages the zFS file systems. You have a choice between a batch job or using the **zfsadm attach** command. Figure 7-5 on page 296 shows a sample batch JCL job to attach the aggregate and in Figure 7-6 on page 296 you will see the use of the **zfsadm attach** command that attaches the aggregate.

```
//ZFSATT   JOB 'zFS attach',CLASS=C,MSGCLASS=X,
//             NOTIFY=&SYSUID,MSGLEVEL=(1,1)
/*JOBPARM  ROOM=F21,S=SC64
//*-----------------------------------------------------------------*
//* define and format zFS aggreagte file as multifile system         *
//*-----------------------------------------------------------------*
//ATTZFS   EXEC PGM=IOEZADM,
//  PARM='attach -aggregate LUTZ.SC64.HOME.ZFS'
//SYSPRINT DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//
```

Figure 7-5 sample job for attaching an aggregate to the colony adress space

```
zfsadm attach -aggregate LUTZ.SC65.HOME.ZFS
IOEZ00117I Aggregate LUTZ.SC65.HOME.ZFS attached successfully
```

Figure 7-6 The zfsadm command to attach the aggregate

7.1.6 Define a zFS file system inside the aggregate

A zFS aggregate is a data set that contains zFS file systems. The aggregate is a VSAM Linear Data Set (VSAM LDS) and is a container that can contain one or more zFS file systems.

Sufficient space must be available on the volume or volumes, as multiple volumes may be specified on the DEFINE of the VSAM LDS. DFSMS decides when to allocate on these volumes during any extension of a primary allocation. VSAM LDSs greater than 4 GB may be specified by using the extended format and extended addressability capability in the data class of the data set.

A zFS file system is a named entity that resides in a zFS aggregate. A zFS file system can be mounted at a directory into the USS hierarchy. While the term file system is not a new term, a zFS file system resides in a zFS aggregate, which is different from an HFS file system.

In Figure 7-7 on page 297, you can see a batch job to create a zFS file system inside a zFS aggregate. You can also use the **`zfsadm command`**, as shown in Figure 7-8 on page 297, to create a zFS file system.

Restriction: if you plan to define only one zFS file system inside the aggregate, called compat-mode, you don’t need to define and attach the aggregate. You need only mount the zFS VSAM LDS dataset like an HFS file system.

```
//ZFSCREAT JOB 'zFS create file system',CLASS=C,MSGCLASS=X,
//              NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//*-------------------------------------------------------------------*
//* define and format zFS aggreagte file as compat system *
//*-------------------------------------------------------------------*
// SET ZFSNAME='create -filesystem LUTZ.HOME'
// SET AGTNAME='-aggregate LUTZ.SC64.HOME.ZFS '
// SET ZFSSIZE='-size 4000'
//MKEZFS   EXEC PGM=IOEZADM,PARM='&ZFSNAME &AGTNAME &ZFSSIZE'
//SYSPRINT DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//
```

Figure 7-7 Sample batch job to define a zFS file system

```
zfsadm create -filesystem LUTZ.HOME -aggregate LUTZ.SC65.HOME.ZFS -size 1000
IOEZ00099I File system LUTZ.HOME created successfully
```

Figure 7-8 Sample command to define a zFS file system

7.1.7 Creating an HFS file system

If you prefer to use HFS file systems, then allocate the user HFS in exactly the same way you created other HFS files. Choose a data set name that has the user name as one of the qualifiers and a size that provides sufficient space for the user's requirements. If more space is required, you may wish to increase the size of the allocation or you may wish to create additional HFS data sets on different DASD volumes and mount them at different mount points in the user's hierarchy. HFS data sets can reside on non-managed or managed DFSMS DASD volumes. The current limit for each HFS data set is 123 extents and can be spread across 59 volumes. Figure 7-9 shows a sample job to allocate an HFS data set.

```
//HFSALLOC JOB ,'HFS CREATE',NOTIFY=LUTZ,
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),TIME=1440
//ALLOC    EXEC PGM=IEFBR14
//HFS      DD  DSN=LUTZ.HOME.HFS,
//             DISP=(,CATLG,DELETE),
//             SPACE=(CYL,(5,2,1)),
//             DSNTYPE=HFS,
//             STORCLAS=STANDARD
//
```

Figure 7-9 Sample JCL for an HFS allocation

Note: All newly allocated HFS data sets have permission bit settings of 700.

7.1.8 Mounting a file system

A file system must be mounted before it can be accessed. When z/OS UNIX is started, the root file system is mounted as the first file system. All other file systems included on the MOUNT statements in the BPXPRMxx member or mounted with the TSO MOUNT command will be mounted on the root file system or on other file systems.

A file system can be mounted in read/write mode or in read-only mode. If it is mounted in read-only mode, nobody can update any files or directories in that file system. To change the mount mode, the file system must be unmounted and then remounted with a mode of read/write.

A file system must be mounted on a directory. This is called a *mount point*. Use empty directories as mount points. If a directory is not empty, the existing files will be overlayed by the file system which is mounted on the directory. When this file system is unmounted, the existing files will be accessible and visible again.

Special authority is required to mount or unmount a file system. That means the user must have at least one of the following accesses.

- Superuser authority UID(0)
- Read permissions to BPX.SUPERUSER
- Read/Update permissions to SUPERUSER.FILESYS.MOUNT

If a file system does not need to be updated, it could be mounted in read-only mode. This will improve performance. However, if the file system mounted as read-only needs updates, it must be unmounted and remounted again. When a user requires an HFS or zFS file system to be accessed, you need to get it mounted at a mount point off of the root directory to make it available. The preferred place to mount all user HFS or zFS file systems is the /u directory mount point.

Methods to mount a file system

In z/OS UNIX, there are two ways to accomplish this, as follows and as shown in Figure 7-10 on page 299.

- Direct mount

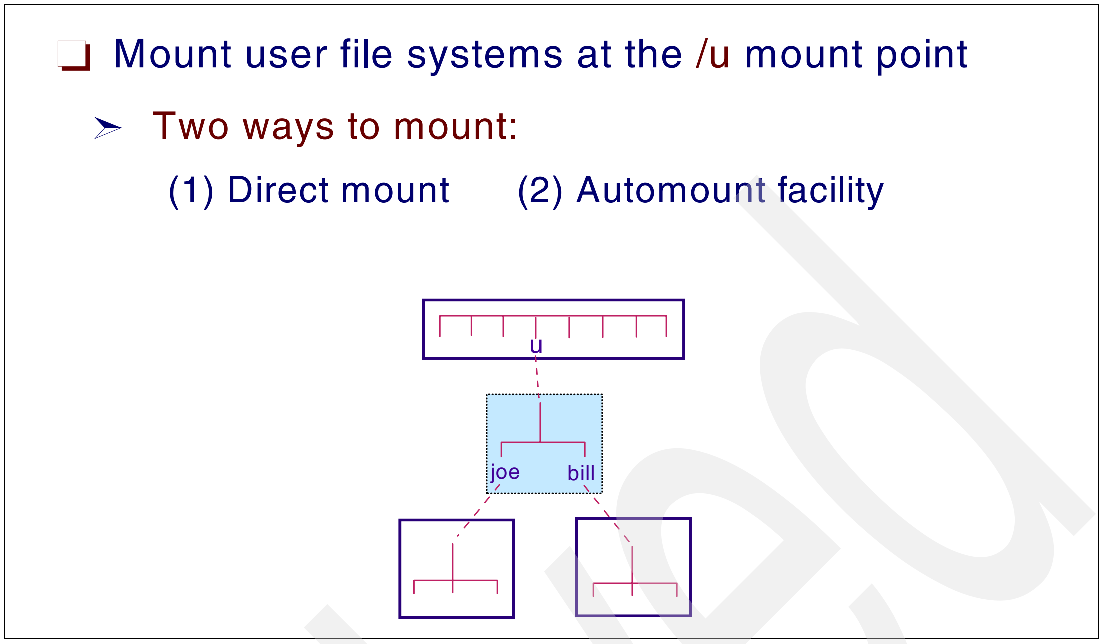
  Allocate an intermediate HFS or zFS data set to be mounted between the root file system and all user file systems.

  Create a mount point using the **`mkdir`** command and issue the **`mount`** command. To make the mount permanent, you will also need to add the HFS or zFS data set name and its mount point to the BPXPRMxx member of parmlib.

- Automount

  You need to customize the automount facility to control all user file systems to automatically mount them when they are needed. This is the preferred method of managing user HFS or zFS data sets because it saves administration time. See 7.3, “Mounting file systems with the automount facility” on page 302.

  The automount facility lets you designate directories as containing only mount points. This is the preferred method of managing user HFS or zFS file systems. As each of these mount points is accessed, an appropriate file system is mounted. The mount point directories are internally created as they are required. Later, when the file system is no longer in use, the mount point directories are deleted.



❑ Mount user file systems at the /u mount point

➤ Two ways to mount:

(1) Direct mount (2) Automount facility

Figure 7-10 Two ways to mount a user file system

7.2 Mounting a file system using direct mount

To mount your previously created file system you need a mount point. To create a mount point you can use the **mkdir** command in z/OS UNIX. Usually, user file systems are mounted under /u, or you can use /home. You should have a separate file system just to contain the user's home directories. These home directories will be empty, and they will act as mount points for the user file systems. Depending on how the installation is organized and on the need for user space, each user can have their own file system, or users in a department can share a system. It is easier to control space usage when each user has his own file system. It can be compared to users having their own z/OS data set for user data, or their own minidisk on VM. How large it should be depends entirely on who will use it and what the user will do. If multiple users share an HFS data set, it is not possible to guarantee space for each user. One user can dominate the space in a file system shared between multiple users.

When making plans for a file structure, it is important to think about a naming convention for the HFS data sets, as shown in Figure 7-11 on page 300. If the automount facility will be used, one of the data set qualifiers should be equal to the directory name where the file system will be mounted.

The /u directory contains the user home directories, and if these directories are placed in separate file systems, most of the user data will be kept out of the root file system.

The UNIX System Service programmer can choose to use only one HFS for all users, or use one HFS per user. If the user HFS is too small, you can mount some of the users to another user's HFS, or increase the user space for the HFS. This will make it easier to manage the HFS.

A recommendation is to name the user home directories /u/userid with the user ID in lowercase, for example /u/joe for user JOE as shown in Figure 7-11 on page 300.

7.2.1 Create a user file system for direct mounts

Following is a suggested method for creating user file systems:

- Leave the /u directory in the root file system empty.
- Create an HFS file called OMVS.<SYSNAME>.USERS.HFS.

  This file system will contain the home directories for all users and will be mount points for the user file systems. The reason for keeping these home directories in a separate file system is to avoid updating the root file system for each new home directory to create or delete. The second qualifier of the DSNAME identifies the z/OS system image it relates to.
- Create a file system for each user or for a department, depending on what is best for the installation. The user file systems can be called OMVS.<SYSNAME>.<userid>.HFS.

  The user file systems will be mounted on the home directories in the OMVS.<SYSNAME>.USERS.HFS file system.

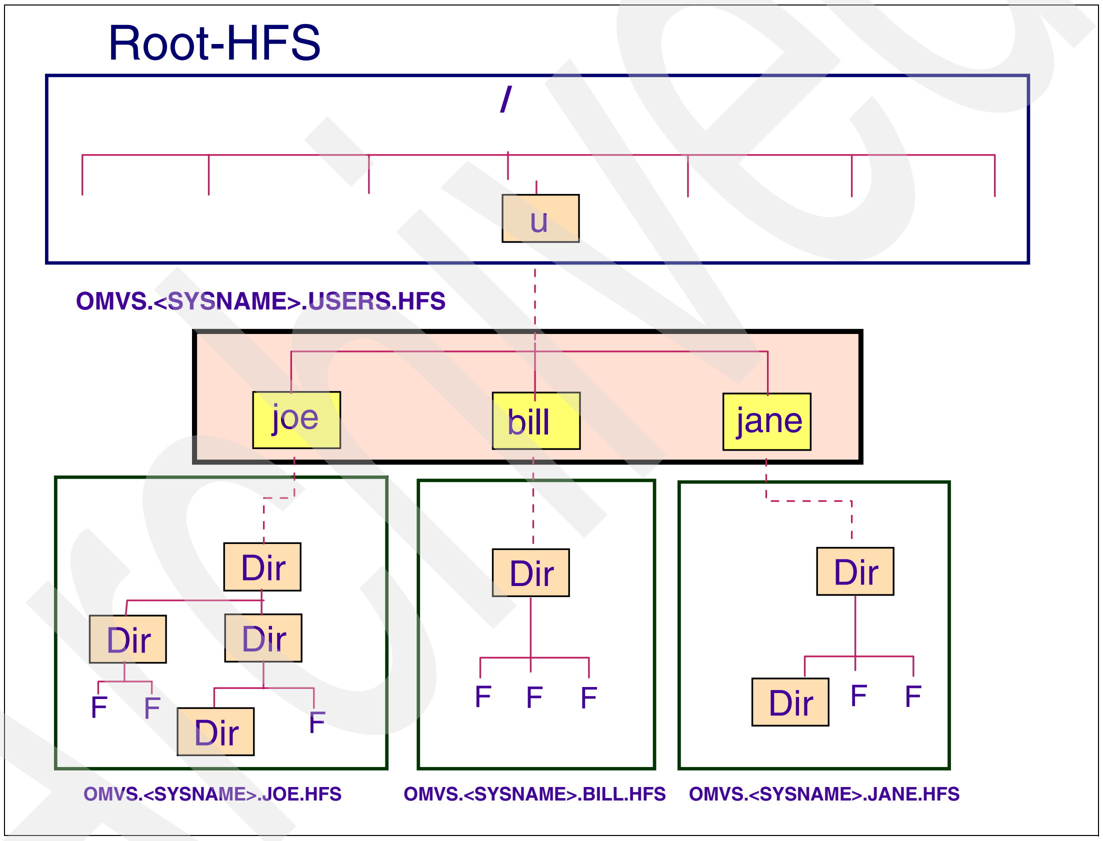


Figure 7-11 Direct mount example for mounting user file systems

Mounting a user file system

A file system can be mounted with the TSO/E MOUNT command or the ISHELL. Superuser authority is required for mounting or unmounting a file system.

The options for the MOUNT command are the same as for the MOUNT statement in the BPXPRMxx member, except for an additional option called WAIT or NOWAIT. This option specifies whether to wait for an asynchronous mount to complete before returning.

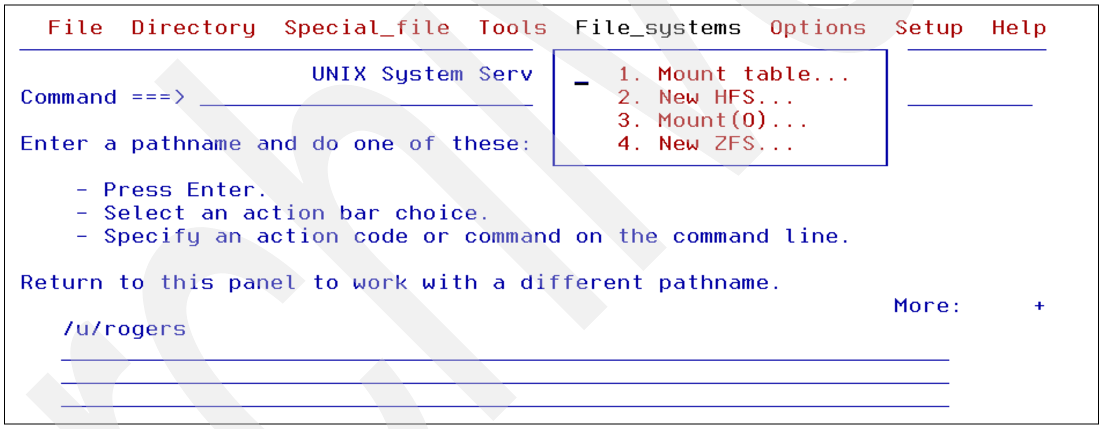
Mounts can be done for a user file system by issuing the `mount` command shown in Figure 7-12. The HFS file system OMVS.SC65.JOE.HFS is mounted on the z/OS UNIX directory /u/joe in the OMVS.SC65.USERS.HFS file system.

```
MOUNT FILESYSTEM('OMVS.SC65.JOE.HFS') TYPE(HFS) MODE(RDWR) MOUNTPOINT('/u/joe')
```

Figure 7-12 Example of an HFS mount statement

The ISHELL is a powerful application based on ISPF. The ISHELL command invokes the z/OS ISPF shell, a panel interface that helps you to set up and manage z/OS UNIX System Services functions such as mounting of file systems. With the ISHELL mount panel shown in Figure 7-13, you can do many things. The eight menu options and what they do are as follows:

File	Edit, browse, delete, copy, rename, print, run, and so on
Directory	List, create, rename, print, find string, and so on
Special_file	New FIFO, link, attribute, delete, rename, and so on
Tools	Processes, Shell commands, run programs, and so on
File_systems	Mount, unmount, change attribute, allocate HFS, and so on
Options	Directory list, edit, browse, settings, and so on
Setup	User and Group Admin., create TTY, RACF permit Field, and so on
Help	Action code, Help, Keys help, About, and so on



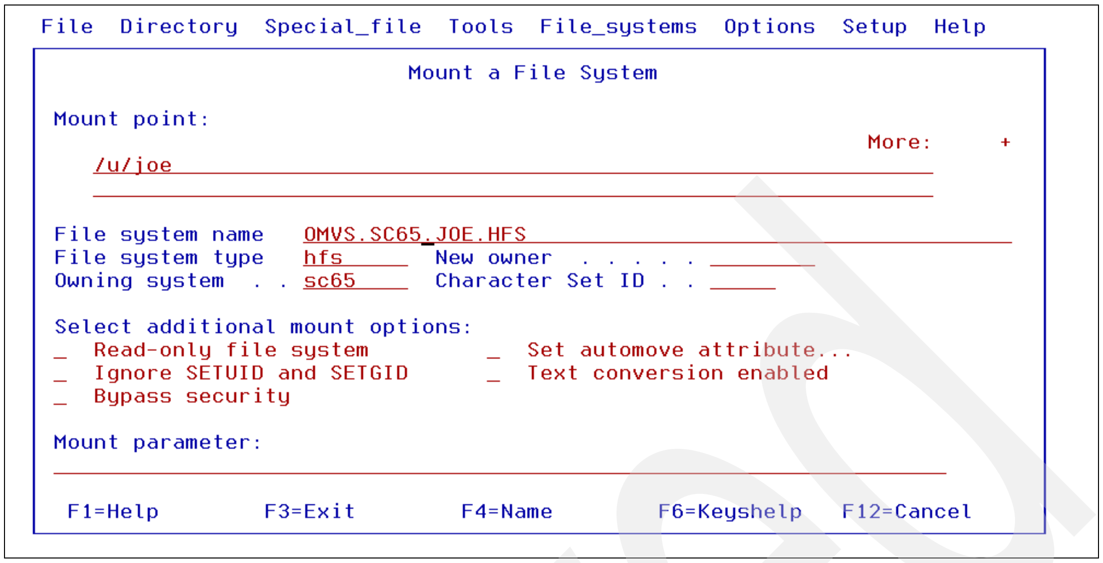
```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
 ──────────────────────────────────────             ┌───────────────────────┐
                            UNIX System Serv        │ _  1. Mount table...  │
Command ===> ________________________               │    2. New HFS...      │ ________
                                                    │    3. Mount(O)...     │
Enter a pathname and do one of these:               │    4. New ZFS...      │
                                                    └───────────────────────┘
    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                               More:     +
   /u/rogers
   ________________________________________________________________
   ________________________________________________________________
   ________________________________________________________________
```

Figure 7-13 ISHELL mount panel

The ISPF shell also provides the administrator with a panel interface for setting up users for z/OS UNIX access, for setting up the root file system, and for mounting and unmounting a file system.

Select the options you require on the panel. The mount point must be a directory. If it is not an empty directory, files in that directory are not accessible while the file system is mounted. Only one file system can be mounted at a directory (mount point) at any one time.

To mount the file system for user joe, select option 3 in Figure 7-13. In Figure 7-14 on page 302, supply the mount point, file system name, file system type, owning system, and any other option required.



```
 File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                      Mount a File System

  Mount point:
                                                             More:     +
     /u/joe


  File system name   OMVS.SC65.JOE.HFS
  File system type   hfs        New owner  . . . . .
  Owning system  . . sc65       Character Set ID . .

  Select additional mount options:
  _  Read-only file system          _  Set automove attribute...
  _  Ignore SETUID and SETGID       _  Text conversion enabled
  _  Bypass security

  Mount parameter:


   F1=Help        F3=Exit        F4=Name        F6=Keyshelp   F12=Cancel
```

Figure 7-14 Mount a file system panel in the ISHELL

Attention: Do not forget that the direct mount is only temporary. After the next IPL, the user file systems are not mounted.

7.3 Mounting file systems with the automount facility

For the mounting of z/OS UNIX user file systems, the automount facility provides the following advantages:

- You do not need to mount the user's file systems at initialization and you do not need to request that they be mounted by an administrator or authorized operator. This makes it easier to add new users, because you can keep your parmlib specification unchanged. This simplifies management of the user file systems.
- You can establish a simple automount policy to manage user home directories.
- A file system that is managed by the automount facility remains unmounted until its mount point is accessed.
- It enables you to reclaim system resources used by a mount if that file system has not been used for a period of time. You can specify how long the file system should remain mounted after its last use.

Usually you have more than one user working on your system. Therefore, it is strongly recommended that you use the z/OS UNIX automount facility. It manages the creation of the mount point and the mount of the user file system for you. Whenever someone accesses a directory managed by the z/OS UNIX automount facility, the mount is issued automatically.

7.3.1 Creating the automount facility

The steps required to define the automount facility environment are described in this section.

Step 1

You need to have the automount FILESYSTYPE active, either by having the statement in the active BPXPRMxx member during IPL as shown or by using the SETOMVS RESET=(xx) command to do it dynamically. The appropriate entry in the BPXPRMxx member is shown as follows:

```
FILESYSTYPE TYPE(AUTOMNT) ENTRYPOINT(BPXTAMD)
```

Step 2

The auto.master file in /etc must contain all mount points that should be managed by automount. An example is shown in Figure 7-15. A map file describes the automount policy for a specific mount point. If you choose to include a system name, you can specify &SYSNAME. instead of <system>, which exploits the new system symbol support of automount introduced in z/OS V1R3. See 7.3.7, “Support of system symbols in the map file” on page 308.

Automount uses the following two HFS files for specifying the policy for which file systems should be automatically mounted when referenced:

/etc/auto.master	This file contains a list of directories to be managed along with their MapName files.
/etc/u.map	This file is the MapName file for a directory.

```
/u             /etc/user.map
```

Figure 7-15 Sample definition in /etc/auto.master

Note: The automount facility in z/OS V1R6 allows the master and map files to reside in MVS data sets. Although the default remains /etc/auto.master, another file name can be specified on the command line. The data set can be a sequential data set or a member of a PDS and can be specified both uppercase and lowercase.

/etc/u.map specifications

The MapName file contains the mapping between a subdirectory of a directory managed by automount and the mount parameters as follows:

name	The name of the mount point directory. * specifies a generic entry for the automount-managed directory.
type	File system type. Default is HFS.
file system	Data set name of the file system to mount. Two special symbols are supported to provide name substitution, where: **<asis_name>** Is used to represent the name exactly, as is. **<uc_name>** Is used to represent the name in uppercase characters.
mode	Mount mode
duration	The minimum amount of time in minutes to leave the file system mounted. The default is nolimit.
delay	The minimum amount of time in minutes to leave the file system mounted after the duration has expired and the file system is no longer in use. The default is 0.
setuid	Can be specified as yes or no. This will support or ignore the SETUID and SETGID mode bits on executable files loaded from the file system.

Note: The following attributes apply to a file system mounted with NOSETUID: SETUID, and SETGID programs are not supported. That is, the UID or GID will not be changed when the program is executed.

An example of the u.map file is shown in Figure 7-16 on page 304.

```
#----------------------------------------------------------------------#
# automount profile for HFS user filesystems                           #
#----------------------------------------------------------------------#
name               *
type               HFS
filesystem         OMVS.&SYSNAME..<uc_name>.HFS
mode               rdwr
duration           1440
delay              360
allocuser          space(3,3) cyl storclas(STANDARD)
```

Figure 7-16 /etc/u.map automount policy for user HFS file systems

Step 3

Issue the start of the automount feature (Figure 7-17). We recommend you place this command in the /etc/rc file so that the automount facility starts during the IPL. The command is shown in Figure 7-15 on page 303.

```
# Start the Automount facility
/usr/sbin/automount
```

Figure 7-17 Start command for the automount facility placed in /etc/rc file

If your automount policy resides in a sequential or PDS data set, you can use the syntax shown in Figure 7-18. Notice the double quotes around the name to avoid unwanted shell processing.

```
/usr/sbin/automount "//sys1.parmlib(amtmst01)"
```

Figure 7-18 Start command

7.3.2 Display the current automount policy

A new option in the **automount** command has been introduced to display the current automount policy in effect. The policy is displayed in a normalized format suitable as input to the automount utility as the map files with minor editing required, as shown in Figure 7-19.

```
@ SC65:/>/usr/sbin/automount -q
/u
name                 *
filesystem           OMVS.<uc_name>.HFS
type                 HFS
mode                 rdwr
duration             1440
delay                360
```

Figure 7-19 Example output for the display automount command

7.3.3 Add to an existing policy

With z/OS V1R6 we have the capability to add new automount managed directories to the existing automount policy. To do this a new flag was added to the command line. The **-a** option, shown in Figure 7-20, indicates that the policy being loaded is to be appended to the existing policy rather than replacing it.

```
/usr/sbin/automount -a
```

Figure 7-20 Add to an existing policy

7.3.4 Support "#" as comment delimiter in the map file

In order to provide a more common syntax with shell script files, z/OS V1R3 supports the "#" character as a comment delimiter in map files, as shown in Figure 7-21 on page 305.

```
#####################
# Automount Map File for /u #
#####################
name               *
type               HFS
filesystem         OMVS.<uc_name>.HFS
mode               rdwr
duration           1440
delay              360
```

Figure 7-21 Example map file with "#" as a comment delimiter

7.3.5 Dynamic HFS allocation in automount policy

Two new keywords have been introduced in z/OS V1R3 in the map file to allocate an HFS dynamically if it is not currently defined at the moment:

allocuser This keyword allocates an HFS only if HFS does not exist and the name matches the user ID.

allocany This keyword allocates an HFS if the HFS does not exist.

The format of these new keywords is as follows:

```
allocuser space_specifications string
allocany space_specifications string
```

Where the `space_specifications` string specifies typical allocation parameters, such as:

```
space(primary-alloc[,secondary alloc])
 cyl | tracks |  block(block size)
 vol(volser[,volser]...)
 maxvol(num-volumes)
 unit(unit-name)
 storclas(storage-class)
 mgmtclas(management-class)
 dataclas(data-class)
```

The following keywords are automatically added:

```
dsn(filesystem)
dsntype(hfs)
dir(1)
```

```
new
```

Map file example

In Figure 7-22 on page 306, the allocany keyword has been added in the map file. In our example, if the HFS does not exist at the moment of the reference of /u/uc_name, an HFS with a primary space of 10 tracks and a secondary space of 5 tracks will be allocated.

```
#####################
# Automount Map File for /u #
#####################
name               *
type               HFS
filesystem         OMVS.<uc_name>.HFS
mode               rdwr
duration           1440
delay              360
allocany           space(10,5) tracks
```

Figure 7-22 Example map file with the allocany keyword

The map file syntax is checked by the automount policy at load time. In case of an error, the key number in error is indicated and the policy fails. However, incorrect usage in the allocation specification is not checked at the time the automount policy is processed and will result in allocation failures on usage.

Incorrect map file example

Figure 7-23 shows what happens if an incorrect specification for allocation is used. In our example, we created an automount policy including an non-existent dataclas(nonexist). The policy is loaded correctly but at the moment the HFS must be allocated, the allocation fails. Allocation failure message IGD01011I is issued.

```
######################
# Automount Map File for /u #
######################
name            *
type            HFS
filesystem      <uc_name>.HFS
mode            rdwr
duration        nolimit
delay           10
allocany        space(10,5) tracks maxvol(3) dataclas(nonexist)

Messages on SYSLOG:

IEF196I IKJ56893I DATA SET USER1.HFS NOT ALLOCATED+
IKJ56893I DATA SET USER1.HFS NOT ALLOCATED+
IEF196I IGD01011I DATA SET ALLOCATION REQUEST FAILED -
IEF196I ACS DATACLAS ROUTINE RETURNED NONEXIST
IEF196I WHICH DOES NOT EXIST
IGD01011I DATA SET ALLOCATION REQUEST FAILED - 923
ACS DATACLAS ROUTINE RETURNED NONEXIST
WHICH DOES NOT EXIST
```

Figure 7-23 Example of allocation failure for automount

For more information, see *z/OS UNIX System Services Command Reference,* SA22-7802.

7.3.6 Generic match on lowercase names

z/OS V1R3 introduces the new keyword, lowercase, that ensures that a generic match will be done only for lowercase names.

Previously, when automount tried to resolve a lookup request, it attempted to find a specific entry. If a specific entry did not exist for the name being looked up, it attempted to use the generic entry (name *). The generic match could be in lowercase or uppercase. Then, automount automatically mounts the HFS data set based on the MapName policy, which indicates the name of the HFS to be mounted. The name of the HFS to be mounted can include the following special symbols to provide name substitution:

<asis_name> This represents the exact name of the subdirectory to be automounted. If the name is in uppercase, the substitution name in the HFS name will be in uppercase. If the name is in lowercase, the substitution name will result in lowercase. That means that we are using the <asis_name> keyword as shown in Figure 7-24.

> **Note:** The access to /u/testauto will result in a catalog error, whereas the access to /u/TESTAUTO will succeed.

<uc_name> This represents the name of the subdirectory to be automounted in uppercase characters. Note than in this case, the /u/user1 and /u/USER1 mount point directories map the same file system.

> **Note:** The new keyword, *lowercase,* makes it possible to define whether a generic entry will match names with lowercase or not. Two options are available:

lowercase[YES] This indicates that only names in lowercase (special characters are also allowed) will match the * specification.

lowercase[NO] This is the default and indicates that any names will match the * specification.

```
######################
# Automount Map File for /u #
######################
name            *
type            HFS
filesystem      <asis_name>.HFS
mode            rdwr
duration        nolimit
delay           10
allocany        space(10,5) tracks
```

Figure 7-24 Automount map file

```
####################
# Automount Map File for /u #
####################
name                *
type                HFS
filesystem          OMVS.<uc_name>.HFS
mode                rdwr
duration            1440
delay               360
allocany            space(10,5) tracks maxvol(3)
lowercase           yes
```

Figure 7-25 Automount map file with lowercase keyword

For example, if we activate the policy described in Figure 7-25, because lowercase=yes is specified, access by /u/USER1 will not match the generic entry, resulting in a failure for the automount load. But accessing with /u/user1 will be successful.

Note: Note that the lowercase keyword with <asis_substitution> in the HFS name will also result in an error when automount is requested.

7.3.7 Support of system symbols in the map file

z/OS V1R3 introduces support of system symbols in the map file to provide name substitution in the file system name.

Figure 7-26 shows the use of system symbols &SYSNAME and &SYSPLEX for the automount policy.

Note: Symbol substitution is done when the automount policy is loaded, not when the rule is used to resolve a mountpoint.

```
@ SC65:/>/usr/sbin/automount -q
/u2

name                  *
filesystem            OMVS.&SYSNAME..<uc_name>.HFS
type                  HFS
allocany                        space(10,5) tracks
mode                  rdwr
duration              1440
delay                 360

/u

name                  *
filesystem            OMVS.&SYSPLEX..<uc_name>.HFS
type                  HFS
allocany              space(10,5) tracks
mode                  rdwr
duration              1440
delay                 360
```

Figure 7-26 Map files including system symbols

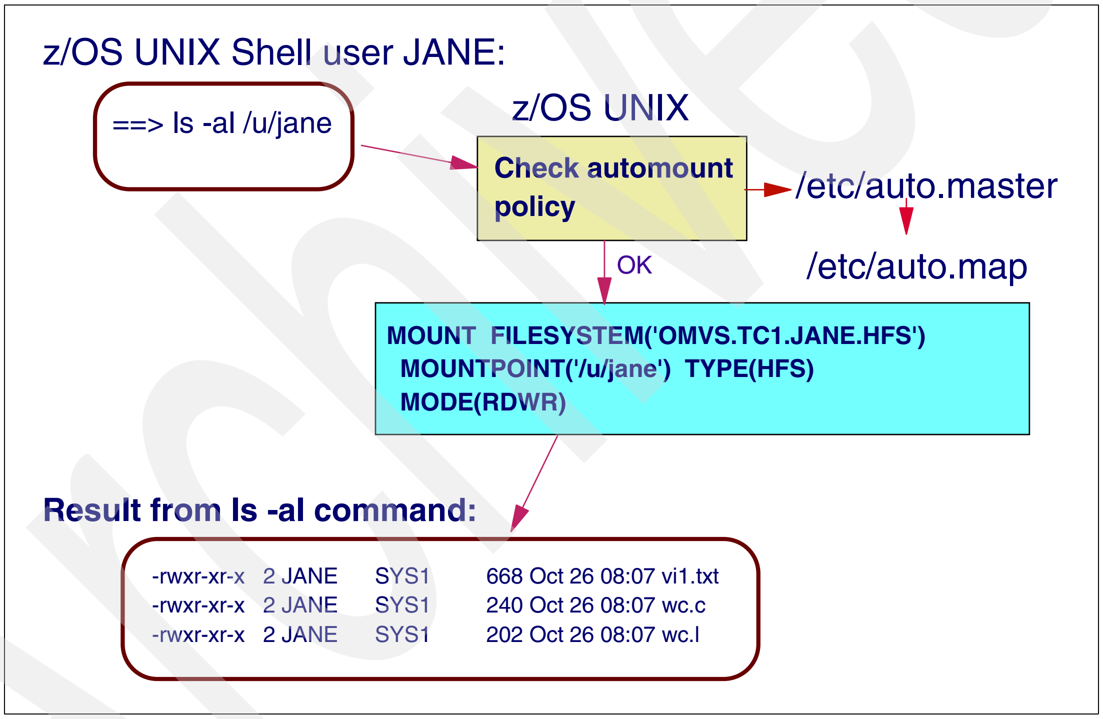
Attention: The use of <SYSTEM> will be withdrawn in a future release; use &SYSNAME. instead.

7.3.8 Using the automount facility for general users

The automount facility lets you designate directories as containing only mount points. This is the preferred method of managing user HFS or zFS data sets. As shell user JANE accesses the home directory in Figure 7-27, the file system of JANE is mounted.

When JANE issues the command and references /u, a check is made to see if an automount policy exists with a /u in the /etc/auto.master file. If yes, the /etc/auto.map is accessed to determine the parameters for the mount of the file system belonging to JANE. The **`mount`** command is issued by automount and JANE receives a response, as shown in Figure 7-27.

The mount point directories are internally created as they are required. Later, when the file system is no longer in use, the mount point directories are deleted.



```
MOUNT  FILESYSTEM('OMVS.TC1.JANE.HFS')
  MOUNTPOINT('/u/jane')  TYPE(HFS)
  MODE(RDWR)
```

```
-rwxr-xr-x   2 JANE     SYS1        668 Oct 26 08:07 vi1.txt
-rwxr-xr-x   2 JANE     SYS1        240 Oct 26 08:07 wc.c
-rwxr-xr-x   2 JANE     SYS1        202 Oct 26 08:07 wc.l
```

Figure 7-27 User JANE accessing her files results in an automount mount

Try to think of automount as an administrator that has total control over a directory. When a name is accessed in this directory, it checks its policy to see what file system is supposed to be associated with that name. If it finds one, it (logically) does an **`mkdir`** followed by a **`mount`** and quietly moves out of the way. Once out of the way, the root directory of that newly mounted file system is now accessed as that name.

With automount active and the correct automount policy in place, there is no need to create a directory for JANE with the **`mkdir`** command; the directory JANE is dynamically allocated and the OMVS.SC65.JANE.HFS data set is automatically mounted at the /u/jane mount point, as shown in Figure 7-28 on page 310. Later, if the /u/jane file system has not been accessed based on the duration specified in the automount policy, the OMVS.SC65.JANE.HFS data set is automatically unmounted.

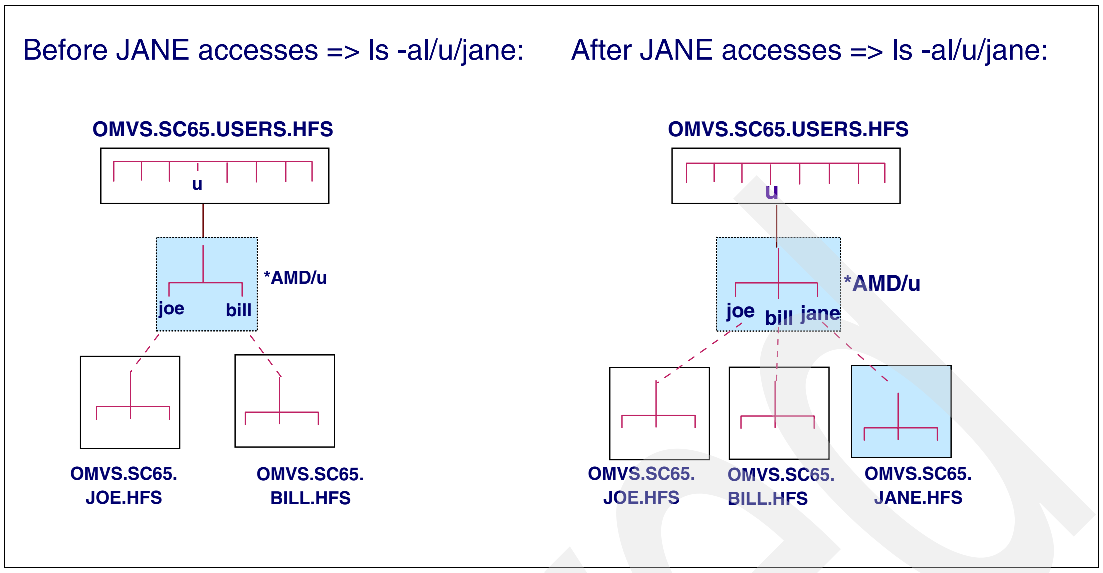


Figure 7-28 The file system of JANE is not mounted

7.4 Setting up started tasks

Started tasks usually do not need a separate file system as general users do. We recommend that you define the started task user IDs as shown in Figure 7-29. Use the /tmp directory as temporary space for the file system. If a special task requires a different home directory, you can create another one.

Attention: To set up a started task, it is very important that you know exactly what the started task does. This means, for example, which C-API functions would be used. The reason why this is important is that you should define the appropriate RACF permissions for the started tasks. The easiest way is to give the started task a UID 0. For security reasons this may not be acceptable, and the started task should have permission settings as required.

```
AU  BPXSTC DFLTGRP(OMVSGRP) OWNER(LUTZ) NOPASSWORD
ALU BPXSTC OMVS (HOME('/tmp') PROGRAM('/bin/sh') UID( &UID ))
RDEFINE STARTED BPXSTC.* STDATA(USER(BPXSTC) GROUP(OMVSGRP)
```

Figure 7-29 Sample RACF definitions for a started task

An example of the started task JCL can be see in Figure 7-30.

```
//BPXSTC  PROC
//* -------------------------------------------------------------------
//* demonstrate the execution of ls inside a stc
//* -------------------------------------------------------------------
//OMVS     EXEC PGM=BPXBATCH,PARM='PGM /bin/ls -la',REGION=0M
//STDOUT   DD PATH='/dev/console',PATHOPTS=(OWRONLY)
```

Figure 7-30 Example of a start task that runs a z/OS UNIX command

7.5 Environment variables

When a program starts, environment variables are made available to it. These consist of strings of the form name=value, where name is the name associated with the environment variable, and its value is represented by the characters in value. UNIX systems traditionally pass information to programs through the environment variable mechanism.

There are global variables for all shell users. Each user can override these variables with an individual set of variables. You can also change any of the values for the duration of your session (or until you change them again). You enter the name of the environment variable and equate it to a new value.

Some environment variables used by the shell are PATH and TZ.

To display variables, use the **set** command, as follows:

```
PATH="/usr/lpp/Printsrv/bin:/bin:."
SHELL="/bin/sh"
STEPLIB="none"
TERM="dumb"
TZ="EST5EDT"
_BPXK_SETIBMOPT_TRANSPORT="TCPIPOE"
_BPX_TERMPATH="OMVS"
```

An environment variable is a variable that describes the operating environment of a process and typically includes information about the home directory, command search path, the terminal in use, and the current time zone.

Setting an environment variable is optional. If a variable is not set, it will have no value. The following is a list of the places where environment variables can be set:

- By the system programmer:
  - RACF user profile
  - /etc/profile
- By the shell users:
  - $HOME/.profile
  - The file named in the ENV environment variable in $HOME/.profile
  - A shell command or shell script

7.6 Code page tables

A code page for a character set determines the graphic characters produced for each hexadecimal code. The code page is determined by the programs and national languages being used.

If the shell is using a locale generated with code pages IBM-1047, an application programmer needs to be concerned about “variant” characters in the POSIX portable character set whose encoding may vary from other EBCDIC code pages. For example, the encodings for the square brackets do not match on code pages IBM-037 and IBM-1047.

For most countries, MVS uses one code page and the z/OS UNIX shell uses a different one. To complicate the matter, many users have workstations that use an ASCII-based code page. When moving data between these environments, the data may have to be converted between the code pages to maintain the same characters. This may result in different hexadecimal codes, depending on the code pages.

There are code page conversion tables located in SYS1.LINKLIB which can be used to convert between the z/OS code page and the shell code page for the national languages supported by the z/OS UNIX shell.

The source for these code page tables can be found in SYS1.SAMPLIB. They can be used as samples for creating a new table if an installation has special requirements.

7.6.1 Specifying a code page

The OMVS command has a CONVERT option that lets you specify a conversion table for converting between code pages for the shell accesses by a TSO/E user. The table you want to specify depends on the code pages you are using in MVS and in the shell. For example, if you are using code page IBM-037 in MVS and code page IBM-1047 in the shell, specify the following when you enter the OMVS command:

```
OMVS CONVERT((BPXFX111))
```

The BPXFX111 translation table is for z/OS (code page 00033) to z/OS UNIX (code page 1047) translation. This would be used by most shell users in the U.S.

IBM has supplied many code pages for various countries with the product. If you were in Switzerland you might invoke the shell with the BPXFX450 conversion table. If you leave out the data set name, the normal z/OS search order is used to find the module.

If you are accessing the shell through the rlogin or Telnet interface, you do not use the OMVS command. You must change to the appropriate code page by issuing the **chcp** command. This could be issued by the user or be put in a profile for them.

7.7 Setting the time zone

The time zone is an environment variable used by the time service to set the correct time for z/OS UNIX and for applications running on z/OS UNIX.

The activation order is as follows:

- First the settings in /etc/init.options.
- Then the /etc/rc becomes active at kernel initialization time.
- Afterwards, the settings in /etc/profile are read.
- If a user has a .profile in his HOME directory that has its own TZ value specified, this setting will be the active one for this user if he enters the shell.

It is possible to specify a time zone variable in four different files. Table 7-2 shows the files and shows when changes made in these files become active.

Table 7-2 This table shows when changes in these four files become active

Files	After z/OS UNIX restart	After next user enters the shell	Immediately after change
/etc/init.options	YES	NO	NO

Files	After z/OS UNIX restart	After next user enters the shell	Immediately after change
/etc/rc	YES	NO	NO
/etc/profile	YES	YES	NO
$HOME/.profile	YES	YES	NO

If the time zone variable is changed, you must restart z/OS UNIX or re-IPL z/OS.

Recommendation: It is recommended to have a TZ setting active in /etc/init.options and in /etc/profile. If there are daemons to start via the /etc/rc file, export the TZ variable before starting the daemons. If you want the correct time on any replies inside the shell, you will have to specify the TZ variable at least in the user profiles ($HOME/.profile) if no specifications were made in the system-wide profile /etc/profile where all users are affected.

7.7.1 User-defined variables

Shell users can set their own values for the environment variables using any of the following methods:

- A $HOME/.profile file. A sample is provided in /samples/.profile, which can be copied to a shell user's home directory and modified.

  Add the environment variable _BPX_SHAREAS=YES to the user profile. This will cause all shell commands that run in separate processes to create the processes in the same address space as the shell.
- The ENV variable, shown in Figure 7-31, that is specified in $HOME/.profile can be set to a file name which is a shell script that sets environment variables.
- A shell user can use the **env** command to display the environment variables for his/her session, and to change any of these variables. The change will only last for the length of the session.

The .profile file must be located in a user's home directory. Figure 7-31 shows the contents of the sample provided by IBM. One of the variables in the .profile is called ENV, and this variable can be used to point to another file where a user can add environment variables and shell commands that he/she wants to perform when the shell is invoked.

```
ENV=$HOME/.setup
export ENV
# Append your home directory to the current path.
PATH=$PATH:$HOME:
# Set the default editor to ed.
EDITOR=ed
# Set the prompt to display your login name, and current
# directory.
PS1='$LOGNAME':'$PWD':' >'
# Export the variable settings so that they are known
# to the system.
export PATH EDITOR PS1
```

Figure 7-31 /u/jane.profile

8

Exploitation

In this chapter, the following topics are discussed:

- BookManager BookServer
- DFS SMB
- HTTP Server
- Infoprint Server
- Java
- NFS
- Text Search
- Tivoli Storage Manager

8.1 BookManager BookServer

The IBM BookManager BookServer is a specialized World Wide Web server that lets information providers make BookManager electronic libraries, containing electronic books, bookshelves, and bookcases available on the Internet. You can read and search for information in these libraries using any Web browser such as Netscape Navigator or Internet Explorer. Hypertext links within a BookManager electronic book can link to any Internet resource.

8.1.1 Publish on the Web

BookManager BookServer translates your BookManager book text into HTML and serves it to a Web browser when requested. BookManager BookServer translates the pictures in your books into GIF format (if not already stored in the book in a Web-ready format) and serves them up at the same time. This means you can maintain one copy of your source in its original word processor format and use it on multiple platforms and media, such as softcopy on a workstation or the Web, or traditional hardcopy.

8.1.2 Read BookManager books on the Web

Use your favorite Web browser to access BookManager electronic books, bookshelves, and bookcases on the Web, taking full advantage of BookManager's powerful search capabilities and easy toolbar navigation.

Additional information

See *IBM BookManager BookServer for World Wide Web for z/OS: Getting Started,* SC31-8814*.*

8.2 DFS SMB

The SMB[1] support provides a server that makes Hierarchical File System (HFS) files and data sets available to SMB clients. The data sets supported include sequential data sets (on DASD), partitioned data sets (PDS), partitioned data sets extended (PDSE) and Virtual Storage Access Method (VSAM) data sets. The data set support is usually referred to as Record File System (RFS) support.

The SMB protocol is supported through the use of TCP/IP on z/OS. This communication protocol allows clients to access shared directory paths and shared printers. Personal Computer (PC) clients on the network use the file and print sharing functions that are included in their operating systems. Supported SMB clients include Microsoft Windows 98, Windows NT® 4.0 Workstation, and Windows 2000 Professional. At the same time, these files can be shared with local z/OS UNIX applications and with DCE DFS clients.

Note: Throughout this topic, references are made to HFS. Unless otherwise stated, HFS is a generic reference that includes HFS, ZFS, TFS, and AUTOMNT file system data. If you are in a sysplex with Shared HFS, SMB support of ZFS is limited to ZFS compatibility mode file systems.

In addition, Windows SMB clients can make remote print requests to z/OS printers that are connected to the Infoprint Server for z/OS (OS/390 Version 2 Release 8 or later).

[1] Server Message Block (SMB) is a protocol for remote file/print access used by Windows clients. This protocol is also known as Common Internet File System (CIFS).

8.2.1 SMB support features

PC users work with files on their computers. Files are used to store data, programs, and other information. Files are stored on a disk on the computer. There may be several disks on the computer. Each of these disks is referred to by a different drive letter (for example, A:, B:, C:, D:, etc.). PC users can read or write files on different disks on their computer by using the appropriate drive letter in the file name (for example, D:\dir1\file1).

PC users also work with printers attached to their computers. When a print request is made, the PC user can choose which printer the print request should go to.

Restriction: The z/OS SMB server supports basic file and print serving. It does not necessarily support all functions that a Windows file server supports. For example, the z/OS SMB server does not support Kerberos authentication or DFS. There may be other functions that are not supported.

SMB support allows PC users to be able to access files that reside on a z/OS system remotely. That is, PC users can access files that are not located on their computer. Remote files simply appear to the PC user on one or more separate drive letters. PC users can "connect" an unused drive letter to a "shared resource" on a remote computer. This is sometimes referred to as "mapping a network drive". This capability is provided by software that resides on the PC (the client), in combination with software that resides on the remote computer (the server). There must also be a TCP/IP network connection between the PC and the remote computer.

In addition, SMB support allows Windows PC users to be able to use remote printers that are attached to a z/OS system. Remote printers simply appear to be additional printers that are available to the PC user. Remote printers are installed on PCs using existing commands or install utilities.

8.2.2 SMB processes

SMB support provides a server process that makes file data and printers available to PC users. It allows an administrator to define shared directories and shared printers. It also handles PC requests to connect to the server process to satisfy file or print requests.

Another SMB process is the control process (also known as the DFS Control Task). It oversees the server process. When SMB support is started, it is really the DFS Control Task that is started. The DFS Control Task, in turn, starts the server process. If the server process ends abnormally for some reason, the DFS Control Task can automatically restart the server process.

8.2.3 Shared directories

In order to allow PCs to access remote files (located on a z/OS system), one or more shared directories must be created on the z/OS system. Distributed File Service administrators make files available to SMB clients by creating shared directories. A shared directory is given a share name and specifies a directory path name in a file system. Any directory can be shared with clients. To access shared directories from a PC, clients can map a network drive by choosing an available drive letter and mapping it to a computer name and a share name, or they can use Universal Naming Convention (UNC) mapping. The computer name is the name of the Distributed File Service SMB server and the share name is the name of the shared directory created on that server. After this is done, the remote files can be read or written as though they were local files.

8.2.4 Shared printers

Distributed File Service administrators make z/OS printers available to Windows PCs by creating shared printers. A shared printer is given a share name and specifies a z/OS Infoprint Server printer definition name. To access a remote printer from a Windows PC, clients can install a remote printer or they can use commands. After this is done, the remote printers can be used as though they were local printers.

Additional information

See *z/OS Distributed File Service SMB Administration,* SC24-5918.

8.3 HTTP Server

The HTTP Server is a scalable, high-performance Web server that brings you state-of-the-art security, dynamic caching capabilities, advanced server statistic reporting, and site indexing. It allows you to exploit Java to build dynamic, personalized Web sites and use the Platform for Internet Content Selection (PICS) to both rate and filter Web content. With the HTTP Server, you can establish an effective presence on the World Wide Web, reach customers and suppliers around the world, and conduct secure electronic commerce.

8.3.1 Additional information

See *z/OS HTTP Server Planning, Installing, and Using,* SC34-4826.

8.4 Infoprint Server

Infoprint Server is an optional feature of z/OS that uses z/OS UNIX System Services. This feature is the basis for a total print serving solution for the z/OS environment. It lets you consolidate your print workload from many servers onto a central z/OS print server, as shown in Figure 8-1 on page 319.

Infoprint Server delivers improved efficiency and lower overall printing cost with the flexibility for high-volume, high-speed printing from anywhere in the network. With Infoprint Server, you can reduce the overall cost of printing while improving manageability, data retrievability, and usability.

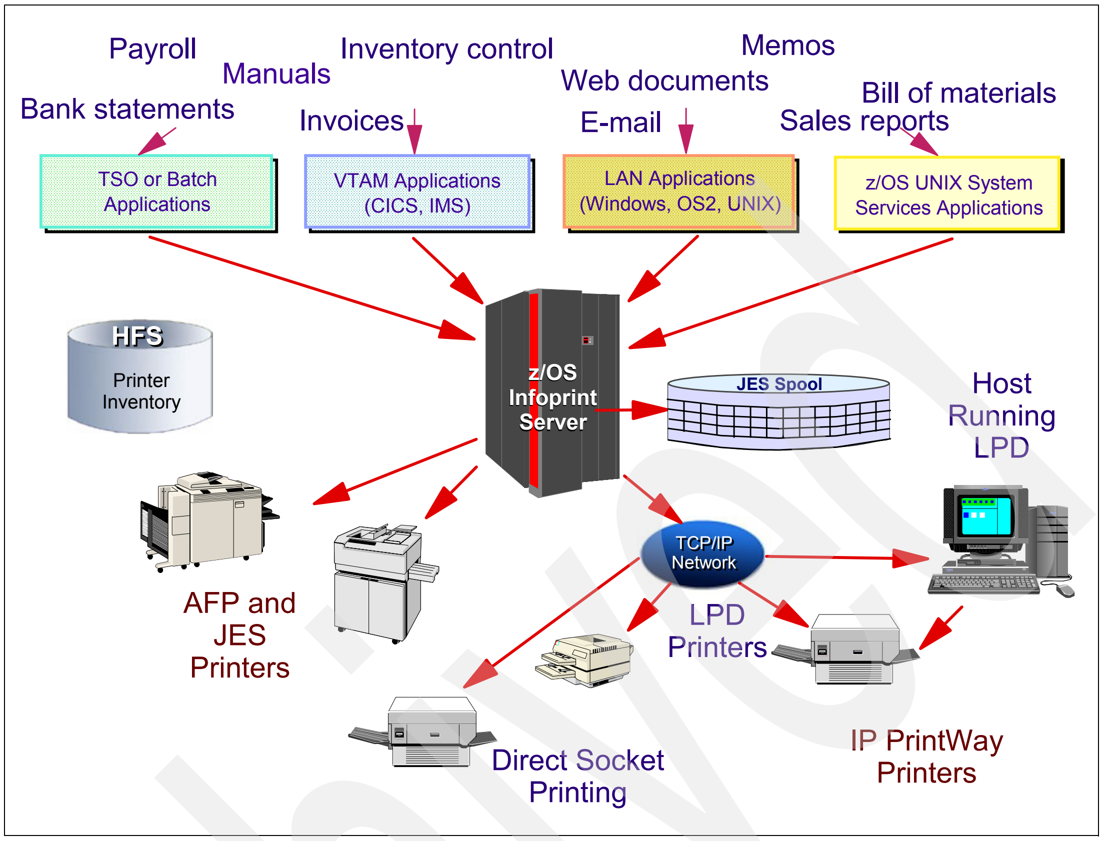


Figure 8-1 Infoprint Server overview

8.4.1 Printing from UNIX System Services

From the z/OS UNIX Shell you can print to any printer defined in the Printer Inventory of the z/OS Infoprint Server. You can print on local printers attached directly to z/OS, or on remote printers in a TCP/IP LAN network.

Your system administrator assigns a name to each printer defined in the Printer Inventory. To print, you need to know this name. A printer in the Printer Inventory can be a physical printer or a pool of physical printers that can print the same types of files. Or, your administrator can define more than one printer name for the same physical printer, so that you can use a different printer name for printing files with different characteristics.

The z/OS Infoprint Server attempts to validate that your file can print on the selected printer before accepting your print request. For example, if the printer you select cannot print the type of data (PostScript, PCL, and so on) in your file, the z/OS Infoprint Server does not accept your request and sends you a message.

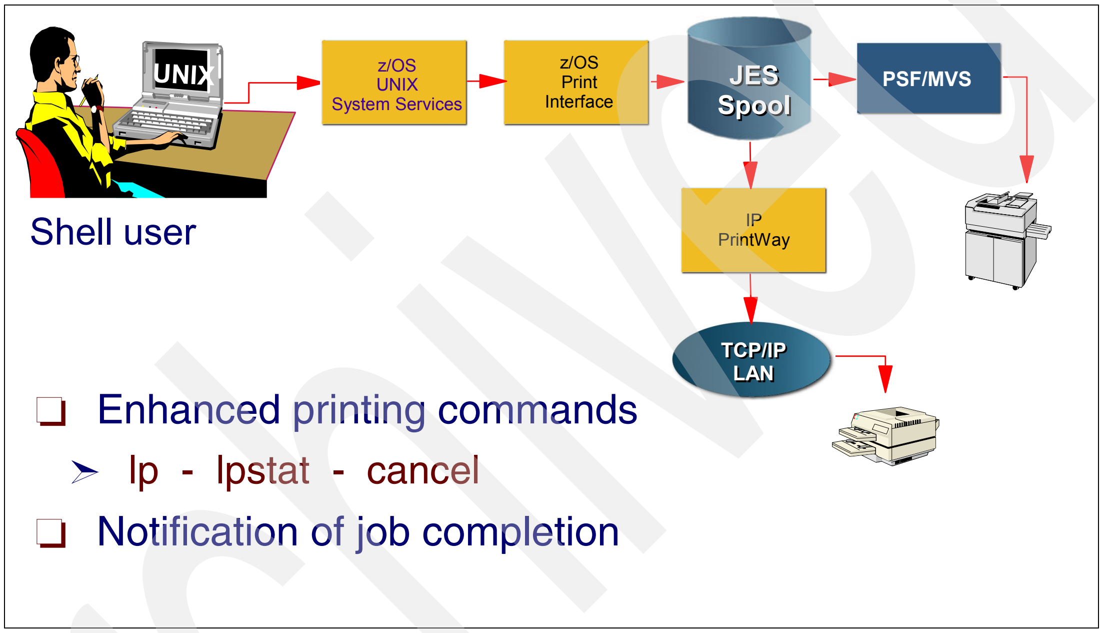
Note: A new environment variable was introduced with z/OS v1R6, _BPX_UNLIMITED_OUTPUT=YES. With this variable set, warning message are displayed when output limits are being reached. To use this variable you must have read access to the BPX.UNLIMITED. OUTPUT security profile in the FACILITY class.

Enhanced printing commands

The z/OS UNIX printing commands provide by Infoprint Server, provided in /usr/lpp/Printsrv/bin and shown in Figure 8-2 on page 320, have enhanced function over the commands of the same name described in the *z/OS UNIX System Services Command Reference,* SA22-7802. For example, when printing on AFP™ printers, you can specify options such as duplexing or a special overlay. You can also query the status of your print request, and you can cancel a print request. These printing commands adhere to the UNIX standards in XPG4.2, so that you do not need to change your UNIX applications when you port them to z/OS.

Make sure that the correct path to the commands is in /etc/profile as follows:

PATH=/usr/lpp/Printsrv/bin:/bin:



- Enhanced printing commands
  - lp - lpstat - cancel
- Notification of job completion

Figure 8-2 Printing from the shell session

8.4.2 UNIX commands with Infoprint Server

The **lp**, **lpstat**, and **cancel** commands use TCP/IP protocol to send print requests to the Print Interface. They send commands to the port number specified in the Print Interface configuration file.

lp	Send a job to a printer
lpstat	Query printers, locations, and status of jobs
cancel	Cancel a print job

These commands are modified to be used with the Print Interface and are placed in the HFS as follows:

/usr/lpp/Printsrv/bin

The lp command prints one or more files, or sends the files to an e-mail destination. The address of the printer, or the e-mail address list, is specified in the printer definition in the Infoprint Server Printer Inventory, which your administrator manages. The files can be:

- MVS data sets, such as partitioned data sets or sequential data sets
- Hierarchical file system (HFS) files

- Lists of printable files

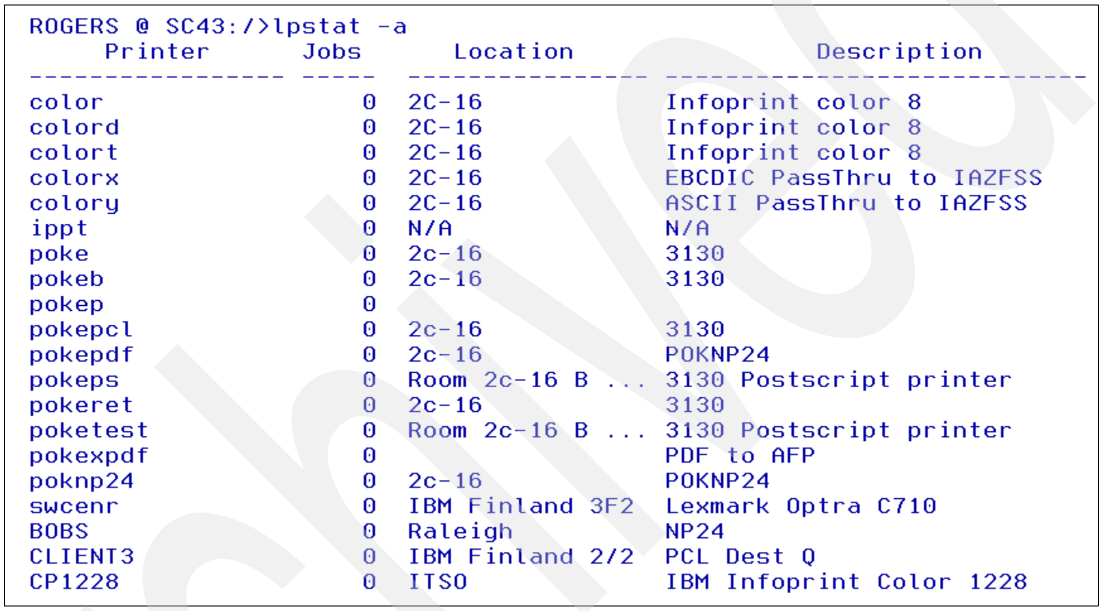
The `lp` command returns an Infoprint Server job ID, which you can use to query or cancel the job.

Query printers in the Printer Inventory

A z/OS UNIX user can query the status of all printers defined in the Printer Inventory as follows:

```
lpstat -a
```

Returned to the user's screen, shown in Figure 8-3, are printers in the Infoprint Server Printer Inventory (not all printers are shown due to the total number).



```
ROGERS @ SC43:/>lpstat -a
     Printer       Jobs      Location                  Description
----------------- ----- ---------------- ------------------------------
color               0   2C-16            Infoprint color 8
colord              0   2C-16            Infoprint color 8
colort              0   2C-16            Infoprint color 8
colorx              0   2C-16            EBCDIC PassThru to IAZFSS
colory              0   2C-16            ASCII PassThru to IAZFSS
ippt                0   N/A              N/A
poke                0   2c-16            3130
pokeb               0   2c-16            3130
pokep               0
pokepcl             0   2c-16            3130
pokepdf             0   2c-16            POKNP24
pokeps              0   Room 2c-16 B ... 3130 Postscript printer
pokeret             0   2c-16            3130
poketest            0   Room 2c-16 B ... 3130 Postscript printer
pokexpdf            0                    PDF to AFP
poknp24             0   2c-16            POKNP24
swcenr              0   IBM Finland 3F2  Lexmark Optra C710
BOBS                0   Raleigh          NP24
CLIENT3             0   IBM Finland 2/2  PCL Dest Q
CP1228              0   ITSO             IBM Infoprint Color 1228
```

Figure 8-3 z/OS UNIX shell user issues lpstat -a to display all printers

The other options for the `lpstat` command are:

-d Query default printer Example: lpstat -d
-o Query specified printer and jobs Example: lpstat -o poke
-p Query specified printer Example: lpstat -p poke
-t Query all printers and jobs Example: lpstat -t
-u Query all printers and jobs by user ID Example: lpstat -u ROGERS
-a Query names and locations of all printers Example: lpstat -a

Send a print request

UNIX System Services users can use the `lp` command to print data sets to printers defined to the Print Interface. The UNIX user issues the `lp` command, as shown in Figure 8-4, specifying the destination (**-d pokeps**) as the Print Interface defined printer. The data set to be printed is an MVS data set: test.jcl

To send a print request to print an MVS data set owned by the submitting user ID to the Print Interface, issue the following:

```
ROGERS @ SC43:/>lp -d pokeps //test.jcl
AOP007I Job 67289 successfully spooled to pokeps.
ROGERS @ SC43:/>

 ===>
```

Figure 8-4   z/OS UNIX user submitting a print request

The job ID returned in the message is PS067289.

Note: If you do not specify any files on the command line, or if you specify a dash (-) for the file name, **lp** prints from standard input.

Notification of job completion

Specifying the **-m** option notifies you by electronic mail when the file is removed from the system spool for any reason. Some reasons are:

- The file has finished printing.
- The file has been transmitted to a local area network (LAN) printer or to the z/OS UNIX sendmail function. You might receive notification before the file has finished printing or been sent to the e-mail destination. You might receive notification even though a transmission error has occurred.

  If your administrator has requested that Infoprint Server retain files on the system spool after transmission, you receive notification after the retention time expires.
- The operator has deleted the file.

Cancel a print request

Use the **cancel** command to cancel a job. For example, you realize that you need to make some changes in the file that you just sent to print on pokeps. If you don't remember the job ID that the **lp** command returned, use the **lpstat** command to query all the jobs that you submitted to pokeps.

To cancel a print request that has not yet been printed, issue:

```
cancel 67289
```

Print request examples

To send a job to print or to send one or more files to print, use the **lp** command. For example, to print three copies of myfile1 and myfile2 on pokeps, enter:

```
lp -d pokeps -n 3 myfile1 myfile2
```

- Find out where the printers are

  Use the **lpstat** command to query printer names and locations. For example, to see the names and locations of all printers known to the Printer Inventory, enter:

  ```
  lpstat -a
  ```
- Find out if a job is printing

  You can also use the **lpstat** command to query the status of a job. For example, you submitted several jobs to print and want to know if any of them are printing. To query all your jobs submitted to printer pokeps, enter:

  ```
  lpstat -o pokeps
  ```

To determine which jobs have been submitted to each printer, specify:

```
lpstat -t
```

This command shows all the printers defined to the Print Interface and all the jobs queued to the Print Interface printers, as shown in Figure 8-5 on page 323.

```
Printer: poke
 Job   Owner    Status   Format   Size    File
----- -------- --------- ------ --------  ------------------------------
  285 ROGERS   pending   text       3149  ROGERS.TEST.JCL
  286 ROGERS   pending   text       3109  ROGERS.TEST.JCL
  287 TCPIPOE  pending   text       2960  //test.jcl
  288 pc-user  pending   text       2997  test.jcl
  289 ROGERS   pending   text       3108  TEST.JCL
  290 ROGERS2  pending   pcl       20902  ... About the IBM AFP Printer"
  291 ROGERS2  pending   pcl       19019  Printing "Options Dialog"
  296 ROGERS   pending   text       3109  ROGERS.TEST.JCL
  297 ROGERS2  pending   pcl       41436  readme95 - Notepad
  298 ROGERS2  pending   pcl       41436  readme95 - Notepad
  311 ALCIDES  pending   text       2960  //test.jcl

Printer: pokeps
 Job   Owner    Status   Format   Size    File
----- -------- --------- ------ --------  ------------------------------
  292 ROGERS2  pending   modca     19422  scop.AOI
  293 ROGERS2  pending   pcl       41436  readme95 - Notepad
  294 ROGERS2  pending   pcl       41436  readme95 - Notepad
  295 ROGERS2  pending   pcl       41436  readme95 - Notepad
  299 ROGERS2  pending   modca      3650  word.AOI
  301 ROGERS2  pending   modca     19506  word.AOI
  302 ROGERS2  pending   modca     19506  word.AOI
  304 ROGERS2  pending   modca     19422  word.AOI
```

Figure 8-5 Sample output from the lpstat -t command

8.5 Java support on z/OS

Java support for z/OS has been available for a considerable period of time. In this chapter we discuss the latest available Java level and tell you about some considerations you may face.

8.5.1 What is Java?

Java is a language whose specifications are maintained by SUN Microsystems. The most important task of SUN Microsystems is to guard the Java philosophy:

Develop once, run everywhere

Java is packaged as a set of tools, better known as the Software Development Kit, or SDK. The SDK consists of a number of things:

- First of all there is the runtime environment, known as the Java Virtual Machine (JVM™). This is the layer between the Java runtime code and the z/OS runtime environment. Basically, the JVM is the computing engine that executes Java applications on the zSeries platform and it runs as a UNIX process in z/OS. Each time the JVM is started, an address space is spawned. See also Figure 8-6 on page 324.

- And then there is a set of APIs, part of the SDK—a constantly growing collection. These APIs are a set of Java classes, which you can freely reuse while developing an application.

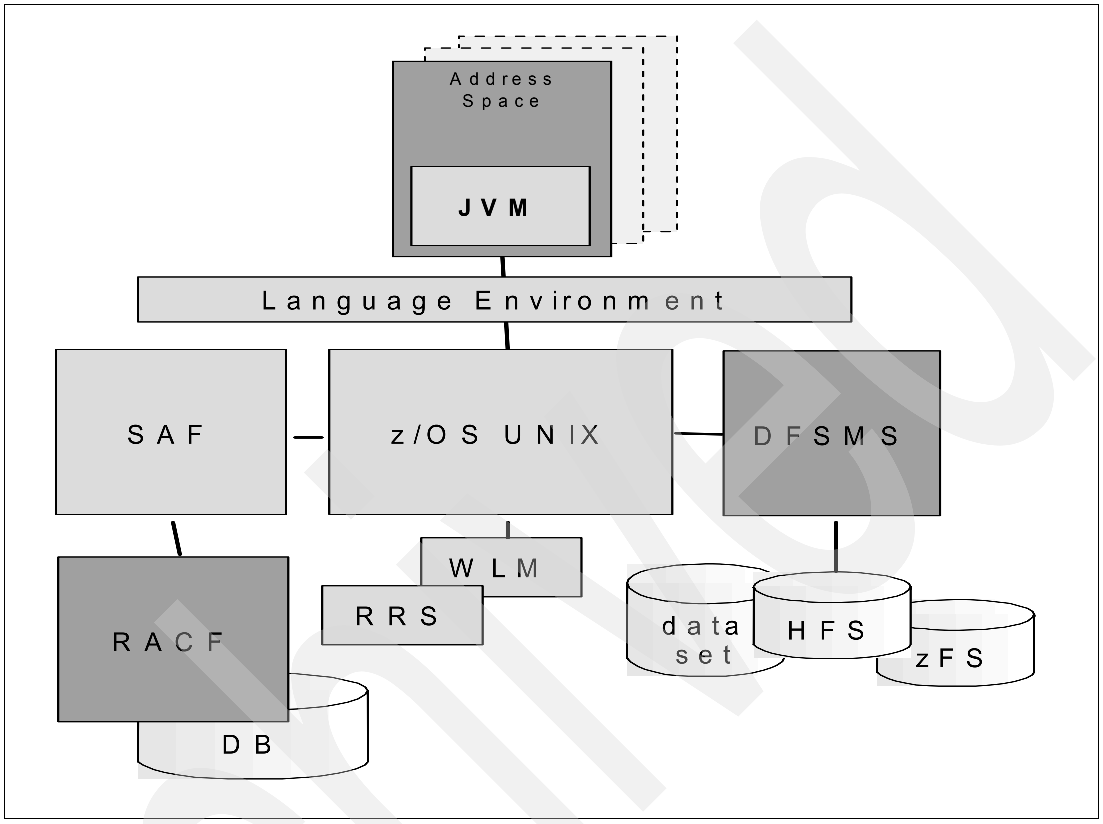


Figure 8-6 Java Virtual Machine on z/OS

8.5.2 SDK installation and setup

At this moment there are two versions of the Java2 Technology Edition:

- IBM SDK for z/OS, Java 2 Technology Edition, Version 1.4
- Java 2 Technology Edition, SDK1.3.1

IBM SDK for z/OS, Java 2 Technology Edition, Version 1.4

This version provides a full-function Software Development Kit (SDK) at the Java 2 technology level, compliant with the Sun SDK 1.4 APIs. For more information on the actual list of supported APIs, check the following URL:

http://java.sun.com/j2se/1.4/docs/api/

Before June 2003, Java 2 version 1.4 was only supported by z/OS V1.4. But PTF UQ77468 enables you to run Java 2 V1.4 also on:

- z/OS Version 1 Release 2 or above
- z/OS.e Version 1 Release 3 or above

Java 2 Technology Edition, SDK1.3.1

This version provides a complete Java 2 Technology Development Kit SDK at 1.3 level for the zSeries and S/390 platforms. For the complete API function list, check the following URL:

http://java.sun.com/j2se/1.3/docs/api/

Java 2 Technology Edition V1.3 requires either:

- OS/390 Version 2 Release 8 or above
- z/OS Version 1 Release 1 or above

Note: It is highly recommended to install the latest version of the Java 2 Technology Edition. Java 2 is to be continuously improved with new APIs and performance enhancements. Also Java 2 V1.4 is strongly compatible with previous versions of the Java 2 platform. Almost all existing programs should run on Java 2 V1.4 without modification. Although they are very rare, there are some minor incompatibilities with V1.3 that may arise in exceptional circumstances. The Internet can provide you with the latest information on this subject at URL:

http://java.sun.com/j2se/1.4/compatibility.html

Downloading the kit

Both versions of the Java 2 Technology Edition are available in both SMP/E and non-SMP/E format.

However, the SMP/E format of Java 2 Technology Edition version 1.4 is only available via a product order at IBM Software Manufacturing. Version 1.3 is still downloadable from the Internet in the SMP/E format. The non-SMP/E format of Java 2 Technology Edition version 1.4 is available from the Web. There is no charge for either product.

Note: Be aware that installing a new version of Java in your standard SMP/E zone will delete the previous version. This could give you some trouble, if you are still planning to keep an older version around.

The non-SMP/E Java 2 V1.4 can be downloaded at URL:

http://www-1.ibm.com/servers/eserver/zseries/software/java/getsdk14.html

Follow the installation instructions after you download the non-SMP/E format.

Note: Like many other z/OS products, Java 2 Technology Edition version 1.4 has its list of prerequisites. Check the the following Web site for the latest APARs that need to be applied:

http://www-1.ibm.com/servers/eserver/zseries/software/java/prereqs14.html

Verifying the installation

Your path should contain the binary directory where Java is located. Type:

```
export PATH=/usr/lpp/java/J1.4/bin:$PATH
```

Now verify that Java is correctly installed:

```
java -fullversion
```

Java should reply with the correct version and build date of the SDK, as shown in Figure 8-7.

```
PATRICK @ SC64:/u/patrick>java -fullversion
java full version "J2RE 1.4.1 IBM z/OS Persistent Reusable VM build cm141-200305
23b"
PATRICK @ SC64:/u/patrick>
```

Figure 8-7 Java version

8.5.3 Considerations when using Java

Using Java in batch

Instead of executing your Java program directly by hand from the z/OS UNIX shell prompt, it is also possible to run a Java application as a traditional batch job in z/OS. Batch jobs written in the Job Control Language (JCL) and submitted by a user run in a separate address space. The actual Java program then runs under z/OS UNIX System Services, also known as USS.

The utility to execute Java programs in batch is BPXBATCH. Or use BPXBATSL, which provides an alternate entry point in BPXBATCH. See 9.6, “BPXBATCH” on page 376 for more information about this utility.

Running Java in batch is very useful for long-running Java programs, or server type Java programs. But there is another great advantage in running Java as a batch program, and that has to do with the region size.

The region size is very much related to the JVM heap size. The heap is the runtime data area from which memory for all class instances and arrays is allocated. The heap is created at virtual machine startup. Heap storage for objects is reclaimed by an automatic management system known as the *garbage collector*. See also 8.5.4, “Garbage collection” on page 328.

Region size

Java programs can take up a lot of system resources, sometimes even more than you may expect. Therefore, it is always necessary to examine your region size and allow for heap and stack storage requirements, plus LE and Java code, and LE internal control blocks.

The region size describes the amount of storage in which a user is allowed to run or execute programs. This value determines what kinds of programs (depending on their size) and how many programs are executable at the same time.

TSO users

In principle, TSO users get the region size from the logon panel to their programs. See Figure 8-8 on page 327. If a user enters the z/OS UNIX shell by issuing a TSO OMVS command, the parent process doesn't run in the user's address space. Under control of the WorkLoad Manager, an address space is created that gets the same region size as the TSO/E user itself.

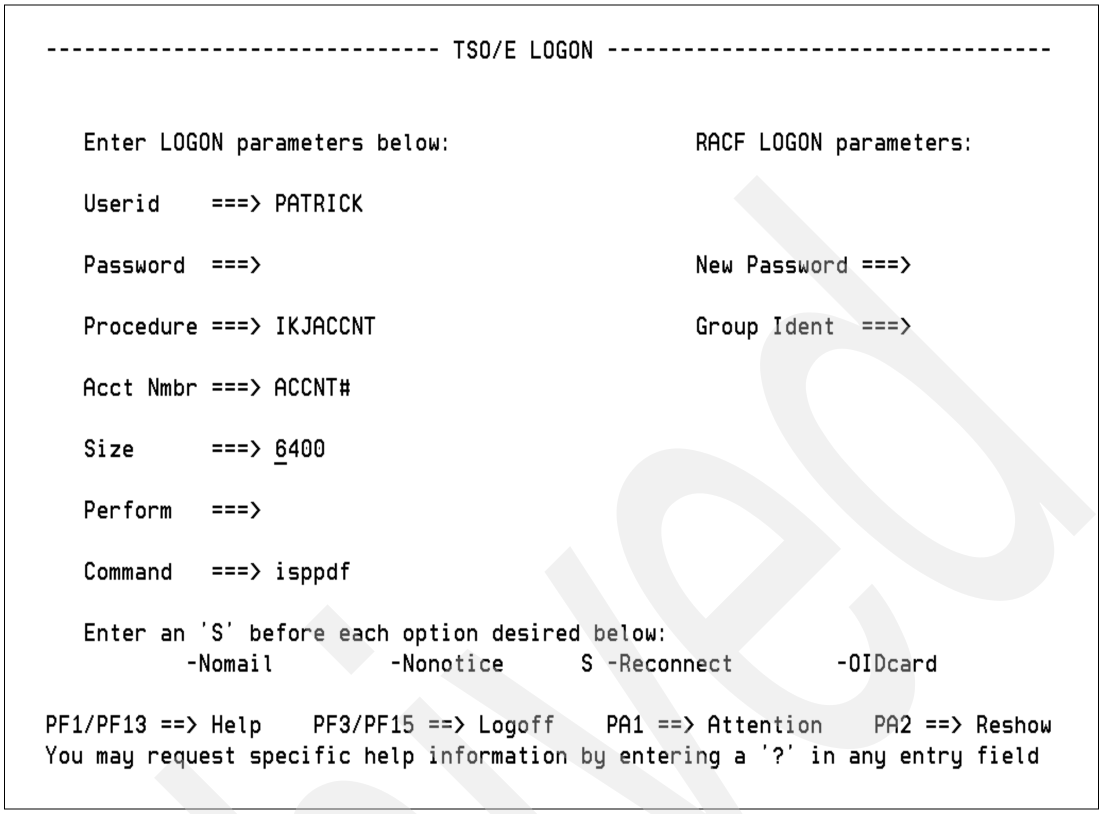

```
------------------------------ TSO/E LOGON ----------------------------------


   Enter LOGON parameters below:                 RACF LOGON parameters:

   Userid    ===> PATRICK

   Password  ===>                                New Password ===>

   Procedure ===> IKJACCNT                       Group Ident  ===>

   Acct Nmbr ===> ACCNT#

   Size      ===> 6400

   Perform   ===>

   Command   ===> isppdf

   Enter an 'S' before each option desired below:
           -Nomail         -Nonotice      S -Reconnect        -OIDcard

PF1/PF13 ==> Help    PF3/PF15 ==> Logoff    PA1 ==> Attention    PA2 ==> Reshow
You may request specific help information by entering a '?' in any entry field
```

Figure 8-8 TSO logon screen

The maximum region size you can specify on your TSO logon panel depends on the settings in your TSO profile in RACF. You can check the maximum size with the LISTUSER command:

```
TSO lu patrick tso
```

The LISTUSER command provides you with information about the initial logon size (SIZE) and the maximum size (MAXSIZE) you are able to specify on your logon panel.

```
TSO INFORMATION
---------------
ACCTNUM= ACCNT#
PROC= IKJACCNT
SIZE= 00006072
MAXSIZE= 00000000
UNIT= SYSALLDA
USERDATA= 0000
COMMAND= isppdf
```

In our example you see an initial size of around 6 MB and an unlimited maximum size.

Batch users

If BPXBATCH is used, there is a BPXBATCH REGION parameter that describes the region size; however, started procedures that create z/OS UNIX processes use the REGION parameter on the EXEC statement.

```
//BPXBATCH EXEC PGM=BPXBATCH,REGION=0M
```

It is very useful to use the BPXBATCH approach. Java programs can easily take more memory than you can give them when using the TSO logon panel.

Telnet and rlogin users

Telnet and rlogin users get their region size from the ASSIZEMAX parameter in the users' OMVS segment. Since OS/390 V2R8, it is possible to add individual z/OS UNIX user limits to the users' OMVS segment.

Use the following command to display a user's OMVS segment information:

```
TSO lu patrick omvs
```

Below you can see the OMVS segment information for a specific user. This user has an ASSIZEMAX of 20 MB specified.

```
OMVS INFORMATION
----------------
UID= 0000068216
HOME= /u/patrick
PROGRAM= /bin/sh
CPUTIMEMAX= NONE
ASSIZEMAX= 0020480000
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE
```

There is also another setting in the SYS1.PARMLIB(BPXPRMxx) member, called the MAXASSIZE parameter. MAXASSIZE is the system-wide limit and represents the maximum size for an address space created by one of the z/OS UNIX daemons. With the ASSIZEMAX parameter in your users' OMVS segment, it is possible to set an even higher limit for individual users.

Note: IEFUSI is a user exit where an installation can set the region size and region limit for all programs that run under this job step. Make sure this exit does not change the region size setting for the z/OS UNIX process.

8.5.4 Garbage collection

Garbage collection is part of the JVM. This task is carried out in the background to reclaim unusable space and is very important for the performance of the JVM.

The heap size given to a Java program can be specified using an initial and a maximum heap size parameter. An example of user-defined heap size would look like this:

```
java -Xms20MB -Xmx35MB -Xminf0.2 HelloWorld
```

Where `-Xms` specifies the initial heap size of 20 MB and `-Xmx` the maximum heap size of 35 MB. This means that our Java program will start with 20 MB of memory.

There is another parameter called `-Xminf` that represents the minimum percentage of free space for the heap. In our case it is set to 0.2, which means 20%. If the total amount of free heap drops below 20%, the JVM will expand the heap size. It will continue to do so until it reaches the value specified as the maximum heap size, `-Xmx`, or if the Address Space limit has been reached. See 8.5.3, "Considerations when using Java" on page 326 for more information about the maximum region size.

Note: For performance reasons it is recommended to use fixed-size heap, which means the `-Xms` value is equal to the `-Xmx` value. This way no heap expansion or contraction occurs and this can lead to significant performance gains in some situations.

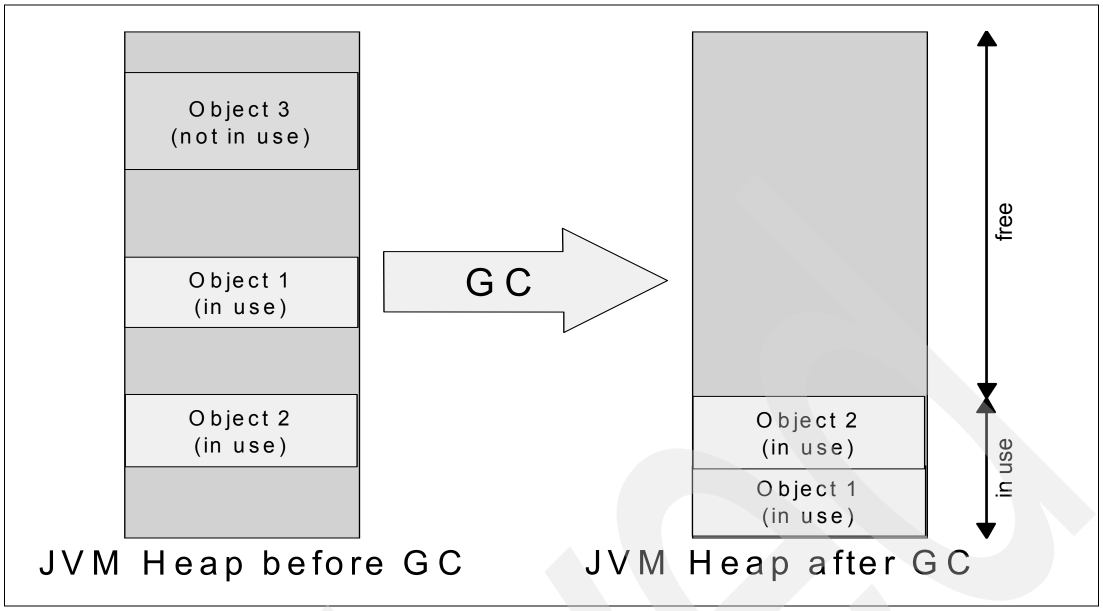


Figure 8-9 Garbage Collection in effect

When a Java program makes a request for some storage and the JVM is unable to comply, an allocation failure occurs. See also verbosegc logs in Figure 8-10. This allocation failure triggers the Garbage Collection process, as shown in Figure 8-9. Garbage Collection is the process of automatically freeing objects that are no longer referenced by the program.

It is better to avoid very frequent Garbage Collections. Tune the Garbage Collection frequency by specifying enough heap size for your program. Giving your program too much heap size will most likely lead to other resource problems, so try to find some balance.

```
<<GCí0ù: Expanded System Heap by 65536 bytes
<<AFí1ù: Allocation Failure. need 144 bytes, 0 ms since last AF>
<<AFí1ù: managing allocation failure, action=1 (0/533658112) (3145728/3145728)>
<<GC(1): GC cycle started Mon Jul 28 15:37:48 2003
<<GC(1): freed 493486384 bytes, 92% free (496632112/536803840), in 336 ms>
   <GC(1): mark: 304 ms, sweep: 31 ms, compact: 1 ms>
   <GC(1): refs: soft 0 (age >= 32), weak 0, final 15868, phantom 0>
<<AFí1ù: completed in 365 ms>
<<AFí2ù: Allocation Failure. need 56 bytes, 322198 ms since last AF>
<<AFí2ù: managing allocation failure, action=1 (0/533658112) (3145728/3145728)>
<<GC(2): GC cycle started Mon Jul 28 15:43:11 2003
<<GC(2): freed 490277016 bytes, 91% free (493422744/536803840), in 334 ms>
   <GC(2): mark: 296 ms, sweep: 38 ms, compact: 0 ms>
   <GC(2): refs: soft 0 (age >= 32), weak 0, final 31689, phantom 0>
<<AFí2ù: completed in 340 ms>
```

Figure 8-10 Verbosegc log

A good approach would be to use the verbosegc parameter. This parameter enables verbosegc logs, which you can use to determine the best heap size settings for your program. Enabling verbosegc can have a performance impact on an application, although it is very common for server applications to keep verbosegc enabled at all times.

Note: The behavior of the Garbage Collector is related by the heap size. A small heap may produce more frequent, but shorter GC cycles. A large heap size, on the other hand, will produce less frequent but longer GC cycles.

Common Java parameters

Common Java parameters are, for example:

-Xms*<size>*

This option sets the initial size of the heap. If this option is not specified, a value of half the platform-dependent default value is used. Default: 1000 KB/2 = 500 KB (z/OS default).

-Xmx*<size>*

This option sets the maximum heap size. If this option is not specified, a platform dependent value is used. Default: 64 MB (z/OS default).

-Xminf*<size>*

This option specifies the minimum percentage of free space for the heap. If this option is not specified, the default value is used. Default: 0.3 (that is, 30%)(z/OS default).

-Xmaxf*<size>*

This option specifies the maximum percentage of free space for the heap. If this option is not specified, the default value is used. Default: 0.6 (that is, 60%)(z/OS default).

Note: For a full list of parameters see also the *New IBM Technology Features Persistent Reusable Java Virtual Machines*, SC34-6201.

8.5.5 Tuning Java and LE runtime options

The Language Environment (LE) handles all the runtime services that Java applications use, such as:

- Message handling
- Storage management
- Condition handling
- Math functions

In fact, the LE provides a common runtime environment for IBM versions of certain high-level languages. Because the IBM JVM for z/OS is written in C, the LE is not able to tell the difference between pthreads created by the JVM, or pthreads created by another C application. Therefore, heap storage and stack storage requirements by Java programs are treated by the LE as if it was a C program.

The Java heap is managed by the Garbage Collector and is malloc'ed from the LE heap. Storage is acquired by using the C library function malloc(). If the call succeeds, a process is given the requested amount of virtual memory pages from the operating system.

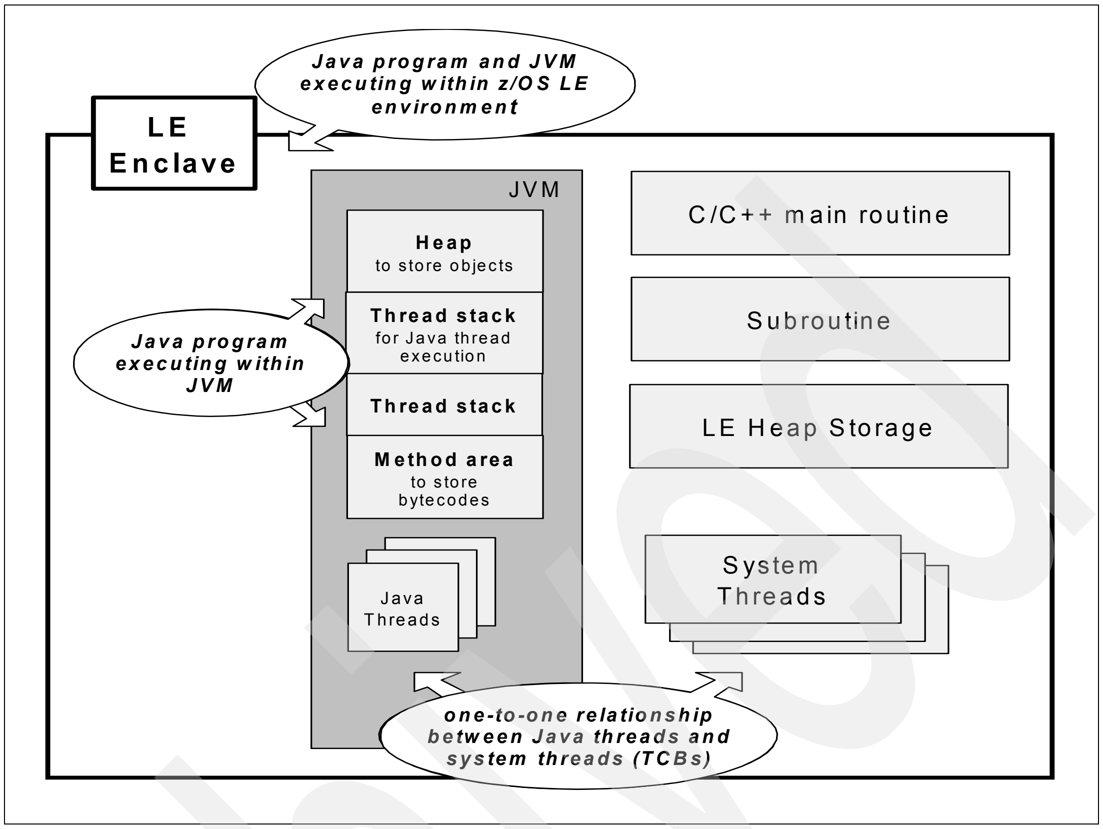

Java program and JVM executing within z/OS LE environment
LE Enclave
JVM
Heap to store objects
Thread stack for Java thread execution
Thread stack
Method area to store bytecodes
Java Threads
Java program executing within JVM
C/C++ main routine
Subroutine
LE Heap Storage
System Threads
one-to-one relationship between Java threads and system threads (TCBs)

Figure 8-11 JVM in the z/OS LE environment

There is a one-to-one relationship between Java threads and system threads (see Figure 8-11). Because each Java thread is mapped to a native z/OS thread, it allows us to create multitasking and multiprocessor applications.

Controlling the storage allocation

Specify the RPTSTG(ON) runtime option to generate a report of the storage a program used during its run. The storage report, shown in Example 8-1, provides statistics that can help you understand how space is being consumed.

Examine the report and check the number of segments allocated. The report can give you a good hint, to start out with a new initial size. Any of the runtime options can be overridden using the _CEE_RUNOPTS environment variable.

Example 8-1 storage report

```
Storage Report for Enclave main 07/28/03 4:36:38 PM
Language Environment V01 R04.00

    STACK statistics:
      Initial size:                                  16384
      Increment size:                                 4096
      Maximum used by all concurrent threads:        11968
      Largest used by any thread:                    11968
      Number of segments allocated:                      1
      Number of segments freed:                          0
    THREADSTACK statistics:
      Initial size:                                  16384
      Increment size:                                 4096
```

```
  Maximum used by all concurrent threads:             16632
  Largest used by any thread:                          3432
  Number of segments allocated:                           5
  Number of segments freed:                               0
XPLINK STACK statistics:
  Initial size:                                       65536
  Increment size:                                     16384
  Largest used by any thread:                         38304
  Number of segments allocated:                           1
  Number of segments freed:                               0
XPLINK THREADSTACK statistics:
  Initial size:                                       65536
  Increment size:                                     16384
  Largest used by any thread:                          8112
  Number of segments allocated:                           5
  Number of segments freed:                               0
LIBSTACK statistics:
  Initial size:                                        1024
  Increment size:                                      1024
  Maximum used by all concurrent threads:                 0
  Largest used by any thread:                             0
  Number of segments allocated:                           0
  Number of segments freed:                               0
THREADHEAP statistics:
  Initial size:                                        4096
  Increment size:                                      4096
  Maximum used by all concurrent threads:                 0
  Largest used by any thread:                             0
  Successful Get Heap requests:                           0
  Successful Free Heap requests:                          0
  Number of segments allocated:                           0
  Number of segments freed:                               0
HEAP statistics:
  Initial size:                                     4194304
  Increment size:                                    524288
  Total heap storage used (sugg. initial size):    19632200
  Successful Get Heap requests:                       16143
  Successful Free Heap requests:                      12810
  Number of segments allocated:                          44
  Number of segments freed:                              32
HEAP24 statistics:
  Initial size:                                        8192
  Increment size:                                      4096
  Total heap storage used (sugg. initial size):           0
  Successful Get Heap requests:                           0
  Successful Free Heap requests:                          0
  Number of segments allocated:                           0
  Number of segments freed:                               0
ANYHEAP statistics:
  Initial size:                                      786432
  Increment size:                                    131072
  Total heap storage used (sugg. initial size):      309376
  Successful Get Heap requests:                         283
  Successful Free Heap requests:                        150
  Number of segments allocated:                           1
  Number of segments freed:                               0
BELOWHEAP statistics:
  Initial size:                                       24576
  Increment size:                                      2048
  Total heap storage used (sugg. initial size):           0
```

```
      Successful Get Heap requests:                           0
      Successful Free Heap requests:                          0
      Number of segments allocated:                           0
      Number of segments freed:                               0
    Additional Heap statistics:
      Successful Create Heap requests:                        5
      Successful Discard Heap requests:                       0
      Total heap storage used:                          2277376
      Successful Get Heap requests:                           5
      Successful Free Heap requests:                          0
      Number of segments allocated:                           5
      Number of segments freed:                               0
    Largest number of threads concurrently active:            6
End of Storage Report
```

Storage tuning:

- Use the RPTSTG(ON) option to get storage reports.
- Use the values returned by the RPTSTG(ON) option as the size of the initial blocks of storage for the HEAP, ANYHEAP, BELOWHEAP, STACK, and LIBSTACK runtime options. For example:

```
HEAP(19M,2M,ANY;KEEP,8K,4K)
STACK(54K,16K,ANY,KEEP,52K,13K)
```

For more information on the Language Environment Run-Time Options, see chapter 12 of *z/OS Language Environment Customization,* SA22-7564.

Note: When you use an SDK version below 1.4 you will not find XPLINK STACK and XPLINK THREADSTACK in your storage report. XPLink support is new for SDK 1.4 and works in combination with the heap storage management algorithm known as *heap pools*. See also *XPLink: OS/390 Extra Performance Linkage,* SG24-5991 for more information on when and where to use XPLink.

8.5.6 Enhanced z/OS linkage and heap pools

The latest version of SDK, version 1.4, is capable of taking advantage of enhanced z/OS linkage capabilities (XPLINK) for greatly improved performance. In most common cases, this will be done completely transparently. However, any application code that creates a JVM itself and interacts with the JVM via Java Native Interface (JNI) or any other “call” interface must create the Language Environment (LE) enclave specifying that LE should set up an XPLINK environment.

This XPLINK LE enclave must be in place prior to creating the JVM. There are a variety of ways to accomplish this. They include setting a runtime option (XPLINK(ON)), or recompiling the launching application code to be XPLINK.

The following environment runtime option should be set to accomplish this:

```
_CEE_RUNOPTS=XPLINK(ON)
```

The overall performance of the C runtime library functions, like **malloc()** for allocating memory, can improve by up to 28% when used with XPLink. But heap pools should be used also, because the performance will degrade when XPLINK is used, without the use the heap pools storage algorithm.

For more information on this feature, see *XPLink: OS/390 Extra Performance Linkage,* SG24-5991and *z/OS Language Environment Programming Guide,* SA22-7561.

Heap pools

Storage allocations were already improved in OS/390 V2R10 with the introduction of an optional heap storage management algorithm, better known as heap pools. This algorithm is designed to improve performance of mulithreaded C/C++ applications with high usage of **malloc()**, **calloc()**, r**ealloc()** and **free()**. The goal of heap pools is to reduce the overhead of many small storage requests, which are very typical for C/C++ programs. When active, heap pools virtually eliminate contention for heap storage. Remember that Java storage requirements are treated as C/C++ storage requirements by the Language Environment.

To set up the Language Environment heap pools storage options, we should create a storage report first, to examine the most optimal settings for our environment. Run your application with the heap pools storage report turned on:

```
export _CEE_RUNOPTS=”HEAPPOOLS(ON) RPTSTG(ON)”
```

You may remember the RPTSTG(ON) parameter from 8.5.5, “Tuning Java and LE runtime options” on page 330.

Note: When you use BPXBATSL to run your application and you want to specify _CEE_RUNOPTS, you have to omit the quotes (“ “). Otherwise you may receive the following message:

```
CEE3608I The following messages pertain to the invocation command run-time options.
CEE3611I The run-time option “HEAPPOOLS was an invalid run-time option or is not
supported in this release of Language Environment.
```

After you run your application with the heap pools storage report, you will see at the end of that report a heap pools summary like the one shown in Figure 8-12. In that report you see the suggestions for the pools. When used, the heap pools feature creates a number of pools, each consisting of arrays of tuned storage elements. This will eventually make allocation and freeing of elements simple.

```
HeapPools Summary:
    Cell  Extent   Cells Per  Extents    Maximum     Cells In
    Size  Percent  Extent     Allocated  Cells Used  Use
    ----------------------------------------------------------
       8    10       52428         1         682         667
      32    10       20971         1        1410        1405
     128    10        6168         1        2711        2710
     256    10        3177         1        1454        1454
    1024    10         812         3        2362        2347
    2048    10         408         3         959         953
    ----------------------------------------------------------
    Suggested Percentages for current Cell Sizes:
      HEAPP(ON,8,1,32,1,128,5,256,5,1024,30,2048,24)
    Suggested Cell Sizes:
      HEAPP(ON,80,,136,,520,,1224,,1600,,2048,)
  Largest number of threads concurrently active:           19
```

Figure 8-12 HeapPools summary after the first run

Now run the application again, but this time use the following Language Environment HeapPools storage option from `Suggested Cell Sizes`:

```
export _CEE_RUNOPTS=”HEAPP(ON,80,,136,,520,,1224,,1600,,2048,) RPTSTG(ON)”
```

```
HeapPools Summary:
     Cell  Extent   Cells Per  Extents    Maximum     Cells In
     Size  Percent  Extent     Allocated  Cells Used  Use
     ----------------------------------------------------------
       80    10       9532          1       3300        3284
      136    10       5825          1       1591        1591
      520    10       1588          2       2681        2680
     1224    10        680          3       1399        1386
     1600    10        521          1        328         325
     2048    10        408          1        256         241
     ----------------------------------------------------------
     Suggested Percentages for current Cell Sizes:
       HEAPP(ON,80,4,136,3,520,17,1224,21,1600,7,2048,7)
     Suggested Cell Sizes:
       HEAPP(ON,80,,136,,360,,704,,1240,,2048,)
   Largest number of threads concurrently active:          19
EEnd of Storage Report
```

Figure 8-13 HeapPools Summary report after the second run

Again a storage report is created for this application, as shown in Figure 8-13. For the final HeapPools runtime option, you should use `Suggested Percentages for current Cell Sizes` from the second run. In our example this should be:

```
export _CEE_RUNOPTS="HEAPP(ON,80,4,136,3,520,17,1224,21,1600,7,2048,7)"
```

Make sure you turn off the storage report after you finish tuning your heap pools, because in a non-XPLINK environment it could have a very negative impact on performance. In an XPLINK environment, the storage report has a minimal impact on performance. Also keep in mind that this heap pool tuning was very specific for this particular application. Another or changed application could show a completely different storage report, which may necessitate new heap pool tuning.

8.5.7 Reusable JVM

Persistent Reusable Java Virtual Machine is an enhancement of the Java 2 Technology Edition. It is a solution that provides faster processing of Java applications in a transaction processing environment. These short and repetitive transactions usually run in subsystems, such as CICS and DB2.

The Persistent Reusable JVM allows the use of multiple JVMs that share classes, and for each of these to be reset, thereby distributing the cost of starting the JVM over multiple runs.

As shown in Figure 8-14 on page 336, the Persistent Reusable JVM consists of a master JVM and several worker JVMs that together make a JVMSet. The master JVM controls the JVMSet by providing a system heap that contains the core API, as loaded by the bootstrap class loader, and shareable classes. This system heap is available to all worker JVMs. Because JVMs in a JVMSet all share a common system heap of system classes, the time to start a new JVM in the JVMSet is significantly reduced. The Persistent Reusable JVM provides the ability to process hundreds of transactions in a JVM. This JVM is reset between transactions or whenever necessary.

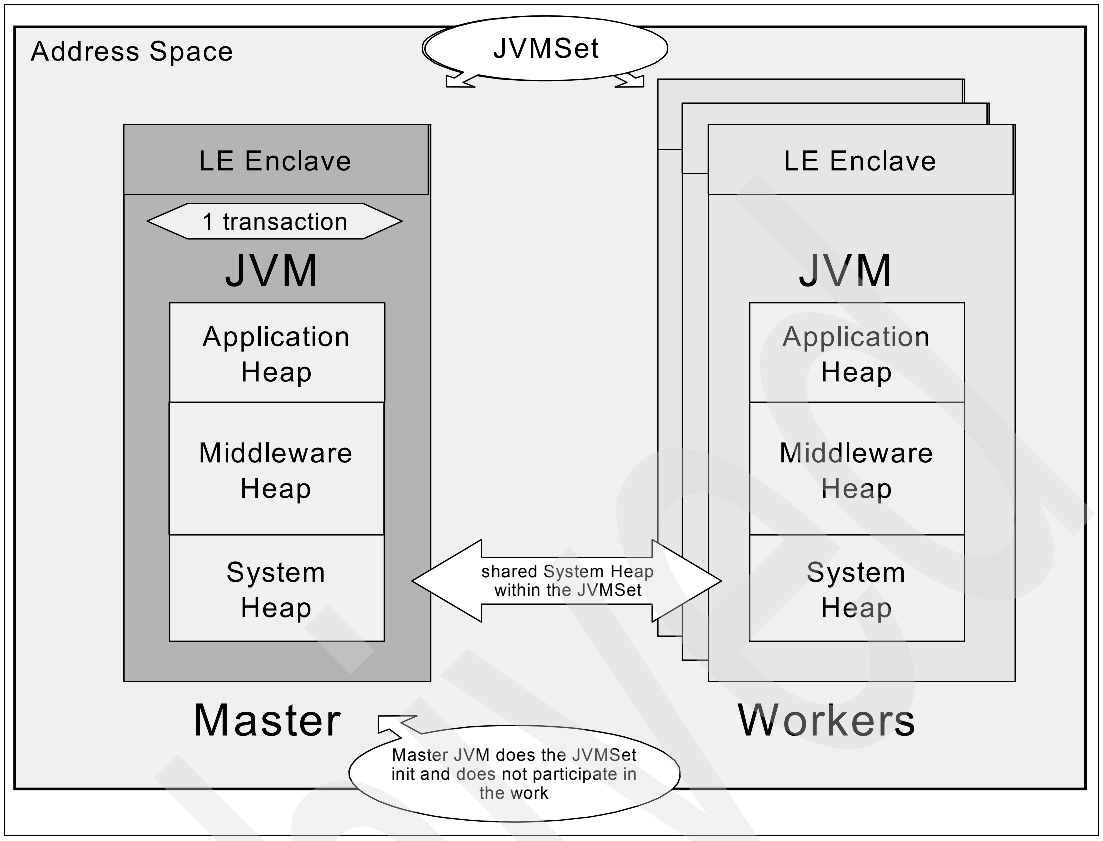


Figure 8-14 Persistent Reusable JVM

Split heaps are used by the Persistent Reusable JVM, as shown in Figure 8-14. This provides a way of grouping objects by their expected lifetime. The heaps are used as follows:

- System heap

  Contains class objects that persist for the lifetime of the JVM and are loaded just once.

- Middleware heap

  Contains class objects that have a life expectancy longer than a single transaction and that persist across JVM resets.

- Application heap or Transient heap

  Contains objects with a life expectancy tied to the transaction. At JVM reset this heap is discarded and recreated.

Middleware scenarios

Persistent Reusable JVM can be a good solution for the following scenarios:

- CICS, DB2/390: single application per JVM at any one time
  - Multiple JVMs per address space to achieve scalability.
    - CICS: 100 JVMs per address space
    - DB2: 60 JVMs per address space
  - Isolation infringements cause JVM restarts.
- IMS: single JVM per address space
  - Reduced startup time

- WebSphere for z/OS: single JVM per address space
  - Multiple long-running JVMs per image

Note: A good sample program for a Persistent Reusable JVM is provided in *New IBM Technology Features Persistent Reusable Java Virtual Machines,* SC34-6201*.*

8.6 NFS

A client is a computer or process that requests services on the network. A server is a computer or process that responds to a request for service from a client. A user accesses a service, which allows the use of data or other resources.

Figure 8-15 illustrates the client-server relationship. The upper right portion of the figure shows the z/OS NFS server. The lower right portion of the figure shows the z/OS NFS client. The left portion of the figure shows various NFS clients and servers which can interact with the z/OS NFS server and client. The center of the figure shows the Transmission Control Protocol/Internet Protocol (TCP/IP) network used to communicate between the clients and servers.

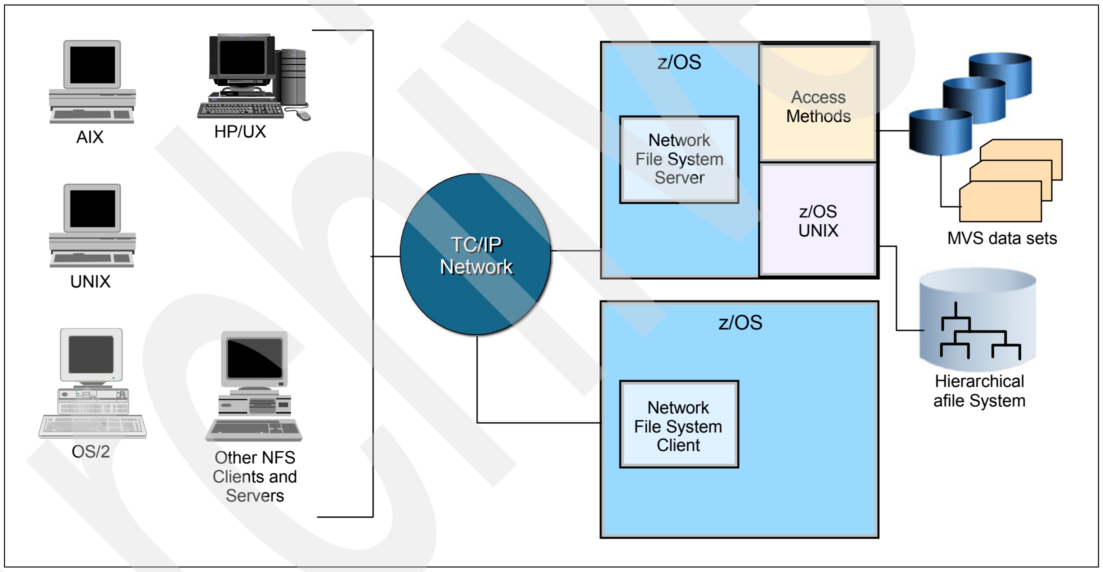

AIX
HP/UX
UNIX
OS/2
Other NFS Clients and Servers
TC/IP Network
z/OS
Network File System Server
Access Methods
z/OS UNIX
MVS data sets
Hierarchical afile System
z/OS
Network File System Client

Figure 8-15 Client-server relationship

With the z/OS NFS server, you can remotely access z/OS conventional data sets or z/OS UNIX files from workstations, personal computers, and other systems that run client software for the Sun NFS version 2 protocols, the Sun NFS version 3 protocols, and the WebNFS™ protocols over a TCP/IP network.

The z/OS NFS server acts as an intermediary to read, write, create, or delete z/OS UNIX files and MVS data sets that are maintained on an MVS host system. The remote z/OS data sets or z/OS UNIX files are mounted from the host processor to appear as local directories and files on the client system. This server makes the strengths of a zSeries host processor (storage management, high-performance disk storage, security, and centralized data) available to the client platforms.

With the z/OS NFS client you can allow basic sequential access method (BSAM), queued sequential access method (QSAM), virtual storage access method (VSAM), and z/OS UNIX users and applications transparent access to data on systems that support the Sun NFS version 2 protocols and the Sun NFS version 3 protocols. The remote NFS server can be z/OS, UNIX, AIX, or other systems. The z/OS NFS client is implemented on z/OS UNIX and implements the client portion of the Sun NFS version 2 protocols and the Sun NFS version 3 protocols.

NFS uses the communication services provided by TCP/IP, a suite of protocols that includes the remote procedure call (RPC) and External Data Representation (XDR) protocols. RPC allows a program on one machine to start a procedure on another machine, as if the procedure were local. XDR resolves the differences in data representation of different machines.

NFS, then, can be used for file sharing between platforms and file serving (as a data repository).

If you are using NFS as a file server, HFS might be a better choice than using conventional MVS data sets, because of its UNIX-like features.

8.6.1 Using z/OS UNIX files

The NFS server enables the client user remote access to z/OS UNIX files from a client workstation.

z/OS UNIX provides a hierarchical file system (HFS) for z/OS. The HFS file system is similar to a UNIX file system. All z/OS UNIX files reside in a directory, which in turn is a file in a higher level directory. The highest level directory is called the root directory.

When client users mount files from your server system, you use a common HFS prefix to distinguish z/OS UNIX files from conventional z/OS data sets. You see z/OS UNIX files in a standard UNIX format on your workstation, but the files are stored on a z/OS host system.

Using the NFS, the client can mount all or part of the z/OS UNIX file system and make it appear as part of your local file system. From there the client user can create, delete, read, write, and treat the host-located files as part of the workstation's own file system. For more information about z/OS UNIX see z/OS UNIX System Services User's Guide.

z/OS UNIX enhancements

z/OS UNIX file system support provides these enhancements over conventional z/OS data sets:

- Support for hierarchical directories
- File names up to 255 characters in length
- Path names up to 1023 characters in length
- Mixed case names and special characters, except nulls and slash characters, in file and path names
- UNIX-style access permission support
- Group and user ID support at a file level
- Ability to link conventional MVS data sets to a POSIX path name

NFS protocol compliance

NFS provides full NFS protocol compliance for accessing the HFS file system.

8.6.2 Using conventional z/OS data sets

Using the NFS, you can access conventional z/OS data sets from a client workstation, personal computer, or any client system using software for the NFS protocol.

In z/OS, a file is called a data set. NFS allows client users to mount conventional z/OS data sets from their workstations. It presents the information to them in the form of a UNIX (or AIX) or DOS file, though the information is actually stored on a z/OS-owned DASD.

The files for an operating system are organized into a file system. The UNIX and DOS environments use a file system that is a hierarchy of directories. Conventional z/OS, in contrast to z/OS UNIX, uses a non-hierarchical file system in which groups of data sets are referred to by specifying a high-level qualifier (HLQ).

The z/OS HLQ can include the first (leftmost) qualifier of data sets, or the first and second qualifiers, or the first, second, and third qualifiers, and so on. For example, NEILOC is the HLQ for the files named NEILOC.TEST.DATA and NEILOC.PROJ7.SCHED, while NEILOC.TEST is the HLQ of NEILOC.TEST.DATA and NEILOC.TEST.DOCS.

Mounting z/OS data sets onto a client mount point

To access a z/OS file system from your client, client users use the mount command to create a temporary link (until unmounted) between specific z/OS data sets and your UNIX directory (preferably empty) or an unused logical drive on their workstations. The empty UNIX directory or logical drive is called a mount point.

Client users use a z/OS HLQ in the mount command to specify which z/OS data sets to mount at a mount point. The z/OS data sets beginning with the specified HLQ appears as files under the mount point.

Client users can also perform a mount using a fully qualified data set name or an alias to a user catalog, but not the catalog name itself. Only Integrated Catalog Facility (ICF) cataloged data sets are supported by the z/OS NFS server. Tape data sets and generation data sets are not supported.

Some client platforms support TCP in addition to UDP. Users can choose either TCP or UDP to access the server. The default protocol option depends on the NFS client platform.

Note: When directly mounting on a fully qualified data set name, the server must return the mount size as part of getting the attributes for the mount. This can slow down the completion of the mount command.

Crossing file systems - NFS server

Crossing file systems means the NFS client can also potentially be a server, and remote and local mounted file systems can be freely mixed. This leads to some problems when a client travels down the directory tree of a remote file system and reaches the mount point on the server for another remote file system. Allowing the server to follow the second remote mount would require loop detection, server lookup, and user revalidation. When a client does a lookup on a directory on which the server has mounted a file system, the client sees the underlying directory instead of the mounted directory.

The NFS server does not support crossing file systems. For example, if a server has a file system called /usr and mounts another file system on /usr/src, a client can also mount /usr, but the server will not see the mounted version of /usr/src. A client could perform remote mounts that match the server's mount points to maintain the server's view. In this example, the client would also have to mount /usr/src in addition to /usr, even if the mounts are from the same server.

Creating conventional z/OS data sets

Client users can create z/OS data sets from a client system using the NFS. The default data set creation attributes specified by the system administrator are used to create z/OS data sets, unless the user overrides them. These attributes determine how the z/OS data sets are structured and where they are stored. Client users can override the default data set creation and processing attributes for a mount point when issuing the mount command. In addition, you can override these attributes at file creation time.

Data set serialization and sharing

The z/OS NFS server handles data set serialization and sharing differently, depending on the type of data set; see Table 8-1.

Table 8-1 Data set serialization and sharing

PS	The server insures the physical sequential data set read/write integrity by SVC 99 dynamic allocation with exclusive option whenever a physical sequential data set is opened for output. Otherwise, it allocates with share option.
VSAM	The server dynamically allocates a VSAM data set with share option and allows the VSAM access method to manage data sharing using the share options specified during data set definition.
PDS/E	The server dynamically allocates a PDS/E data set with share option and allows the PDS/E functions to manage the serialization of the PDS/E data set and its members.
PDS	For read and write, the z/OS NFS server issues ENQ SHR on QNAME=SYSDSN and RNAME=dataset_name (through an SVC 99). For write, the server issues an exclusive ENQ against QNAME=SPFEDIT and RNAME=dataset_name.member_name, in addition to the serialization of resources by SVC 99. For all z/OS users who are allocating their data set with exclusive status, this provides write protection. It only provides read integrity for ISPF users.

8.6.3 Supported servers for the z/OS NFS client

The z/OS NFS client supports all servers that implement the server portion of the Sun NFS Version 2 or Version 3 protocols.

A mount parameter vers(x), where x is either 2 or 3, is provided to make the z/OS NFS client communicate with the server at the specified protocol level. The z/OS NFS client also communicates at the highest protocol level that is supported by the server if no level is specified.

If no version is specified and if the server:

- Only supports NFS version 2 protocol, then the z/OS NFS client will use NFS version 2 protocol to communicate
- Supports both the NFS version 2 and 3 protocols, then z/OS NFS client will use NFS version 3 protocol to communicate

If vers(2) is specified, then use NFS version 2 protocol to communicate with the server.

If vers(3) is specified, then use NFS version 3 protocol to communicate with the server. The z/OS NFS client fails the **mount** command if the server does not support NFS version 3 protocol.

8.6.4 WebNFS support

The z/OS NFS server supports the WebNFS protocol. WebNFS specification extends the semantics of NFS versions 2 and 3 protocols to allow clients to obtain file handles without the mount protocols. The z/OS NFS server supports the public file handle and multi-component lookup features as well as other additional requirements as described in RFC 2055.

A new keyword, *public*, is added for the system administrator to specify the public paths that the public file handle can access. A public path for conventional z/OS data and a public path for HFS data can both be specified. When a lookup request comes in from an NFS client and an absolute path name is specified, it is matched with the public paths to determine which public path it is trying to reference. If a relative path is specified and both HFS and z/OS public paths are defined, then the lookup request is processed relative to the HFS public path.

The following are restrictions for the WebNFS support provided by the z/OS NFS server in this release:

- Export Spanning Pathnames - Lookup requests, which reference files or directories outside of the exported public path, will result in an error condition.
- Symbolic Links - A symbolic link embedded in a multi-component pathname lookup request will result in an error condition. However, if the final component is a symbolic link, the server will return the file handle of the symbolic link and let the client evaluate it. External links, which are special cases of symbolic links, will be handled similarly.
- Native Path - Only canonical pathnames will be supported.

8.6.5 Native ASCII support

The z/OS NFS client and server support applications running on z/OS V1R2 (and higher) in a Native ASCII environment. Applications can operate on files in either EBCDIC or ASCII format as well as other data formats defined with a Coded Character Set Identifier (CCSID).

Native ASCII support is provided with a mechanism called file tagging where the file is defined with a tag to identify the CCSID to use for data conversion. File tagging is defined in the appropriate z/OS UNIX System Services (USS) publications. The z/OS NFS client and server provide the necessary support to provide data conversion between different CCSIDs specified for the client and server. The z/OS NFS client cln_ccsid and srv_ccsid parameters will also be supported by the z/OS NFS server to identify the CCSID to be used in the data conversion.

Additional information

See *Network File System Customization and Operation,* SC26-7417.

8.7 Text Search

This topic introduces you to the Text Search Engine environment and to some of the concepts that are applicable to the Text Search Engine.

8.7.1 The Text Search Engine environment

The Text Search Engine consists of a server component, a client component, and resources, for example, dictionaries and thesaurus files. You can install these components in the following combinations on any machine:

- The client component and resources
- The server component and resources
- Both the client and server components and resources

Figure 8-16 on page 342 shows how you might set up your Text Search Engine environment.

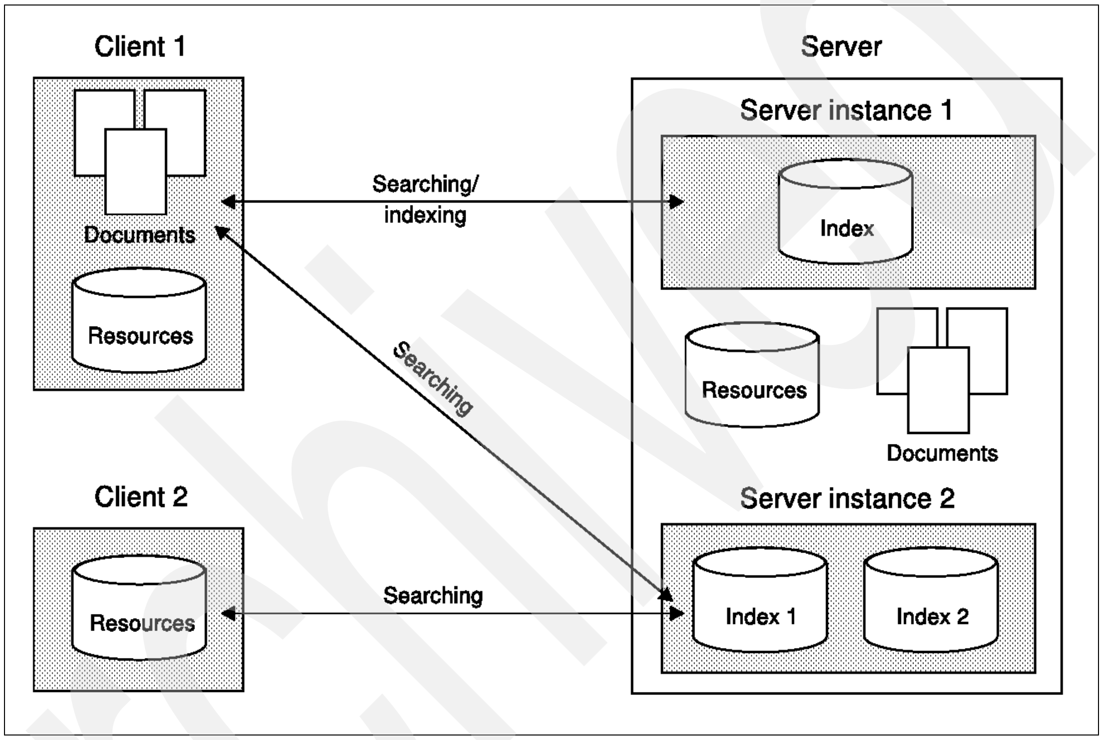


Figure 8-16 Text Search environment

Client The client manages access to the Text Search Engine server. For example, it is the interface for building and maintaining indexes, and it provides access to search and result-list handling. It contains a dynamically loadable module for programming text applications and a command-line interface for issuing administration commands that access the API functions. Configuration files on the client define where resources are located, and the communication method used for connecting to the Text Search Engine server. A client can connect to different servers located on different machines.

Server A Text Search Engine server is known as a *search server instance*. Several server instances can be configured on the same machine. A set of indexes is owned by a server instance. This means that server instances are independent of one another and can run in parallel. Each server instance:

- Maintains one set of indexes.
- Allows for up to 100 parallel processes for client sessions.
- Uses one set of configuration files defining resources and the default behavior of the server instance.

A server is a daemon process that must be running to be connected to by a client.

Resources Resources include dictionaries, thesaurus files, stop-word lists, and abbreviation files. Dictionaries support the linguistic processing of documents during indexing and retrieval. The U.S. English dictionary is always installed on every machine. When you install a dictionary, the stop words for that language are also installed.

Thesaurus files can be used during a search for expanding query terms. Two sample thesaurus files are available; one for ngram indexes (imonthes.*) and one for all the other index types (imothes.*).

8.7.2 Client/server communication

The Text Search Engine supports the following communication methods for a client/server environment:

TCP/IP TCP/IP allows clients on the same or different machines to access the Text Search Engine server. The server starts a daemon process that controls access to the index from each of the clients. It also starts communication processes to accept client requests.

If you have a lot of clients running in parallel and receive an error message that the server is busy, you should increase the number of client processes running in parallel. To do this, use the **`imcfgsv`** command to increase the number of running tasks.

Local Local communication allows clients to be connected to a server only if they are on the same machine. The server starts a daemon process that controls access to the index only. A client can connect to only one server at any one time (in one process).

Note: Local communication is not thread reentrant. Before you can use local communication, make sure that the applications using this mode serialize the client requests within the application.

8.7.3 Text Search Engine concepts

The following are general concepts that apply throughout the Text Search Engine components. These concepts are especially important in a heterogeneous environment with, for example, clients on workstation platforms and the server on z/OS.

Configuration files There are several files available for configuring the Text Search Engine. These are in a flat-file format and can be changed using any editor available on your system.

Character data Whenever character data is passed in interfaces, especially across systems or environments, character encoding problems might arise. To overcome this problem, the Text Search Engine has rules for input data.

There are rules for:

- Text Search Engine names (such as index names or server names)
- External names (such as document names)

- Document identifiers (and document group identifiers)
- Document text
- Search terms

Additional information

See *Text Search - Programming the Text Search Engine,* SH12-6717.

8.8 Tivoli Storage Manager

Tivoli Storage Manager (TSM) is an enterprise-wide storage management application for the network. It provides automated storage management services to multivendor workstations, personal computers, and local area network (LAN) file servers. TSM includes the following components:

Server Allows a server system to provide backup, archive, and space management services to workstations. The server maintains a database and recovery log for TSM resources, users, and user data.

The server controls storage called storage pools. These are groups of random and sequential access media that store backed-up, archived, and space-managed files.

You can set up multiple servers in your enterprise network to balance storage, processor, and network resources. TSM allows you to manage and control multiple servers from a single interface that runs in a web browser (the enterprise console).

Administrative interface

Allows administrators to control and monitor server activities, define management policies for client files, and set up schedules to provide services at regular intervals. Administrative functions are available from an administrative client command line and from a Web browser interface. A server console is also available.

Backup-archive client

Allows users to maintain backup versions of their files, which they can restore if the original files are lost or damaged. Users can also archive files for long-term storage and retrieve the archived files when necessary. Users themselves or administrators can register workstations and file servers as client nodes with a TSM server.

Application program interface (API)

Allows users to enhance existing applications with back up, archive, restore, and retrieve services. When users install the TSM API client on their workstations, they can register as client nodes with a TSM server.

TSM also supports the following client programs:

Tivoli Data

Protection for applications (application clients) allows users to perform online backups of data that is used by applications such as database programs. After the database initiates a backup or restore, the application client acts as the interface to TSM. The TSM server then applies its storage management functions to the data. The application client can perform its functions while users are working, with minimal disruption.

Tivoli Space

Manager provides space management services for workstations on some platforms. Tivoli Space Manager users can free workstation storage by migrating less frequently used files to server storage. These migrated files are also called space-managed files. Users can recall space-managed files automatically simply by accessing them as they would normally. Tivoli Space Manager is also known as the hierarchical storage management (HSM) client.

Figure 8-17 shows an example of a client/server environment with TSM. In this example, an administrator uses an administrative interface to monitor the system, for example, the administrative client program that is installed on a workstation. An administrator can also monitor a server by using a Web browser with the appropriate Java support.

The backup-archive client program and HSM client program have been installed on workstations connected through a LAN and registered as client nodes. From these client nodes, users can back up, archive, or migrate files to the server.

Using rules in TSM policies that are assigned to files, the server stores client files on disk or tape volumes in server storage. Server storage is divided into storage pools that are groups of storage volumes.

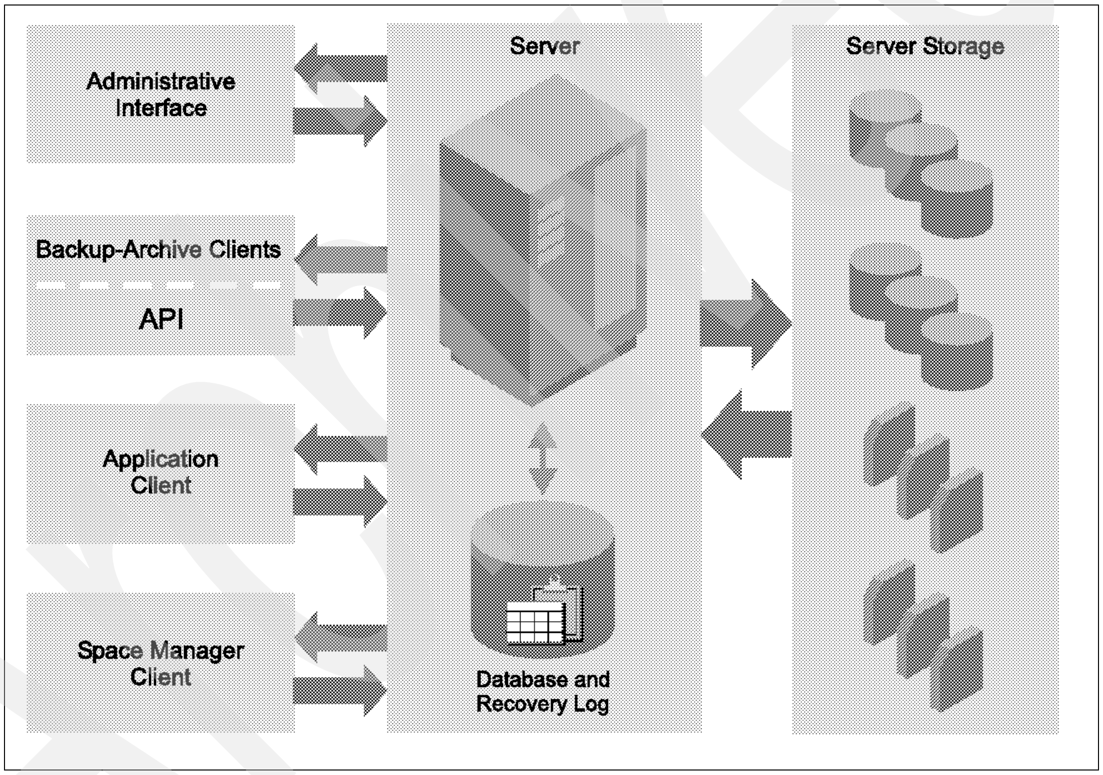


Figure 8-17 Sample client/server environment

Additional information

See *Tivoli Storage Manager for MVS and OS/390: Administrator's Guide,* GC35-0377.

9

Interacting with z/OS UNIX

In this chapter, the following topics are discussed:

- z/OS UNIX operator commands
- Using interactive interfaces, the z/OS UNIX shell, and the ISHELL
- Direct logon to the z/OS UNIX shell
- The BPXBATCH and BPXBATSL utilities
- Running a shell script

9.1 Commands to monitor z/OS UNIX

There are many different reasons to monitor z/OS UNIX, for example:

- To kill a process
- To see how much space a directory needs
- To activate a new configuration for z/OS UNIX
- To find a file located in the HFS
- To change your local directory

Therefore, you need to know which commands show the information that you want, and which commands provide useful information without incurring too much overhead.

Commands can be divided into two types: commands you issue from a console, and commands you issue from the z/OS UNIX shell.

z/OS console commands

The following examples are z/OS (MVS) commands that are issued from MCS or Extended MCS consoles:

`D OMVS,A=ALL`	Displays the information about all running processes
`D OMVS,O`	Displays the information defined in the BPXPRMxx
`D OMVS,F`	Displays the information about the mounted HFS
`D OMVS,PID=<pid>`	Displays the information about the process ID
`D A,OMVS`	Displays the information about kernel and data spaces
`F BPXOINIT,FORCE,PID=<PID>`	Forces a process ID to end
`SET OMVS=xx`	Sets a new configuration of BPXPRMxx member

z/OS UNIX shell commands

The following commands can be issued from the OMVS shell:

`pwd`	Displays the current pathname
`ls -alWE`	Displays the contents and extended attributes
`ps -ef`	Displays the information about all running processes
`man ls`	Displays information about syntax and use of the command
`df -P`	Displays the information about the mounted HFS
`find / -name setup.sh`	Searches the HFS from the root to find the file specified

9.1.1 Interprocess communication signals

Signals are used for communication between processes to inform about events and trigger actions. Generally, signals are initiated by the kernel, but the kill system call can be used to terminate a process.

- **Wait and Exit** - A parent process can wait on the exit of a child. The wait() function is used for waiting for any child process, while the waitpid() function is used for waiting for a particular child process.

  Both exit() and _exit() terminate a process and generate status information which is available for the parent process waiting with wait() or waitpid(). When using exit(), these cleanup routines are not invoked. Generally, exit() is used for a graceful exit from a program, while _exit() is used for abnormal terminations.

- **Signal() and sigaction()** are equivalent functions that are used to catch a signal and determine what to do with it.

The function for sending signals is called kill(). A process can send a signal to another process or group of processes if it has permission to do so. A process can also send a signal to itself.

Signals are used for system event notification, or they can be used for process synchronization. For example, a process might want to wait for a signal to know that another process has opened a pipe, written a file, or completed a task that the current process needs to wait for.

A process can choose what to do when it receives a signal:

- Execute a signal handling function.
- Ignore the signal.
- Restore the default action of a signal.

Kill() accepts several different signal codes. Examples are:

- SIGABND - Abend
- SIGCHLD - Child termination
- SIGKILL - Cancel a process
- SIGSTOP - Stop a process

From a program's point of view, signals are asynchronous. That means a program can, in principle, receive a signal between any two instructions.

9.1.2 Kill a process

The best way to end a process is to issue the **kill** command. Use the **D OMVS** operator command or the **ps** command to display all the active processes. Then issue the **kill** command, specifying the signal and the PID (process identifier) for the process.

- Start by sending a SIGTERM signal:

  ```
  kill -s TERM pid
  ```

  where `pid` is the process identifier.

- If that does not work, try sending a SIGKILL signal:

  ```
  kill -s KILL pid
  ```

  where `pid` is the process identifier.

If some of the processes still exist after both of these signals are sent, they are terminated with a 422-1A3 ABEND, as shown in Figure 9-1.

```
IEF450I STEVEZ IKJACCT IKJACCNT - ABEND=S422 U0000 REASON=000001A3 952
        TIME=07.58.28
```

Figure 9-1 IEF450I message

If after all of these steps, some non-permanent processes still exist, the shutdown request is aborted and a `BPXI061E` message is issued:

```
BPXI061E OMVS SHUTDOWN REQUEST ABORTED
```

Superkill

It can happen that z/OS UNIX processes that become hung cannot be terminated via the kill() service. That situation required intervention by the MVS operator to CANCEL the address space containing the z/OS UNIX process.

In z/OS V1R6 the new *superkill* OMVS function enables you to do the following:

- Cancel hung USS processes using UNIX semantics
- Cancel your own hung process from the shell
- Use the enhanced console support to give operators and automated console applications additional flexibility

The superkill command bypasses the Language Environment, in contrast to the “normal” kill command. Superkill employs a 422 non-retryable abend, directed to the initial thread of the target process. Only one abend per process is allowed. Restrictions have been put in place to ensure that the asynchronous nature of the abend is limited to processes that are truly hung. Limiting the abend to a single process at a time also avoids abusive use.

There are four ways to invoke the superkill:

BPX1KIL/ BPX4KIL	These are the USS-callable assembler services. A superkill can be sent by setting the PPSDSUPERKILL bit.
__superkill()	This is the C/C++ service.
kill -K [pid...][job-identifier ...]	The shell command.
F BPXONIT,SUPERKILL=pid	The operator console command.

Restrictions on the use of superkill are:

- It cannot target Group IDs or -1
- Must be authorized to send the target process a signal
- Must be preceded by a normal sigkill signal

9.2 z/OS UNIX interactive interfaces

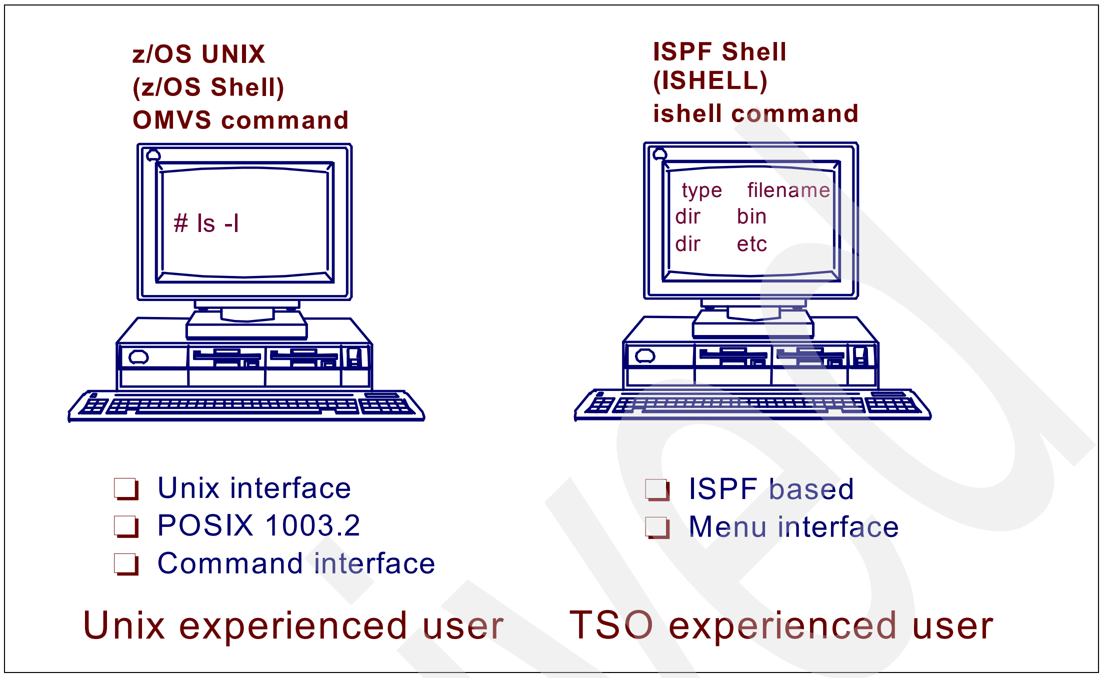


Figure 9-2 Overview of z/OS UNIX shell and ISHELL

Figure 9-2 shows an overview of the two interactive interfaces, z/OS UNIX shell and the ISHELL. In addition, there are some TSO/E commands to support z/OS UNIX, but they are limited to certain functions such as copying files and creating directories.

The z/OS UNIX shell provides the environment that has the most functions and capabilities. Shell commands can easily be combined in pipes or shell scripts and thereby become powerful new functions. A sequence of shell commands can be stored in a text file which can be executed. This is called a shell script. The shell supports many of the features of a regular programming language.

There are some TSO commands that provide support for UNIX System Services. An important command is the OMVS command, which invokes the z/OS UNIX shell. The ISHELL command invokes the ISPF shell. The ISHELL is a good starting point for users familiar with TSO and ISPF that want or need to use z/OS UNIX.

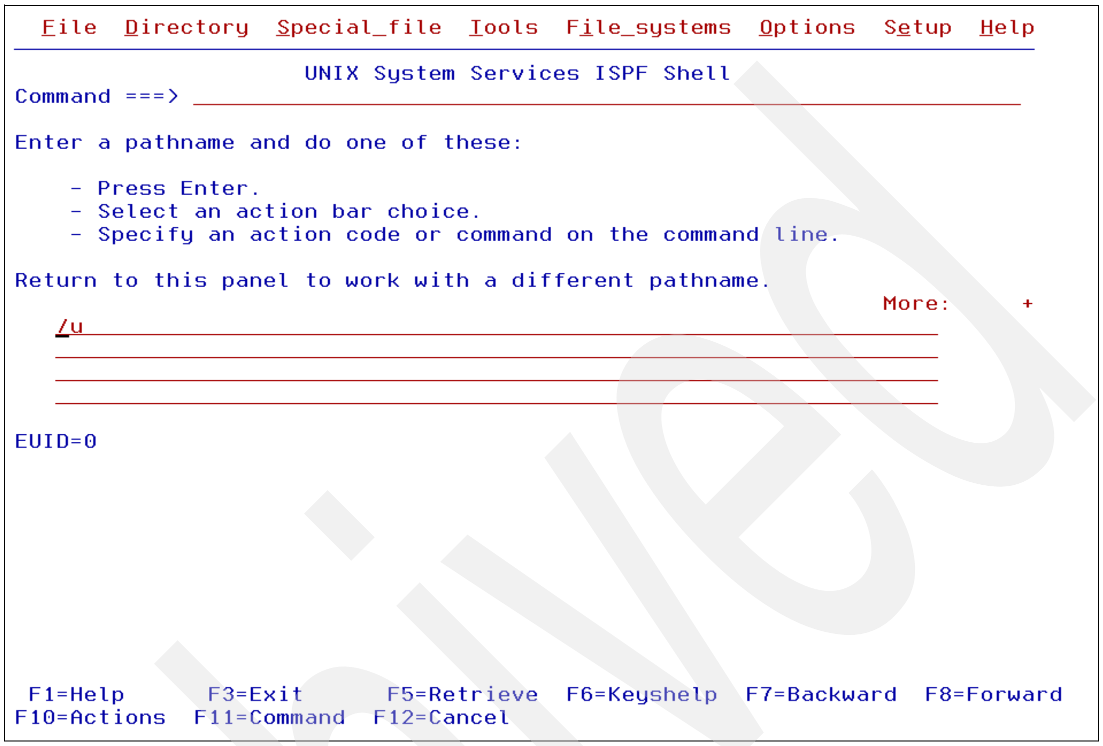
Once the ISHELL command is executed, it provides so-called Common User Access® (CUA®) panels, where users can work with the hierarchical file system. There are also panels for mounting and unmounting file systems and for doing some z/OS UNIX administration. In addition the ISHELL is an ISPF dialog for users and system administrators which can be used instead of shell commands to perform many tasks related to file systems, files, and z/OS UNIX user administration.

The two types of interacting with the z/OS UNIX shell can be a good starting point for:

- Those whose primary interactive computing environment is a UNIX or AIX workstation and who find the z/OS shell programming environment familiar.
- Those whose primary interactive computing environment is TSO/E and ISPF and who can do much of their work in that environment.

9.3 Using the ISHELL

From ISPF option **6**, type `ISHELL` to get to the ISHELL main panel. See Figure 9-3.



```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
 ────────────────────────────────────────────────────────────────────────────
                     UNIX System Services ISPF Shell
Command ===> ____________________________________________________________

Enter a pathname and do one of these:

    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                             More:     +
   /u_________________________________________________________________
   ___________________________________________________________________
   ___________________________________________________________________
   ___________________________________________________________________

EUID=0








 F1=Help      F3=Exit      F5=Retrieve  F6=Keyshelp  F7=Backward  F8=Forward
F10=Actions  F11=Command  F12=Cancel
```

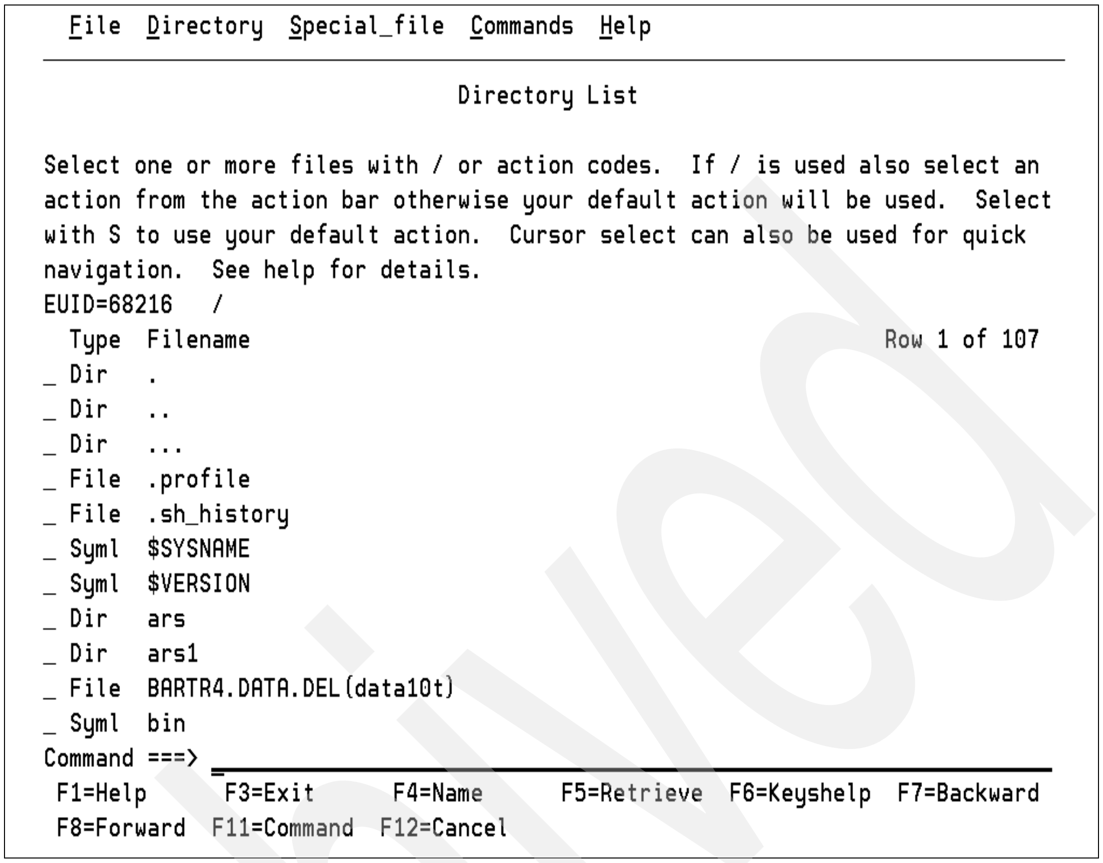
Figure 9-3 ISHELL main panel

Display the root directory

When you press Enter, a list of the files that exist in the root directory are displayed. Figure 9-4 on page 353 provides a picture of what you can expect to see when you enter the directory structure through the ISHELL.

From here you can list, browse, edit, copy, and delete files and directories in the HFS file system. In the next topic we show some of the newest features in the ISHELL.

For administrative use of ISHELL, check *z/OS UNIX System Services User's Guide,* SA22-7801 for more detailed information.



```
  File  Directory  Special_file  Commands  Help
────────────────────────────────────────────────────────────────────────────────
                                Directory List

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=68216   /
  Type  Filename                                                  Row 1 of 107
_ Dir   .
_ Dir   ..
_ Dir   ...
_ File  .profile
_ File  .sh_history
_ Syml  $SYSNAME
_ Syml  $VERSION
_ Dir   ars
_ Dir   ars1
_ File  BARTR4.DATA.DEL(data10t)
_ Syml  bin
Command ===> ________________________________________________________________
 F1=Help      F3=Exit      F4=Name      F5=Retrieve  F6=Keyshelp  F7=Backward
 F8=Forward  F11=Command  F12=Cancel
```

Figure 9-4 ISHELL Directory List

9.3.1 ISHELL enhancements

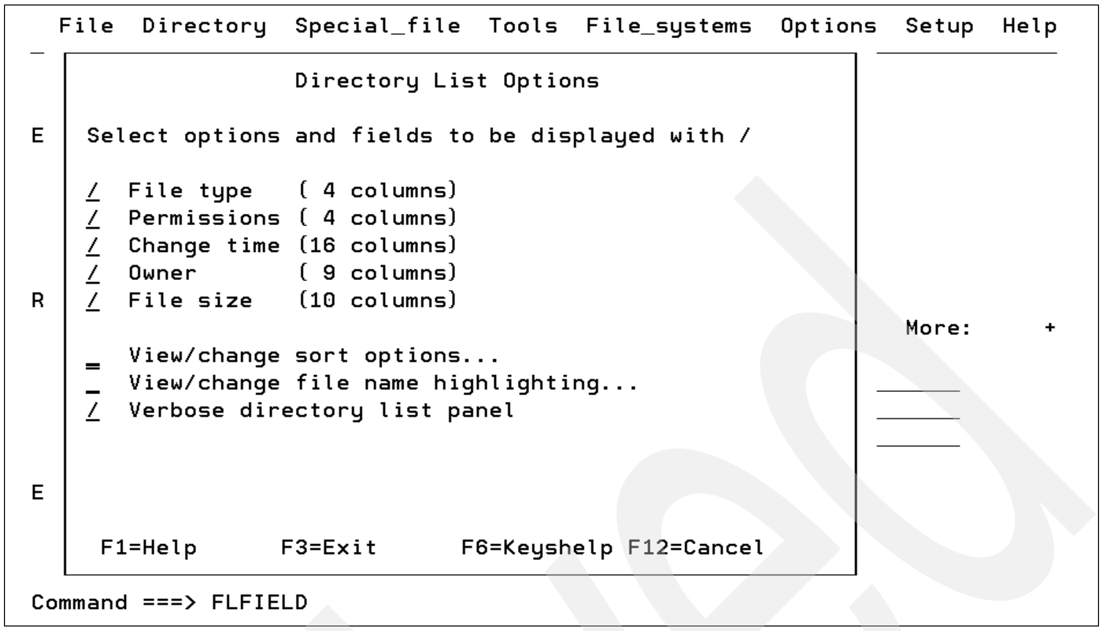
New features have been added to improve the ISHELL functionality on z/OS V1R3. The ISHELL is a user interface that allows users to work with menus rather than with sometimes cryptic commands. In z/OS V1R3, many new features that have been requested over the years to improve ISHELL functionality and panel navigation are implemented. These new features are a significant upgrade to the ISHELL and many of them are part of the Directory List options, which list files in a particular directory. Other enhancements include sorting and highlighting support, as well as easy ways to access information.

Many of the new features are part of the Directory List, which lists files in a particular directory. The ISHELL enhancements include:

- Sorting
- Support for colorful highlighting
- Execute actions based on cursor location
- Easy ways to access information

Directory List enhancements

Before going to the Directory List panel, you can choose which fields on the Directory List you want to have displayed. From the ISHELL panel choose **Options** → **Option1 - Directory list...**, then choose which fields you want to display by typing a slash (/) next to the option, as shown in the Directory Options List in Figure 9-5.



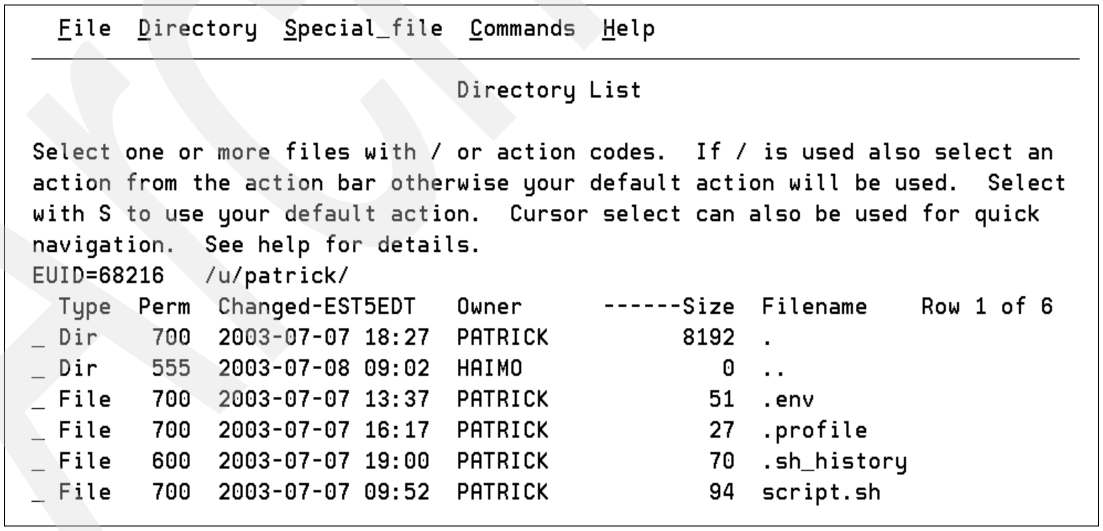
```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
_
                  Directory List Options

E   Select options and fields to be displayed with /

    /  File type   ( 4 columns)
    /  Permissions ( 4 columns)
    /  Change time (16 columns)
    /  Owner       ( 9 columns)
R   /  File size   (10 columns)
                                                            More:     +
    =  View/change sort options...
    _  View/change file name highlighting...
    /  Verbose directory list panel


E

     F1=Help      F3=Exit      F6=Keyshelp F12=Cancel

Command ===> FLFIELD
```

Figure 9-5   Directory List options

Note: The bottom three Directory List options in Figure 9-5 are new options in z/OS V1R3 and higher.

Directory List panel enhancements

The Directory List panel is available through the ISHELL panel, **Directory → Option 1, List Directory,** or by specifying a file pathname (see Figure 9-6).



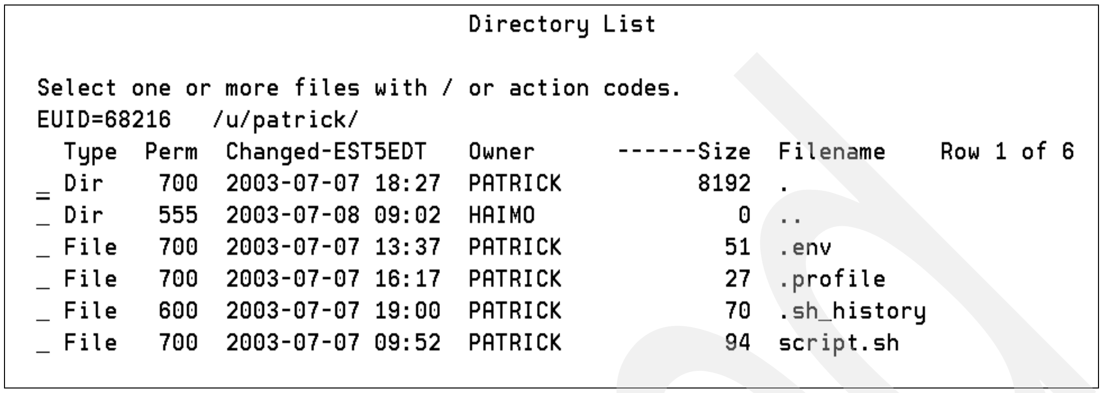
```
  File  Directory  Special_file  Commands  Help
──────────────────────────────────────────────────────────────────────────────
                              Directory List

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=68216   /u/patrick/
  Type  Perm  Changed-EST5EDT   Owner      ------Size  Filename    Row 1 of 6
_ Dir    700  2003-07-07 18:27  PATRICK          8192  .
_ Dir    555  2003-07-08 09:02  HAIMO               0  ..
_ File   700  2003-07-07 13:37  PATRICK            51  .env
_ File   700  2003-07-07 16:17  PATRICK            27  .profile
_ File   600  2003-07-07 19:00  PATRICK            70  .sh_history
_ File   700  2003-07-07 09:52  PATRICK            94  script.sh
```

Figure 9-6   Directory List panel

The following changes are made to the Directory List panel shown in Figure 9-6:

- The effective user ID (EUID) is displayed on the panel, so you can know the authority you have at any particular moment. In the example shown, EUID=68216 indicates that the current UID is accessing the ISHELL.

- Times are displayed in local time instead of Greenwich time, as in previous releases.
- The Action Bar can be removed by typing `noab` in the command line, as shown in Figure 9-7. If you want to see the action bar again, type `ab` in the command line.



```
                                  Directory List

  Select one or more files with / or action codes.
  EUID=68216   /u/patrick/
    Type  Perm  Changed-EST5EDT   Owner      ------Size  Filename    Row 1 of 6
  _ Dir    700  2003-07-07 18:27  PATRICK          8192  .
  _ Dir    555  2003-07-08 09:02  HAIMO               0  ..
  _ File   700  2003-07-07 13:37  PATRICK            51  .env
  _ File   700  2003-07-07 16:17  PATRICK            27  .profile
  _ File   600  2003-07-07 19:00  PATRICK            70  .sh_history
  _ File   700  2003-07-07 09:52  PATRICK            94  script.sh
```

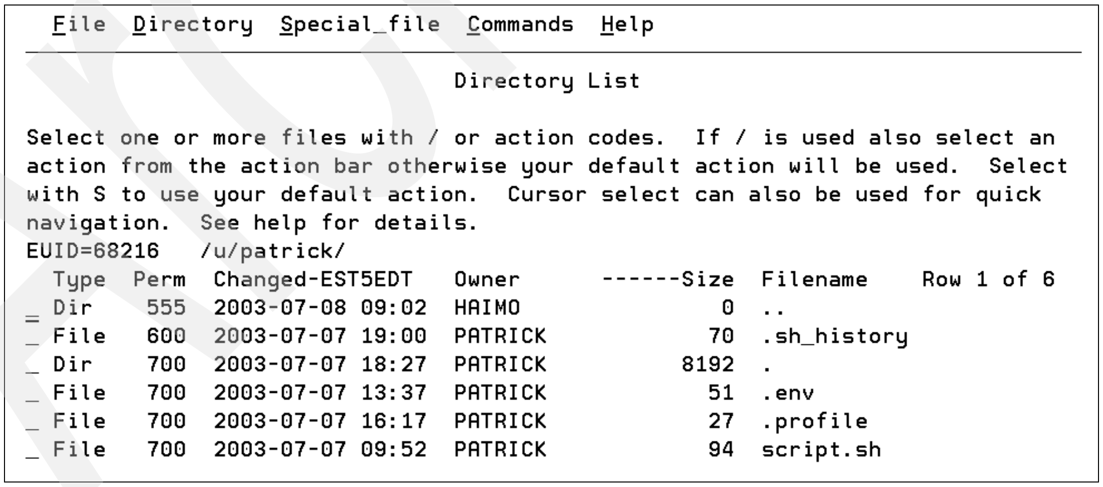
Figure 9-7 Changing the Directory List panel to a new format

9.3.2 Using the cursor on the Directory List panel

Most areas of the Directory List panel are cursor-sensitive. In a “sensitive” area, when you place the cursor under a value and press Enter, a new window appears that allows you to change the value of the selected field. The following actions can be done through the sensitive areas in the Directory List panel.

Sorting by column header

By placing the cursor under any of the column headers on the Directory List panel shown in Figure 9-6 on page 354 and pressing Enter, the directory list will be sorted on that column. For example, placing the cursor on the Perm column header and pressing Enter will sort the permission bit settings, as shown in Figure 9-8.



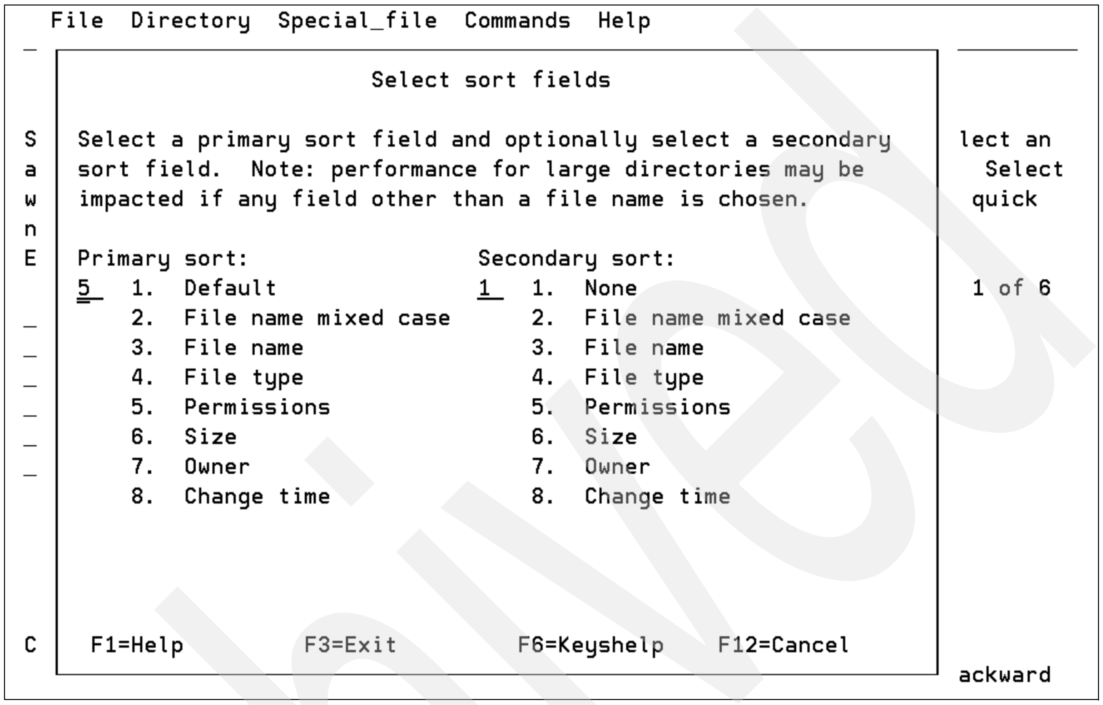
```
  File  Directory  Special_file  Commands  Help
 ------------------------------------------------------------------------------
                                  Directory List

 Select one or more files with / or action codes.  If / is used also select an
 action from the action bar otherwise your default action will be used.  Select
 with S to use your default action.  Cursor select can also be used for quick
 navigation.  See help for details.
 EUID=68216   /u/patrick/
   Type  Perm  Changed-EST5EDT   Owner      ------Size  Filename    Row 1 of 6
 _ Dir    555  2003-07-08 09:02  HAIMO               0  ..
 _ File   600  2003-07-07 19:00  PATRICK            70  .sh_history
 _ Dir    700  2003-07-07 18:27  PATRICK          8192  .
 _ File   700  2003-07-07 13:37  PATRICK            51  .env
 _ File   700  2003-07-07 16:17  PATRICK            27  .profile
 _ File   700  2003-07-07 09:52  PATRICK            94  script.sh
```

Figure 9-8 Sorting by column header list

Sort fields in Directory List panel

Entries in the directory list panel can be sorted by any field (even if the field is not displayed) with a primary and secondary sort field. To set the sort options, you can access the Sorting Options panel shown in Figure 9-9 on page 356 by using one of the following options:

1. Type `sort` on the command line from the Directory List panel.
2. From the Directory List panel, select **Commands** → **Option 2 - Sort**.
3. From the ISHELL panel, select **Options** → **Option1 - Directory List**. Then type / on the column **View/change sort options...**.



```
  File  Directory  Special_file  Commands  Help
_                                                               __________
                        Select sort fields

S   Select a primary sort field and optionally select a secondary   lect an
a   sort field.  Note: performance for large directories may be      Select
w   impacted if any field other than a file name is chosen.          quick
n
E   Primary sort:                  Secondary sort:
    5   1.  Default                1   1.  None                      1 of 6
_       2.  File name mixed case       2.  File name mixed case
_       3.  File name                  3.  File name
_       4.  File type                  4.  File type
_       5.  Permissions                5.  Permissions
_       6.  Size                       6.  Size
_       7.  Owner                      7.  Owner
        8.  Change time                8.  Change time




C    F1=Help        F3=Exit        F6=Keyshelp    F12=Cancel
                                                                  ackward
```

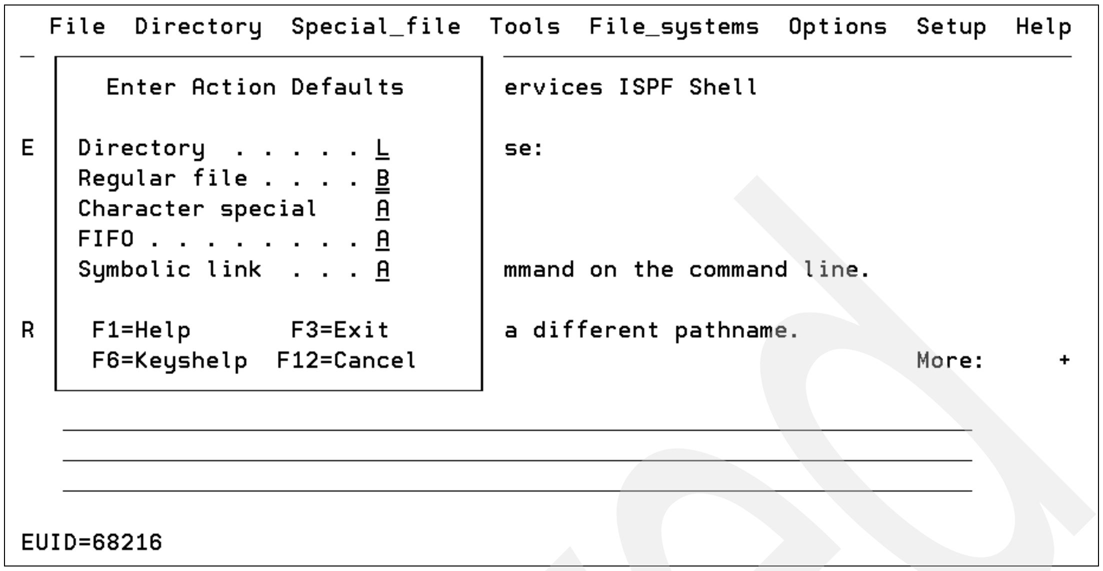
Figure 9-9 Sort panel sort fields

Note: In Figure 9-9, selecting option 5, Permissions, does exactly the same sort as shown in "Sorting by column header" on page 355.

Accessing files with cursor on the Type header

By placing the cursor on the Type field for any file and pressing Enter, the default action is taken on that file from the ISHELL panel; select **Options** → **Option 2 - Default Actions**, as shown in Figure 9-10 on page 357.

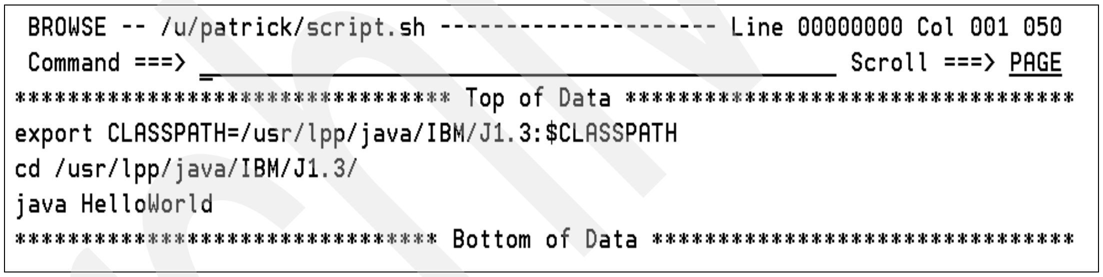
Notice that for a regular file, the default action is to browse the file.



```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
_
    Enter Action Defaults      ervices ISPF Shell

E  Directory  . . . . . L      se:
   Regular file . . . . B
   Character special    A
   FIFO . . . . . . . . A
   Symbolic link  . . . A      mmand on the command line.

R   F1=Help      F3=Exit       a different pathname.
    F6=Keyshelp  F12=Cancel                               More:     +

EUID=68216
```

Figure 9-10   Default actions window settings

Therefore, when you place the cursor under the Type field (File) for file script.sh in Figure 9-6 on page 354, the next panel displayed is an ISPF browse mode panel for file script.sh, as shown in Figure 9-11.



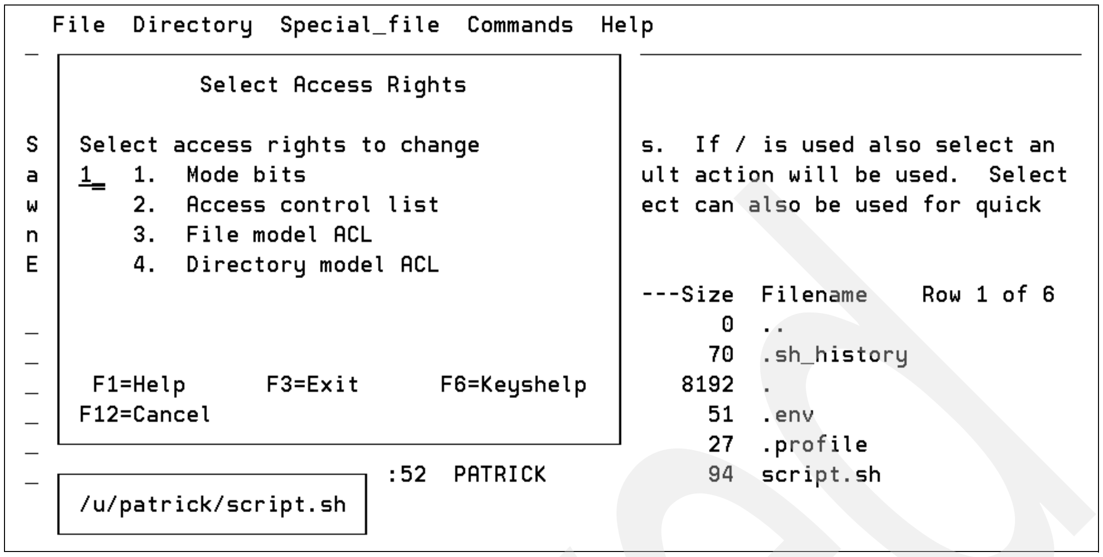
```
 BROWSE -- /u/patrick/script.sh -------------------- Line 00000000 Col 001 050
 Command ===> _                                             Scroll ===> PAGE
********************************* Top of Data **********************************
export CLASSPATH=/usr/lpp/java/IBM/J1.3:$CLASSPATH
cd /usr/lpp/java/IBM/J1.3/
java HelloWorld
******************************** Bottom of Data ********************************
```

Figure 9-11   Using the cursor to enter browse mode for a file

File mode bits of FSP

By placing the cursor on the permissions field for any file shown in Figure 9-6 on page 354 and pressing Enter, a window appears that allows you to change the permissions, setuid, setgid, or sticky bit by selecting option 1, as shown in Figure 9-12 and Figure 9-13 on page 358.

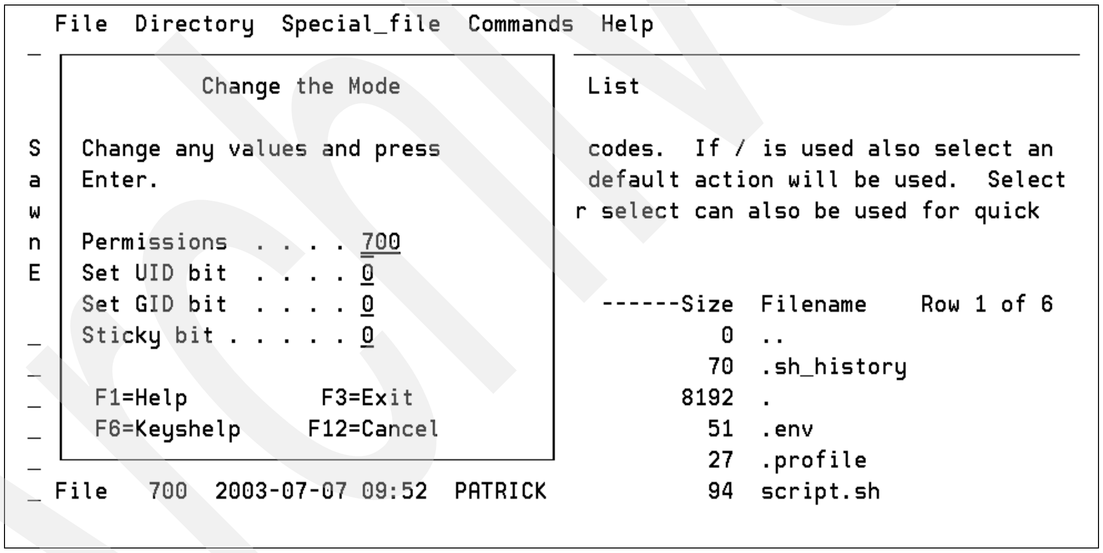
In addition, you can also choose to modify the ACL types from this window, but only if the ACL already exists. Refer to *z/OS UNIX System Services Planning*, GA22-7800, for more information about ACLs.



```
  File  Directory  Special_file  Commands  Help
_ ┌──────────────────────────────────────────┐ ─────────────────────────────────
  │          Select Access Rights            │
  │                                          │
S │ Select access rights to change           │ s.  If / is used also select an
a │ 1_  1.  Mode bits                        │ ult action will be used.  Select
w │     2.  Access control list              │ ect can also be used for quick
n │     3.  File model ACL                   │
E │     4.  Directory model ACL              │
  │                                          │ ---Size  Filename    Row 1 of 6
_ │                                          │       0  ..
_ │                                          │      70  .sh_history
_ │  F1=Help     F3=Exit      F6=Keyshelp    │    8192  .
_ │ F12=Cancel                               │      51  .env
_ └──────────────────────────────────────────┘      27  .profile
_ ┌───────────────────────┐  :52  PATRICK            94  script.sh
  │ /u/patrick/script.sh  │
  └───────────────────────┘
```

Figure 9-12 Select permission change window

When you select option 1, a panel like Figure 9-13 appears to allow changes to the FSP fields.

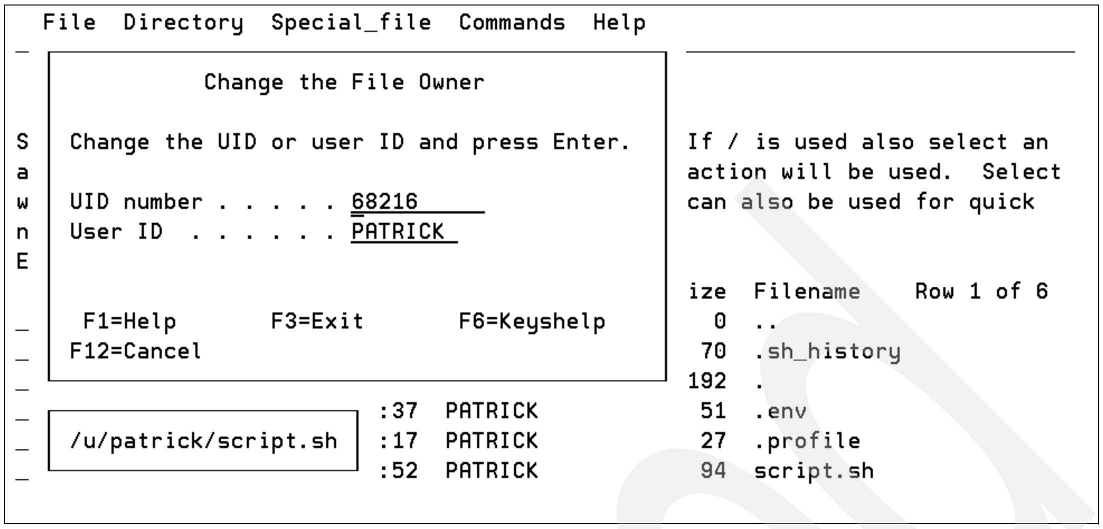


```
  File  Directory  Special_file  Commands  Help
_ ┌─────────────────────────────────────┐ ─────────────────────────────────
  │          Change the Mode            │  List
  │                                     │
S │ Change any values and press         │  codes.  If / is used also select an
a │ Enter.                              │  default action will be used.  Select
w │                                     │ r select can also be used for quick
n │ Permissions  . . . . 700            │
E │ Set UID bit  . . . . 0              │
  │ Set GID bit  . . . . 0              │  ------Size  Filename    Row 1 of 6
_ │ Sticky bit . . . . . 0              │           0  ..
_ │                                     │          70  .sh_history
_ │  F1=Help         F3=Exit            │        8192  .
_ │  F6=Keyshelp    F12=Cancel          │          51  .env
_ └─────────────────────────────────────┘          27  .profile
_ File   700  2003-07-07 09:52  PATRICK            94  script.sh
```

Figure 9-13 Change permissions, setuid, setgid, or sticky bit

Change the owner

By placing the cursor under the owner field for any file shown in Figure 9-6 on page 354 and pressing Enter, a window is displayed that permits the assignment of a new file owner for the file, as shown in Figure 9-14 on page 359.

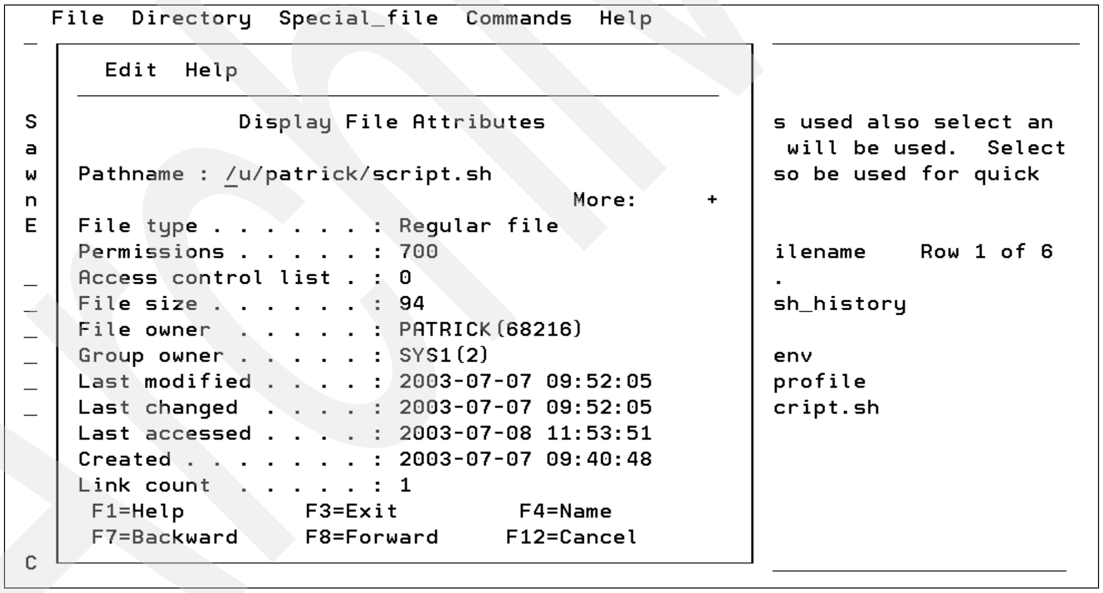


```
  File  Directory  Special_file  Commands  Help
_  ____________________________________________  ___________________________
  |            Change the File Owner            |
  |                                             |
S | Change the UID or user ID and press Enter.  |  If / is used also select an
a |                                             |  action will be used.  Select
w | UID number . . . . . 68216                  |  can also be used for quick
n | User ID  . . . . . . PATRICK                |
E |                                             |
  |                                             |  ize  Filename    Row 1 of 6
_ |  F1=Help      F3=Exit      F6=Keyshelp      |    0  ..
_ | F12=Cancel                                  |   70  .sh_history
_ |_____________________________________________|  192  .
_  ______________________  :37  PATRICK             51  .env
_ | /u/patrick/script.sh |  :17  PATRICK            27  .profile
_ |______________________|  :52  PATRICK            94  script.sh
```

Figure 9-14   Change the file owner

Display the file attributes

By placing the cursor under either the Size or Changed-EST5EDT field for a file in Figure 9-6 on page 354, the file attributes are displayed, as shown in Figure 9-15. This is equivalent to selecting the `a` action code.



```
  File  Directory  Special_file  Commands  Help
_  ______________________________________________  _______________________
  |  Edit  Help                                   |
  | ____________________________________________  |
S |            Display File Attributes            |  s used also select an
a |                                               |   will be used.  Select
w | Pathname : /u/patrick/script.sh               |  so be used for quick
n |                                 More:     +   |
E | File type . . . . . . : Regular file          |
  | Permissions . . . . . : 700                   |  ilename    Row 1 of 6
_ | Access control list . : 0                     |  .
_ | File size . . . . . . : 94                    |  sh_history
_ | File owner  . . . . . : PATRICK(68216)        |
_ | Group owner . . . . . : SYS1(2)               |  env
_ | Last modified . . . . : 2003-07-07 09:52:05   |  profile
_ | Last changed  . . . . : 2003-07-07 09:52:05   |  cript.sh
  | Last accessed . . . . : 2003-07-08 11:53:51   |
  | Created . . . . . . . : 2003-07-07 09:40:48   |
  | Link count  . . . . . : 1                     |
  |  F1=Help       F3=Exit        F4=Name         |
  |  F7=Backward   F8=Forward    F12=Cancel       |
C |_______________________________________________|  _______________________
```

Figure 9-15   Displaying the file attributes using the cursor

Displaying the complete file name

By placing the cursor under any file name in Figure 9-6 on page 354, a new window appears that shows the full path name for that file. This is useful when a file name is truncated on the Directory list panel. The complete pathname is displayed, as shown in Figure 9-16 on page 360.

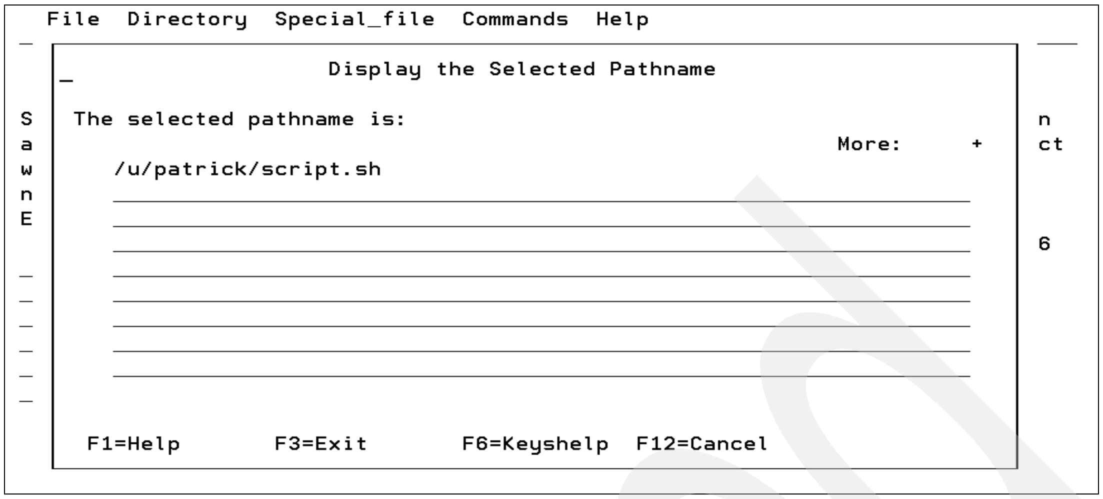

```
  File  Directory  Special_file  Commands  Help
_                                                                         ____
      _                Display the Selected Pathname
S                                                                          n
a    The selected pathname is:                                             ct
w                                                      More:     +
n       /u/patrick/script.sh
E
                                                                           6
_
_
_
_
_
_

       F1=Help        F3=Exit        F6=Keyshelp   F12=Cancel
```

Figure 9-16 Displaying the complete file name

9.3.3 Displaying colors on the Directory List panel

Entries in the Directory List can be highlighted with different colors based on a number of different methods. To set the highlighting options, use one of the following:

- From the Directory List panel, type `colors`.
- On the Directory List Panel, select **Commands** → **Option 3 - Colors**.
- On the ISHELL panel, select **Options** → **Option1 - Directory List**; then type / on the column View/change file name highlighting....

By choosing one of the options, the Highlighting Options Panel is displayed (Figure 9-17).

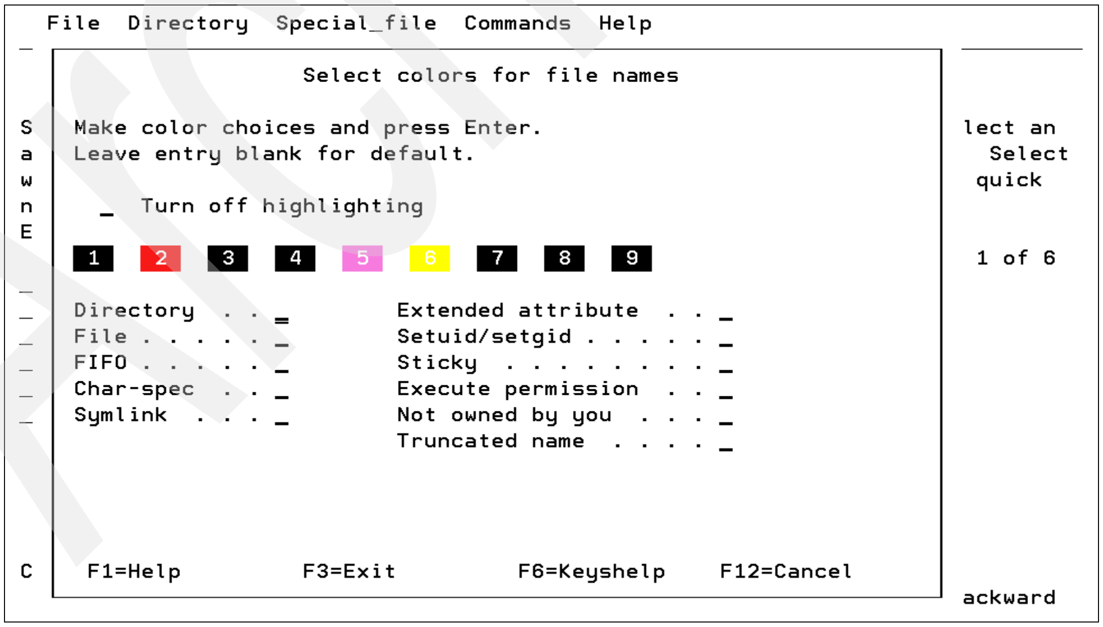

```
  File  Directory  Special_file  Commands  Help
_                                                                  ________
                    Select colors for file names

S    Make color choices and press Enter.                            lect an
a    Leave entry blank for default.                                   Select
w                                                                    quick
n      _  Turn off highlighting
E
     1    2    3    4    5    6    7    8    9                        1 of 6
_
_    Directory  . . _        Extended attribute  . . _
_    File . . . . . _        Setuid/setgid . . . . . _
_    FIFO . . . . . _        Sticky  . . . . . . . . _
_    Char-spec  . . _        Execute permission  . . _
_    Symlink  . . . _        Not owned by you  . . . _
                             Truncated name  . . . . _



C      F1=Help         F3=Exit         F6=Keyshelp     F12=Cancel
                                                                   ackward
```

Figure 9-17 Highlighting options panel

9.4 Invoking the z/OS UNIX shell

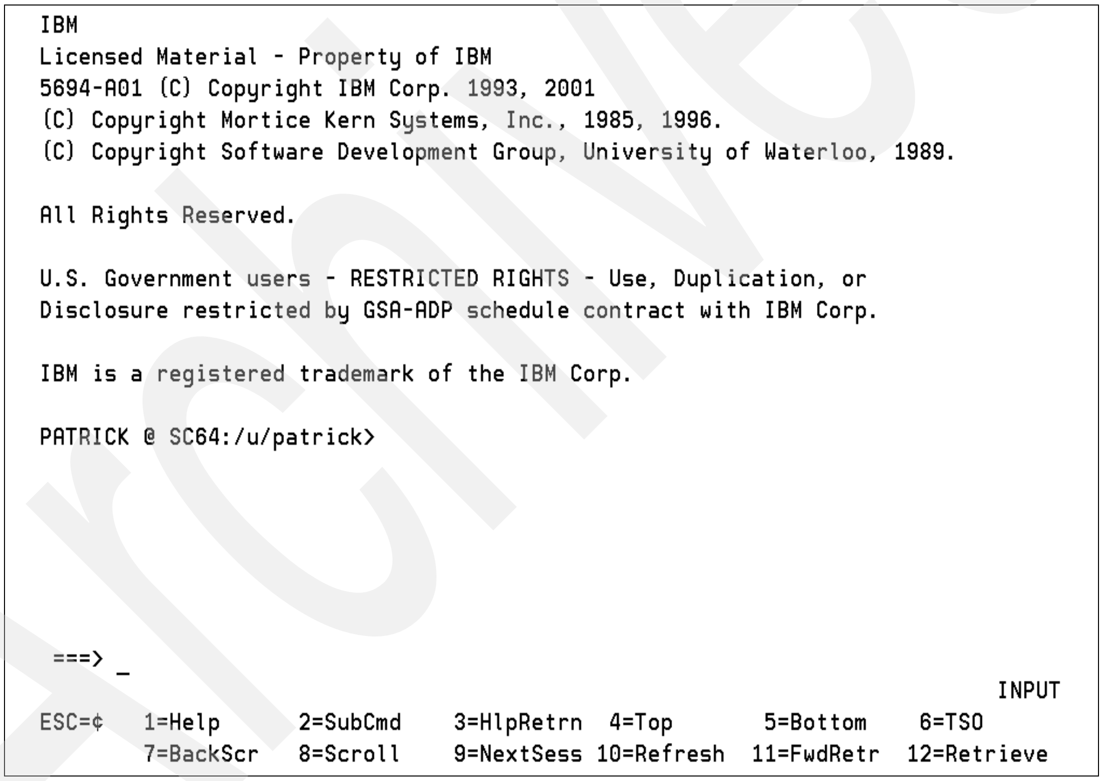
One of the user benefits of UNIX System Services is the z/OS UNIX shell. The shell is a command processor that you use to:

- Invoke shell commands or utilities that request services from the system (similar to TSO/E).
- Write shell scripts using the shell programming language (similar to REXX).
- Run shell scripts, Java language programs, and C-language programs interactively in the foreground, in the background, or in batch (again, similar to REXX).

The shell can be invoked through a TSO/E command called OMVS. Like the OBROWSE, OEDIT, and ISHELL commands, the OMVS command can also be added to an ISPF selection panel, or entered as a TSO/E command.

After the OMVS command is entered, the system initializes the shell for that user.

If successful, the user is presented with the startup screen.



```
IBM
Licensed Material - Property of IBM
5694-A01 (C) Copyright IBM Corp. 1993, 2001
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

PATRICK @ SC64:/u/patrick>






 ===> _
                                                                    INPUT
ESC=¢   1=Help      2=SubCmd    3=HlpRetrn  4=Top       5=Bottom    6=TSO
        7=BackScr   8=Scroll    9=NextSess 10=Refresh  11=FwdRetr  12=Retrieve
```

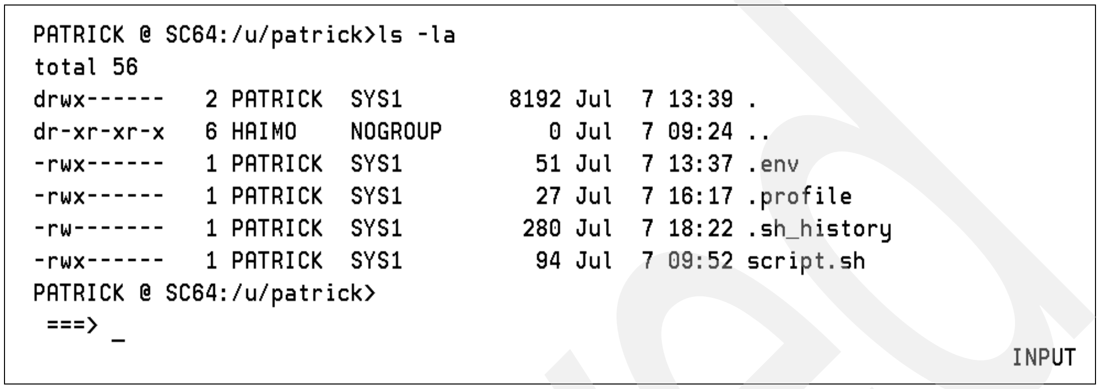
Figure 9-18 z/OS UNIX shell startup screen

At this point, you can issue shell commands. This can be done near the bottom of your screen.

Note: The cent sign (¢) special character is the default used to escape from certain routines. These special characters may differ from country to country, depending on the code page used to interpret characters.

9.4.1 Using z/OS UNIX shell commands

The interactive shell feature comes with multiple commands and utilities that are in most cases the same as the ones that come with a UNIX system. The distinction between a command and a utility is specified in the POSIX 1003.2 standard. To the end user there is no difference between a command and a utility.



```
PATRICK @ SC64:/u/patrick>ls -la
total 56
drwx------   2 PATRICK  SYS1        8192 Jul  7 13:39 .
dr-xr-xr-x   6 HAIMO    NOGROUP        0 Jul  7 09:24 ..
-rwx------   1 PATRICK  SYS1          51 Jul  7 13:37 .env
-rwx------   1 PATRICK  SYS1          27 Jul  7 16:17 .profile
-rw-------   1 PATRICK  SYS1         280 Jul  7 18:22 .sh_history
-rwx------   1 PATRICK  SYS1          94 Jul  7 09:52 script.sh
PATRICK @ SC64:/u/patrick>
 ===> _
                                                               INPUT
```

Figure 9-19 A shell command

Shell commands often have options (also known as flags) that you can specify and they usually take an argument, such as the name of a file or directory. The format for specifying the command begins with the command name, then the option or options, and finally the argument, if any. When you take a look at Figure 9-19, you see that there is a command line at the top of the screen and the following command is shown:

```
ls -la /u/patrick/
```

Where:

ls is the command name, **-la** are the options.

This command lists the files and directories. If the pathname is a file, **ls** displays information on the file according to the requested options. If it is a directory, **ls** displays information on the files and subdirectories therein. You can get information on a directory itself using the **-d** option.

9.4.2 History file

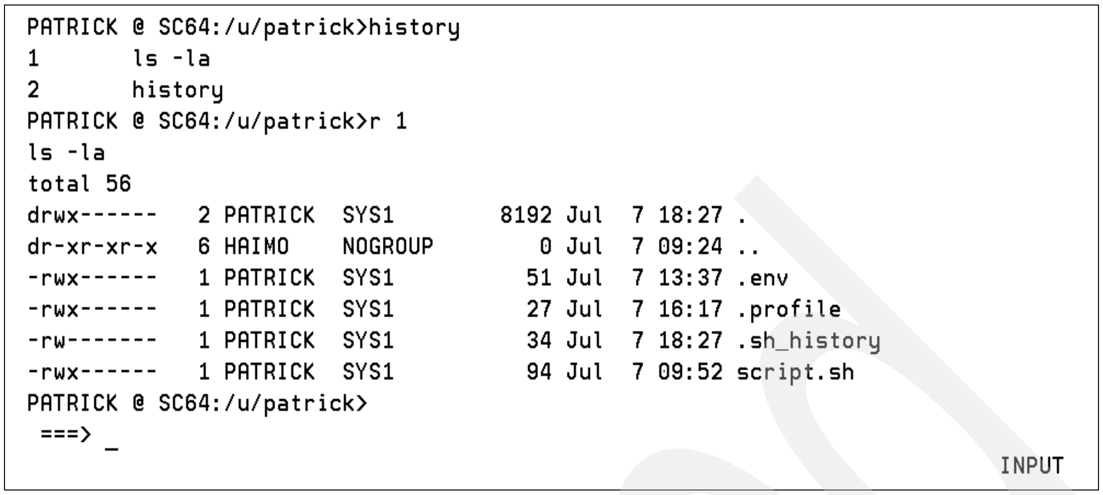
Each command that you enter in the shell is recorded in a file under your home directory. The name of this file is .sh_history and it is called the history file. The default directory to put the history file is set to $HOME/.sh_history. Enter the command:

```
history
```

The shell displays the current content of your history file and puts a number in front of each command.

You can easily rerun any of the commands in the history file by typing an `r` followed by the number of the command you want to use. See Figure 9-20 on page 363.

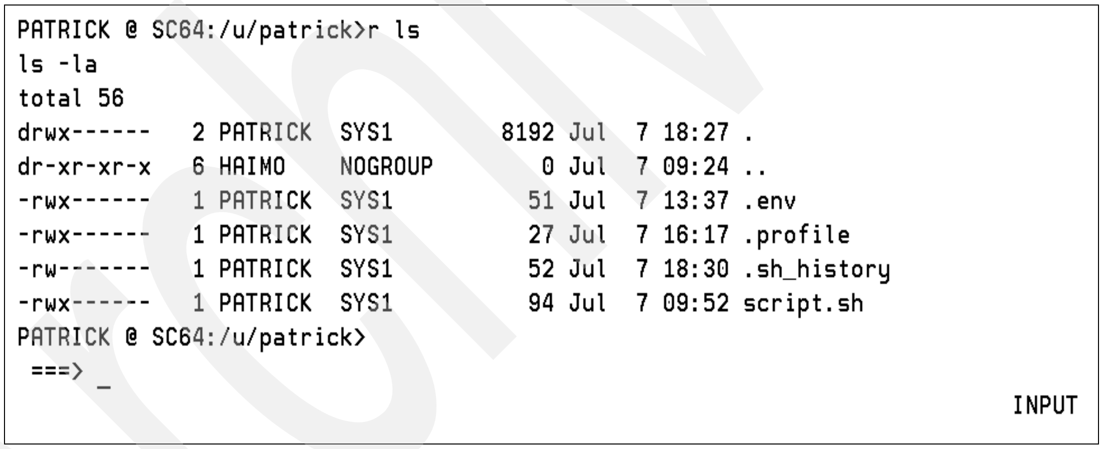
Note: Remember that the home directory for a user is set in its corresponding OMVS segment in RACF. The $HOME variable is then set automatically from the RACF user profile at login time.



```
PATRICK @ SC64:/u/patrick>history
1       ls -la
2       history
PATRICK @ SC64:/u/patrick>r 1
ls -la
total 56
drwx------   2 PATRICK  SYS1        8192 Jul  7 18:27 .
dr-xr-xr-x   6 HAIMO    NOGROUP        0 Jul  7 09:24 ..
-rwx------   1 PATRICK  SYS1          51 Jul  7 13:37 .env
-rwx------   1 PATRICK  SYS1          27 Jul  7 16:17 .profile
-rw-------   1 PATRICK  SYS1          34 Jul  7 18:27 .sh_history
-rwx------   1 PATRICK  SYS1          94 Jul  7 09:52 script.sh
PATRICK @ SC64:/u/patrick>
 ===> _
                                                                INPUT
```

Figure 9-20   The history command

We showed you how to rerun a command by typing an **r** followed by the number from the history file. You can also rerun the most recent command by typing an **r** followed by a string of characters. The shell will than check the history file and search for the most recent command, that begins with the given string. This saves you time, because you don’t have to look up the exact number again from the history file. See Figure 9-21.



```
PATRICK @ SC64:/u/patrick>r ls
ls -la
total 56
drwx------   2 PATRICK  SYS1        8192 Jul  7 18:27 .
dr-xr-xr-x   6 HAIMO    NOGROUP        0 Jul  7 09:24 ..
-rwx------   1 PATRICK  SYS1          51 Jul  7 13:37 .env
-rwx------   1 PATRICK  SYS1          27 Jul  7 16:17 .profile
-rw-------   1 PATRICK  SYS1          52 Jul  7 18:30 .sh_history
-rwx------   1 PATRICK  SYS1          94 Jul  7 09:52 script.sh
PATRICK @ SC64:/u/patrick>
 ===> _
                                                                INPUT
```

Figure 9-21   The redo command

Another way to rerun commands when you are using the OMVS interface is to use the function keys for retrieving commands:

Retrieve	This key is usually activated by pressing the PF12 key. It performs a “backward retrieving” function.
FwdRetr	This key is activated by pressing the PF11 key and is used within the Retrieve key, to retrieve commands. It gives you the opportunity to scroll forwards again, if you pressed the PF12 key one time to many.

9.4.3 Customizing $HOME/.profile

When you start the z/OS UNIX shell, it uses information in three files to set up environment variables that determine system and user defaults and preferences. The information is gathered in these files, in this order:

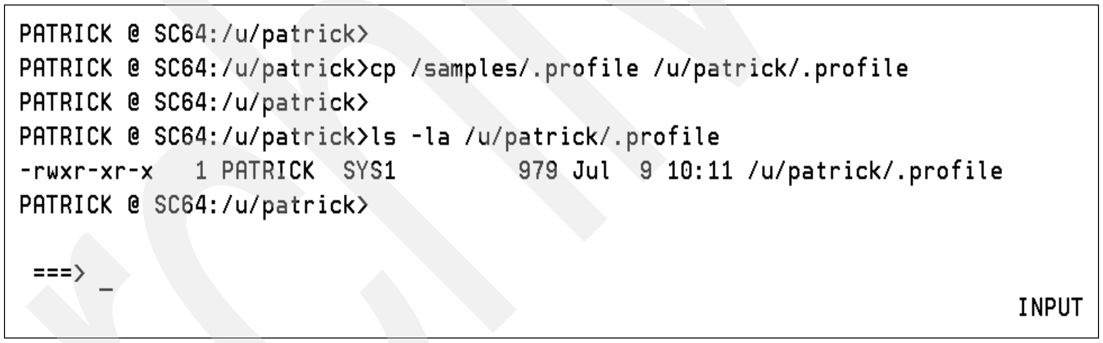
1. etc/profile
2. $HOME/.profile (notice the dot ".")
3. The file that the ENV variable specifies (for example, $HOME/.env or $HOME/.setup)

Note: Later settings take precedence. For example, the values set in $HOME/.profile override those in /etc/profile.

The etc/profile is discussed in 2.3.16, "Step 16 - Customize /etc/profile" on page 78. It is the first file that is searched and sets up a default system-wide user environment. This file is normally set up by the system programmer or administrator to reflect any system-wide requirements such as setting the local time zone. The /etc/profile file is also used to set c89/cc compiler and runtime library environment variables.

An example for customizing the /etc/profile can be found in /samples/etc. Simple copy /samples/etc to /etc/profile and adjust it to your system-wide needs.

The $HOME/.profile file (where $HOME is a variable for the individual user's home directory) is an individual user profile. You can override any values that are set in the /etc/profile file by coding them in your own $HOME/.profile file. A sample $HOME/.profile file is supplied in /samples/.profile, and should be copied into each user's home directory. Figure 9-22 shows the commands to copy the /samples/.profile file into a user's home directory. Type in OMVS from ISPF option 6 to enter the z/OS UNIX shell.



```
PATRICK @ SC64:/u/patrick>
PATRICK @ SC64:/u/patrick>cp /samples/.profile /u/patrick/.profile
PATRICK @ SC64:/u/patrick>
PATRICK @ SC64:/u/patrick>ls -la /u/patrick/.profile
-rwxr-xr-x   1 PATRICK  SYS1         979 Jul  9 10:11 /u/patrick/.profile
PATRICK @ SC64:/u/patrick>

 ===> _
                                                                     INPUT
```

Figure 9-22 Copying the .profile file into the user's home directory

Table 9-1 shows some of the environment variables the user may use in the .profile file.

Table 9-1 Important variables that you can change

Variable	Explanation
PATH	Standard search path for executables; you change it for your own search order. Do not forget to append the original path.
CLASSPATH	Standard search path for Java classes.
PS1	The default appearance of your command prompt.
EDITOR	Set your default editor.
_LC_*	Variable that defines your language settings in your shell.
TZ	Sets the current time zone.

Edit the $HOME/.profile

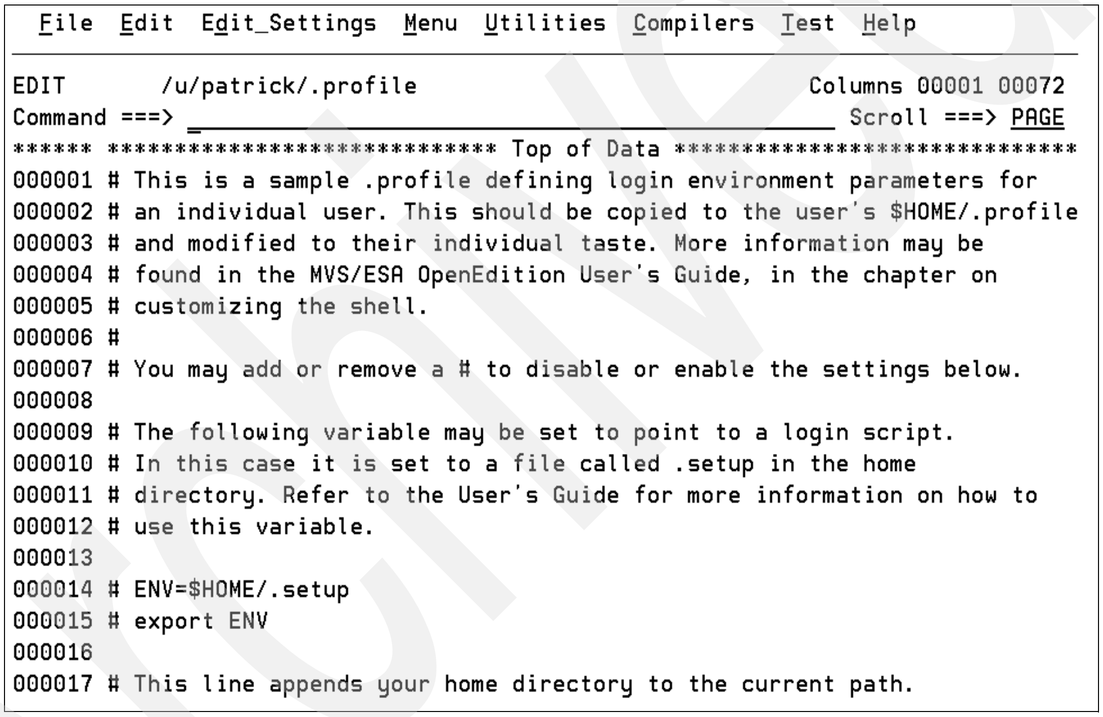
Because we just copied the $HOME/.profile from the samples directory, it still needs some modifications to make it work for our user.

To edit the $HOME/.profile file we can use the **oedit** command, which enables us to edit a file within the hierarchical file system (HFS). This command uses the TSO/E OEDIT command and must run in the foreground. From the command prompt, type:

```
oedit .profile
```

After you execute the command, **oedit** will get you into TSO/E OEDIT, as shown in Figure 9-23.

From here on you can edit the $HOME/.profile file, as you would edit any other file under native MVS. Just use normal commands, such as up (PF7), down (PF8), line insert (i), line copy (c), or line move (m), to do the work.



```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
──────────────────────────────────────────────────────────────────────
EDIT       /u/patrick/.profile                             Columns 00001 00072
Command ===> _                                                Scroll ===> PAGE
****** ***************************** Top of Data ****************************
000001 # This is a sample .profile defining login environment parameters for
000002 # an individual user. This should be copied to the user's $HOME/.profile
000003 # and modified to their individual taste. More information may be
000004 # found in the MVS/ESA OpenEdition User's Guide, in the chapter on
000005 # customizing the shell.
000006 #
000007 # You may add or remove a # to disable or enable the settings below.
000008
000009 # The following variable may be set to point to a login script.
000010 # In this case it is set to a file called .setup in the home
000011 # directory. Refer to the User's Guide for more information on how to
000012 # use this variable.
000013
000014 # ENV=$HOME/.setup
000015 # export ENV
000016
000017 # This line appends your home directory to the current path.
```

Figure 9-23 TSO/E OEDIT

Note: You can specify a number of file names. The TSO/E OEDIT command is invoked once for each file. However, if you do not specify a file name at all, the Edit Entry panel is displayed (see Figure 9-24 on page 366).

Environment file

We have a special file available to put all the environment definitions in. This is the so-called environment file. The name of the environment file is your own choice, but you have to specify it. This is done by putting a line in your $HOME/.profile that tells the shell where the environment file is.

Enter the edit mode for the $HOME/.profile file by using the **oedit** command. Edit lines 14 and 15 in the sample you copied to your own $HOME directory. Change these lines from:

```
# ENV=$HOME/.setup
export ENV
```

to:

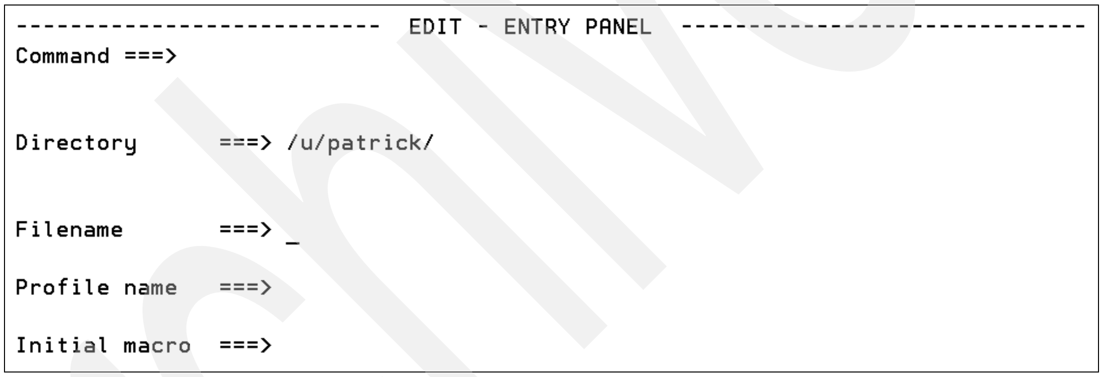
```
ENV=$HOME/.env
export ENV
```

Exit the edit mode by using the **`exit`** command or press PF3. You now have specified that the $HOME/.env file will be your environment file. This file still doesn't exist and we will have to create it.

Make sure your working directory is set to your $HOME directory. If not, enter **`cd $HOME`** to get back to your home directory. After that, run the following command from the command prompt in the z/OS UNIX shell:

```
oedit
```

Because no file name was specified, the Edit Entry Panel appears. Your cursor will be positioned at Filename and you can type the name of the file you want to create. See also Figure 9-24. Specify `.env` at the cursor position and press Enter. Once again you will be in TSO/E OEDIT mode, but this time to create a completely new file in the hierarchical file system.



```
---------------------------  EDIT - ENTRY PANEL  ------------------------------
Command ===>


Directory      ===> /u/patrick/


Filename       ===> _

Profile name   ===>

Initial macro  ===>
```

Figure 9-24 Edit entry panel

At this point we don't have too many variables that we want to put into our environment file. But we will run a shell script later that executes a Java program. See 9.4.7, "REXX, CLISTs, and shell scripts" on page 369. Therefore, we put a CLASSPATH statement in our environment file for later use:

```
export CLASSPATH=/u/patrick:$CLASSPATH
```

You can now exit the TSO/E EDIT mode by pressing PF3 to return to the z/OS UNIX shell. Once you exit and reenter the z/OS UNIX shell, the shell uses etc/profile, $HOME/.profile, and $HOME/.env to set up the environment.

Note: Although you don't have to put your environment definitions into the environment file, we still advise you to do so, instead of using $HOME/.profile or etc/profile. It is a common rule to put as few definitions as possible into $HOME/.profile and as many as possible into your environment file.

9.4.4 Command prompt

You can see in Figure 9-18 on page 361, that the command prompt says:

```
PATRICK @ SC64:/u/patrick>
```

This command prompt provides you with some information. We configured it to show the logon name of the user, the hostname, and the current directory.

The layout of the command prompt has been set using the PS1 shell environment variable. In our environment this was done in etc/profile. But you can just as well put it into your $HOME/.profile.

To establish this command prompt layout, we must set the following environment variable:

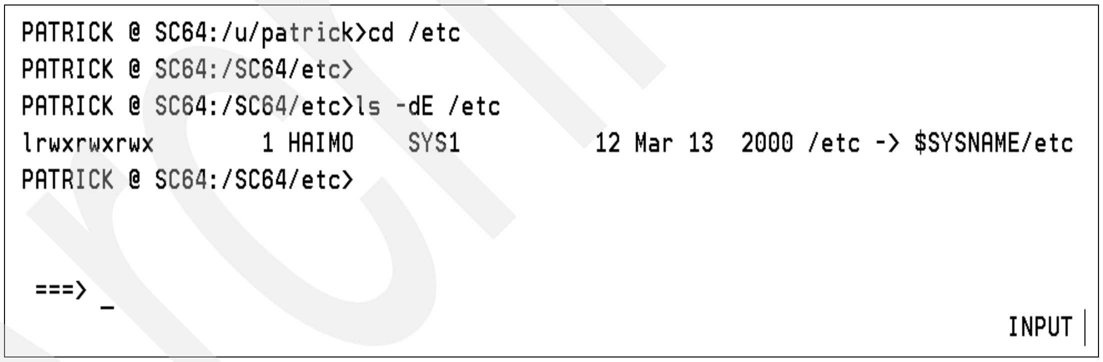
```
export PS1='$LOGNAME @ $HOSTNAME:$PWD>'
```

Logical path names

On the PS1 environment variable you can easily decide how your z/OS UNIX shell prompt should look. We just gave you a practical example of that, but sometimes the layout of your prompt can be a little bit confusing for the users. To understand this we have to explain some things.

By default the shell commands **cd**, **pwd** and the PWD variable all use physical directory path names. For example, assume /etc is a symbolic link to /SC64/etc, and that it is your current directory. See also Figure 9-25, where we used the **cd** command.

For more information on symbolic links, read "Symbolic links" on page 21 and "Symbolic and external links with a sticky bit" on page 21.



```
PATRICK @ SC64:/u/patrick>cd /etc
PATRICK @ SC64:/SC64/etc>
PATRICK @ SC64:/SC64/etc>ls -dE /etc
lrwxrwxrwx       1 HAIMO    SYS1         12 Mar 13  2000 /etc -> $SYSNAME/etc
PATRICK @ SC64:/SC64/etc>


 ===> _
                                                                          INPUT
```

Figure 9-25 Symbolic link

The prompt that is now displayed may look somewhat confusing. Instead of just /etc, it now displays:

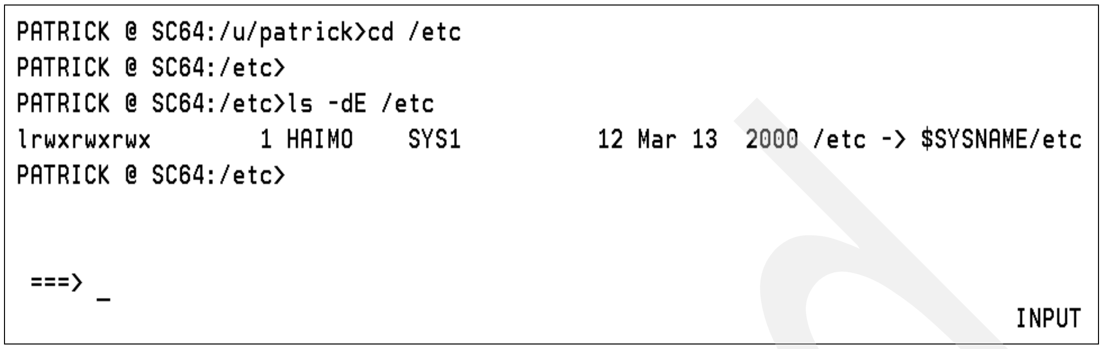
```
PATRICK @ SC64:/SC64/etc>
```

It shows the complete physical directory name, /SC64/etc, and that may seem a bit strange, if you expect just to see /etc. In addition, your PS1 environment settings also show the $HOSTNAME, which is SC64. That can make it even more confusing.

To make things simpler to watch and to understand, you can use the **set** command, so that it only shows the logical path names in directories with symbolic links. To do that, just add the following command to your $HOME/.profile or let your z/OS UNIX administrator add it to the etc/profile:

```
set -o logical
```

Once you set this command, your z/OS UNIX shell will *not* show the physical directory anymore, when there is a symbolic link involved. See Figure 9-26.



```
PATRICK @ SC64:/u/patrick>cd /etc
PATRICK @ SC64:/etc>
PATRICK @ SC64:/etc>ls -dE /etc
lrwxrwxrwx        1 HAIMO    SYS1          12 Mar 13  2000 /etc -> $SYSNAME/etc
PATRICK @ SC64:/etc>



 ===> _
                                                                            INPUT
```

Figure 9-26

Using the following command:

```
ls -dE /etc
```

shows you where the physical directory path name resolves to. See also Figure 9-26.

Note: The **`ls`** command we executed to display the /etc directory also displayed the permission bits settings. There is a lower case “l” in front of that. The first character identifies the file type and in this case it is an “l”, meaning that we are dealing here with a symbolic link.

9.4.5 Built-in shell variables

There is a list of built-in shell variables. You can make them available in the environment for all commands that run from the shell by exporting them. Exporting is done with the **`export`** command.

Practically every variable can become an environment variable. To do this you must export it with the command:

```
export variablename
```

Where `variablename` is the name of the variable that you wish to add to the environment. Or it can be a list of variable names separated by blanks.

Some built-in variables are basically environment variables. Well-known examples of environment variables are shown in Table 9-2.

Table 9-2 Built-in shell variables

Variable	Purpose
LOGNAME	Contains the user login name. This is set automatically from the RACF user profile when the user logs in.
PWD	Contains the name of the working directory. When the shell starts, the working directory name is assigned to PWD unless the variable already has a value.
HOME	Contains your home directory. This is also the default directory for the **`cd`** command. The HOME variable is set automatically from the RACF user profile when the user logs in.

For a full list of built-in shell variables, see *z/OS UNIX System Services Command Reference,* SA22-7802. Chapter 2 (see the SH command) gives a description of all the built-in shell variables available in the z/OS UNIX shell.

9.4.6 Subcommand mode

Once you enter the z/OS UNIX shell, you can enter the subcommand mode by pressing the subcommand function key. This is usually done by pressing the Esc or cent (¢) key on your keyboard. When you switch to subcommand mode, the command prompt changes to OMVS Subcommand ==>.

While in subcommand mode, you can enter subcommands on the command line or use function keys. In subcommand mode you can, for example, return temporarily to TSO/E command mode.

Note: A status indicator and shell session number (if more than one session has been started) are displayed in the lower right-hand corner of the screen.

9.4.7 REXX, CLISTs, and shell scripts

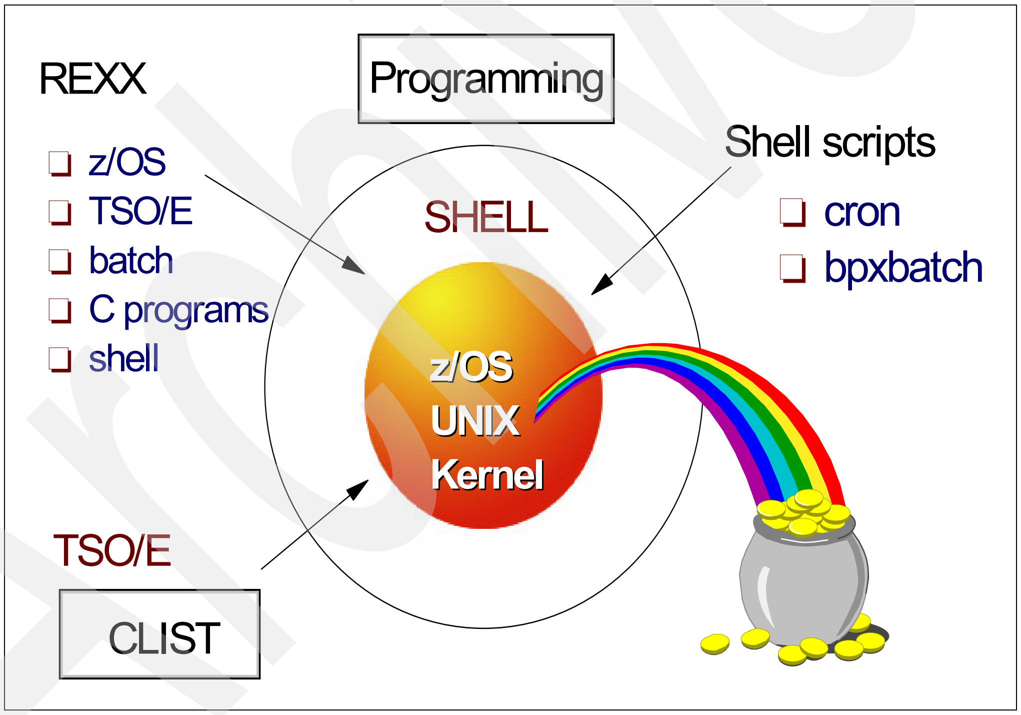


Figure 9-27 REXX, CLISTs, and shell scripts

The shell programming environment with its shell scripts provides function similar to the TSO/E environment with its command lists (CLISTs) and the REstructured eXtended eXecutor (REXX) execs. The CLIST language is a high-level interpreter language that lets you work efficiently with TSO/E. A CLIST is a program, or command procedure, that performs a given task or group of tasks.

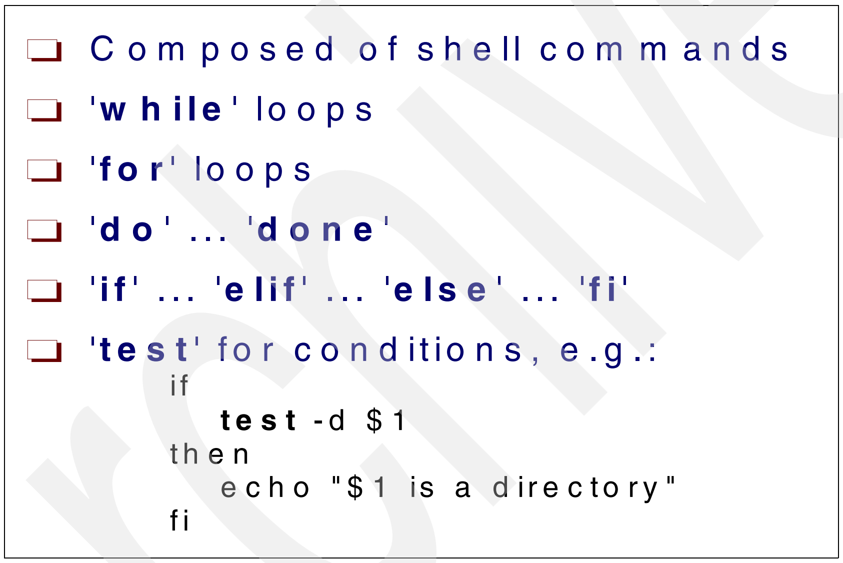
The REXX language is a high-level interpreter language that enables you to write programs in a clear and structured way. You can use the REXX language to write programs called REXX programs, or REXX execs, that perform given tasks or groups of tasks. REXX programs can run in any z/OS address space. You can run REXX programs that call z/OS UNIX services in TSO/E, in batch, in the shell environment, or from a C program.

In the z/OS UNIX shell, command processing is similar to command processing for CLISTs. You can write executable shell scripts (a sequence of shell commands stored in a text file) to perform many programming tasks. They can run in any dubbed z/OS address space. They can be run interactively, using cron, or using BPXBATCH. With its commands and utilities, the shell provides a rich programming environment.

Performance improvement: To improve performance when running shell scripts, add to the export statement in **/etc/profile** or **$HOME/.profile**:

```
_BPX_SPAWN_SCRIPT=YES
_BPX_SHAREAS=YES
```

Shell script syntax



- Composed of shell commands
- '**while**' loops
- '**for**' loops
- '**do**' ... '**done**'
- '**if**' ... '**elif**' ... '**else**' ... '**fi**'
- '**test**' for conditions, e.g.:

```
if
    test -d $1
then
    echo "$1 is a directory"
fi
```

Figure 9-28 Shell script syntax

As you can see in Figure 9-28, a shell script is composed of different types of code. In a script file, you can use z/OS UNIX shell commands, flow control constructions like if-then-else, variables, environment settings, and so on.

If you are familiar with programming languages, you will have no problems writing your first shell script.

If you are setting environment variables through a shell script, consider the following:

Any variables set in a shell script are set only while the script is running and do not affect the shell that invoked the script (unless the script is sourced by running it with the **.** (dot) command).

To run a shell script in your current environment without creating a new process, use the **.** (dot) command. You could run the compile shell script this way:

. script-file-name

You can improve shell script performance by setting the _BPX_SPAWN_SCRIPT environment variable to YES.

```
if
 test -f java_memory.class
then
  echo "The Java program is present. Let's execute it now!"
  java java_memory
elif
 test -f java_memory.java
 then
     echo "The Java source is there. Let's compile it first!"
     javac java_memory.java
     echo "Now we will executed the Java code."
     java java_memory
  else
   echo "there is no Java source present to compile"
fi
```

Figure 9-29 Shell script example

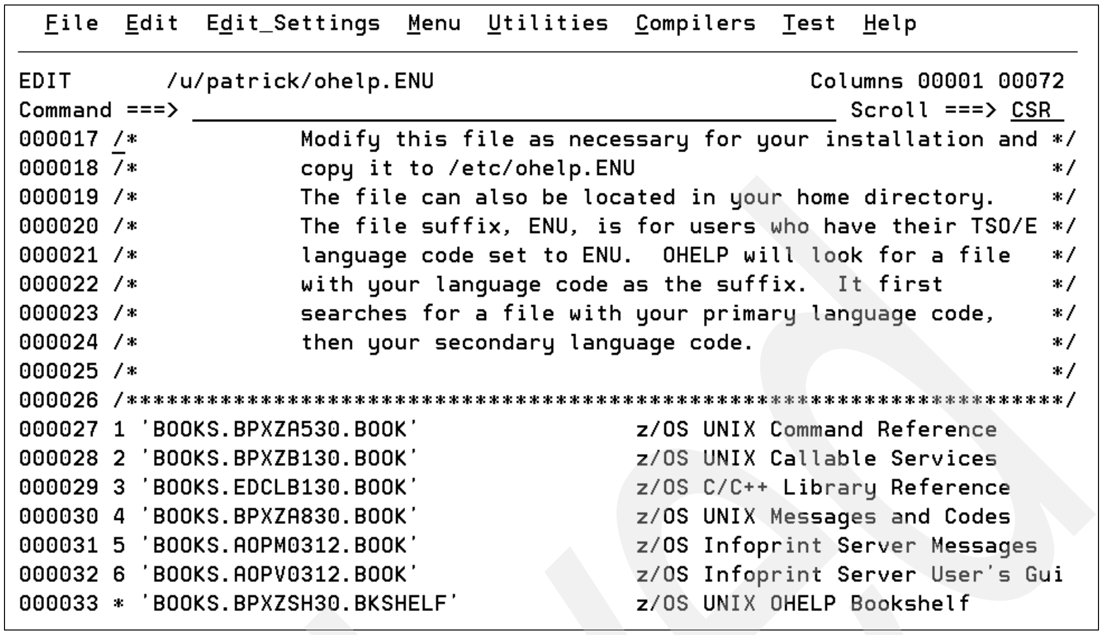
Figure 9-29 shows an example of a shell script. This script checks whether there is a specific Java class available. If it is present, it is executed. If it is not present, the script checks to see if the source is available. If yes, the source is compiled and the Java program is executed.

9.4.8 Help facilities

An optional step for the installation of the UNIX System Services environment is to install books and a bookshelf® for the online help facility. This facility enables you to find information about shell commands, shell messages, callable services, and C functions. Once the books are installed, this facility is accessed via the TSO/E command OHELP. The help facility uses the BookManager READ element.

In order to be able to use this facility, it is necessary to have the BookManager READ product installed on your system.

If you have BookManager READ installed, you can easily set up the OHELP configuration. For that, you have to copy the sample file /samples/ohelp.ENU to the /etc directory or to your $HOME directory. After that, open the file for editing with the OEDIT command, for example. Change the listed books and bookshelf so they match the data set naming convention in your z/OS environment. See also Figure 9-30 on page 372.



```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       /u/patrick/ohelp.ENU                           Columns 00001 00072
Command ===> ________________________________________________ Scroll ===> CSR
000017 /*           Modify this file as necessary for your installation and */
000018 /*           copy it to /etc/ohelp.ENU                                */
000019 /*           The file can also be located in your home directory.     */
000020 /*           The file suffix, ENU, is for users who have their TSO/E */
000021 /*           language code set to ENU.  OHELP will look for a file   */
000022 /*           with your language code as the suffix.  It first        */
000023 /*           searches for a file with your primary language code,    */
000024 /*           then your secondary language code.                      */
000025 /*                                                                   */
000026 /*******************************************************************/
000027 1 'BOOKS.BPXZA530.BOOK'              z/OS UNIX Command Reference
000028 2 'BOOKS.BPXZB130.BOOK'              z/OS UNIX Callable Services
000029 3 'BOOKS.EDCLB130.BOOK'              z/OS C/C++ Library Reference
000030 4 'BOOKS.BPXZA830.BOOK'              z/OS UNIX Messages and Codes
000031 5 'BOOKS.AOPM0312.BOOK'              z/OS Infoprint Server Messages
000032 6 'BOOKS.AOPV0312.BOOK'              z/OS Infoprint Server User's Gui
000033 * 'BOOKS.BPXZSH30.BKSHELF'           z/OS UNIX OHELP Bookshelf
```

Figure 9-30 ohelp.ENU

For more detailed information, see *z/OS UNIX System Services Planning,* GA22-7800 on installing books for the OHELP command.

You can include as many books as you want. To obtain information from the books, simply type the following command:

```
OHELP refid name
```

The `refid` operand represents a specific book you want to search, and the `name` operand represents the element for which you are looking for information. Every book is assigned a refid, so if you know it, just include it.

If you do not specify a refid, but include the element you want to search, by default you are pointed to the book with the refid of 1. This is normally set up as the command reference book.

```
OHELP name
```

If you only specify a refid, and do not include an element, as in the next command, you are shown the table of contents of the book identified by refid.

```
OHELP refid
```

If you want to see the list of available books and the refids associated with them, just type the OHELP command.

The OHELP command can be entered in TSO/E, ISPF, but *not* in the z/OS UNIX shell. The help information is displayed in a BookManager session and therefore it doesn't work from the z/OS UNIX shell. To display help information about a shell command, while working from within the z/OS UNIX shell, use the **`man`** command.

Man command

The **`man`** command can provide you with help information about a shell command. It works from within the z/OS UNIX shell. The syntax is as follows:

```
man command_name
```

Before you can access the man pages (help files), you have to enable them.

1. Depending on your Bookserver data set, you have to do one of the following:
   - If you are not using the default IBM-supplied prefix on data set EPH.SEPHTAB, you have to create the SYS1.PARMLIB member EPHWP00. In this member you should define the new prefix of the SEPHTAB data set. It should contain:

     ```
     DSN=<newprefix>.SEPHTAB
     ```
   - If you rename the SEPHTAB data set to another suffix, then you have to create a file called /etc/booksrv/bookread.conf. Let it contain:

     ```
     DSN=fully.qualified.dsn.where.members.are
     ```

     Note: You may decide not to use the etc/booksrv/bookread.conf file. In that case you should set an environment variable to let the shell know where to find the BookRead configuration file. This is done by using the EPHBookReadConfig environment variable.

2. Also, you have to set the MANPATH environment variable to let the shell know where to find the man pages:

   ```
   MANPATH=/usr/man/%L
   ```

Note: For more information on the OHELP facilities and the `man` command facilities in the UNIX System Services environment, refer to *z/OS UNIX System Services User's Guide,* SA22-7801.

9.5 Direct login to the z/OS UNIX shell

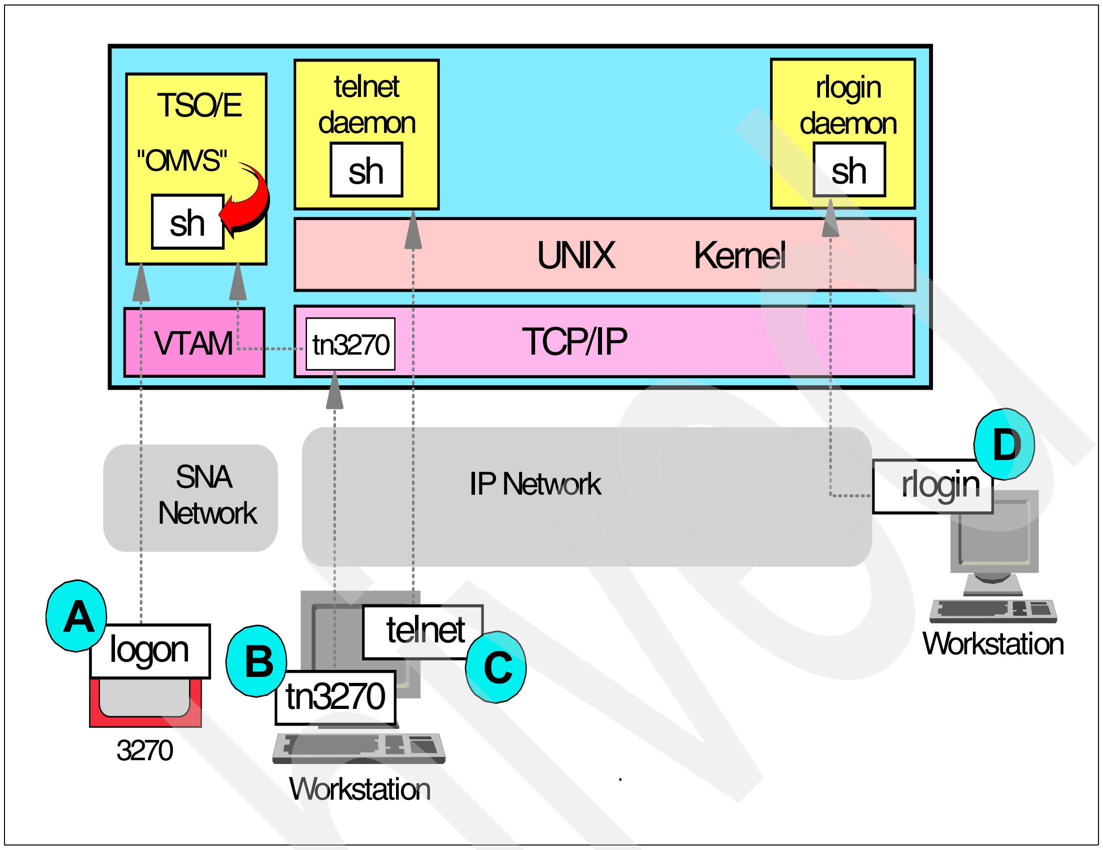


Figure 9-31 Logging into the z/OS UNIX shell

In 9.4, "Invoking the z/OS UNIX shell" on page 361 we showed how to access the z/OS UNIX shell with the TSO/E OMVS command. This is done by either:

- A TSO user, logged in via SNA and VTAM®. **(A)** See Figure 9-31.
- A workstation user that uses the TCP/IP tn3270 protocol to log on to a TSO user ID via IP 3270(E) Telnet Server. **(B)** See Figure 9-31.

In both cases the user is able to access the z/OS UNIX shell via the TSO/E OMVS command after login. This is through tn3270 or via SNA and VTAM.

Besides using the ISHELL or the OMVS command after your TSO/E logon, you can also decide to log in directly to the z/OS UNIX shell. As shown in Figure 9-31, you can choose using one of the following solutions:

telnet The telnet support comes with the TCP/IP z/OS UNIX feature. It also uses the inetd daemon which must be active and set up to recognize and receive the incoming telnet requests.**(C)** See Figure 9-31.

rlogin When the inetd daemon is set up and active, you can use rlogin to connect to the z/OS UNIX shell from a workstation that has rlogin client support. Also it has to be connected via TCP/IP or Communications Server to the MVS system. To login, use the `rlogin` (remote log in) command syntax supported at your site. **(D)** See Figure 9-31. The remote login command `rlogin` is commonly found on UNIX systems.

There are some differences between the asynchronous terminal support (direct shell log in) and the 3270-terminal support (the OMVS command):

- You cannot switch to TSO/E. However, you can use the TSO shell command to run a TSO/E command from your shell session.
- You cannot use the ISPF editor (this includes the **`oedit`** and TSO/E OEDIT commands, which invoke ISPF edit).

Telnet login

The Telnet protocol is usually part of the TCP/IP suite of protocols, which enables you to connect to a remote computer over a network. We will now demonstrate a direct login to the z/OS UNIX shell by using the Telnet client software from our workstation. We currently have Windows 2000 Professional running on our workstation. You can use the Telnet client provided with Windows 2000 to connect to a remote computer, log on to it, and interact with it.

The Telnet client is a command-line application, so it may look very similar to users of UNIX-based Telnet clients. To activate a Telnet client session, just start up a command prompt session in Windows 2000.The Telnet command is specified as follows:

```
telnet   [host  [port]]
```

where:

host	Specifies the hostname or IP address of the remote computer to connect to.
port	Specifies the port number or service name.

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>telnet wtsc64oe
```

Figure 9-32 telnet login command

Figure 9-32 shows the actual command we used to set up a direct login Telnet session to the z/OS UNIX shell.

After the message `EZYTE27I` you can specify your user ID. Press Enter and you will be asked to specify your password, as follows:

```
EZYTE27I login: patrick
EZYTE28I patrick Password:
```

```
EZYTE27I login: patrick
EZYTE28I patrick Password:
IBM
Licensed Material - Property of IBM
5694-A01 (C) Copyright IBM Corp. 1993, 2001
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

PATRICK:/u/patrick: >
```

Figure 9-33 the z/OS UNIX shell

Figure 9-33 shows the z/OS UNIX shell after being entered from a Telnet login. Remember that from this session you cannot switch to TSO/E or use the ISPF editor.

Note: 4.3, “Customizing for inetd and rlogind daemons” on page 201 and 4.4.2, “Otelnetd daemon” on page 208 have more details about this. We just wanted to give you a picture of the methods to invoke the shell.

9.6 BPXBATCH

BPXBATCH is an MVS utility that you can use to run shell commands or shell scripts to run executable files through the MVS batch environment. You can invoke BPXBATCH:

- In JCL
- From the TSO/E READY prompt
- From TSO CLISTs and REXX execs
- From a program

BPXBATCH has logic in it to detect when it is run from JCL. If the BPXBATCH program is running as the only program on the job step task level, it sets up the stdin, stdout, and stderr and executes the requested program. If BPXBATCH is not running as the only program at the job step task level, the requested program will run as the second step of a JES batch address space from JCL in batch.

If run from any other environment, the requested program will run in a WLM initiator in the OMVS subsys category.

See Figure 9-34 on page 377 for a sample BPXBATCH job.

```
//DELETE   EXEC PGM=IEFBR14
//STDOUT   DD  PATH='/tmp/&SYSUID..stdout',
//             PATHOPTS=(OCREAT,OWRONLY),
//             PATHMODE=SIRWXU,PATHDISP=(DELETE)
//STDERR   DD  PATH='/tmp/&SYSUID..stderr',
//             PATHOPTS=(OCREAT,OWRONLY),
//             PATHMODE=SIRWXU,PATHDISP=(DELETE)
//*
//BPXBATCH EXEC PGM=BPXBATCH,REGION=0M
//STDIN    DD  PATH='/u/patrick/script.sh',
//             PATHOPTS=(ORDONLY)
//STDERR   DD  PATHOPTS=(OWRONLY,OCREAT),PATHMODE=SIRWXU,
//             PATH='/tmp/&SYSUID..stderr'
//STDOUT   DD  PATHOPTS=(OWRONLY,OCREAT),PATHMODE=SIRWXU,
//             PATH='/tmp/&SYSUID..stdout'
//STDENV   DD  DUMMY
//*
//COMPOUT   EXEC PGM=IKJEFT01,DYNAMNBR=300,COND=EVEN
//SYSTSPRT  DD SYSOUT=*
//HFSOUT    DD PATH='/tmp/&SYSUID..stdout'
//HFSERR    DD PATH='/tmp/&SYSUID..stderr'
//STDOUTL   DD SYSOUT=*,DCB=(RECFM=VB,LRECL=243,BLKSIZE=247)
//STDERRL   DD SYSOUT=*,DCB=(RECFM=VB,LRECL=243,BLKSIZE=247)
//SYSPRINT  DD SYSOUT=*
//SYSTSIN   DD DATA,DLM='/>'
 ocopy indd(HFSOUT) outdd(STDOUTL)
 ocopy indd(HFSERR) outdd(STDERRL)
/>
//
```

Figure 9-34 BPXBATCH running a shell script in the z/OS UNIX shell environment

BPXBATCH makes it easy for you to run, from your TSO/E session, shell scripts or z/OS C executable files that reside in HFS files.

With BPXBATCH, you can allocate the MVS standard files stdin, stdout, and stderr as HFS files for passing input. If you do allocate these files, they must be HFS files. You can also allocate MVS data sets or HFS text files containing environment variables (stdenv). If you do not allocate them, stdin, stdout, stderr, and stdenv default to /dev/null. Allocate the standard files using the data definition PATH keyword options, or standard data definition options for MVS data sets.

The BPXBATCH default for stderr is the same file defined for stdout. For example, if you define stdout to be /tmp/&SYSUID..stdout and do not define stderr, then both printf() and perror() output is directed to /tmp/&SYSUID..stdout.

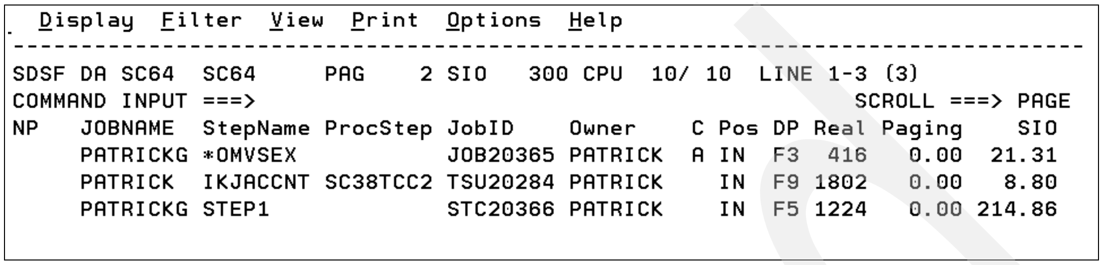
Note: BPXBATCH can only write stdout and stderr to files within the hierarchical file system—not to MVS data sets. You can't use SYSOUT=* for these data sets.

Performance considerations

```
export CLASSPATH=$HOME:$CLASSPATH
java -Xms16MB -Xmx16MB -Xtgc -Xverbosemongc java_memory
```

Figure 9-35 The shell script "script.sh"

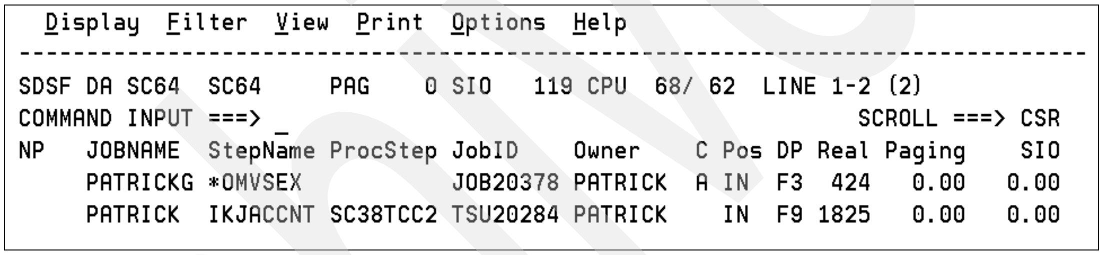
As you can see in Figure 9-34 on page 377, the BPXBATCH sample program is not running as the only program at the job step task level. BPXBATCH is running a shell script, which contains a small Java program (see Figure 9-35 on page 377). Therefore, the requested program will run as the second step of a JES batch address space from JCL in batch. See Figure 9-36 for the additional address space at runtime.



```
  Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF DA SC64  SC64     PAG    2 SIO   300 CPU  10/ 10  LINE 1-3 (3)
COMMAND INPUT ===>                                         SCROLL ===> PAGE
NP   JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real Paging    SIO
     PATRICKG *OMVSEX           JOB20365 PATRICK  A IN  F3  416   0.00  21.31
     PATRICK  IKJACCNT SC38TCC2 TSU20284 PATRICK    IN  F9 1802   0.00   8.80
     PATRICKG STEP1             STC20366 PATRICK    IN  F5 1224   0.00 214.86
```

Figure 9-36 Without _BPX_SHAREAS

To improve performance in the z/OS UNIX shell, set _BPX_SHAREAS to YES. The z/OS UNIX shell will now run foreground processes in the same address space that the shell is running in, which saves the overhead of a fork() and exec(). Figure 9-37 shows the combined address space in SDSF.



```
  Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF DA SC64  SC64     PAG    0 SIO   119 CPU  68/ 62  LINE 1-2 (2)
COMMAND INPUT ===> _                                       SCROLL ===> CSR
NP   JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real Paging    SIO
     PATRICKG *OMVSEX           JOB20378 PATRICK  A IN  F3  424   0.00   0.00
     PATRICK  IKJACCNT SC38TCC2 TSU20284 PATRICK    IN  F9 1825   0.00   0.00
```

Figure 9-37 With _BPX_SHAREAS

Figure 9-38 shows the _BPX_SHAREAS variable. It also shows another setting that can be set to gain some performance improvements, _BPX_SPAWN_SCRIPT=YES. Setting this variable to YES eliminates the additional overhead of the fork. In the default processing, the spawn callable service determines that a file is not an HFS executable or a REXX exec, but a script instead. That leads to the spawn failing with ENOEXEC and the shell then forks another process to run the input shell script. This overhead can be overcome by setting the _BPX_SPAWN_SCRIPT to YES.

```
export _BPX_SHAREAS=YES
export _BPX_SPAWN_SCRIPT=YES
```

Figure 9-38 Environment settings for performance improvement

CPU time limit

The time limit for using a shell is the same as the TSO/E timeout. In determining the time, the system does not count the processing time for shells running in separate address spaces or processes forked by the shell.

```
TOTAL CPU TIME=   .00  TOTAL ELAPSED TIME=  1.15
```

Figure 9-39 Without _BPX_SHAREAS=YES

If you specify environment variable _BPX_SHAREAS=YES, then the shell processes and possibly one shell command are created in local processes. The CPU time consumed by local processes comes out of the TSO/E address space's time limit.

```
TOTAL CPU TIME=  2.20  TOTAL ELAPSED TIME=  1.15
```

Figure 9-40   With _BPX_SHAREAS=YES

If you decide to run the OMVS command with the SHAREAS option, or by setting the environment variable _BPX_SHAREAS=YES, two or more processes can be running in the same address space. In this case, SMF provides process identification only for the first process in the address space. However, resource consumption is accumulated for all processes that are running.

9.6.1 BPXBATSL

BPXBATSL is an alias of BPXBATCH, but it provides an alternate entry point in BPXBATCH. This forces a program to run using a local spawn instead of fork/exec as normal BPXBATCH does. As a result of that a program will run faster, because of less execution overhead.

Another advantage of using BPXBATSL is that DD statements will still be available after your program gets control from BPXBATSL.

Note: BPXBATSL and SH parms will not work together as far as DD names are concerned. You may receive `BPXM018I BPXBATCH FAILED BECAUSE SPAWN (BPX1SPN) OF /BIN/LOGIN FAILED WITH RETURN CODE 0000009D REASON CODE 0B1B0473`.

BPXBATSL sample

We now show an example of a BPXBATSL job. In Figure 9-41 you see JCL that invokes BPXBATSL with program bpxrexx.

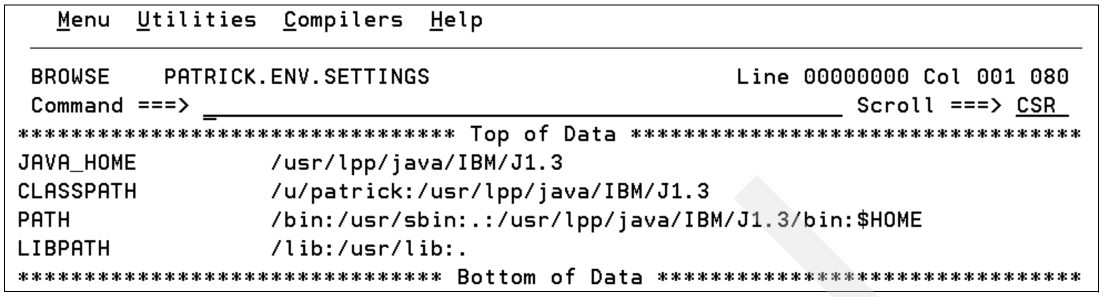
This REXX program uses the information in DD statements ENVIR and CONFIG to execute a Java program. The output is written to DD statements JAVAOUT and JAVAERR in SDSF.

This is something that is not possible with BPXBATCH, because it doesn't allow your program to use DD statements or to write your output directly to SDSF.

```
//RUNBATSL JOB (999,POK),'PATRICK JOB',CLASS=A,MSGCLASS=T,
// NOTIFY=&SYSUID,TIME=1440,REGION=0M
//BPXBATSL EXEC PGM=BPXBATSL,PARM='PGM bpxrexx'
//JAVAOUT  DD  SYSOUT=*,DCB=(RECFM=VB,LRECL=243,BLKSIZE=247)
//JAVAERR  DD  SYSOUT=*,DCB=(RECFM=VB,LRECL=243,BLKSIZE=247)
//ENVIR    DD  DISP=SHR,DSN=PATRICK.ENV.SETTINGS
//CONFIG   DD  DISP=SHR,DSN=PATRICK.CFG.SETTINGS
```

Figure 9-41   RUNBATSL JCL

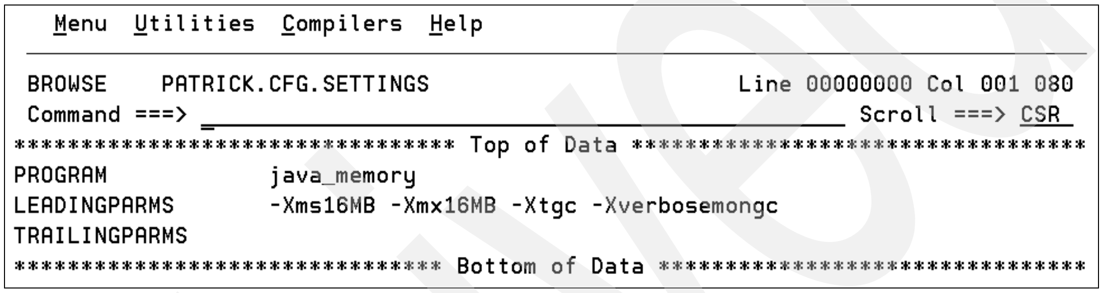
The Java program that we want to execute in our sample needs some environment settings to be set. When you use BPXBATSL, your default environment, which you normally get from /etc/profile and /$HOME/.profile, will not be set up for you. With BPXBATSL you have to set up your own environment from scratch. Figure 9-42 on page 380 shows a data set called PATRICK.ENV.SETTINGS, which is used by bpxrexx to set up the environment.



```
   Menu  Utilities  Compilers  Help
 ──────────────────────────────────────────────────────────────────────────────
 BROWSE    PATRICK.ENV.SETTINGS                      Line 00000000 Col 001 080
 Command ===> _                                                Scroll ===> CSR
********************************* Top of Data **********************************
JAVA_HOME         /usr/lpp/java/IBM/J1.3
CLASSPATH         /u/patrick:/usr/lpp/java/IBM/J1.3
PATH              /bin:/usr/sbin:.:/usr/lpp/java/IBM/J1.3/bin:$HOME
LIBPATH           /lib:/usr/lib:.
******************************** Bottom of Data ********************************
```

Figure 9-42 Environment data set

The bpxrexx program also reads data set PATRICK.CFG.SETTINGS to find out which Java program it should run, including additional parameters. See Figure 9-43.



```
   Menu  Utilities  Compilers  Help
 ──────────────────────────────────────────────────────────────────────────────
 BROWSE    PATRICK.CFG.SETTINGS                      Line 00000000 Col 001 080
 Command ===> _                                                Scroll ===> CSR
********************************* Top of Data **********************************
PROGRAM           java_memory
LEADINGPARMS      -Xms16MB -Xmx16MB -Xtgc -Xverbosemongc
TRAILINGPARMS
******************************** Bottom of Data ********************************
```

Figure 9-43 Configuration data set

At runtime, bpxrexx first reads the environment data set (Figure 9-42) and executes some `export` statements. After that it checks the configuration data set (Figure 9-43) for the actual Java program to run and the extra parameters it needs.

Example 9-1 bpxrexx sample

```
/*   rexx   */
/*******************************************************************/
/*                                                                 */
/* This simple REXX EXEC can be used in combination with JCL       */
/* RUNBATSL to execute Java programs within the z/OS UNIX shell    */
/* using BPXBATSL                                                   /
/* Author:  Patrick Bruinsma - IBM                                 */
/* Date:    16th july 2003                                         */
/*                                                                 */
/*******************************************************************/
/*                                                                 */
/* Setting up the environment using the ENVIR dd statement.        */
/*                                                                 */
address mvs 'execio * diskr' ENVIR '(stem s.'
 do i=1 to s.0
 parse var s.i var1 var2
  part1=space(var1)                /* remove blanks                */
  part2=space(var2)                /* remove blanks                */
  deel1=delword(part1,2)           /* remove trailing linenumber   */
  deel2=delword(part2,2)           /* remove trailing linenumber   */
 call environment deel1,deel2      /* set environment variable     */
 end
/*                                                                 */
/* Executing the z/OS UNIX shell command.                          */
/*                                                                 */
address mvs 'execio * diskr' CONFIG '(stem c.'
 num1=words(c.1)                  /* Determine the number of words  */
 num2=words(c.2)                  /* Determine the number of words  */
 num3=words(c.3)                  /* Determine the number of words  */
/*                                                                  */
/* Resolve the first word in the line                               */
/*                                                                  */
do i=1 to c.0
line1=c.i
   part1=delword(line1,2)
   front1=space(part1)
/*                                                                  */
/* Process the first word of the string                             */
/*                                                                  */
if front1='PROGRAM' then
 do
   cmd1=c.i
   cmd2=delword(cmd1,1,1)              /* delete first word           */
   cmd3=space(cmd2)                    /* remove blanks               */
   delnum=num1-1                       /* start del from word number  */
   cmd4=delword(cmd3,delnum)           /* remove trailing line number */
 end
if front1='LEADINGPARMS' then
 do
   lead1=c.i
   lead2=delword(lead1,1,1)            /* delete first word           */
   lead3=space(lead2)                  /* remove blanks               */
   delnum=num2-1                       /* start del from word number  */
   lead4=delword(lead3,delnum)         /* remove trailing line number */
 end
if front1='TRAILINGPARMS' then
 do
   trail1=c.i
   trail2=delword(trail1,1,1)          /* delete first word           */
   trail3=space(trail2)                /* remove blanks               */
   delnum=num3-1                       /* start del from word number  */
   trail4=delword(trail3,delnum)       /* remove trailing line number */
 end
end
COMMAND='java '||lead4||cmd4||trail4
call bpxwunix COMMAND,,'DD:JAVAOUT','DD:JAVAERR'
```

Note: The REXX function bpxwunix() is new in z/OS UNIX V1R4 and enables you to run shell commands.

10

Tools, functions, and programming interfaces

This chapter describes some helpful tools for use with z/OS UNIX and points to some helpful API functions with code examples.

The z/OS UNIX home page on the World Wide Web has the latest technical news, customer stories, tools, and FAQs (frequently asked questions). You can visit it at:

http://www.ibm.com/servers/eserver/zseries/zos/unix/

Some of the tools available from the Web site are ported tools, and some are home-grown tools designed for z/OS UNIX. All this code works in our environment at the time we make it available, but is not officially supported. Each tool has a README file that describes the tool and any restrictions on its use.

The simplest way to reach these tools is through the z/OS UNIX home page. From the home page, click on Tools and Toys. The code is also available through anonymous ftp from:

ftp://ftp.software.ibm.com/s390/zos/unix/

If you'd like help with customizing z/OS UNIX, then check out the Web-based wizard. You can access it at:

http://www.ibm.com/servers/eserver/zseries/zos/wizards/

This wizard builds two BPXPRMxx parmlib members; one with system processing parameters and one with file system statements. It also builds a batch job that does the initial RACF security setup for z/OS UNIX. Whether you are installing z/OS UNIX for the first time or are a current user who wishes to verify settings, you can use this wizard.

The following tools and programming interfaces are described:

- Useful tools
- REXX functions
- Programming samples for REXX and C
- Using BPX.JOBNAME and jobname activation for USS processes

- C Shell
- Disabling usage of SMF exit IEFUSI for UNIX processes
- USS hard links used with SMP/E
- OMVS syntax checker
- Storage limits for UNIX processes
- Using BPXCOPY
- Using the magic number
- Enhanced ASCII and file tagging

10.1 Useful tools for z/OS UNIX

Table 10-1 shows a list of some helpful tools for use with z/OS UNIX. The utilities listed here are provided with z/OS or available from the Tools and Toys Web site.

Table 10-1 Overview of helpful tools

Tool	Explanation	Location
skulker	Deletes older files in /tmp for example.	/samples
copytree	Makes a copy of a whole directory tree.	/samples
openssh	Establishes an ssl-based secure connection to other machines.	Tools and Toys Web site
USSTools	A collection of useful functions and extensions.	Tools and Toys Web site

10.1.1 Skulker

Skulker finds files that are candidates for deletion in a directory, based on the file age specified by *daysold*. The general calling conventions are:

```
skulker [-irw] [-l logfile] directory daysold
```

Table 10-2 shows the calling parameters for skulker.

Table 10-2 Parameters for skulker

Parameter	Explanation
-i	Displays the files that are candidates for deletion, and prompts the user to stop or continue with file removal. Do not use this option if you are invoking skulker from a cron job. If skulker is invoked with -i from a cron job, no files will be deleted. A message will be mailed to the caller, showing the skulker output that includes the message “Request canceled.”
-r	Moves recursively through subdirectories, finding both files and subdirectories that are equal to or older than the specified number of days.
-w	Does not remove files, but sends a warning to the owner of each old file (using mailx) that the file is a candidate for deletion.
-l	Specifies a logfile to store a list of files that have been deleted, are candidates for deletion, or for which warnings have been mailed; and any errors that may have occurred.

The following call of skulker will delete all files in directory /tmp that are older than 29 days, and produce a logfile.

```
skulker -l /u/lutz/skulker.log /tmp/ 30
```

Sample output of the skulker command shows the output that skulker produces.

```
skulker run on Tue Jul  8 17:03:56 EDT 2003
Files removed on Tue Jul  8 17:03:56 EDT 2003:
/tmp/inetd-stdout
/tmp/inetd-stderr
/tmp/HESSCAN.out
/tmp/HESSCAN.err
```

Figure 10-1   Sample output of the skulker command

You can run skulker periodically with the cron daemon or other tools, such as OPC, with a batch job. Maybe you can send a warning message to the owning user before deleting the files, as follows:

```
skulker -w -l /u/lutz /tmp/ 20
skulker -l /u/lutz /tmp/ 30
```

In this case, the owning user would be informed if the files are equal to or older than 20 days. Then the user has 10 days to save the files if needed. In the second call, all files equal to or older than 30 days are deleted.

10.1.2 Copytree

Copytree is a tool to clone a whole directory structure including subdirectories under another directory, preserving all file attributes (including permission bits and ACLs). It may also be used to check a tree for structural integrity. Copytree has been available from the z/OS UNIX tools disk on the Internet for a long time. Since z/OS V1R3, a supported version is shipped by IBM and resides in the /samples directory.

Notes:

- The REXX functions are available as package REXXFUNC from the z/OS UNIX tools disk. Beginning with z/OS V1R4, they are shipped as a part of z/OS.
- Be aware that copytree cannot handle files greater than or equal to 2 GB in TSO and that it requires the REXX functions for files greater than or equal to 2 GB if run in a shell environment.

Copytree calling conventions are as follows:

```
copytree -aos sourcedir targetdir
```

Figure 10-3 gives detailed descriptions of the calling parameters.

Table 10-3   Parameters for copytree

Parameter	Explanation
-a	Do not issue 30,000 node limit warning.
-o	Do not preserve file ownership.
-s	Attempt to set effective UID to 0 before starting.

Parameter	Explanation
sourcedir	This is the pathname for the source directory where the copy begins (note: pathname, not file system name).
targetdir	This is the pathname for the target directory. This directory must exist and must be empty. The permissions and other attributes of the target directory are not modified to be the same as the source directory. If the <targetdir> is not specified, copytree runs in a mode to check the source file tree.

Figure 10-2 shows a sample output of the copytree tool.

```
Licensed Material - Property of IBM
5694-A01 (C) Copyright IBM Corp. 1993, 2001
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

LUTZ § SC64:/Z04RD1/samples>copytree -s /tmp /u/lutz/tmp
Copying /tmp to /u/lutz/tmp
Scanning for file nodes...
Skipping mountpoint: /tmp/..
Skipping mountpoint: /tmp/zfs
Processing 9 nodes
Creating directories
Scanning for file nodes...
Skipping mountpoint: /tmp/..
Skipping mountpoint: /tmp/zfs
Processing 9 nodes
Creating directories
Creating other files
Setting file attributes
Creating mount points

*******************

Copy complete.  Error count= 0
Directory errors:   0
Directories copied: 1
File errors:        0
Files copied:       8
Symlink errors:     0
Symlinks copied:    0
Char-spec errors:   0
Char-spec copied:   0
FIFO errors:        0
FIFOs copied:       0
Sparse file count:  0
LUTZ § SC64:/Z04RD1/samples>
```

Figure 10-2 Sample output of copytree

10.1.3 OpenSSH

OpenSSH is a suite of network connectivity tools that provide secure encrypted communications between two untrusted hosts over an insecure network.

OpenSSH software tools support the SSH1 and SSH2 protocols. The tools provide shell functions where network traffic is encrypted and authenticated. OpenSSH is based on client and server architecture. It runs the sshd daemon process on the AIX host and waits for the connection from clients. It supports public key and private key pairs for authentication and encryption of channels to ensure secure network connections and host-based authentication. For more information about OpenSSH, see the following Web site:

http://www.openssh.org

This site provides the man page information for the OpenSSH commands.

To set up the tools in the past, it was necessary to download the binaries from the z/OS UNIX Tools and Toys Web site. The following files were needed:

```
openssh-3.5p1-ebcdic-bin.pax.tgz
openssl-0.9.7a-ebcdic-bin.pax
```

Then you had to put these files into a temporary directory on z/OS UNIX. You could use ftp to download them. To unpack the openssh binaries, the following commands could be used:

```
pax -pe -rvf openssh-3.5p1-ebcdic-bin.pax.tgz
pax -pe -rvf openssl-0.9.7a-ebcdic-bin.pax
```

The default output directory was /usr/local.

Important: OpenSSH is now available as a unpriced feature, named IBM Ported Tools for z/OS, that runs on z/OS V1R4 or higher.

This means it is no longer necessary to get it from the OpenSSH Web site, and it is officially supported. The initial version was OpenSSH 3.5p1 (based on OpenSSL 0.9.7b). With APAR OA10315 it gets upgraded to OpenSSH 3.8.1p1. This is related to OpenSSL 0.9.7d.

IBM Ported Tools for z/OS User's Guide, SA22-7985 presents information you need to set up and use the OpenSSH.

10.1.4 The ssh daemon

The ssh daemon, sshd, is listening for incoming connections. It usually uses TCP/IP port 22. First we generate private and public keys, as shown in Figure 10-3 on page 388. Each host has a host-specific RSA key (normally 1024 bits) used to identify the host. Additionally, when the daemon starts, it generates a server RSA key (normally 768 bits). This key is normally regenerated every hour if it has been used, and is never stored on disk.

Whenever a client connects, the daemon responds with its public host and server keys. The client compares the RSA host key against its own database to verify that it has not changed. The client then generates a 256-bit random number. It encrypts this random number using both the host key and the server key, and sends the encrypted number to the server. Both sides then use this random number as a session key that is used to encrypt all further communications in the session.

The rest of the session is encrypted using a conventional cipher, currently Blowfish or 3DES, with 3DES being used by default. The client selects the encryption algorithm to use from those offered by the server. Next, the server and the client enter an authentication dialog. The client tries to authenticate itself using *.rhosts* authentication. *.rhosts* authentication combined with the RSA host key is normally disabled because it is fundamentally insecure, but can be enabled in the server configuration file if desired. System security is not improved unless rshd, rlogind, and rexecd are disabled (thus completely disabling rlogin and rsh in the machine).

```
ssh-keygen -t rsa1 -f /usr/local/etc/ssh_host_key -N " "
Generating public/private rsa1 key pair.
Your identification has been saved in /usr/local/etc/ssh_host_key.
Your public key has been saved in /usr/local/etc/ssh_host_key.pub.
The key fingerprint is:
87:67:1a:99:35:37:12:aa:1c:77:f6:64:40:08:65:c1 LUTZ§WTSC64OE

ssh-keygen -t dsa -f /usr/local/etc/ssh_host_dsa_key -N " "
Generating public/private dsa key pair.
Your identification has been saved in /usr/local/etc/ssh_host_dsa_key.
Your public key has been saved in /usr/local/etc/ssh_host_dsa_key.pub.
The key fingerprint is:
e6:3b:54:a1:18:bc:f7:0e:6d:13:ac:de:65:d5:9e:fd LUTZ§WTSC64OE

ssh-keygen -t rsa -f /usr/local/etc/ssh_host_rsa_key -N " "
Generating public/private rsa key pair.
Your identification has been saved in /usr/local/etc/ssh_host_rsa_key.
Your public key has been saved in /usr/local/etc/ssh_host_rsa_key.pub.
The key fingerprint is:
17:0a:95:63:ab:9f:71:51:e1:64:7b:f8:68:8b:35:28 LUTZ§WTSC64OE
```

Figure 10-3 Sample key generation

When this is done, you can start the ssh daemon. We recommend the use of a started task to start the ssh daemon. Figure 10-4 shows an example of the started task we used.

```
//LUTZSTC  PROC
//* -------------------------------------------------------------------
//* inkove the start xecution of ls inside a stc
//* -------------------------------------------------------------------
//SSHD     EXEC PGM=BPXBATSL,                                          *
//         PARM='PGM /usr/local/sbin/sshd -d',REGION=0M
//STDOUT   DD PATH='/dev/console',PATHOPTS=(OWRONLY)
//STDERR   DD PATH='/dev/console',PATHOPTS=(OWRONLY)
//STDENV   DD PATH='/usr/local/etc/sshd_env',PATHOPTS=(ORDONLY)
//  PEND
```

Figure 10-4 Stc for the ssh daemon

Figure 10-5 shows the output of the ssh daemon. You must at least turn on debug level 1 to see any messages from the ssh daemon.

```
debug1: sshd version OpenSSH_3.5p1
debug1: private host key: #0 type 0 RSA1
debug1: read PEM private key done: type RSA
debug1: private host key: #1 type 1 RSA
debug1: read PEM private key done: type DSA
debug1: private host key: #2 type 2 DSA
debug1: Bind to port 22 on 9.12.6.31.
Server listening on 9.12.6.31 port 22.
```

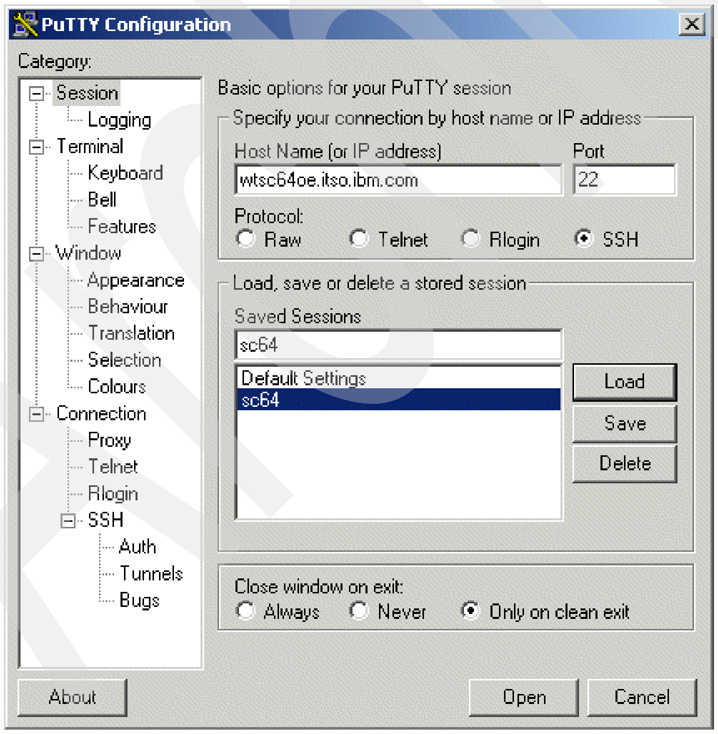
Figure 10-5 Output of the sshd daemon

Note: The userid set for a started task needs read access to the facility class profile BPX.DAEMON. Furthermore, the sshd module needs to be defined program-controlled.

If the started task is running well, you can try to establish a client session to the server. We used the free window utility *putty*. You will find the putty tools for download on the Web site:

`http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`

If you start PuTTY, the dialog shown in Figure 10-6 appears. You must configure the destination host name or IP address and choose the application protocol. In the host name field, enter the address of the sshd daemon. In the protocol selection field, mark SSH and press Open. After a few seconds, enter your userid and password for the remote host. This data is encrypted before transfer.



Basic options for your PuTTY session

Specify your connection by host name or IP address

Host Name (or IP address): wtsc64oe.itso.ibm.com

Port: 22

Protocol: Raw Telnet Rlogin [✓ SSH]

Load, save or delete a stored session

Saved Sessions: sc64

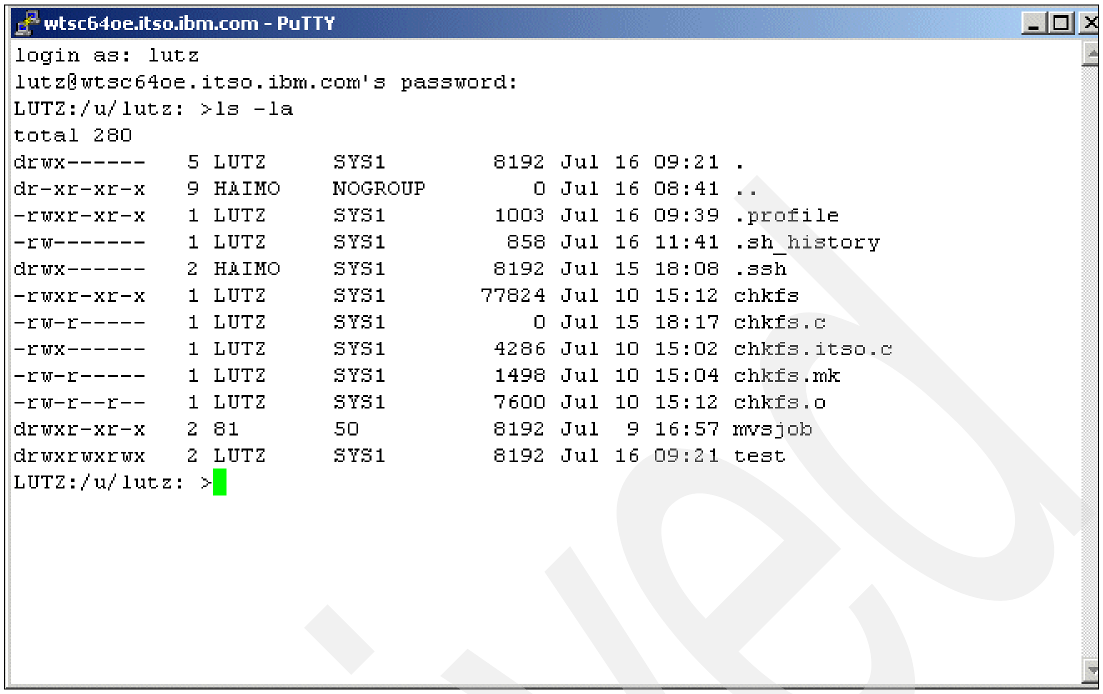
Default Settings

sc64

Close window on exit: Always Never [✓ Only on clean exit]

Figure 10-6 Sample setting for PuTTY

If the connection in established you will see a session screen like the one shown in Figure 10-7 on page 390.



```
login as: lutz
lutz@wtsc64oe.itso.ibm.com's password:
LUTZ:/u/lutz: >ls -la
total 280
drwx------    5 LUTZ     SYS1        8192 Jul 16 09:21 .
dr-xr-xr-x    9 HAIMO    NOGROUP        0 Jul 16 08:41 ..
-rwxr-xr-x    1 LUTZ     SYS1        1003 Jul 16 09:39 .profile
-rw-------    1 LUTZ     SYS1         858 Jul 16 11:41 .sh_history
drwx------    2 HAIMO    SYS1        8192 Jul 15 18:08 .ssh
-rwxr-xr-x    1 LUTZ     SYS1       77824 Jul 10 15:12 chkfs
-rw-r-----    1 LUTZ     SYS1           0 Jul 15 18:17 chkfs.c
-rwx------    1 LUTZ     SYS1        4286 Jul 10 15:02 chkfs.itso.c
-rw-r-----    1 LUTZ     SYS1        1498 Jul 10 15:04 chkfs.mk
-rw-r--r--    1 LUTZ     SYS1        7600 Jul 10 15:12 chkfs.o
drwxr-xr-x    2 81       50          8192 Jul  9 16:57 mvsjob
drwxrwxrwx    2 LUTZ     SYS1        8192 Jul 16 09:21 test
LUTZ:/u/lutz: >
```

Figure 10-7 Sample PuTTY session

10.1.5 USSTools

This package is a collection of functions and extensions for the USS environment. Most of the tools included in this package are designed and useful for one or more of the following situations:

- You are working on z/OS UNIX-related tasks as a BPX.SUPERUSER with a real UID<>0.
- You are using a German or the US terminal emulation and are working in the OMVS shell with the default z/OS USS environment.
- You are editing or browsing UNIX files composed in the default z/OS USS code page. Editing of UNIX files (conversion to and from IBM-1047) is supported for all emulation code pages supported by iconv.
- You want to use the **su** shell command to switch to another userid in the Standard or the C Shell environment.

Table 10-4 provides a list of the commands and functions included in the package.

Table 10-4 USSTools commands and functions

USSTools function	Explanation
AOBROWSE, AOEDIT, SAOEDIT	Browse, edit or edit in SU mode ASCII UNIX files from TSO or the OMVS shell using your host emulation code page. You must run at z/OS V1.4 or have installed the REXX function package REXXFUNC (available from the UNIX Tools Web page).
GOBROWSE, GOEDIT, SGOEDIT	Browse, edit or edit in SU mode UNIX files (created in the standard z/OS UNIX code page IBM-1047) from TSO or the OMVS shell using the German code page conversion table BPXFX273 available with this package.

USSTools function	Explanation
UOBROWSE, UOEDIT, SUOEDIT	Browse, edit or edit in SU mode UNIX files (created in the standard z/OS UNIX code page IBM-1047) from TSO or the OMVS shell using the US code page conversion table BPXFX111 available with z/OS UNIX.
XOBROWSE, XOEDIT, SXOEDIT	Browse, edit or edit in SU mode UNIX files (created in the standard z/OS UNIX code page IBM-1047) from TSO or the OMVS shell using your host emulation code page. You must run at z/OS V1.4 or have installed the REXX function package REXXFUNC (available from the UNIX Tools Web page).
%OMVS	Start an OMVS shell with the German or the US conversion table to support input data entered and show correctly all the UNIX output data according to a German or the US terminal emulation used.
PSINFO	Display information about active processes. The command is supported from TSO and from a UNIX shell.
RXSHELL, RXSUSH	Run UNIX commands from the TSO foreground or within a TSO batch job. RXSUSH tries to switch to SU mode before running a UNIX command.
REXX	Run a REXX command processor to process SYSCALL, Shell, TSO and REXX commands. The command is supported from TSO and the OMVS shell.
SOEDIT	Edit UNIX files in SU mode using OEDIT. The command is supported from TSO and the OMVS shell.
SU	Run UNIX-related commands in SU mode from the TSO foreground or TSO batch.
“SU -” support	Support full shell initialization for the Standard and the C Shell environment when switching to another userid using the `su` command.
SWSU	Switch to SU mode (temporary) or just run one command with superuser authority in a USS shell environment.

Note: The “su -” support has been added for the `su` command in z/OS V1R5. The function provided with the USSTools package is only needed for elder z/OS releases.

Examples

In this section we provide some examples using USSTools commands. Furthermore, we add here some comments on this package.

Editing UNIX files and working in the OMVS shell

When working in a UNIX shell environment, you have two main choices regarding code page setting. One possibility is to change the UNIX code page to the terminal emulation code page you use. This removes all the problems with character input and display when entering commands in the OMVS shell or editing files from TSO or the OMVS shell.

However, all the standard shell scripts supplied with z/OS and other products installed need to be converted to your emulation code page. Furthermore, it is recommended to modify some UNIX tools to reflect the changed code page.

The other possibility is to keep the standard UNIX code page (IBM-1047) and live with the problems caused by the differences between this and your emulation code page. In our opinion, this second choice is the better solution.

The editing commands provided with this package (all emulation code pages) and the conversion tables available for OMVS (German and US code page) are a good means to overcome these small restrictions and problems.

Running UNIX commands in superuser mode (SU mode)

When running UNIX commands in SU mode as a BPX.SUPERUSER, you need to switch to SU mode first. In most situations it is enough to switch the effective UID to zero. And in the TSO foreground this was the standard way to do this (for example, the ISHELL command did it that way until z/OS V1.3).

However, there are several UNIX commands, like **find**, **pax**, and **test**, that test the user's authority not based on the effective but the real UID set. And the result is that these commands fail in several situations although the process is running with an effective UID of zero.

Note: This behavior of UNIX commands may be changed in the future, but is still this way at the moment. This fact has therefore to be taken into account.

The utility SU just switches the effective UID to zero while RXSUSH (REXX SU shell) supports running with a real and effective UID set to zero, if you run at level z/OS V1.3 or above.

Note: The interface that makes it possible to do this in a REXX procedure (for example, in the TSO foreground) was introduced with z/OS V1.3. The ISHELL command probably was the first application that exploited this new function.

In order to demonstrate the difference, we provide two samples. In Figure 10-8 we show a couple of commands and the results when using the command **su rxshell**, including the failing **find** command.

```
tso su rxshell
 Enter command input data or "exit" to exit processing:
id+
 uid=888(HERING) gid=1(IBMDE#01) euid=0(STCUSER) groups=0(SYS1)
 Enter command input data or "exit" to exit processing:
ls -d test.sumode+
 drwx------         2 STCUSER  SYS1        8192 Oct 19 17:42 test.sumode
 Enter command input data or "exit" to exit processing:
ls -E test.sumode+
 total 16
 -rw-r--r--  --s-  1 STCUSER  SYS1          26 Oct 19 17:42 testfile.a
 -rw-r--r--  --s-  1 STCUSER  SYS1          26 Oct 19 17:42 testfile.b
 Enter command input data or "exit" to exit processing:
cat test.sumode/testfile.a+
 aaaaaaaaaaaaaaaaaaaaaaaaa
 Enter command input data or "exit" to exit processing:
find test.sumode+
 test.sumode
 find: FSUM6512 unable to access "test.sumode": EDC5111I Permission denied.
 *** non-zero return code: Rc(1).
 Enter command input data or "exit" to exit processing:
exit
```

Figure 10-8 Failing UNIX find command if only effective UID set to zero is used

Figure 10-9 on page 393 shows the successful results if **rxsush** is used instead.

```
tso rxsush
UID setting switched to 0...
Enter command input data or "exit" to exit processing:
id+
 uid=0(STCUSER) gid=1(IBMDE#01) groups=0(SYS1)
 Enter command input data or "exit" to exit processing:
cd /u/hering && find test.sumode+
 test.sumode
 test.sumode/testfile.a
 test.sumode/testfile.b
 Enter command input data or "exit" to exit processing:
exit
```

Figure 10-9 Successful find results when real and effective UID are set to zero

For more information about USSTools, see the documentation on the UNIX tools Web site (under Tools and Toys) or get the package together with the description and install it as suggested.

10.2 REXX functions and interfaces

This chapter briefly lists and describes the REXX functions and some further interfaces.

The z/OS UNIX REXX functions extend the REXX language on z/OS for z/OS UNIX tasks. With the exception of bpxwunix() and syscalls(), these functions must be run in a z/OS UNIX environment.

The z/OS UNIX REXX functions include functions for standard REXX I/O, and for accessing common file services and environment variables.

Note: All of the functions are fully enabled for large files (>2 GB).

All numbers that are used as input to the functions must be integers. The default precision for REXX is 9 digits. If arithmetic is used on large numbers, be sure to change your precision appropriately, using the NUMERIC DIGITS statement.

The REXX functions have been available as a tools package from the UNIX Tools disk on the Internet for a long time. Beginning with z/OS V1.4, they are now added to the system and officially supported.

Table 10-5 New REXX functions

REXX function	Explanation
bpxwunix()	The bpxwunix() function runs the shell by passing a single command similar to the **sh -c** command. It does not run a login shell.

REXX function	Explanation
bpxwdyn()	BPXWDYN is a text interface to a subset of the SVC 99 (dynamic allocation) and SVC 109 (dynamic output) services. It supports data set allocation, unallocation, concatenation, and adding and deleting output descriptors. BPXWDYN is designed to be called from REXX, but may be called from several programming languages, including Assembler, C, and PL/I. This interface makes dynamic allocation and dynamic output services easily accessible to programs running outside of a TSO environment; however, this interface also functions in a TSO environment.
charin()	Returns a string of up to *length* characters read from the stream specified by name. The location for the next read is the current location increased by the number of characters returned. This function does no editing of the data.
charout()	Returns the number of characters remaining after an attempt to write a string to the stream specified by name. The location for the next write is the current location increased by the number of characters returned.
chars()	Returns the number of characters remaining in the input stream specified by name. For persistent streams, this is the number of characters between the current read location and the end of the stream. If the stream was created by the stream **`popen`** command, **`chars()`**—while the process is active or bytes remain in the stream—returns either 1 or the number of bytes in the stream. After the process has terminated and the stream is empty, **`chars()`** returns a value of 0.
chmod()	Changes the mode for the specified pathname. It returns 0 if the mode for the specified pathname is changed; otherwise, it returns the system call error number.
convd2e()	Converts timestamp to POSIX epoch time, and returns the time in seconds past the POSIX epoch (1/1/1970).
directory()	Returns the full pathname to the current directory, first changing it to *newdirectory* if the argument is supplied and you have access to that directory.
environment()	Queries and alters environment variables. The stem __environment. is not altered through this service. That stem contains the environment variables on entry to the REXX program, and is available for your use. Alterations of the environment are used on subsequent calls to the stream **`popen`** command and ADDRESS SH.
exists()	Returns the full pathname for the specified file. If the file does not exist, the function returns a null string.
getpass()	Prints prompt on the controlling TTY and reads and returns one line of input with terminal echo suppressed.
linein()	Returns one line or no lines from the stream specified in name, and sets the location for the next read to the beginning of the next line. The data is assumed to be text. The newline character is the line delimiter and is not returned. A null string is returned if no line is returned; this appears exactly the same as a null line in the file. Use **`chars()`** or **`lines()`** to determine if you are at the end of a file. Use **`stream()`** to determine if there is an error condition on the stream.

REXX function	Explanation
lineout()	Returns 1 line or 0 lines that are remaining to be written after an attempt to write a string to the stream specified by name. A newline character is written following the string. If an error occurs on the write, some data may be written to the stream, and the function returns the value 1.
lines()	Returns 1 if data remains in the stream; otherwise it returns 0. Programs should check for a value of 0 or nonzero.
outtrap()	Enables or disables the trapping of output from commands run using ADDRESS TSO, and returns the name of the variable in which trapped output is stored. If trapping is off, the word OFF is returned. Note that outtrap does not trap output for ADDRESS SH or any other command environment besides TSO. To trap shell command output, see “Running UNIX commands in TSO batch” on page 398.
procinfo()	Retrieves information about one or more processes.
rexxopt()	Sets, resets, or queries z/OS UNIX REXX options.
sleep()	Places the process in a signals-enabled wait, and returns after the wait expires. If a signal interrupts the wait, the function returns the number of seconds remaining for the wait, otherwise it returns 0.
stream()	Returns the state of the stream or the result of the command.
submit()	Submits a job to the primary subsystem (JES), returning the job ID of the submitted job.
syscalls()	Establishes the SYSCALL environment or ends it; or establishes or deletes the signal interface routine (SIR).

In 10.3, “Programming examples for REXX” on page 395 we provide some examples. See *z/OS Using REXX and z/OS UNIX System Services,* SA22-7806 for more detailed information.

10.3 Programming examples for REXX

In this section we show examples using the REXX functions and also other programming interfaces that are available.

Determine whether z/OS UNIX works in sharing mode

Figure 10-10 shows a sample REXX script that provides information on whether z/OS UNIX works with SYSPLEX(YES).

```
/* REXX -----------------------------------------------------------------*/
/* function : checks wether z/OS UNIX works with SYSPLEX(YES)         */
/*----------------------------------------------------------------------*/
 numeric digits 12
 cvt=c2x(storage(10,4))                   /* get asdress of cvt    */
 ecvt=c2x(storage(d2x(x2d(cvt)+140),4))   /* get asdress of ecvt   */
 ocvt=c2x(storage(d2x(x2d(ecvt)+240),4))
 oext=c2x(storage(d2x(x2d(ocvt)+12),4))
 sysplexflag=storage(d2x(x2d(oext)+8),1)

 if bitand(sysplexflag ,'20'x ) = '20'x
   then say 'z/OS UNIX works with file sharing.'
   else say 'z/OS UNIX works without file sharing.'
exit
```

Figure 10-10 Sample for getting information about file system sharing

Accessing the TCP/IP stack

Figure 10-11 on page 397 shows a sample that demonstrates how to use REXX functions to connect to the local TCP/IP stack. We use the REXX IP API functions shown in Table 10-6. For further information, see *z/OS Communications Server IP Application Programming Interface Guide,* SC31-8788.

Table 10-6 Used REXX functions

Function	Description
Initialize	Initialize and connect to the local TCP/IP stack.
Version	Get the TCP/IP version that is used.
GetHostName	Determine the local host name.
GetHostID	Provide the local TCP/IP home address.
GetDomainName	Get the domain that is valid for the local host.
Resolve	Resolve a TCP/IP address from a given host name.
Terminate	Disconnects from the TCP/IP stack.

```
/* REXX --------------------------------------------------------------*/
/* function : connects to tcpip stack                                 */
/*--------------------------------------------------------------------*/
 call syscalls 'ON'

 tcprc = Socket('Initialize','TCPIP')
 if (subword(tcprc,1,1) = 0)
    then say 'connect successful to stc' subword(tcprc,4,1)
    else do
     say 'Connecting to tcpip stack failed' subword(tcprc,1,1)
     return
    end
tcprc = Socket('Version')
 if (subword(tcprc,1,1) = 0)
    then say 'tcpip stack version is' subword(tcprc,4,1)
    else do
     say 'getting tcpip version failed' subword(tcprc,1,1)
     return
    end
 tcprc = Socket('GetHostName')
 if (subword(tcprc,1,1) = 0)
    then say 'local host name' subword(tcprc,2,1)
    else do
     say 'getting host name failed' subword(tcprc,1,1)
     return
    end
tcprc = Socket('GetHostID')
 if (subword(tcprc,1,1) = 0)
    then say 'local host id' subword(tcprc,2,1)
    else do
     say 'getting host id failed' subword(tcprc,1,1)
     return
    end
 tcprc = Socket('GetDomainName')
 if (subword(tcprc,1,1) = 0)
    then say 'local domain' subword(tcprc,2,1)
    else do
     say 'getting damain failed' subword(tcprc,1,1)
     return
    end
dnsdest = 'wtsc65.itso.ibm.com'
 tcprc = Socket('Resolve',dnsdest)
 if (subword(tcprc,1,1) = 0)
    then say 'dns resolv' subword(tcprc,2,1)
    else do
     say 'dns resolv failed' subword(tcprc,1,1)
     return
    end
 tcprc = Socket('Terminate','TCPIP')
 if (subword(tcprc,1,1) = 0)
    then say 'disconnect from tcpip successfull'
    else say 'disconnect from tcpip failed' subword(tcprc,2,1)
exit
```

Figure 10-11   Sample REXX to connect to the TCP/IP stack

Running UNIX commands in TSO batch

This example demonstrates how to use the REXX function Bpxwunix() to run UNIX commands and directly get the output to SYSOUT. Figure 10-12, Figure 10-13 and Figure 10-14 show a small REXX, the TSO JC, and the output provided with SYSTSPRT.

```
/* REXX ***************************************************************/
/*    Procedure: USSCMD                                               */
/*  Description: Run USS shell command                                */
/*               Property of IBM  (C) Copyright IBM Corp. 2003        */
/*    Format is: usscmd uss_cmd                                       */
/**********************************************************************/

Trace O
Parse Arg uss_cmd
Call Bpxwunix uss_cmd,, "DD:SYSTSPRT"
If result<>0 Then Say "Rc("!!result!!")"
Exit result
```

Figure 10-12 REXX to run a USS command

```
//UNIXJOB  JOB ,'TSOBATCH Job',NOTIFY=&SYSUID.,REGION=0M
//* -------------------------------------------------------------------
//TSOBATCH EXEC PGM=IKJEFT1A
//SYSEXEC  DD DSNAME=&SYSUID..UNIX.REXX.EXEC,DISP=SHR
//SYSTSPRT DD SYSOUT=*,LRECL=256,RECFM=V
//SYSTSIN  DD DATA,DLM=##
usscmd pwd;id;env
usscmd lxxxx
usscmd ls -E .
##
//* -------------------------------------------------------------------
```

Figure 10-13 TSO JCL to run UNIX commands

```
READY
usscmd pwd;id;env
/u/hering
uid=888(HERING) gid=2(SYS1) groups=1047(USSTEST)
_=/bin/env
READY
usscmd lxxxx
Rc(127)
```

Figure 10-14 Output provided with SYSTSPRT

Information about UNIX processes

The REXX function procinfo() makes it possible to retrieve almost all desirable information about a specific process or all UNIX processes. If the utility PSINFO (available with the USSTools package, described in 10.1.5, "USSTools" on page 390) is used from a UNIX shell to get information about a specific process by providing the process ID, procinfo() is used to get some of the important information. Figure 10-15 on page 399 shows a sample of this.

```
#> ps -e | grep inetd
         5 ?        0:00 /usr/sbin/inetd
#> psinfo -p 5
Userid   : STCUSER
Jobname  : INETD7
ASID     : 28("1C"x)
TTY      : ?
Threads  : 1
PID      : 5
PPID     : 1
EUID     : 0
RUID     : 0
SUID     : 0
EGID     : 0
RGID     : 0
SGID     : 0
Size     : 442368
Starttime: 2003-10-10 08:58:40 (GMT)
Cmdline  : /usr/sbin/inetd /etc/inetd.conf
#>
```

Figure 10-15 Displaying information about a specific UNIX process

If PSINFO is run without parameters you a description of the parameters and the output data. Figure 10-16 shows the excerpts that explain the fields shown in Figure 10-15.

```
#> psinfo
...
Userid    - Process MVS userid
Jobname   - Process jobname
ASID      - Address space number
TTY       - Controlling TTY or ?
Threads   - Number of threads
PID       - Process ID
PPID      - Parent process ID
EUID      - Effective UID
RUID      - Real UID
SUID      - Saved set UID
EGID      - Effective GID
RGID      - Real GID
SGID      - Saved set GID
Size      - Region size
Starttime - Process start time (GMT)
Cmdline   - Process start command

#>
```

Figure 10-16 Description of the information shown with PSINFO for a specific process

Retrieving the file size for large files

The utility program COPYTREE that was available from the UNIX Tools disk is now provided officially since z/OS V1.3. It is used to copy UNIX file structures or to check for file system consistency. In order to access the exact size value for a file of 2 GB or above, it uses the stream function because the SYSCALL interface commands like **`stat`** only provide precise values up to 2 GB. Higher values are given rounded up to MB.

Figure 10-17 shows an excerpt of COPYTREE testing whether the SYSCALL size information is in MB only or not. If the answer is YES for a UNIX environment, the stream() function is used instead, while for TSO, using this type of implementation, there is nothing left to do because the stream() function is not supported in a TSO environment.

```
...
if datatype(paths.i.st_size,'W')=0 then
     if tso then
        do
        say 'File to large for TSO environment:' src
        sum.s_isreg.1=sum.s_isreg.1+1
        return
        end
      else
        paths.i.st_size=stream(src,'c','size')
...
```

Figure 10-17 Excerpt from the utility program COPYTREE

Running TSO commands

The TSO command environment (Address TSO) can be used from a z/OS UNIX REXX environment, and is initialized with either:

```
Address TSO
```

or

```
Address TSO command
```

In this statement, `command` may be any TSO/E command, CLIST, or REXX exec that can run in a batch TSO TMP (terminal monitor program). And this TSO TMP is running in a separate address space and process from your REXX program.

It does not provide you with the capability to use TSO commands to affect your REXX environment, or to have REXX statements or other host command environments affect your TSO process. So both environments are completely separated.

By default, all command output is directed to your REXX process's standard output stream. You can use the **`outtrap()`** function to trap command output in variables.

Note: Before z/OS V1.4, you could use only the **`tso`** shell command to run TSO commands. This command does not support authorized TSO commands. To do that in a pre-z/OS V1.4 system, there is a tsocmd utility available from the UNIX Tools disk.

Figure 10-18 on page 401 shows two commands addressed to TSO together with the results. File tsoc is a REXX script that uses Address TSO to execute TSO commands. It takes the TSOALLOC variable structure (similar to the shell command **`tso`**) and allocates these libraries in share mode.

This makes it possible to use **`cn`**, which is a REXX procedure simply by its name because it is allocated in the SYSEXEC library chain in both situations. Because **`cn`** contains authorized TSO commands, the standard shell command **`tso`** fails.

```
HERING:/u/hering:$> tso -t "cn d omvs"
cn d omvs
IKJ56652I You attempted to run an authorized command or program.  This is not su
pported under the Dynamic TSO Environment.
IKJ55308I THE CONSOLE COMMAND HAS TERMINATED.+
IKJ55308I THE CONSOLE COMMAND DETECTED INTERNAL ERROR 20 DURING PROCESSING.
IKJ55323I GETMSG PROCESSING HAS TERMINATED.  A CONSOLE SESSION IS NOT ACTIVE.
IKJ55308I THE CONSOLE COMMAND HAS TERMINATED.+
IKJ55308I THE CONSOLE COMMAND DETECTED INTERNAL ERROR 20 DURING PROCESSING.
IKJ56652I You attempted to run an authorized command or program.  This is not su
pported under the Dynamic TSO Environment.
HERING:/u/hering:$> tsoc cn d omvs
cn d omvs
 BPXO042I 10.24.12 DISPLAY OMVS 628
 OMVS     000F ACTIVE          OMVS=(4A)
HERING:/u/hering:$>
```

Figure 10-18 Running TSO commands from a UNIX shell environment

The REXX procedure tsoc used in the sample of Figure 10-18 is shown in Figure 10-19.

```
/* REXX **************************************************************/
/*    Procedure: tsoc                                                */
/*  Description: Run a TSO command in TMP TSO                        */
/*               Property of IBM  (C) Copyright IBM Corp. 2003       */
/*    Format is: tsoc tso_cmd                                        */
/*********************************************************************/

Trace O
Parse Arg tso_cmd
If tso_cmd="" Then Do
  Say "Please append a TSO command."
  Exit 2
End

tso_libs = Environment("TSOALLOC")
Do Forever
  Parse Var tso_libs dd_name ":" tso_libs
  If dd_name="" Then Iterate
  dd_libstr = Translate(Environment(Strip(dd_name)))
  dd_libraries = ""
  Do Forever
    Parse Var dd_libstr dd_lib ":" dd_libstr
    If dd_lib="" Then Iterate
    dd_libraries = dd_libraries "'"||Strip(dd_lib)||"'"
    If dd_libstr="" Then Leave
  End
  Address TSO "ALLOCATE DATASET("||Strip(dd_libraries)||") DDNAME("||,
    dd_name||") SHR REUSE"
  If tso_libs="" Then Leave
End
Say tso_cmd
Address TSO tso_cmd
Exit rc
```

Figure 10-19 Sample REXX tsoc

10.4 Programming example for C

Example 10-1 shows a C program that uses C API functions for z/OS UNIX. This example reads the current mounted file systems with the function **w_getmntent**. Furthermore, we use the z/OS UNIX API function **w_statfs** to determine details about the mounted file system and some other functions.

Example 10-1 Sample C source

```
 /*******************************************************************
  *                                                                 *
  * project      : ITSO redbook SG24-7035                           *
  *                                                                 *
  * function     : checks filesystems utilization                   *
  * requirements :                                                  *
  *                                                                 *
  * history      : 10.07.2003 Lutz       first creation             *
  *                                                                 *
  *******************************************************************/

  #define  _OPEN_SYS
  #include <stdio.h>
  #include <stdarg.h>
  #include <errno.h>
  #include <string.h>
  #include <time.h>
  #include <sys/mntent.h>
  #include <sys/statfs.h>
  #include <sys/utsname.h>
 /*******************************************************************
  * get systinformations (date/time/sysname)                        *
  *******************************************************************/
 void getenvinfo(char *systemname,char *systemdate,char *systemdate2)
  {
   struct utsname sysinfo;
   time_t ltime;
   struct tm *newtime;
/* get sysid from z/OS UNIX */
    if (uname(&sysinfo) < 0)
      memcpy( systemname , "unknown", 8);
    else
     {
      memcpy( systemname , sysinfo.nodename, 5);
     }
/* get system date and time    */
   time(&ltime);
   memcpy( systemdate, ctime(&ltime),25);
   newtime = localtime(&ltime);
   memcpy( systemdate2, asctime(newtime),25);
  }
 /*******************************************************************
  * get filesystems total/uesd blocks                               *
  *******************************************************************/
 void getFSinfo( char fs[    ] )
  {
  struct w_statfs buf;
  if (w_statfs(fs, &buf, sizeof(buf)) == -1)
    perror("error getting filesystem information for %s\n");
  else
   {
```

```
   printf("%10.1fMB %10.1fMB %5.1f%% \n",
    ((double)buf.statfs_used_space  * buf.statfs_blksize/1024/1024),
    ((double)buf.statfs_total_space * buf.statfs_blksize/1024/1024),
    ((((double)buf.statfs_used_space  * buf.statfs_blksize)*100)/
      ((double)buf.statfs_total_space * buf.statfs_blksize)));
  }
 }
/*********************************************************************
 * main program                                                      *
 *********************************************************************/
 int main( int argc, char * argv[] )
 {
   int entries, entry;
   char linie[ 100 ];
   char s1[ 30 ];
   char s2[ 30 ];
   char s3[ 30 ];
   struct
    {
     struct w_mnth   header;
     struct w_mntent mount_table[ 10 ];
    } work_area; char  *parm_ptr;

   strcpy( linie, "_");
   for (entry=0; entry<72; entry++)
    {
     strcat( linie, "_" );
    };

   getenvinfo( s1, s2, s3 );
   printf("CHKFS for ITSO Redbook SG 24-7035 \n");
   printf("running on system %s at %s %s",s1, s2, s3);
   printf("%-40s %12s %12s %6s\n","filesystem","used","total","Util");
   printf("%s\n",linie);
   memset(&work_area, 0x00, sizeof(work_area)); /* clear */ do
    {
     if ((entries = w_getmntent((char *) &work_area,
                                sizeof(work_area))) == -1)
       perror("error getting fs-info");

     else for (entry=0; entry<entries; entry++)
      {
       printf("%-40s ",
       /*     work_area.mount_table[ entry 1.mnt_fsname, */
              work_area.mount_table[ entry ].mnt_mountpoint);
       getFSinfo( work_area.mount_table[ entry ].mnt_fsname );
       if (work_area.mount_table[ entry ].mnt_parmoffset |= 0)
        { parm_ptr = ((char *)&(work_area.mount_table[ entry ] )) +
                    work_area.mount_table[ entry ].mnt_parmoffset;
         printf(" mount parameter is %s",*parm_ptr);
        }
      }
    } while (entries > 0);
  return;
 }
```

To create the load module, you can use the c89 command line compiler. Our compiler call is as follows:

```
c89 -O  -o chkfs chkfs.c
```

For special compiling we recommend the creation of a make file.

Notes:

- Since z/OS 1.3, the c89 compiler invokes, per default, the z/OS C/C++ compiler CBDBDRV changes.
- Since z/OS 1.3, the default linker option COMPAT has changed to PM4. This means that modules linked into HFS or PDSE data sets do not run under z/OS 1.2 and prior. You should use the link option COMPAT=PM3 for compatibility with prior releases. Furthermore, you can use the _C89_PVERSION variable to control the kind of load module.

Figure 10-20 shows the output of the chkfs program. It shows a list of mounted file systems on system SC64. The z/OS UNIX API function **w_statfs** provides still more information about a file system that you could use for your own requirements.

```
CHKFS for ITSO Redbook SG 24-7035
running on system SC64 at Thu Jul 10 15:09:27 2003
 Thu Jul 10 15:09:27 2003
filesystem                                    used        total   Util
__________________________________________________________________________
/SC63/web/pki6a                              0.4MB       21.1MB   1.8%
/SC63/web/pki6                               0.4MB       21.1MB   1.8%
/pp/db2olap                                 68.0MB       80.1MB  84.9%
/SC63/web/pki5a                              0.4MB       21.1MB   1.8%
/pp/db2nx                                   18.5MB       21.0MB  88.0%
/SC63/web/pki5                               0.4MB       21.1MB   1.8%
/pp/netdata                                 10.5MB       19.5MB  53.7%
/SC63/web/pki4a                              0.4MB       21.1MB   1.8%
/pp/DWC                                     21.6MB       28.1MB  76.8%
/SC63/web/pki4                               0.4MB       21.1MB   1.8%
/SC63/web/pki2a                              1.4MB       21.1MB   6.8%
/SC63/web/pki2                               1.3MB       21.1MB   6.3%
/SC63/var                                    0.8MB        6.3MB  13.2%
/SC63/etc                                    3.3MB       37.3MB   8.8%
/SC63/dev                                    0.0MB        7.0MB   0.5%
/Z04RA1                                   2130.2MB     2134.0MB  99.8%
/SC63                                        1.1MB        1.4MB  78.6%
/                                            4.5MB        7.8MB  57.3%
```

Figure 10-20 Output from chkfs

Also, you can link the chkfs module as a z/OS load module to run without BPXBATCH or BPXBATSL. Figure 10-21 on page 405 shows sample JCL for linking the z/OS module. You can also see a call of the linked program.

```
//LUTZLNK  JOB LK,MSGCLASS=X,CLASS=D,REGION=32M,
//             NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//LKED     EXEC PGM=IEWL,
//         PARM='NCAL,LIST,RENT,LET'
//SYSLMOD  DD DISP=SHR,DSN=LUTZ.LOAD
//CHKFS    DD PATH='/u/lutz/chkfs',PATHOPTS=(ORDONLY)
//SYSPRINT DD SYSOUT=*
//SYSLIN   DD *
  INCLUDE CHKFS
  NAME  CHKFS(R)
/*
//GO       EXEC PGM=CHKFS
//STEPLIB  DD DISP=SHR,DSN=LUTZ.LOAD
//SYSPRINT DD SYSOUT=*
//
```

Figure 10-21 Sample link JCL

10.5 Using BPX.JOBNAME for USS processes

The environment variable _BPX_JOBNAME is used by the exec() callable service to change the jobname of the new process image. The jobname change is allowed only if the invoker has appropriate privileges and is running in a space created by fork(). If these conditions are not met, the environment variable is ignored. You can specify a string of one to eight alphanumeric characters. Incorrect specifications are ignored.

Appropriate privileges for setting up the jobname include either superuser authority or READ permission to the BPX.JOBNAME FACILITY class profile.

10.5.1 Using _BPX_JOBNAME in /etc/rc

Beginning with OS/390 V2R6, setting of _BPX_JOBNAME was only accepted from shell environments if the command was started in the background. Because the daemon processes may be started from /etc/rc and the process running this script ends immediately after the script is completed, a sleep statement had to be added near the end of the script. Otherwise the daemon processes did not have enough time to protect themselves against a hangup signal when the parent process ended.

Beginning with OS/390V2R7, this "&" at the end of the daemon startup commands is no longer needed to make the new jobname work. So, instead of using the commands shown in Figure 10-22 in /etc/rc, you can use the lines shown in Figure 10-23 on page 406.

```
# Start the INET daemon
_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf &
# Start the SYSLOG daemon
_BPX_JOBNAME='SYSLOGD' /usr/sbin/syslogd -f /etc/syslog.conf &
# Start the CRON daemon
_BPX_JOBNAME='CRON' /usr/sbin/cron &
sleep 5
echo /etc/rc script executed, `date`
```

Figure 10-22 /etc/rc with daemons started in the background

```
# Start the INET daemon
_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf
# Start the SYSLOG daemon
_BPX_JOBNAME='SYSLOGD' /usr/sbin/syslogd -f /etc/syslog.conf
# Start the CRON daemon
_BPX_JOBNAME='CRON' /usr/sbin/cron
echo /etc/rc script executed, `date`
```

Figure 10-23 /etc/rc with daemons started in the foreground

Figure 10-24 shows a sample using _BPX_JOBNAME to activate a specific jobname.

```
HERING:/u/hering:$> _BPX_JOBNAME=TEST sleep 180
```

Figure 10-24 UNIX command using envvar _BPX_JOBNAME

Figure 10-25 displays the processes related to the command entered to demonstrate that the new jobname is used.

```
d omvs,u=hering
 BPXO040I 01.08.48 DISPLAY OMVS 576
 OMVS     000D ACTIVE          OMVS=(LD,OE,FS)
 USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
...
 HERING   HERING   0041    83886671         1 MR---- 17.28.35      5.97
   LATCHWAITPID=         0 CMD=EXEC
 HERING   HERING   0041    50332436  83886671 1W---- 01.06.59      5.97
   LATCHWAITPID=         0 CMD=sh -L
 HERING   TEST     00FF    67109657  50332436 1SI--- 01.07.44       .00
   LATCHWAITPID=         0 CMD=sleep 180
...
```

Figure 10-25 OMVS display showing processes related to the UNIX command

10.6 C Shell

A shell is a program that provides a user interface. With a shell, users can type in commands and run programs with z/OS UNIX. The main function a shell performs is to read in from the terminal, run commands, and display the output of the commands back to the terminal.

The C Shell with all its commands and tools provides a real programmer's interface. It is specifically designed to have syntax similar to the C programming language.

Similar to the case of the standard she, several files are provided as samples and need to be customized.

The comparison between the standard shell and the C Shell in Table 10-7 on page 407 shows which files are related and what has to be customized from the system programmer's point of view.

Table 10-7 Comparison between the standard shell and the C Shell

	Default shell (/bin/sh)	C Shell (tcsh)
Run when a subshell is opened	N/A	/etc/csh.cshrc set path = (/bin)
Run when the shell is a login shell	/etc/profile export LANG=C export PATH=/bin	/etc/csh.login setenv LANG C
Run when a subshell is opened	$HOME/.setup	$HOME/.cshrc
Run when the shell is a login shell	$HOME/.profile	$HOME/.login
Run when a login shell is left	N/A	$HOME/.logout
"Delete" environment variable	unset TEMPVAR	unsetenv TEMPVAR

For "sh" the usage of a .setup file is controlled by setting the environment variable ENV. In the C Shell we have two files (/etc/csh.cshrc and $HOME/.cshrc) that are automatically run if they exist when a subshell is started.

All the C Shell scripts do not need to have the execute bit set; permission bits set to 644 are sufficient. They are run in the sequence listed in Table 10-7. First the system files in /etc are "sourced" and then the files in the user's home directory.

The names and syntax for several commands and also the way tasks are done may differ somewhat. For example, some environment variables are not set in the system login shell script but activated using the `set` command in a script run whenever a subshell is created.

Note: The user shell customization scripts begin with a period (.), so they do not usually appear when the user types the `ls` command. In order to see all files beginning with periods, the -a option is used with the `ls` command.

Fore more information, refer to *z/OS UNIX System Services Command Reference,* SA22-7802, *z/OS UNIX System Services Planning,* GA22-7800, and *z/OS UNIX System Services User's Guide,* SA22-7801.

10.7 Disabling use of SMF exit IEFUSI for UNIX processes

Starting with OS/390 V2R8, you can control the amount of resources that are consumed by individual OS/390 UNIX or z/OS UNIX users. This is supported by adding individual limits within the user's OMVS segment. This includes a setting for the user's UNIX processes or more precisely their address space storage.

Because you can now limit the storage available for a user's UNIX processes, there is no need for using IEFUSI in this situation and you need not manage complex coding. We now provide some information on how to disable exit IEFUSI for UNIX.

Append SUBSYS entries to parmlib members IEFSSNxx and SMFPRMxx. After a new IPL you may use OMVS entries in the SMF parmlib member to control exit calls for OMVS processes.

It seems that OMVS does not support the `setssi` command to dynamically activate a new subsystem.

When activating the new SMF settings (for example, set smf=xx) you can ignore the message

```
IEE968I NOTIFICATION OF SUBSYS OMVS FAILED -
IEE968I    SUBSYSTEM DOES NOT EXIST
```

This occurs because OMVS does not support the SSI Notify function.

Figure 10-26 provides an excerpt of an SSN parmlib member with subsystem OMVS included.

```
SUBSYS SUBNAME(JES2) /* JES2 AS PRIMARY SUBSYSTEM */
  PRIMARY(YES) START(NO)
SUBSYS SUBNAME(SMS) /* DEFINE SMS SUBSYSTEM */
  INITRTN(IGDSSIIN)
  INITPARM('ID=02,PROMPT=DISPLAY')
...
SUBSYS SUBNAME(RACF) INITRTN(IRRSSI00) INITPARM('#')
SUBSYS SUBNAME(OMVS)
```

Figure 10-26 IEFSSNxx parmlib member

Figure 10-27 shows some lines of an SMF parmlib member. Note that exit IEFUSI has been left out in the OMVS `SUBSYS` line, which prevents calling IEFUSI for creation of OMVS processes.

```
ACTIVE                          /* ACTIVE SMF RECORDING             */
...
SYS(NOTYPE(14:19,62:69,99),EXITS(IEFU83,IEFU84,IEFACTRT,
              IEFUSI,IEFUJI,IEFU29),NOINTERVAL,NODETAIL)
...
SUBSYS(OMVS,EXITS(IEFU83,IEFU84,IEFACTRT,IEFUJI,IEFU29))
```

Figure 10-27 SMFPRMxx parmlib member

10.8 USS hard links used with SMP/E

After completing the needed customization, the man pages can be displayed successfully. And each time you display pages for a specific command, they are copied to /var/man/C.

Figure 10-28 on page 409 shows a sequence of UNIX commands and output to demonstrate this.

```
HERING:/u/hering:$> ls -E /var/man/C
total 96
-rw-rw-rw-  --s-  1 HERING   VSMUNIX     6737 Apr  3 00:05 chmod.1.bpxa5mst
-rw-rw-rw-  --s-  1 STCUSER  VSMUNIX     4159 May  7 14:30 df.1.bpxa5mst
-rw-rw-rw-  --s-  1 STCUSER  VSMUNIX     2038 May  7 14:31 du.1.bpxa5mst
-rw-rw-rw-  --s-  1 STCUSER  VSMUNIX    21393 Apr  3 15:19 tar.1.bpxa5mst
-rw-rw-rw-  --s-  1 HERING   VSMUNIX     3163 Jul  2 09:42 ulimit.1.bpxa5mst
HERING:/u/hering:$> man -w pg
/usr/man/C/man1/bpxa5mst.book
HERING:/u/hering:$> man pg
  pg -- Display files interactively


  Format

  pg [-cefnst] [-p prompt] [- screen] [+line] [+/pattern/] [file ...]
...
HERING:/u/hering:$> ls -E /var/man/C
total 120
-rw-rw-rw-  --s-  1 HERING   VSMUNIX     6737 Apr  3 00:05 chmod.1.bpxa5mst
-rw-rw-rw-  --s-  1 STCUSER  VSMUNIX     4159 May  7 14:30 df.1.bpxa5mst
-rw-rw-rw-  --s-  1 STCUSER  VSMUNIX     2038 May  7 14:31 du.1.bpxa5mst
-rw-rw-rw-  --s-  1 HERING   VSMUNIX     8305 Jul  9 19:36 pg.1.bpxa5mst
-rw-rw-rw-  --s-  1 STCUSER  VSMUNIX    21393 Apr  3 15:19 tar.1.bpxa5mst
-rw-rw-rw-  --s-  1 HERING   VSMUNIX     3163 Jul  2 09:42 ulimit.1.bpxa5mst
HERING:/u/hering:$> man -w pg
/var/man/C/pg.1.bpxa5mst
HERING:/u/hering:$>
```

Figure 10-28 Sample man commands and files

When a UNIX file structure is installed using SMP/E, not just the UNIX file name is referenced, but also a simple MVS name qualifier with a maximum of eight uppercase characters is used that identifies the file. This means that you finally have two directory entries that refer to the same file. Entries pointing to the same file are called hard links.

We demonstrate this with the file /usr/man/C/man1/bpxa5mst.book that is referenced in Figure 10-28.

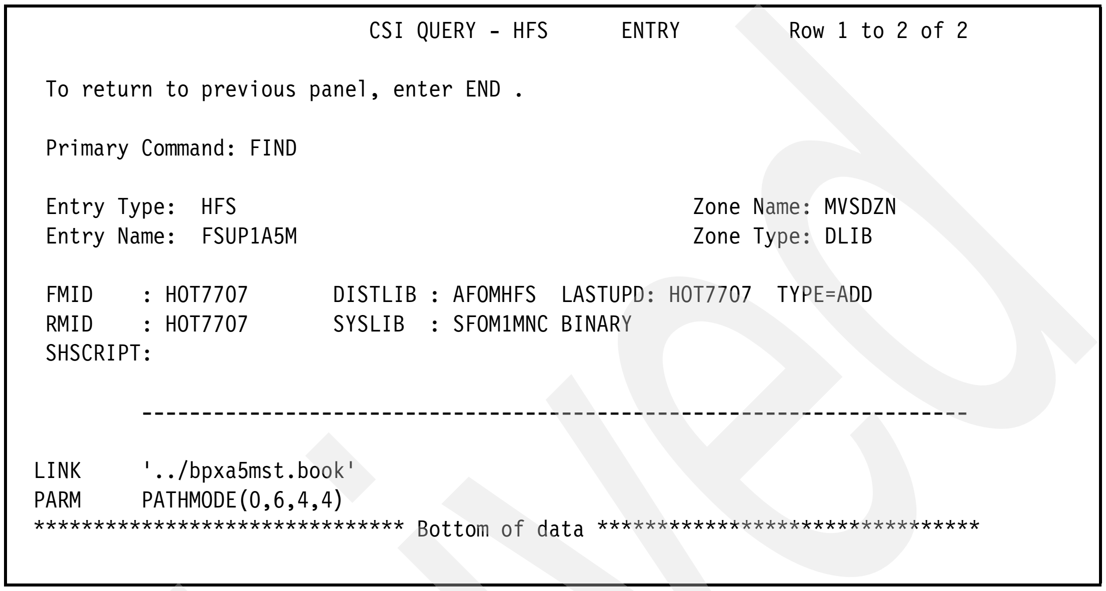
```
HERING:/u/hering:$> ls -Ei /usr/man/C/man1
total 3704
 8908 drwxr-xr-x        2 STCUSER  VSMUNIX     8192 May 30  2002 IBM
 8906 -rw-r--r--  --s-  2 STCUSER  VSMUNIX  1777664 May 30  2002 bpxa5mst.book
 8907 -rw-r--r--  --s-  2 STCUSER  VSMUNIX   110592 May 30  2002 ipeou01.book
HERING:/u/hering:$> ls -Ei /usr/man/C/man1/IBM
total 3688
 8907 -rw-r--r--  --s-  2 STCUSER  VSMUNIX   110592 May 30  2002 FOMOMANE
 8906 -rw-r--r--  --s-  2 STCUSER  VSMUNIX  1777664 May 30  2002 FSUP1A5M
HERING:/u/hering:$>
```

Figure 10-29 Listing files and hard links in the IBM subdirectory

The number displayed in column one of the ls output listing in Figure 10-29 is the inode of the file or directory. In a UNIX file system this number identifies a file or directory as it is used only once. And the inodes of files bpxa5mst.book and FSUP1A5M are identical. Therefore, FSUP1A5M is a hard link of bpxa5mst.book.

The number in column four is the link count. It provides the total number of hard links defined for a file. So these two files are the complete set.

Figure 10-30 shows the result of an SMP/E CSI query for FSUP1A5M, and the relation to bpxa5mst.book is obviously given.



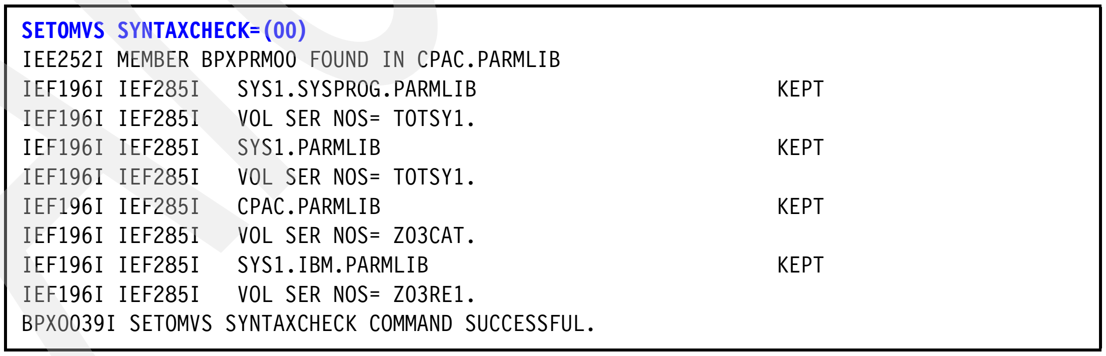
```
                              CSI QUERY - HFS      ENTRY        Row 1 to 2 of 2

   To return to previous panel, enter END .

   Primary Command: FIND

   Entry Type:  HFS                                       Zone Name: MVSDZN
   Entry Name:  FSUP1A5M                                  Zone Type: DLIB

   FMID     : HOT7707       DISTLIB : AFOMHFS  LASTUPD: HOT7707  TYPE=ADD
   RMID     : HOT7707       SYSLIB  : SFOM1MNC BINARY
   SHSCRIPT:

           ---------------------------------------------------------------------

  LINK     '../bpxa5mst.book'
  PARM     PATHMODE(0,6,4,4)
 ******************************* Bottom of data ********************************
```

Figure 10-30 SMP/E CSI query of HFS entry FSUP1A5M

10.9 OMVS syntax checker

Before doing an IPL, you can use the SETOMVS SYNTAXCHECK operator command to check the syntax of the BPXPRMxx member that you specify. It will not verify the validity of HFS data sets or mount points. Any syntax errors are sent to the hardcopy log.

Figure 10-31 shows successful syntax checking.



```
SETOMVS SYNTAXCHECK=(00)
IEE252I MEMBER BPXPRM00 FOUND IN CPAC.PARMLIB
IEF196I IEF285I   SYS1.SYSPROG.PARMLIB                         KEPT
IEF196I IEF285I   VOL SER NOS= TOTSY1.
IEF196I IEF285I   SYS1.PARMLIB                                 KEPT
IEF196I IEF285I   VOL SER NOS= TOTSY1.
IEF196I IEF285I   CPAC.PARMLIB                                 KEPT
IEF196I IEF285I   VOL SER NOS= Z03CAT.
IEF196I IEF285I   SYS1.IBM.PARMLIB                             KEPT
IEF196I IEF285I   VOL SER NOS= Z03RE1.
BPXO039I SETOMVS SYNTAXCHECK COMMAND SUCCESSFUL.
```

Figure 10-31 Successful syntax checking

In Figure 10-32 on page 411 we provide a sample of a check that finds syntax errors.

```
SETOMVS SYNTAXCHECK=(77)
IEE252I MEMBER BPXPRM77 FOUND IN CPAC.PARMLIB
ASA003I SYNTAX ERROR IN PARMLIB MEMBER=BPXPRM77 ON LINE 5,
POSITION 2: ILESYSTYPE WAS SEEN, WHERE ONE OF
(MAXPROCSYS MAXPROCUSER MAXUIDS MAXFILEPROC
MAXPTYS FILESYSTYPE ROOT MOUNT
CTRACE MAXTHREADTASKS)
WOULD BE CORRECT.
DETECTING MODULE IS BPXIPMX1. INPUT LINE:
 ILESYSTYPE TYPE(HFS)           /* Filesystem type HFS      */
ASA004I PARSING OF PARMLIB MEMBER=BPXPRM77
CONTINUED AT FILESYSTYPE, LINE 9.
DETECTING MODULE IS BPXIPMX1. INPUT LINE:
FILESYSTYPE TYPE(TFS)           /* Type of file system to   */
BPXO023I THE PARMLIB MEMBER BPXPRM77 CONTAINS SYNTAX ERRORS.
REFER TO HARD COPY LOG FOR MESSAGES.
IEF196I IEF285I   SYS1.SYSPROG.PARMLIB                         KEPT
IEF196I IEF285I   VOL SER NOS= TOTSY1.
IEF196I IEF285I   SYS1.PARMLIB                                 KEPT
IEF196I IEF285I   VOL SER NOS= TOTSY1.
IEF196I IEF285I   CPAC.PARMLIB                                 KEPT
IEF196I IEF285I   VOL SER NOS= Z03CAT.
IEF196I IEF285I   SYS1.IBM.PARMLIB                             KEPT
IEF196I IEF285I   VOL SER NOS= Z03RE1.
```

Figure 10-32 Syntax checking with errors

10.10 Storage limits for UNIX processes

When a new UNIX process is create, it inherits characteristics from the parent process or environment. This is true especially for limits like MAXFILEPROC, MAXFILESIZE, and MAXASSIZE. To demonstrate this we take the storage size as an example.

The userid that we use has a maximum storage limit set in its OMVS segment. This is shown in Figure 10-33.

```
lu hering noracf omvs
USER=HERING

OMVS INFORMATION
----------------
UID= 0000000888
HOME= /u/hering
PROGRAM= /bin/sh
CPUTIMEMAX= NONE
ASSIZEMAX= 0990000000
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE
```

Figure 10-33 Listing an OMVS segment with an ASSIZE limit set

Figure 10-34 on page 412 shows a sequence of commands to list or modify a user's current storage limit. The user opens an OMVS shell from TSO and the UNIX process inherits limits that are active for that TSO session.

```
HERING:/u/hering:$> query.storlimit
Maximum storage limit is:  990000000
Current storage limit is:   42967040
Current storage size is :    2473984
HERING:/u/hering:$> setstor.current 550000000
HERING:/u/hering:$> query.storlimit
Maximum storage limit is:  990000000
Current storage limit is:  550000000
Current storage size is :    2473984
HERING:/u/hering:$> setstor.current 440000000
HERING:/u/hering:$> query.storlimit
Maximum storage limit is:  990000000
Current storage limit is:  440000000
Current storage size is :    2473984
HERING:/u/hering:$> setstor.current 2073984
Syscall Service     = setrlimit (rlimit_as) st.
Syscall Return Code = 0
OMVS Return Value   = -1
OMVS Return Code    = 79
OMVS Reason Code    = BC6033F
OMVS Return Code Explanation - EINVAL: The parameter is incorrect
OMVS Reason Code Explanation - JrSoftBelowUsage: An attempt was made to lower a
soft limit below the current usage for the resource.
HERING:/u/hering:$> setstor.current 999000000
Syscall Service     = setrlimit (rlimit_as) st.
Syscall Return Code = 0
OMVS Return Value   = -1
OMVS Return Code    = 79
OMVS Reason Code    = BC6033E
OMVS Return Code Explanation - EINVAL: The parameter is incorrect
OMVS Reason Code Explanation - JrSoftExceedsHard: An attempt was made to raise a
 soft limit above its hard limit.
HERING:/u/hering:$>
```

Figure 10-34 Modifying the storage limit in an OMVS shell environment

Processes in the same address space share the storage available for UNIX processes, of course. If a limit is changed in one process it is also active for the other one. This can be easily demonstrated by opening a second OMVS shell and then switching between these two shell windows while modifying the current storage limit.

- Press PF2 (=SubCmd) and enter open.
- Use PF9 (=NextSess) to switch between these windows.

Note: The initial UNIX processes in OMVS shell sessions are located within the TSO address space.

Figure 10-35 on page 413 shows another situation. A non-superuser tries to set the current and the maximum limit to unlimited. This is only allowed in superuser mode.

```
HERING:/u/hering:$> query.storlimit
Maximum storage limit is:  990000000
Current storage limit is:  630000000
Current storage size is :     671744
HERING:/u/hering:$> setstor.unlimited
Syscall Service     = setrlimit (rlimit_as) st.
Syscall Return Code = 0
OMVS Return Value   = -1
OMVS Return Code    = 8B
OMVS Reason Code    = BC6033C
OMVS Return Code Explanation - EPERM: The operation is not permitted
OMVS Reason Code Explanation - JrRaiseHardLimit: An attempt was made to raise a
hard limit without superuser authority.
HERING:/u/hering:$> su
HERING:/u/hering:#> setstor.unlimited
HERING:/u/hering:#> query.storlimit
Maximum storage limit is: 2147483647
Current storage limit is: 2147483647
Current storage size is :    1376256
HERING:/u/hering:#> exit
HERING:/u/hering:$> query.storlimit
Maximum storage limit is:  990000000
Current storage limit is:  630000000
Current storage size is :     671744
HERING:/u/hering:$>
```

Figure 10-35 Trying to set the maximum limit to unlimited

The three commands used in these shell sessions are REXX shell scripts. The following three scripts are listed for your information.

REXX shell script query.storlimit

This script, shown in Figure 10-36 on page 414, provides current storage limits.

```
/* REXX ************************************************************/
/*    Procedure: query.storlimit                                   */
/*  Description: Provide current storage limits                    */
/*               Property of IBM  (C) Copyright IBM Corp. 1999, 2003  */
/*    Format is: query.storlimit                                   */
/*******************************************************************/

Trace O
Call Syscall_Cmd "getrlimit" rlimit_as "st."
Say "Maximum storage limit is:" Right(st.2,10)
Say "Current storage limit is:" Right(st.1,10)
Call Syscall_Cmd "getpid"
get_pid = retval
Call Syscall_Cmd "getpsent ps."
Do i=1 To ps.0
  If ps.i.ps_pid=get_pid Then Do
    Say "Current storage size is :" Right(ps.i.ps_size,10)
    Leave i
  End
End
Exit 0

Syscall_Cmd:
  Parse Arg syscall_cmd
  Address SYSCALL syscall_cmd
  not_OK = (rc<>0 | retval<0 | retval=0 & (errno<>0 | errnojr<>0))
  OK = (not_OK = 0)
  If not_OK Then Do
    Say "Syscall Service     =" syscall_cmd
    Say "Syscall Return Code =" rc
    Say "OMVS Return Value   =" retval
    Say "OMVS Return Code    =" errno
    Say "OMVS Reason Code    =" errnojr
    Exit 1
  End
Return
```

Figure 10-36 Script query.storlimit lists a user's storage limits

REXX shell script setstor.current

This script, shown in Figure 10-37 on page 415, sets the current storage limit.

```
/* REXX ***************************************************************/
/*    Procedure: setstor.current                                      */
/*  Description: Set the current storage limit                        */
/*               Property of IBM  (C) Copyright IBM Corp. 1999, 2003  */
/*    Format is: setstor.current new_curlimit                         */
/**********************************************************************/

Trace 0
Parse Source . . myname . . . . omvs .
myname = Substr(myname,Lastpos("/",myname)+1)
Parse Arg new_curlimit
If new_curlimit="" Then Do
  Say "To modify the current storage limit append a new value:"
  Say myname "new_curlimit"
  Exit 2
End

Call Syscall_Cmd "getrlimit (rlimit_as) st."
st.1 = new_curlimit
st.2 = st.2
Call Syscall_Cmd "setrlimit (rlimit_as) st."
Exit 0

Syscall_Cmd:
  Parse Arg syscall_cmd, exit_on_error
  exit_on_error = (exit_on_error="")
  Address SYSCALL syscall_cmd
  not_OK = (rc<>0 | retval<0 | retval=0 & (errno<>0 | errnojr<>0))
  OK = (not_OK = 0)
  If not_OK Then Do
    Say "Syscall Service     =" syscall_cmd
    Say "Syscall Return Code =" rc
    Say "OMVS Return Value   =" retval
Say "OMVS Return Code    =" errno
    Say "OMVS Reason Code    =" errnojr
    is_omvs_range = X2d(Left(Right(errnojr,8,"0"),4))<=X2d(20FF)
    syscall_rc = rc
    If errno<>"A3" & errno<>"A4" & is_omvs_range Then
      show_reason = 1
    Else
      show_reason = 0
    If syscall_rc=-21 Then
      Say "The first parameter is in error."
    Address SYSCALL "strerror" errno errnojr "err."
    If rc=0 & retval>=0 Then Do
      If err.se_errno<>"" Then
        Say "OMVS Return Code Explanation -" err.se_errno
      If show_reason & err.se_reason<>"" Then
      If show_reason & err.se_reason<>"" Then
        Say "OMVS Reason Code Explanation -" err.se_reason
    End
    If exit_on_error Then Exit 1
  End
Return not_OK
```

Figure 10-37 Script setstor.current modifies a user's current storage size

REXX shell script setstor.unlimited

This script, shown in Figure 10-38, sets the maximum storage limit to unlimited.

```
/* REXX **************************************************************/
/*    Procedure: setstor.unlimited                                  */
/*  Description: Set the maximum storage limit to unlimited         */
/*               Property of IBM  (C) Copyright IBM Corp. 1999, 2003  */
/*    Format is: setstor.unlimited                                  */
/*********************************************************************/

Trace 0
st.1 = rlim_infinity
st.2 = rlim_infinity
Call Syscall_Cmd "setrlimit (rlimit_as) st."
Exit 0

Syscall_Cmd:
  Parse Arg syscall_cmd, exit_on_error
  exit_on_error = (exit_on_error="")
Address SYSCALL syscall_cmd
  not_OK = (rc<>0 | retval<0 | retval=0 & (errno<>0 | errnojr<>0))
  OK = (not_OK = 0)
  If not_OK Then Do
    Say "Syscall Service     =" syscall_cmd
    Say "Syscall Return Code =" rc
    Say "OMVS Return Value   =" retval
    Say "OMVS Return Code    =" errno
    Say "OMVS Reason Code    =" errnojr
    is_omvs_range = X2d(Left(Right(errnojr,8,"0"),4))<=X2d(20FF)
    syscall_rc = rc
    If errno<>"A3" & errno<>"A4" & is_omvs_range Then
      show_reason = 1
    Else
      show_reason = 0
    Address SYSCALL "strerror" errno errnojr "err."
    If rc=0 & retval>=0 Then Do
      If err.se_errno<>"" Then
        Say "OMVS Return Code Explanation -" err.se_errno
      If show_reason & err.se_reason<>"" Then
        Say "OMVS Reason Code Explanation -" err.se_reason
    End
If exit_on_error Then Exit 1
  End
Return not_OK
```

Figure 10-38 Script setstor.unlimited tries to set current and maximum limit to unlimited

10.11 Using BPXCOPY to load files into the UNIX file structure

BPXCOPY allows you to copy a sequential data set or a partitioned data set or a PDSE member into a USS file system file. You can invoke BPXCOPY from JCL using **EXEC PGM=BPXCOPY** or through API functions in several situations.

BPXCOPY allows a BPX.SUPERUSER to act in superuser mode by automatically switching to SU mode during operation. This provides the means for SMP/E to allow a BPX.SUPERUSER to perform all install and maintenance tasks that need to be done in SU mode. SMP/E uses the API functions to work with BPXCOPY.

In Figure 10-39 we provide an excerpt from the USSTools install job (USSTools is described in “USSTools” on page 390). It copies a PDS member to the UNIX file structure and sets additional file attributes as needed for the file.

```
//UNIXJOB  JOB ,'Install USS Tools',NOTIFY=&SYSUID.
...
//  SET   USSTOOLS=&SYSUID..USSTOOLS             <== This PDS Data Set
...
//  SET   UNIXPATH='/u/bin'                      <== HFS Directory
...
//* -------------------------------------------------------------------
//* Copy member SWSU to the HFS structure
//* -------------------------------------------------------------------
//COPYSWSU EXEC PGM=BPXCOPY,
//  PARM='ELEMENT(BPXSWSU) TYPE(TEXT) LINK(''../swsu'')
//             PATHMODE(0,7,5,5) NOSHAREAS UID(0)'
//SYSUT1   DD DSN=&USSTOOLS.(SWSU),DISP=SHR
//SYSUT2   DD PATH='&UNIXPATH./IBM'
//SYSTSPRT DD SYSOUT=*
...
```

Figure 10-39 Using BPXCOPY in a job to copy a PDS member to the UNIX file structure

If the copy processing is successful you get a message to SYSOUT as shown in Figure 10-40.

```
BPXF150I MVS DATA SET WITH DDNAME SYSUT1 SUCCESSFULLY COPIED INTO TEXT HFS FILE
/u/bin/IBM/BPXSWSU.
```

Figure 10-40 Message shown in SYSOUT when copy processing is successful

10.12 Using the magic number

Most UNIX systems support a feature called the magic number (#!).The magic number is a numeric or string constant in a file that indicates the file name of the executable program to be run. When a script file starts with `#!`, the kernel invokes the specified filename as the script file interpreter.

Prior to OS/390 UNIX V2R8, the OS/390 UNIX kernel did not support the magic number, so it treated the magic number as a comment.

We demonstrate the usage of the magic number by running two scripts from the standard UNIX shell environment. In Figure 10-41 on page 418 the listing of the script shows that line 1 just is a comment line. The script is interpreted by /bin/sh therefore.

In the script output you see that envvar “shell” is not set. Furthermore, the syntax used in the script is obviously invalid for the SH shell.

```
$> cat bin/test.magic
## /bin/tcsh
echo "shell=" $shell
echo "SHELL=" $SHELL
if ( ${#argv} == 0 ) then
   echo "No arguments passed to this script"
else
   echo "At least one argument passed to this script"
endif
$> test.magic
shell=
SHELL= /bin/sh
test.magic 8: FSUM7332 syntax error: got EOF, expecting fi
$>
```

Figure 10-41 Running a shell script in the standard shell with invalid SH syntax

The only difference in the second example (Figure 10-42) is that the first line of the script is no longer a comment but starts with the magic number.

```
$> cat bin/test.magic
#! /bin/tcsh
echo "shell=" $shell
echo "SHELL=" $SHELL
if ( ${#argv} == 0 ) then
   echo "No arguments passed to this script"
else
   echo "At least one argument passed to this script"
endif
$> test.magic
shell= /bin/tcsh
SHELL= /bin/sh
No arguments passed to this script
$> test.magic xx yy
shell= /bin/tcsh
SHELL= /bin/sh
At least one argument passed to this script
$>
```

Figure 10-42 Running a C Shell script from the standard shell using the magic number

In this situation the USS kernel's spawn and exec services recognize /bin/tcsh as the program to be run. So the script is interpreted by the C Shell this time. You see that envvar “shell” is set (a further indication that /bin/tcsh is active). And in both calls you get reasonable output as the syntax is valid for the C Shell environment.

10.13 Enhanced ASCII functionality

The enhanced ASCII functionality allows z/OS UNIX System Services to deal with files that are in both ASCII and EBCDIC format. The enhanced ASCII support makes it easier to port applications developed on ASCII platforms to z/OS platforms by providing a limited ASCII-to-EBCDIC translation and vice versa. ASCII uses codepage ISO8859-1, while z/OS UNIX System Services by default uses the EBCDIC codepage IBM-1047. The enhanced ASCII functionality is limited to these two codepages.

The solution includes:

- Automatic codeset conversion (autoconversion)

  File tagging
- C/C++ compiler options
- Language Environment runtime library (RTL) enhancements

We recommend that you limit the enabling of autoconversion to the smallest environment possible. It is important to understand that file tagging and autoconversion are independent operations. You can tag files without enabling autoconversion and vice versa.

10.13.1 ASCII support overview

z/OS and the zSeries processor are an EBCDIC platform. Extended Binary-Coded-Decimal Interchange Code (EBCDIC) is an 8-bit code that gives 256 possible combinations and was originally developed for the IBM S/360. Today EBCDIC is used by the zSeries and iSeries™ (formerly AS/400®) of IBM servers. This means that the zSeries processors have programs that are compiled to handle EBCDIC data, EBCDIC encoding of characters, and devices that are configured for EBCDIC. UNIX applications on other platforms are encoded in the ASCII code set. This includes programs, literal strings within programs, and data.

Before this support, application programs that ran under z/OS UNIX System Services had to be compiled in EBCDIC format, and they expected data encoded in EBCDIC. The **`iconv`** command was available to convert files from ASCII to EBCDIC. Enhanced ASCII introduces automatic conversion of data between ASCII and EBCDIC.

If you implement the new enhanced ASCII support, the EBCDIC nature of the z/OS platform remains. However, if you compile your C program as ASCII, the EBCDIC nature of the z/OS platform can be partially hidden.

This support is limited to z/OS UNIX files. It does not apply to MVS data sets, even though they can be accessed by z/OS UNIX System Services.

New ASCII support

This support is introduced to help the porting of ASCII applications to the z/OS platform. Specifically, the support provides the ability to:

- Build ASCII-based applications by producing object code with ASCII string literals and character constants and a flag that identifies applications as ASCII or EBCDIC.
- Use Unicode-based wide characters (wchar_t) in ASCII-based applications.
- Transparently call native ASCII run-time library functions from ASCII-based applications.
- Process user-defined ASCII multi-byte code pages with user-supplied code set-related methods.
- Create ASCII-based local objects which allow processing of ASCII data natively at run-time.

There are several ways to read or write to a file. The callable services BPX1RED and BPX1WRT, which are the ones used when **`read()`**, **`write()`**, **`pread()`**, and **`pwrite()`** are issued, are the only services supported for automatic conversion. Only regular files, and named and unnamed pipes are supported. Socket and directory files are not supported.

10.14 Automatic conversion

Figure 10-43 shows a program that is reading or writing from and to an ASCII file. The program does not need to know that the file is in ASCII; the Logical File System (LFS) checks and finds that it is okay to perform the automatic conversion. Nothing indicates that an automatic conversion has occurred. You can use `Ctrace` to find it out.

Automatic conversion of files from ASCII to EBCDIC and vice versa is controlled globally by the AUTOCVT(ON) or (OFF) statement in the BPXPRMxx parmlib member. AUTOCVT can be overridden locally by the individual programs at thread level.

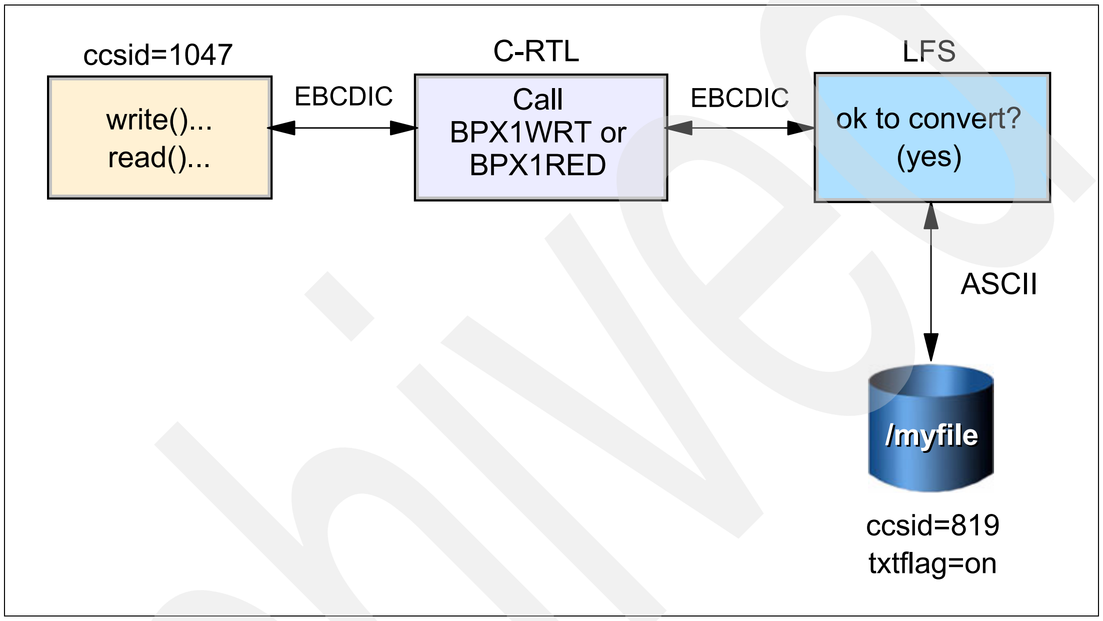


Figure 10-43 Automatic conversion of an ASCII file

The coded character set identifier (CCSID) for enhanced ASCII functionality is a 16-bit value, a number that represents a character set used by file tagging. It identifies the current character set of text strings within a program. This is stored in the file tag of new files or used for the automatic conversion of old files when autoconversion is in effect.

Note: If AUTOCVT is set to enable enhanced ASCII, the performance of your z/OS UNIX environment is affected because every read and write operation for a file must be checked. That means all HFS file systems and every file in the file system is checked to see if conversion is necessary. It is recommended that you use AUTOCVT(OFF). If you want to enable enhanced ASCII using another method, see 10.15, "File tagging" on page 422.

10.14.1 Autoconversion

The enhanced ASCII functionality provides a limited automatic conversion (autoconversion) from ASCII to EBCDIC and vice versa. There may be many reasons why you do not want to use autoconversion. We recommend that you limit the enabling of autoconversion to the smallest possible environment available.

Several conditions must be met before autoconversion is done:

- The autoconversion must be activated by any of the following:
  - A parameter in parmlib member BPXPRMxx
  - An environment variable

  – A runtime option

- The file must be tagged with txtflag on and contain a valid codeset.
- The program codeset must be different from the file tag codeset.

Programs and files

Automatic conversion is accomplished between programs and files that are tagged with different CCSIDs when a conversion table exists for that CCSID pair in the system.

Autoconversion is only supported between the codesets ISO8859-1 (ASCII) and IBM-1047 (z/OS UNIX System Services EBCDIC). These are the only codesets supported, no other converters exist.

Commands

Most commands that perform file I/O expect text data, so they allow autoconversion.

The autoconversion is controlled globally by the BPXPRMxx parmlib statement:

```
AUTOCVT(ON|OFF)
```

This statement activates or deactivates automatic conversion of text data files using CCSIDs for the program and its associated files. The CCSIDs are specified by the program or by setting the appropriate environment variables at run-time. The system AUTOCVT indicator set by this statement can be overridden by individual programs at the thread level. You can think of AUTOCVT as a controlling switch only for existing programs that do not explicitly establish their own conversion environment.

AUTOCVT(OFF) This option deactivates autoconversion, and is the default value.

AUTOCVT(ON) When this is set, every read and write operation for a file must be checked to see if conversion is necessary. Thus, there is a performance penalty involved, even if no conversion occurs. Therefore, we recommend that you keep AUTOCVT(OFF) and have each program enabled, if possible, for conversion. To override the AUTOCVT setting, use the compile or C run-time environment variables that control conversion or issue **`vacant()`** in your program. The two new **`vacant()`** subcommands, F_CONTROL_CVT and F_SETTAG, are supported as both C run-time library variables and as callable services.

You can use the SETOMVS or SET OMVS commands to change the value of AUTOCVT between ON and OFF. Changing this conversion mode does not affect conversion of opened files for which I/O has already started.

10.14.2 Scope of autoconversion

Automatic conversion can be controlled at different environmental levels.

At the highest level, AUTOCVT(ON) can be specified in the BPXPRMxx parmlib member to enable automatic conversion for the entire z/OS UNIX System Services environment.

At the lowest level, a program can use the new subcommand of **`fcntl()`**, F_CONTROL_CVT to enable autoconversion for a single open file or enable automatic conversion using an environment variable _ BPXK_AUTOCVT, or the FILETAG run-time option, or both. When either of these options is used, it overrides the AUTOCVT system setting.

Autoconversion can also be controlled individually by a single program with one of the following flags in the thread Thli control block:

- ThliCvtOn - Activates autoconversion for this thread.
- ThliCvtOff - Deactivates autoconversion for this thread.

Checks for autoconversion

The following checks are made to determine if autoconversion should be done:

- Is the environment enabled for conversion?
- Is there a file tag that indicates that the file is a candidate for autoconversion?
- Is the program CCSID different from the CCSID in the file tag?

All three checks must be true for the conversion to be done.

Figure 10-44 shows the different options you can use; these must be fulfilled in order for autoconversion to take place.

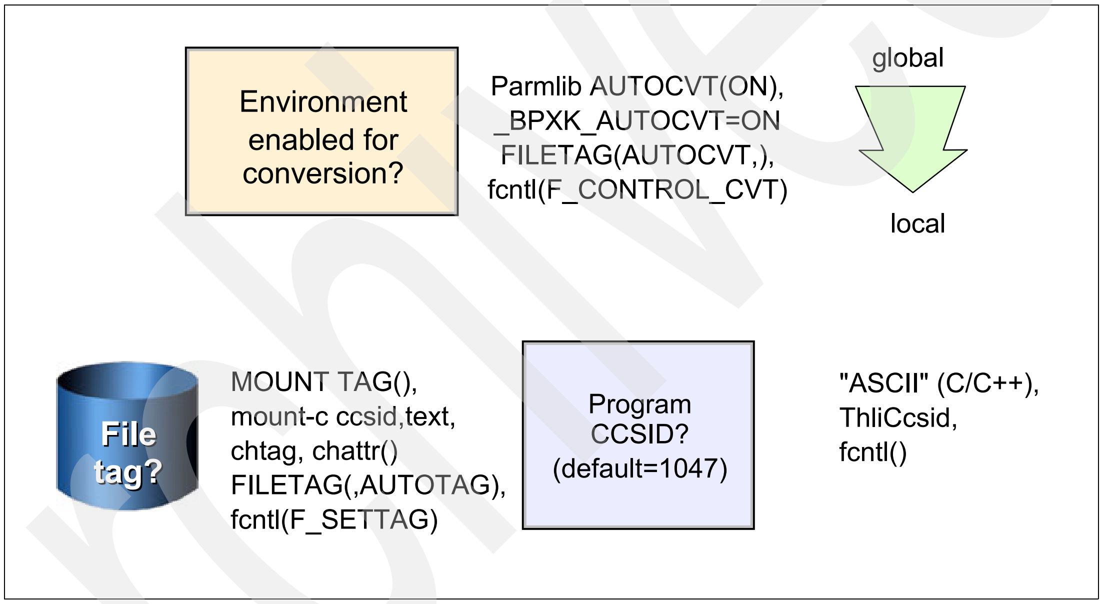

Environment enabled for conversion?
Parmlib AUTOCVT(ON), _BPXK_AUTOCVT=ON FILETAG(AUTOCVT,), fcntl(F_CONTROL_CVT)
global
local
File tag?
MOUNT TAG(), mount-c ccsid,text, chtag, chattr() FILETAG(,AUTOTAG), fcntl(F_SETTAG)
Program CCSID? (default=1047)
"ASCII" (C/C++), ThliCcsid, fcntl()

Figure 10-44 Scope of autoconversion

10.15 File tagging

To complement enhanced ASCII, support for file tagging is also provided. File tags are a way to identify the code set of the text data within files. A file tag is metadata associated with a file; you can think of the file tag as a yellow sticky note on the file containing information about the encoding character set used to write the data in the file.

File tagging and enabling automatic conversion are independent operations. You can tag files without enabling automatic conversion, and vice versa. Remember that enabling automatic conversion for the entire system by using the AUTOCVT statement in the BPXPRMxx member, means that every tagged file becomes subject to conversion by any program that reads from or writes to those tagged files. Because of this, it is possible to have checking done on many tagged files without any conversion occurring.

You should consider enabling automatic conversion in the smallest environment possible by using one of the following methods:

- The _BPXK_AUTOCVT environment variable in a .profile
- The FILETAG run-time option

10.15.1 File tag metadata

File tags are used during automatic codeset conversion, but they are independent from autoconversion.

The file tag metadata contains two fields:

txtflag The txtflag indicates whether a file contains uniformly encoded text data or not. The txtflag can be either on or off.

When the txtflag is on, it indicates that the file is a text file, and uniformly encoded in one specific character codeset, either ASCII or EBCDIC. This means that this file is a candidate for autoconversion. Only files with the txtflag on and a valid codeset are candidates for automatic conversion.

When the txtflag is off, it means the file content is non-uniformly encoded, and that the file is not a candidate for autoconversion.

You can tag your files without using autoconversion, use the filetag for your own information, and perform codeset conversion yourself.

codeset The 16-bit codeset value indicates which code page the data in the file is encoded in. When you tag a file, you can use a character code set name known to the system, or you can use the numeric value called coded character set ID (CCSID). This is a value between 0 and 65536. However, when TEXT is specified, the values of 0 and 65536 are illegal because those values imply no conversion. Other than this, the value is not checked as being valid and the corresponding code page is not checked as being installed.

If a numeric codeset name exists, the CCSID associated with that name will be used. The CCSID values that are associated with names are: 819, which means code page ISO8859-1; and 1047, which means code page IBM-1047. The CCSID can be used both for uniformly encoded text files and for files that contain mixed text and binary data; however, the latter files are not candidates for autoconversion.

The Program CCSID indicates the codeset expected by a C program. Only the values 819 and 1047 are supported. All processes and threads have a default program CCSID of 1047 (EBCDIC). When compiled with the ASCII option, the program CCSID defaults to 819.

10.15.2 How to tag files

There are several ways to tag single files or all files in a file system:

- The TAG parameter in BPXPRMxx
- The `chtag` shell command
- Enhancements to existing shell commands

BPXPRMxx TAG parameter

The ROOT and MOUNT statements in the BPXPRMxx parmlib member have a new parameter, TAG. This parameter specifies whether implicit file tags are assigned to untagged files in the mounted file system. File tagging controls whether a file's data can be converted during file reading and writing.

The format of the TAG parameter is:

```
TAG (NOTEXT | TEXT, CCSID)
```

NOTEXT	Specifies that none of the files in the file system will be automatically converted during file reading and writing. This is the default value.
TEXT	Specifies that each untagged file is implicitly marked as containing pure text data. These files in the file system can be converted by autoconversion.
CCSID	Specifies the option names for the coded character set identifier to be implicitly set for the untagged file.

Note: Either TEXT or NOTEXT, and CCSID, must be specified when TAG is specified.

The tag itself becomes part of the metadata associated with the file, which means that the tag is not permanently stored with the file. The tag is only associated with the file during reading and writing, or when stat()-type functions are issued. The TAG parameter applies to all files in the file system which do not have a file tag. Any file without a tag is implicitly assigned the file tag from the **`mount`** command. The file tag is no longer there when the file system is unmounted.

Files created in the file system after the mount occurs are also implicitly tagged. For example, if you have a file system mounted as ASCII text, you can use OEDIT or vi to create an ASCII text file. Even though OEDIT and vi are EBCDIC programs, autoconversion translates the file to ASCII.

TAG parameter examples

Here are some examples of using the TAG parameter:

TAG(TEXT,819)	Identifies a text file containing ASCII (ISO-8859–1) data.
TAG(TEXT,1047)	Identifies a text file containing EBCDIC ((IBM-1047) data.
TAG(NOTEXT,65536)	Tags a file as containing binary or unknown data.
TAG(NOTEXT,0)	Is the equivalent of not specifying the TAG parameter.
TAG(NOTEXT,273)	Tags a file with the German code set (IBM-273), but the file is ineligible for automatic conversion.

10.15.3 Shell commands for tags

There is a new shell command, **`chtag`**, and enhancements to other existing shell commands to support file tagging:

- **`chtag`**
- **`ls`**
- **`cp`**
- **`iconv`**

The chtag command

With the new shell command **chtag** you can tag files, or change or display information in the file tag. You must have write permission to the file or be a superuser to use the **chtag** command. The format of the command is:

```
chtag -tc codeset file
```

This command tags the specified file as a text file uniformly encoded in the specified codeset. The file becomes a candidate for autoconversion.

If you want to tag all files in a directory and its subdirectories, you can use the **-R** parameter on the command. The pathname must be a directory. The format of this command is:

```
chtag -tc codeset -R dir
```

where:

-t	Indicates that the specified file contains pure text data and, if used alone, sets txtflag=ON.
-c	Sets or changes the codeset. Files that are tagged with this option and contain a valid codeset are candidates for automatic conversion.
-R	Changes the file tag information on all of the files and subdirectories under that directory.

You can tag the file as a file that contains only binary data. The **-b** option sets the txtflag off, and the file is not a candidate for autoconversion. The format of this command is:

```
chtag -b mypgm
```

You can remove the tag from a file by using the **-r** option. The format of this command is:

```
chtag -r file
```

You can display the file tags by using the **-p** option. The format of this command is:

```
chtag -p file
```

The ls command

The **ls** command has a new parameter, **-T**, that can be used to display file tags. This parameter can be combined with other **ls** parameters. The tag information output is the same format as the **chtag -p** command. You can see output from the list command in Figure 10-45.

The cp command

The **cp** command has been enhanced to support file tagging. The format of the command is:

```
cp -O c=codeset source target
```

You can use the **-O** parameter to tag the target file in the copy operation as a text file encoded in the specified code set, as specified with the c= option. Figure 10-45 shows how to copy the file data to the file ebcd and tag the new ebcd file as a text file encoded in EBCDIC codepage 1047. The second command lists the directory with the **-T** parameter on the command to see the file tags.

```
FRAMHUS@SC59:/u/framhus> cp -O c=IBM-1047 data ebcd
FRAMHUS@SC59:/u/framhus> ls -lT
total 16
- untagged    T=off -rw-r--r--   1 STC      SYS1         115 Jun 13 13:25 data
t IBM-1047    T=on  -rw-r--r--   1 STC      SYS1         115 Jun 13 13:30 ebcd
FRAMHUS@SC59:/u/framhus>
```

Figure 10-45 Copy and tag a file

The iconv command

The **iconv** command converts characters in a file from one codepage to another. There are new parameters for the command to support file tagging: **-T**, **-M**, and **-F**. The format of the iconv command with the file tag support is:

```
iconv -T -f ISO8859-1 -t IBM-1047 file
```

where:

-T Specifying **-T** means that the file is tagged as text; this sets the txtflag on, and the codeset will be the same as what you specified in the **-t** option. In Figure 10-46 we convert the file data that is untagged and in EBCDIC codeset to ASCII codeset, and we redirect the output to the new file asci, and that file is tagged.

-F Use the input file's codeset (as defined in the file tag) as the source codeset.

-M Tag a new output file as mixed. That is, the text flag (txtflag) will be off and the value for codeset will be the same as what's specified on the **-t** option.

```
FRAMHUS@SC59:/u/framhus> iconv -T -f IBM-1047 -t ISO8859-1 data > asci
FRAMHUS@SC59:/u/framhus> ls -lT
total 24
t ISO8859-1   T=on  -rw-r--r--   1 STC      SYS1         115 Jun 13 13:53 asci
- untagged    T=off -rw-r--r--   1 STC      SYS1         115 Jun 13 13:25 data
t IBM-1047    T=on  -rw-r--r--   1 STC      SYS1         115 Jun 13 13:30 ebcd
FRAMHUS@SC59:/u/framhus>
```

Figure 10-46 Convert a file from EBCDIC to ASCII and tag it

For more information on file tagging and codeset specifications, see *z/OS UNIX System Services Planning,* GA22-7800 and *z/OS UNIX System Services Command Reference,* SA22-7802.

10.15.4 Accessing data by programs

Figure 10-47 shows an EBCDIC program reading or writing ASCII data.

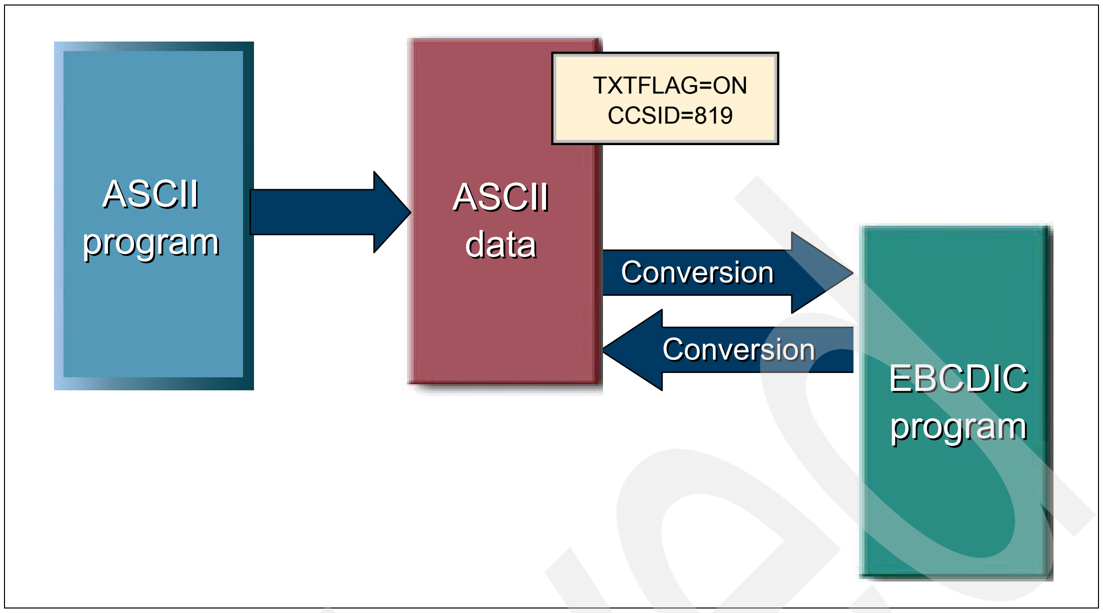


Figure 10-47 ASCII data accessed by an EBCDIC program

Figure 10-48 on page 427 shows an ASCII-compiled program reading or writing EBCDIC data.

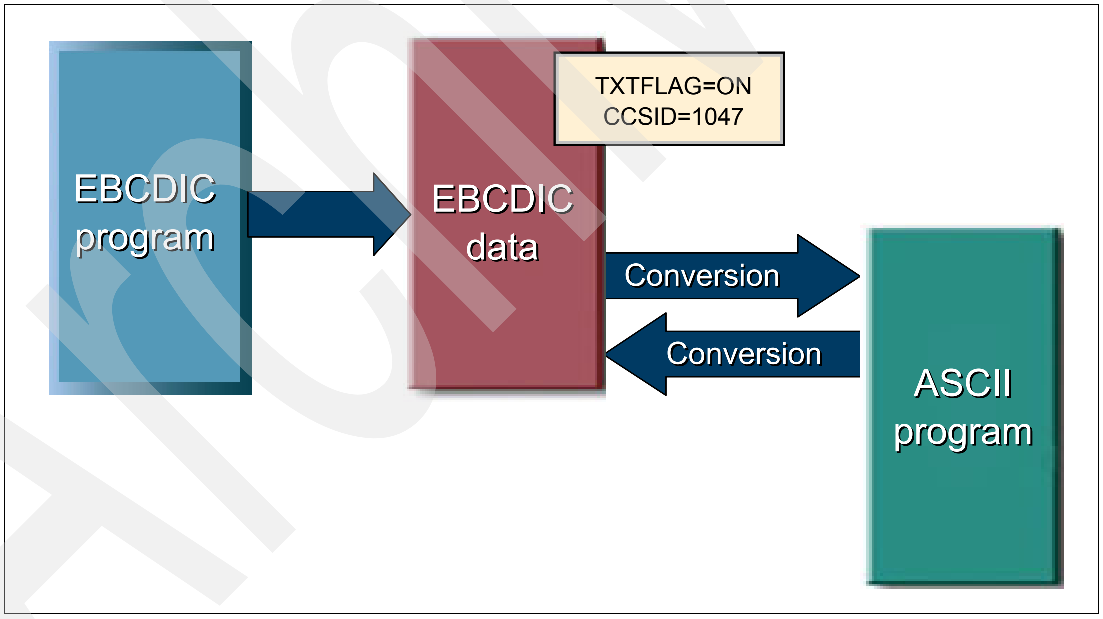


Figure 10-48 EBCDIC data accessed by ASCII program

10.15.5 Other ways to tag files

Files can also be tagged in several other ways.

The mount command

The tag parameter is new for the **mount** command. There are many ways to mount a file system:

- Parmlib statements
- Shell command
- TSO command
- ISPF shell
- BPX2MNT callable service
- REXX syscall

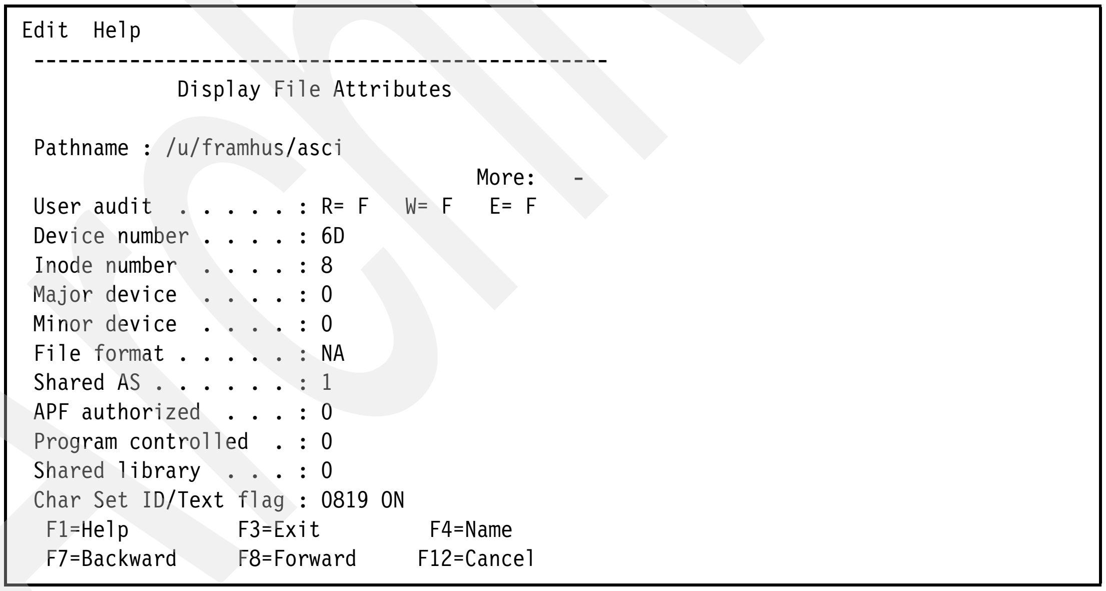
With all these different mount commands, as with the MOUNT statement described in "BPXPRMxx TAG parameter" on page 424, you must specify the setting of the txtflag and the codeset. The following shows a **mount** shell command that tags the file system as text files encoded in ASCII:

```
mount -f user.files -t hfs -c text,819 /u/lib/ascii
```

The following shows the same function issued from REXX:

```
/* rexx */
m.=''
m.mnte_fsname='USER.FILES'
m.mnte_path='/u/lib/ascii'
m.mnte_type='HFS'
m.mnte_filetag='033380000'x /* x'0333' = 819 */
address syscall 'mount m.'
```

There is support in the ISHELL to display the file tag. It is in the display attribute window. The CCSID is shown along with the txtflag set to ON or OFF, as shown in Figure 10-49 on page 428.



```
Edit  Help
 ------------------------------------------------
             Display File Attributes

 Pathname : /u/framhus/asci
                                     More:    -
 User audit  . . . . . : R= F   W= F   E= F
 Device number . . . . : 6D
 Inode number  . . . . : 8
 Major device  . . . . : 0
 Minor device  . . . . : 0
 File format . . . . . : NA
 Shared AS . . . . . . : 1
 APF authorized  . . . : 0
 Program controlled  . : 0
 Shared library  . . . : 0
 Char Set ID/Text flag : 0819 ON
  F1=Help         F3=Exit         F4=Name
  F7=Backward     F8=Forward     F12=Cancel
```

Figure 10-49 ISHELL display file attribute

Redirection

You often use redirection to create a new file. The shell redirection defaults are *no tagging*, and *no autoconversion*.

There are two shell environment variables that can override this default, so that redirected files get tagged, as follows:

_TAG_REDIR_OUT=TXT

```
command > outfile
```

The redirected file is tagged with txtflag on and the codeset is set at the first write to outfile.

The same applies for stderr; here the shell variable is as follows:

```
_TAG_REDIR_ERR=TXT
```

Language environment and C

A new FILETAG run-time option is added for more granular control over how untagged files are set up for conversion, and whether or not open functions will tag new or empty files. The syntax of FILETAG is:

```
FILETAG (AUTOCVT | NOAUTOCVT,AUTOTAG | NOAUTOTAG)
```

The second operand activates or deactivates automatic tagging of new files when created by **`fopen()`**, **`popen()`**, or **`freopen()`**.

Two new C functions, **`_fchattr()`** and **`_chattr()`** are added in this release to allow you to change the attributes of a file, such as, for example, access mode and reference time. These C functions also allow you to tag the file. The provided attribute structure includes a file_tag member. When this structure is populated and passed to either function, the file_tag structure is used to immediately tag the specified file.

Automatic file tagging

When a program issues **`fopen()`** or **`popen()`** with the "text" option, and using the C-RTL FILETAG(,AUTOTAG) run-time option, any new or empty files are automatically tagged at first **`write()`**. Programs that use this form of opening a file are already set up for tagging, and require the least effort to set up automatic conversion.

10.15.6 C/C++

Beginning with z/OS V1.2 there is a new z/OS C/C++ compiler option, ASCII, that instructs the compiler to use ISO8895-1 for its default code page rather than IBM-1047 for character constants and string literals. You can set this option if your program is processing ASCII data natively at execution time. A bit is set in the executable for use at run-time.

The NOASCII compiler option, which is the default, tells the compiler to use the IBM-1047 codepage.

10.15.7 Language environment run-time

A new run-time option, FILETAG is used by the application programmer to specify that HFS files are automatically converted from ASCII to EBCDIC, and whether the files will be automatically tagged with a CCSID when they are opened. This option ensures a more granular control by the application programmer over which files will be applicable for autoconversion, and whether or not new or empty HFS files will be tagged. It is assumed that the application is coded to behave according to the setting of this option.

The format of the run-time option is:

```
FILETAG=(AUTOCVT,AUTOTAG)
```

Where `AUTOCVT` enables autoconversion, and `AUTOTAG` activates the automatic tagging of new or empty files during open.

11

Administration

This chapter describes some administration functions:

- ► Shutting down z/OS UNIX without the need of an IPL
- ► z/OS UNIX file systems and colony address spaces
- ► How to manage HFS data sets
- ► Monitoring z/OS UNIX

11.1 Shutting down z/OS UNIX without re-IPLing

z/OS V1R3 has introduced the ability to shut down and reinitialize the z/OS UNIX environment without the need to IPL the system. This new OMVS option shuts down z/OS UNIX and all the processes that are running under it.

OMVS shutdown allows you to do some reconfiguration that would otherwise have required an IPL, for example:

- Reconfiguring a system from a non-shared HFS system to a shared HFS system
- Implementing a new file structure

There are restrictions and limitations that are not allowed with the OMVS shutdown. In these cases, you need to IPL the system as usual. These limitations include:

- During the cleanup of the resources for z/OS UNIX as part of the shutdown process, some internal failures cannot be resolved using this support, due to their severity.
- OMVS shutdown support cannot be used to install maintenance against z/OS UNIX, because some of the modules are maintained across the shutdown and restart process.
- Installations should avoid using the OMVS restart support as a way to shut down the system with a single command. This causes some unexpected abnormal terminations of address spaces using UNIX System Services that are not shut down in the manner they recommend.
- OMVS restart support is not intended to be used in an unlimited manner to shut down and restart, because some system resources can be lost during the shutdown phase and because of the disruption it causes to the system.

In order to support OMVS shutdowns, a new modify (F) command support for the OMVS address space has been introduced, providing the ability to shut down and then restart the z/OS UNIX environment with the following commands:

```
F OMVS,SHUTDOWN
F OMVS,RESTART
```

11.1.1 Registration support

New registration support has been introduced to allow an application to request special treatment when a shutdown is initiated and to request to receive a new SIGDANGER signal as a warning that shutdown has been initiated and is imminent.

Different kinds of registrations can be implemented, as follows:

- A process or job registered as permanent is not taken down across the shutdown and restart process. Its process-related resources are checkpointed at shutdown time and reestablished at restart time, so the registered permanent process or job can survive the shutdown.
- A process or job registered as blocking delays shutdown until it de-registers or ends. This makes it possible for an application to quiesce itself in a more controlled manner before UNIX System Services starts taking down all processes.
- A process or job registered for notification is notified that the shutdown process is being planned via SIGDANGER signal.

The following command has been modified to include information about what type of registration a specific process has:

```
D OMVS,A=ALL
```

As shown in Figure 11-1, a character P or B, indicating permanent or blocked, has been included in the STATE field.

```
D OMVS,A=ALL
BPXO040I 10.02.18 DISPLAY OMVS 543
OMVS     000F ACTIVE          OMVS=(3A)
USER     JOBNAME ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN BPXOINIT 003C          1          0 MRI--- 07.58.59      .18
  LATCHWAITPID=         0 CMD=BPXPINPR
  SERVER=Init Process                     AF=    0 MF=00000 TYPE=FILE
STC      MVSNFSC5 003B   16908290          1 1R---- 07.59.13       .06
  LATCHWAITPID=         0 CMD=GFSCMAIN
STC      MVSNFSC5 003B   50462724          1 1R---- 07.59.12       .06
  LATCHWAITPID=         0 CMD=BPXVCLNY
STC      MVSNFSC5 003B   50462728          1 1A---- 07.59.14       .06
  LATCHWAITPID=         0 CMD=BPXVCMT
OMVSKERN SYSLOGD5 0041     131081          1 1FI--- 07.59.06       .13
  LATCHWAITPID=         0 CMD=/usr/sbin/syslogd -f /etc/syslog.conf
STC      RMFGAT   0046   84017164          1 1R---P 08.00.01     83.91
  LATCHWAITPID=         0 CMD=ERB3GMFC
TCPIPMVS TCPIPMVS 0043     131085          1 MR---B 08.00.06      8.35
  LATCHWAITPID=         0 CMD=EZBTCPIP
TCPIPMVS TCPIPMVS 0043     131086          1 1R---B 08.00.12      8.35
  LATCHWAITPID=         0 CMD=EZBTTSSL
TCPIPMVS TCPIPMVS 0043     131087          1 1R---B 08.00.12      8.35
```

Figure 11-1 Command now displays process registration

The registration process can be done using the new _shutdown_registration() C function. The BPX1ENV and BPX1SDD callable services have been updated to support shutdown registration.

Attention: Use the F OMVS,SHUTDOWN command carefully because this method will take down other system address spaces. As a result, some system-wide resources may not be completely cleaned up during a shutdown and restart.

Do not use this command to shut down and restart the z/OS UNIX environment on a frequent basis. (If you do, you will eventually have to do a re-IPL.)

11.1.2 Shutting down z/OS UNIX

In order to grant a successful shutdown using the OMVS shutdown support and to control the way in which processes are terminated, remember that the shutdown process will stop all the processes that are running. It is strongly recommended that the following actions taken prior to issuing the OMVS shutdown command:

- Quiesce your batch and interactive workloads.

  Once a shutdown request is accepted, jobs that subsequently attempt to use z/OS UNIX services for the first time will be delayed until the restart occurs, and jobs that are already using z/OS UNIX services as dubbed address spaces are sent termination signals and will end abruptly.

- Quiesce major application and subsystem workloads using z/OS UNIX services in the manner that each application or subsystem recommends.

  That will allow subsystems such as DB2, CICS and IMS, and applications like SAP R/3, Lotus Domino, NetView, and WebSphere to be quiesced in a more controlled manner. The

D OMVS,A=ALL command can be used to determine the applications that require quiescing.

- Unmount all remotely mounted file systems, such as those managed by NFS. Doing so prevents these file systems from losing data.

  Note: As of z/OS V1R3, you can specify that file systems are to be automatically unmounted whenever a system leaves the sysplex.

- Shut down TCP/IP and all TCP/IP applications in the manner that TCP/IP recommends, as well as any colony address spaces. This would potentially include NFS and DFS.

Attention: Failure to perform the necessary shutdown and quiesce of the z/OS UNIX workload prior to using this function may result in abnormal terminations for critical system functions (such as TCP/IP, NFS, DFS, and so on) when shutdown is subsequently done. This may cause many failures on the system that will reduce the likelihood that shutdown will succeed.

Starting the shutdown

The shutdown starts by issuing the F OMVS,SHUTDOWN command and continues as follows:

1. Once the shutdown command has been accepted, a BPXI055I is issued:

   ```
   *BPXI055I OMVS SHUTDOWN REQUEST ACCEPTED
   ```

   SIGDANGER signals are sent to all processes registered for receiving SIGDANGER signals.

2. If any blocking processes are found, shutdown is delayed until these processes end or deregister as blocking, or if an F OMVS,RESTART command is issued to restart. If these blocking processes do not end or deregister in a reasonable amount of time, message BPXI064E is displayed to the console indicating shutdown is delayed.

   In our tests, twelve seconds after the shutdown command was accepted, the BPXI064E message was issued. Message BPXI060I was also issued for each process found to be holding up the shutdown. This message identified the job and address space involved, as follows:

   ```
   *BPXI064E OMVS SHUTDOWN REQUEST DELAYED
   BPXI060I TCPIPMVS RUNNING IN ADDRESS SPACE 0043 IS BLOCKING SHUTDOWN OF OMVS
   BPXI060I TCPIPOE RUNNING IN ADDRESS SPACE 0044 IS BLOCKING SHUTDOWN OF OMVS
   BPXI060I TCPIPB RUNNING IN ADDRESS SPACE 0052 IS BLOCKING SHUTDOWN OF OMVS
   ```

3. Once all blocking processes have ended or deregister as blocking, the shutdown follows by sending a SIGTERM signal to each non-permanent process found and the following messages are received:

```
BPXP010I THREAD 10652BA800000000, IN PROCESS 67239946, WAS 684
TERMINATED BY SIGNAL SIGTERM, SENT FROM THREAD
1065383000000000, IN PROCESS 1, UID 0.
BPXP018I THREAD 1067C3D000000000, IN PROCESS 131109, ENDED 685
WITHOUT BEING UNDUBBED WITH COMPLETION CODE 04EC6000,
AND REASON CODE 0000FF0F.
BPXP018I THREAD 1067AAC000000000, IN PROCESS 131107, ENDED 686
```

Figure 11-2 Messages received after a SIGTERM signal

If any of these processes do not end after receiving the SIGTERM signal, they are sent a SIGKILL signal and the following messages are received:

```
BPXP010I THREAD 106C2BA000000002, IN PROCESS 131198, WAS 789
TERMINATED BY SIGNAL SIGKILL, SENT FROM THREAD
1065383000000000, IN PROCESS 1, UID 0.
BPXP010I THREAD 1066EEC800000000, IN PROCESS 84017176, WAS 792
TERMINATED BY SIGNAL SIGKILL, SENT FROM THREAD
1065383000000000, IN PROCESS 1, UID 0.
```

Figure 11-3 Messages received after a SIGKILL signal

If, after both of these signals are sent and some of the processes still exist, they are terminated with a 422-1A3 ABEND.

```
IEF450I STEVEZ IKJACCT IKJACCNT - ABEND=S422 U0000 REASON=000001A3 952
        TIME=07.58.28
```

Figure 11-4 ABEND message for terminated address spaces

If, after all of these steps, some non-permanent processes still exist, the shutdown request is aborted and the BPXI061E message is issued.

After non-permanent processes have been taken down, the shutdown process continues trying to checkpoint all the permanent processes.

A permanent process cannot be checkpointed; however, a permanent process found using any of the following resources will cause shutdown to be aborted and message BPXI060I to be issued, indicating what resource for which job is causing the problem.

– Shared libraries
– Memory mapped file services
– Map services
– SRB services
– Semaphore services
– Message queue services
– Shared memory services

4. After all non-permanent processes have ended, BPXOINIT is taken down with a 422-1A3 abend.
5. All file systems are unmounted and potentially moved to another system. If for some reason it is not possible to unmount some file systems, a BPXI066E message is issued and shutdown will proceed to the next phase.

   In our tests, we noticed that only one BPXF063I message, which indicates that a file system has been unmounted, is issued. Apparently, it corresponds with the last file system with AUTOMOUNT=N and OWNER of the system being shut down. This can be displayed with the D OMVS,F command.
6. The last step in shutdown processing is to clean up all non-essential kernel and LFS resources, and then the following message is issued:

   ```
   BPXN001I UNIX SYSTEM SERVICES PARTITION CLEANUP IN PROGRESS FOR SYSTEM SC64
   ```

   When this is finished, a BPXI056E is issued indicating that shutdown is complete:

   ```
   *BPXI056E OMVS SHUTDOWN REQUEST HAS COMPLETED SUCCESSFULLY)
   ```

Shutdown differences

In our tests, we found several differences in the shutdown process versus a complete IPL process related to unmount and movement of file systems, as follows:

- File systems mounted with the UNMOUNT keyword, and file systems mounted with the NOAUTOMOVE keyword, are unmounted in the shutdown process, whereas in a complete IPL, file systems mounted with the NOAUTOMOVE keyword remain mounted with no owner associated (and are thus inaccessible for other systems), and only file systems mounted with the UNMOUNT keyword are unmounted when the system is taken down.

  Refer to 6.9.1, “UNMOUNT option” on page 266 for more information about the UNMOUNT option on the **`mount`** command.
- In complete IPL processing, file systems under an automount policy are kept mounted if they have other file systems mounted under them with the AUTOMOVE keyword specified. The shutdown process unmounts file systems mounted with the automount policy, even if they have other file systems mounted on them with the AUTOMOVE keyword.

11.1.3 Restarting z/OS UNIX

The F OMVS,RESTART command restarts the z/OS UNIX environment. This involves the following:

1. Once the RESTART command has been accepted, the following message appears:

   ```
   *BPXI058I OMVS RESTART REQUEST ACCEPTED
   ```

   The first step in the restart process is to reinitialize the kernel and LFS. This includes starting up all physical file systems, as shown in Figure 11-5.

```
BPXF026I FILE SYSTEM HFS.ZOSR03.Z03RD1.ROOT 349 WAS ALREADY MOUNTED.
IEF196I IGD103I SMS ALLOCATED TO DDNAME SYS00055
BPXF013I FILE SYSTEM HFS.SC64.DEV 351 WAS SUCCESSFULLY MOUNTED.
IEF196I IGD103I SMS ALLOCATED TO DDNAME SYS00056
BPXF013I FILE SYSTEM HFS.SC64.ETC 353  WAS SUCCESSFULLY MOUNTED.
IEF196I IGD103I SMS ALLOCATED TO DDNAME SYS00057
BPXF013I FILE SYSTEM HFS.SC64.VAR 355 WAS SUCCESSFULLY  MOUNTED.
BPXF013I FILE SYSTEM /SC64/TMP 356 WAS SUCCESSFULLY MOUNTED.
BPXF203I DOMAIN AF_UNIX WAS SUCCESSFULLY ACTIVATED.
BPXF203I DOMAIN AF_INET WAS SUCCESSFULLY ACTIVATED.
```

Figure 11-5 Restart messages following a system shutdown

2. BPXOINIT is restarted and reestablishes itself as process ID 1.
3. BPXOINIT reestablishes the checkpointed processes as follows:

   All check pointed processes that are still active are reestablished. Those that are not found are not reestablished and will have their checkpointed resources cleaned up.
4. After BPXOINIT completes its initialization, it restarts /etc/init or /usr/sbin/init to begin full function initialization of the z/OS UNIX environment. /etc/init performs its normal startup processing, invoking /etc/rc.
5. After /etc/init has completed full function initialization, a BPXI0041 message is issued indicating that z/OS UNIX initialization is complete:

   ```
   BPXI004I OMVS INITIALIZATION COMPLETE
   ```

In order to monitor the shutdown/restart process, the D OMVS command has been modified to include information about the current state of the process. It shows whether OMVS is shutting down; is shut down; or is restarting. During the shutdown process, the display command also shows a count indicating whether the shutdown request is still proceeding. As long as this count continues to increase, it means that shutdown is still processing, as shown in Figure 11-6.

```
D OMVS
BPXO042I 06.23.52 DISPLAY OMVS 221
OMVS     000F SHUTTING DOWN  27   OMVS=(3A)
```

Figure 11-6 D OMVS command showing current state of shutdown processing

11.2 z/OS UNIX file systems

The following sections explain how you can customize the FILESYSTYPE statements to specify your file systems. These statements define the file systems at OMVS initialization. In Table 11-1 you can see all currently available file systems that you can use with z/OS UNIX.

When you specify SYSPLEX(YES), you must define the file system type for all systems participating in shared HFS. The easiest way to define FILESYSTYPE is to have a single BPXPRMxx member that contains file system information for each system participating in shared HFS. If, however, you decide to define a BPXPRMxx for each system, the FILESYSTYPE statement must be identical on each system.

See 2.3.6, "Step 6 - Customize BPXPRMxx" on page 58 for more information about configuring BPXPRMxx in a sysplex. Facilities required for a particular file system must be initiated on that system. For example, NFS requires TCP/IP, so if you specify a file system type of NFS, you must also initialize TCP/IP when you initialize NFS, even if there is no network connection.

Table 11-1 Available file systems

FS type	Description	Module
AUTOMNT	Handles automatic mounting and unmounting of file systems.	BPXTAMD
CINET	Handles requests for the AF_INET and AF_INET6 family of sockets. This enables many different AF_INET or dual AF_INET/AF_INET6 physical file systems to be active on the system. If you want to use CINET, you must be using z/OS Communication Services (TCP/IP Services). If you use CINET, you cannot use INET.	BPXTCINT
DFSC	Enables a z/OS user or application running in a Distributed Computing Environment (DCE) to access directories and files in the DFS global namespace.	IOECMINI
HFS	Processes file system requests. The HFS statement is necessary if you want to use regular local files.	GFUAINIT
INET	Handles requests for the AF_INET and AF_INET6 family of sockets. You must be using z/OS Communication Services (TCP/IP Services). If you use INET, you cannot use CINET.	EZBPFINI
NFS	Handles Network File System requests for access to remote files. For NFS Client you must create a procedure to run a PFS in a colony address space. For more information, see NFS Customization and Operation, SC26-7029.	GFSCINIT
TFS	Handles requests to the temporary file system (TFS).	BPXTFS

FS type	Description	Module
UDS	Handles socket requests for the AF_UNIX address family of sockets.	BPXTUINT
ZFS	Handles Distributed File Service zSeries file system requests.	IOEFSCM

11.2.1 How to start colony address spaces

To set up a physical file system in a colony address space, create a cataloged procedure in SYS1.PROCLIB to start the colony address space. The following is a sample entry to activate the physical file system NFS inside the colony address space called NFSCLNT.

```
FILESYSTYPETYPE(NFS) ENTRYPOINT(GFSCINIT) ASNAME(NFSCLNT)
```

Restriction: Some physical file systems cannot be initialized in colonies; for example, the INET or CINET sockets file systems and HFS.

Important: The name of the procedure must match the name specified on an ASNAME operand on the FILESYSTYPE statement in BPXPRMxx that starts physical file systems in this colony address space.

11.2.2 Start colony address spaces outside of JES

If you do not want colony address spaces to be started under JES (which is the default), you can change this by including the SUB=MSTR parameter with the ASNAME keyword, which is specified as follows:

```
FILESYSTYPETYPE(NFS) ENTRYPOINT(GFSCINIT) ASNAME(NFSCLNT,'SUB=MSTR')
```

The second value is optional and is a quoted string that is appended to the procedure name when the address space is started. The string can be up to 100 characters long.The start_parms are not validated; they are just passed to the system when the address space is started with an internal start command. The colony address space runs outside of JES control and does not have to be stopped if JES has to be stopped, which facilitates planned shutdowns of individual systems in a shared HFS sysplex. The NFS client, TFS, and zFS physical file systems support running outside of JES.

The following information may help you to decide whether to move these z/OS UNIX colonies outside of JES. The DFS Client PFS does not support being started outside of JES. z/OS UNIX colony address spaces are started procedures. If you do not want to run them under JES, you will need to change any DD SYSOUT= data sets that are specified in these procedures. These must be changed because SYSOUT data sets are only supported under JES. There are three ways you can change these data sets:

- Direct the output to a named data set by changing to DD DSN=.
- Direct the output to a named file by changing to DD PATH=.
- Throw the output away by changing to DD DUMMY.

Additionally, there are some DD names that Language Environment (LE) will open under certain conditions. If these data sets have not been allocated in the procedure, LE dynamically allocates them with SYSOUT=. The DD names are:

SYSIN	For standard input.

SYSPRINT	For standard output. If SYSPRINT does not exist, LE looks for SYSTERM or SYSERR. If one of those exists, it will be used. But LE does not dynamically allocate either SYSTERM or SYSERR.
SYSOUT	For standard error. It is also the default message file DD.
CEEDUMP	For capturing dumps formatted by LE.

If any of these names are not currently used in the colony procedure, you must add them with DD DUMMY. If any of the existing DD SYSOUT= statements are not changed, or any of those dynamically allocated by LE are not added, and an attempt is made to open that DD name, the result will be an ABENDS013. Exactly which DD names are opened and when varies by name and product and the situation. There are also other consequences of running outside of JES you may need to consider:

- SDSF displays will not list the colony address space.
- There will be no JOBLOG or system messages data set.
- System messages will go to SYSLOG.
- SMF recording is different between JES and the master subsystem.

11.2.3 Running a temporary file system in a colony address space

In some situations, you may want to run a temporary file system in a colony address space instead of the kernel address space. Because TFS can use up a lot of kernel virtual storage, there may be some environments in which the kernel can run out of private storage. This can happen on large systems with many shell users or in some Lotus environments. By putting the TFS in a colony, impact on the kernel is reduced, and you can have a larger TFS. To create a cataloged procedure for a temporary file system, the following must be done:

1. Add a FILESYSTYPE statement to your BPXPRMxx member, as follows:

   ```
   FILESYSTYPE TYPE(TFS) ENTRYPOINT(BPXTFS) ASNAME(TFSSTC)
   ```

2. Create the cataloged procedure with the name you used for `ASNAME` in the BPXPRMxx member, and store it in SYS1.PROCLIB, as shown in Figure 11-7.

```
//TFSSTC  PROC
//*******************************************************************
//*  z/OS UNIX TFS procedure                                        *
//*******************************************************************
//TFS      EXEC PGM=BPXVCLNY,
//             REGION=0M,
//             TIME=1440
//SYSIN    DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//
```

Figure 11-7 Sample of TFS started task

3. Make the entries in the security system for running your cataloged procedure.
4. Activate the definitions in the BPXPRMxx member:
   a. During the IPL
   b. Using the SETOMVS RESET=(xx) command, available since OS/390 2.8
5. Mount a TFS file system as shown in Figure 11-8. If you have SYSPLEX(YES) in your system, see Figure 11-9.

A TFS uses private storage for the file system in memory. If you run it in the kernel, then you might run out of virtual storage. However, by starting multiple TFSs in colonies, you can create many temporary files or very large temporary files (about 1.5 GB per TFS colony).

When a TFS is to be used in other situations, it is made available by mounting. You can mount it with a MOUNT or ROOT parmlib statement or via one of the several mount commands supported.

- `TYPE` must specify TFS.
- `FILESYSTEM` must specify a unique name for the file system. We recommend that you specify a unique name across the sysplex. This may make it easier to understand the output produced by commands such as **`df`**.
- `MODE` can be either RDWR or READ.
- `PARM` specifies the amount of virtual storage the file system uses. This is specified as `PARM('-s n')`, where `n` is the approximate size in megabytes. If `PARM` is omitted or invalid, the TFS defaults to 1 MB. If the mount request specifies a size in megabytes that is too large for the address space, the request will fail with an EMVSERR (9D). Try the request again, using a smaller value.

The following is a sample MOUNT statement for a 512 MB in-storage file system mounted over /tmp:

```
   MOUNT FILESYSTEM('/TMP') TYPE(TFS) MOUNTPOINT('/tmp') PARM('-s 512')
```

Figure 11-8 Sample TFS mount command

Because the TFS is a temporary file system, unmounting it causes all data stored in the file system to be discarded. If, after an unmount, you mount another TFS, that file system has only dot (.) and dot-dot (..) and nothing else.

11.2.4 TFS in shared file system mode

A TFS can be used in a shared HFS environment. If you are using a TFS for /tmp, because each system will require its own copy, the `FILESYSTEM` name is required to be different. Hence, the MOUNT statement would need to be specified as in Figure 11-9.

```
MOUNT FILESYSTEM('/TMP&SYSNAME.') TYPE(TFS) MODE(RDWR) NOAUTOMOVE
MOUNTPOINT('/&SYSNAME./tmp') PARM('-s 512')
```

Figure 11-9 Sample mount command for a shared file system

Because &SYSNAME is different on each system, 'TMP&SYSNAME.' will have a different file system name on each system.

11.3 Managing HFS data sets

This section provides information about managing HFS data sets. We include examples of:

- Backup and restore processing of an HFS data set that is mounted read/write
- Recovery processing
- Increasing the file size of an HFS data set
- Converting from single to multi-volume HFS data sets
- Interpreting the different kinds of file size displays

Backup and restore processing for individual files in an HFS uses Tivoli Storage Manager (TSM). For both SMS-managed and non-SMS-managed HFS data sets, DFSMShsm™ can perform automatic volume backup (by invoking DFSMSdss™) and incremental backup.

11.3.1 DFSMSdss dump and restore

To back up and restore an entire HFS data set (not individual files within the HFS data set), you can use DFSMSdss functions:

DFSMSdss DUMP	Use the DFSMSdss DUMP function to dump the file system. The logical dump processing is the recommended method to dump an HFS data set.
DFSMSdss RESTORE	Use the DFSMSdss RESTORE function to restore the dumped file system with a new name (rename). If you want to maintain the original file system name, you must first unmount the HFS.

You can use the MVS D OMVS,F system command to display the currently mounted HFS data sets. For additional information regarding DFSMSdss processing and control statements, refer to the following documentation:

- *z/OS DFSMS/MVS DFSMSdss Storage Administration Reference,* SC26-4929
- *z/OS DFSMS/MVS V1R5 DFSMSdss Storage Administration Guide,* SC26-4930

Important: You can dump an HFS data set from any system in a participating group in a shared HFS sysplex (OS/390 V2R9 and higher). In a pre-OS/390 V2R9 or non-shared environment, you must perform the dump on the same system on which the HFS is currently mounted read/write.

Dump processing

DFSMSdss provides four different kinds of backup processing:

- Logical data set dumps
- Physical data set dumps
- Logical volume dumps
- Physical volume dumps

DFSMSdss can perform either logical or physical processing. If you dump a data set logically, DFSMSdss restores it logically; if you dump it physically, DFSMSdss restores it physically.

Logical processing	Logical processing operates against data sets independently of physical device format. The data sets are located by searching either the catalog or VTOC. If input volumes are specified through the

LOGINDD, LOGINDYNAM, or STORGRP keywords, then data sets are located by searching the VTOCs. Otherwise, data sets are located by searching the catalog.

Physical processing Physical processing moves data at the track-image level and operates against volumes, tracks, and data sets. The data sets are located by searching the VTOC. The processing method is determined by the keywords specified on the command. For example, LOGINDYNAM indicates a logical dump and INDYNAM a physical dump. Each type of processing offers different capabilities and advantages.You can select data sets for DFSMSdss processing by filtering on specified criteria. DFSMSdss can filter on fully qualified or partially qualified data set names (by using the INCLUDE or EXCLUDE keyword) and on various data set characteristics (by using the BY keyword; for example BY(DSORG,EQ,HFS)).

You can filter data sets with any of the following commands:

- Logical dump
- Logical restore
- Physical data set dump
- Physical data set restore

Note: With DFSMSdss V1 R5, the new STORGRP keyword for logical data set dump operations allows filtering by storage group name, in addition to filtering by volume serial numbers. It allows you to dump a complete storage group.

Logical dump

DFSMSdss performs logical processing if you specify the DATASET keyword with the DUMP command, and either no input volume is specified, or LOGINDDNAME, LOGINDYNAM, or STORGRP is used to specify input volumes. If you specify the DATASET keyword with the DUMP command and do not specify input volumes (neither LOGINDYNAM nor LOGINDDNAME), DFSMSdss performs a logical data set dump using information in the catalogs to select data sets. If you specify the DATASET keyword with either LOGINDDNAME or LOGINDYNAM, DFSMSdss performs a logical data set dump using information in the VTOCs to select data sets.

Mounted HFS data sets should be backed up using logical data set dumps. You can dump an HFS data set from any system in a group participating in a shared HFS sysplex (OS/390 2.9 and higher). In a pre-2.9 or non-shared environment, you must perform the dump on the same system on which the HFS is currently mounted read/write.

A logical data set dump provides the quiesce serialization mechanism (using the BPX1QSE callable service) to ensure data integrity. The quiesce allows an HFS data set to be dumped while in use. In a non-shared environment, the dump job must be executed on the same system on which the HFS is currently mounted. Part of the quiesce processing is to perform a sync. This is intended to flush out any buffered data to DASD prior to a logical dump. The same processing will be done in a shared HFS environment. The DFSMSdss process will be finished after unquiesce processing completes successfully.

In DFSMSdss 1.4 and prior releases, the SHARE keyword must be specified when dumping a mounted HFS data set. When an HFS data set is currently mounted, OMVS will have a shared SYSDSN ENQ. Before DFSMSdss 1.5, DFSMSdss obtains an exclusive SYSDSN ENQ if the SHARE keyword is not specified. File and Attributes Management Service (FAMS) is called to perform the quiesce against the HFS data set before dumping it. The quiesce will succeed if the data set is not mounted or if it is mounted on the same system that it is being dumped from. In DFSMSdss 1.5, DFSMSdss no longer obtains a SYSDSN ENQ, so the SHARE keyword is no longer required during logical dump. FAMS is no longer called to perform the quiesce.

Concurrent copy and virtual concurrent copy

DFSMSdss provides the concurrent copy (CC) function that, when used with supported hardware, provides point-in-time data consistency. The data is copied as if no updates have occurred. This function is invoked through the CONCURRENT keyword on the DUMP command. If the source volume is a RAMAC® Virtual Array and CONCURRENT is specified, DFSMSdss uses the Snapshot capability of the RVA to provide a function equivalent to concurrent copy. This function is called CC-compatible Snapshot or Virtual Concurrent Copy (VCC) and is transparent to the user. HFS data sets can be dumped by using concurrent copy (CC or VCC), for shared HFS in OS/390 2.9 and higher:

Concurrent copy (CC)

The concurrent copy function of DFSMSdss is a hardware and software solution that allows you to back up a database or any collection of data at a point-in-time, and with minimum down time, for an HFS or data base. The system serializes access to the data being dumped or copied just long enough for the concurrent copy session to be initialized. This serialization takes a matter of seconds, unlike the quiesce and backup technique, which requires data to be unavailable for the entire duration of the dump, possibly for hours. The copy is logically complete as soon as the concurrent copy environment is initialized. At this point, the original state of the data is “protected” by concurrent copy.

After logical completion, the data is once again available for unrestricted application access. The copy is physically complete once the concurrent copy process finishes copying the data to the output device. After concurrent copy initialization, DFSMSdss releases all the serialization it holds on the data, informs the user that the initialization is complete so that update activity may resume, and begins reading the data.

Be aware, however, that concurrent copy does not remove all data integrity exposures. For example, a DFSMSdss full-volume dump serializes the VTOC of the source volume, but does not serialize the data sets on the volume. This ensures that the existing data sets are not deleted or extended, and new data sets are not allocated. However, there is an exposure in that the data in the existing data sets can be changed. Without concurrent copy, this exposure exists for the entire duration of the dump. With concurrent copy, the exposure exists only during initialization. Logical data set dump processing of HFS data sets, full volume, and physical data set dump operations are processed on a track-by-track basis by DFSMSdss. Refer to the redbook *Implementing Concurrent Copy,* GG24-3990, for details on concurrent copy.

Virtual concurrent copy (VCC)

CC-compatible Snapshot support uses Snapshot to provide a concurrent copy-like function when the source device supports Snapshot but does not support concurrent copy. If you are already using concurrent copy, you do not have to make changes to your JCL to use virtual concurrent copy. To invoke VCC, you specify the CONCURRENT keyword on a DFSMSdss COPY or DUMP statement. You can see the use in Figure 11-10 on page 444.

Important: You can dump an HFS data set from any system in a participating group in a shared HFS sysplex (OS/390 V2R9 and higher). In a pre-OS/390 V2R9 or non-shared environment, you must perform the dump on the same system on which the HFS is currently mounted read/write.

```
//LUTZBACK JOB ,'HFS BACKUP',NOTIFY=LUTZ,
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),TIME=1440
//COPY     EXEC PGM=ADRDSSU
//OUTDD    DD   DISP=(,CATLG,DELETE),SPACE=(CYL,(5,5)),
//          STORCLAS=STANDARD,DCB=BLKSIZE=27998,UNIT=3390,
//          DSN=LUTZ.HFS.BACKUP
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 DUMP                              -
     DS(INCLUDE( LUTZ.HOME.HFS ))   -
     OUTDD(OUTDD)                   -
ALLDATA(*)                     -
     CONCURRENT /*
//
```

Figure 11-10 Sample backup JCL

During CC-compatible Snapshot (VCC), data is snapped from the source location to an intermediate location called the working space data set, and the data is gradually copied to the target location using normal I/O methods. The operation is logically complete after the source data is snapped to the working space data sets and physically complete after the data is moved to the target media. Refer to the redbook *Implementing DFSMSdss SnapShot and Virtual Concurrent Copy,* SG24-5268 for additional information about VCC.

HFS and ALLDATA(*) considerations

The amount of space dumped during logical dump and restore processing is related to the high formatted frame number (HFRFN).

Dumping without specifying ALLDATA(*)

If you do not specify ALLDATA(*) at DUMP time, then DFSMSdss will only dump the storage to the high formatted value (HFRFN). On restore processing, DFSMSdss allocates the target HFS data set based on the HFRFN. This means that DFSMSdss could reduce the amount of space for an HFS data set from the allocated space to the (high) formatted space.

- You cannot reduce the space below the high formatted page by using DFSMSdss DUMP/RESTORE or DFSMShsm MIGRATE/RECALL. To reduce the space of an HFS data set below the high formatted page, you must copy it.
- You can also use the copy tree utility provided by z/OS UNIX System Services. Copytree is a utility that can run under TSO or the shell and is used to make a copy of a file hierarchy preserving all file attributes. Copytree is part of z/OS UNIX and located in /samples.

Dumping with specifying ALLDATA(*)

If you specify ALLDATA(*) at DUMP time, then DFSMSdss allocates the target HFS data set with the same size as the original (source) HFS data set. DFSMSdss will not dump residual data past the high formatted page number when dumping HFS data sets, even if ALLDATA was specified during DUMP. However, the target data set will still be allocated to the high allocated page, if ALLDATA was specified during DUMP.

Logical volume dump

To perform a logical volume dump, you specify DATASET(INCLUDE(**)) with either LOGINDDNAME or LOGINDYNAM. LOGINDDNAME identifies the input volume that contains the data sets to be dumped. LOGINDYNAM specifies that the volumes containing data sets to be dumped be dynamically allocated. You can also specify the SELECTMULTI parameter to select the method for determining how cataloged multi-volume data sets are to be selected during a logical data set dump operation. SELECTMULTI is accepted only when logical volume filtering is specified with either LOGINDDNAME or LOGINDYNAM keywords. If logical volume filtering is not used, the specification of SELECTMULTI is not accepted.

- ALL - Is the default, and specifies that DFSMSdss not dump a multi-volume data set unless the volume list specified by LOGINDDNAME or LOGINDYNAM lists all the volumes that contain a part of the HFS data set.
- ANY - Specifies that DFSMSdss dump a multi-volume data set when any volume in the volume list specified by LOGINDDNAME or LOGINDYNAM contains a part of the HFS data set.
- FIRST - Specifies that DFSMSdss dump a multi-volume data set only when the volume list specified by LOGINDDNAME or LOGINDYNAM lists the volume that contains the first part of the HFS data set.

If either LOGINDDNAME or LOGINDYNAM is specified, DFSMSdss uses logical processing to perform the dump operation. Logical processing is also used if no input volume is specified.

A multi-volume data set that has extents on volumes not specified with LOGINDDNAME or LOGINDYNAM will not be dumped unless you specify SELECTMULTI.

Examples of logical dump processing

You can select data sets for DFSMSdss processing by filtering on criteria you specify. DFSMSdss can filter on fully or partially qualified data set names and on various data set characteristics. The SHARE keyword is required to logically dump mounted HFS data sets in DFSMSdss releases prior to DFSMSdss 1.5. It is no longer required when logically dumping HFS data sets beginning with DFSMSdss Release 1.5. To dump an HFS data set while it is mounted in read/write mode you can specify:

```
DUMP DATASET(INCLUDE(hfs.data.set.name)) -
OUTDDNAME(ddname)
```

You can dump all allocated space using the ALLDATA(*) keyword:

```
DUMP DATASET(INCLUDE(hfs.data.set.name)) -
ALLDATA(*) -
OUTDDNAME(ddname)
```

Using the CONCURRENT (or CC) keyword will minimize the time that the HFS data set is serialized. DFSMSdss can filter on fully qualified or partially qualified data set names (by using the INCLUDE or EXCLUDE keyword) and on various data set characteristics (by using the BY keyword). The next example shows how to select all HFS data sets on one volume by using a filter on DSORG (data set organization). Or you can filter on a partially qualified data set name. In the following sample, all data sets with a high level qualifier of OMVS and with a qualifier starting with HFS will be selected for dump processing. The STORGRP parameter specifies that all of the online volumes in the storage group are dynamically allocated. If a volume in the storage group is not online, that volume is not used for processing. This example can be used to dump all data sets in a storage group.

```
DUMP DATASET(INCLUDE(hfs.data.set.name)) -
CC -
OUTDDNAME(ddname)
```

Physical dump

Physical dump does not use the quiesce service to write cached data to disk, and it is not recommended as a way to back up HFS data sets that are mounted read/write at the same time on several systems. The SHARE keyword should never be specified during a physical dump of an HFS. Since the SHARE keyword applies to the SYSDSN ENQ, it does not provide protection against updates during dump. If SHARE or TOL(ENQF) is specified during a physical dump, then the internal control information and data inside the HFS can change during the dump. This can result in a dump data set that contains a broken HFS data set. This data set may not be usable after it has been restored.

If you must physically dump an HFS that is in use, TOL(ENQF) should be used instead of SHARE. At least, with TOL(ENQF) the user will receive a return code of four along with a warning message if adequate serialization was not provided during dump. For a physical dump of HFS data sets, all of the allocated space is always dumped, regardless of the ALLDATA keyword. A physical dump of HFS data sets that are mounted R/W is not recommended, because quiesce is not available during physical dump. Using SHARE or TOL(ENQF) can result in a dump data set that contains a broken HFS data set, and it may not be usable after it has been restored.

Restore processing

You can use DFSMSdss to restore HFS data sets to DASD volumes from DFSMSdss-produced dump volumes. You can restore HFS data sets to the same or a different device type if you have performed a logical dump. DFSMSdss distinguishes between:

- Logical Data Set Restore

  A logical data set restore is performed if you are restoring from a volume created with a logical dump operation and if you specified the DATASET keyword.

- Physical Data Set Restore

  A physical data set restore is done if you are restoring from a dump volume created by physical dump processing and you specified the DATASET keyword. If the dump volumes resulted from a physical data set dump operation, you must do a physical data set restore.

- Volume Restore

  You can recover a volume or ranges of tracks from a full-volume dump operation.

You can also:

- Rename a data set during a restore
- Replace an existing data set
- Change either the entire name or part of the name
- Create a new data set with a new name instead of replacing the original data set on a DASD volume

During a restore operation, the data is processed the same way it was dumped because physical and logical dump tapes have different formats. If a data set is dumped logically, it is restored logically; If it is dumped physically, it is restored physically. A data set restore operation from a full volume dump is a physical data set restore operation. You cannot restore to an HFS data set when it is currently mounted. You must first unmount the HFS data set before you can restore (REPLACE) to it. Figure 11-11 on page 447 shows a sample job for restoring an HFS file system.

```
//LUTZREST JOB ,'HFS RESTORE',NOTIFY=LUTZ,
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),TIME=1440
//RESTORE  EXEC PGM=ADRDSSU
//INDD     DD   DISP=SHR,DSN=LUTZ.HFS.BACKUP
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 REST                        -
     DS(INCLUDE( LUTZ.HOME.HFS ))   -
     RENAMEU( LUTZ.HOME.HFS, LUTZ.HOME2.HFS )   -
     INDD(INDD)                            -
     CATALOG
/*
//
```

Figure 11-11 Sample JCL for restore

Recovering files and directories

If you detect a problem in a file system, we suggest that you perform the following steps to recover your data as best as possible. Note: We assume that you have taken a DFSMSdss dump or DFSMShsm backup and incremental TSM backups before the failure.

1. Create a new HFS data set and mount it to a different mount point.
2. You must unmount and then remount the affected file system (HFS data set) in read-only mode if the affected file system is flagged in error in the main control block (called the RFS). You can use the CONFIGHFS command to display the RFS error flag for a specific file system.
3. Run the pax copy function or copytree to copy the affected file system into the new file system (HFS data set) to salvage as many files and directories as possible. Some files could have been changed after the last backup action, so this will attempt to copy the files over the new file system.
4. Restore the file system from backup.
   – Depending on the failure, you could restore individual files and directories from TSM backups.
   – If the failure does not allow you to do this, you must restore or recover the complete file system (HFS data set) from a DFSMShsm backup version or a DFSMSdss dump version. Depending on your backup strategy, you may also restore individual files and directories from TSM backups afterwards, because some files could have been backed up after the last DFSMSdss or DFSMShsm backup action.

Now, you have recovered your broken file system (HFS data set). But the restored files and directories will only be as current as the date of the last TSM backup, DFSMShsm backup, or DFSMSdss dump.

11.3.2 Increasing the size of an HFS data set

During the initial allocation of an HFS data set you specify a primary and secondary allocation quantity. Normally during initial allocation, the system allocates the first extent based on the primary allocation. The secondary allocation quantity specifies the amount of additional space to be automatically obtained if more space is needed as users add files and extend existing files. HFS data sets can span up to 59 volumes, with up to 255 total extents for all volumes, and up to 123 extents per volume.

Note: For DFSMS V1R4 and earlier, a file system resides on only one volume. In DFSMS V1R5, an HFS data set can span multiple volumes.

To control or limit the size of an HFS data set, you can define it with no secondary allocation value (zero). Additional extents will not be automatically obtained. Such an HFS data set is limited to the size of the primary allocation.

Note: However, in this case, if candidate volumes are available, the HFS automatically allocates the primary allocation amount on each candidate volume as the HFS is extended to the new volume also with no secondary quantity specified.

As mentioned before, an HFS data set size increases as users add files and extend existing files. It may be necessary to increase the size of an existing HFS data set when it has run out of extents or it can outgrow the space on its volume. In this case, the storage administrator or system programmer responsible for HFS data sets can make more space available by doing one of the following:

- Use DFSMSdss DUMP and RESTORE to:
  - Move the entire full file system to another volume.
  - Restore to a larger preallocated HFS data set.
  - Restore to a multi-volume HFS data set, if it was a single volume HFS data set before.
- Add volumes to the HFS data set with the IDCAMS ALTER ADDVOLUMES command.
- Invoke the new z/OS UNIX **`confighfs`** command to extend the HFS.
- Remove other data sets from the volume on which the full HFS data set resides.
- Remove files from the full file system by either deleting them or by moving them to another file system on another volume. If it is impossible to remove the chosen files from a particular directory in the file system, it may be possible to remove other files from a different directory in the same file system.
- Create a new file system on another volume and move some files from the full file system to the new file system. To avoid problems that might result from this approach, define symbolic links using the original names.

Adding volumes to an HFS file system

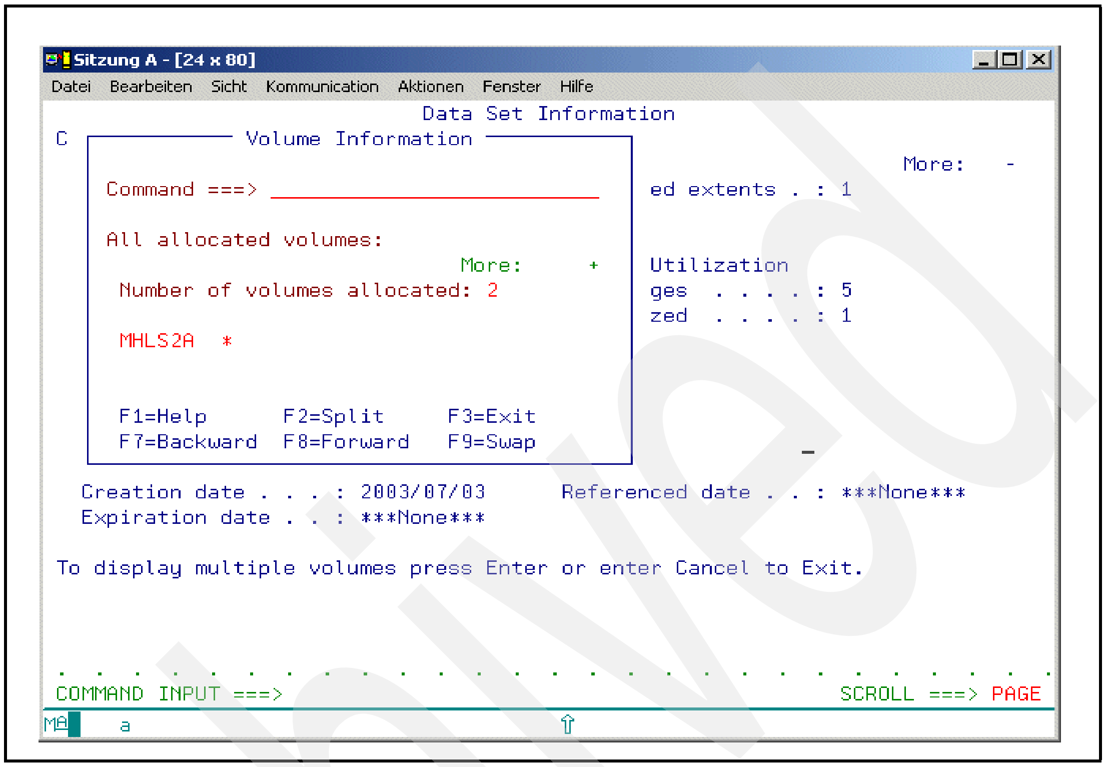
Usually, an HFS dataset can allocate 123 extents on a maximum of 59 volumes.

Use IDCAMS ALTER ADDVOLUMES to add another volume to the HFS volume candidate list. The HFS will have to be unmounted and then mounted again in order for the volume to be available. The HFS data set must be managed by DFSMSsms. Sample JCL for IDCAMS ALTER ADDVOLUMES is shown in Figure 11-12.

```
//LUTZAMS  JOB ,'AMS ALTER',NOTIFY=LUTZ,
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),TIME=1440
//COPY     EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 ALTER LUTZ.HOME.COPY.HFS ADDVOLUME(*)
/*
//
```

Figure 11-12 Sample JCL for adding volumes using IDCAMS

Figure 11-13 shows the status panel of the data set that we expanded. The asterisk in the volume list means that any volume is allocated that is available in the DFDSSsms storage pool.



```
                              Data Set Information
C    ----------- Volume Information -----------
                                                                   More:    -
      Command ===> __________________________     ed extents . : 1

      All allocated volumes:
                                   More:     +     Utilization
       Number of volumes allocated: 2              ges  . . . . : 5
                                                   zed  . . . . : 1
       MHLS2A  *


       F1=Help      F2=Split     F3=Exit
       F7=Backward  F8=Forward   F9=Swap

   Creation date . . . : 2003/07/03      Referenced date . . : ***None***
   Expiration date . . : ***None***

 To display multiple volumes press Enter or enter Cancel to Exit.



 COMMAND INPUT ===>                                            SCROLL ===> PAGE
MA     a
```

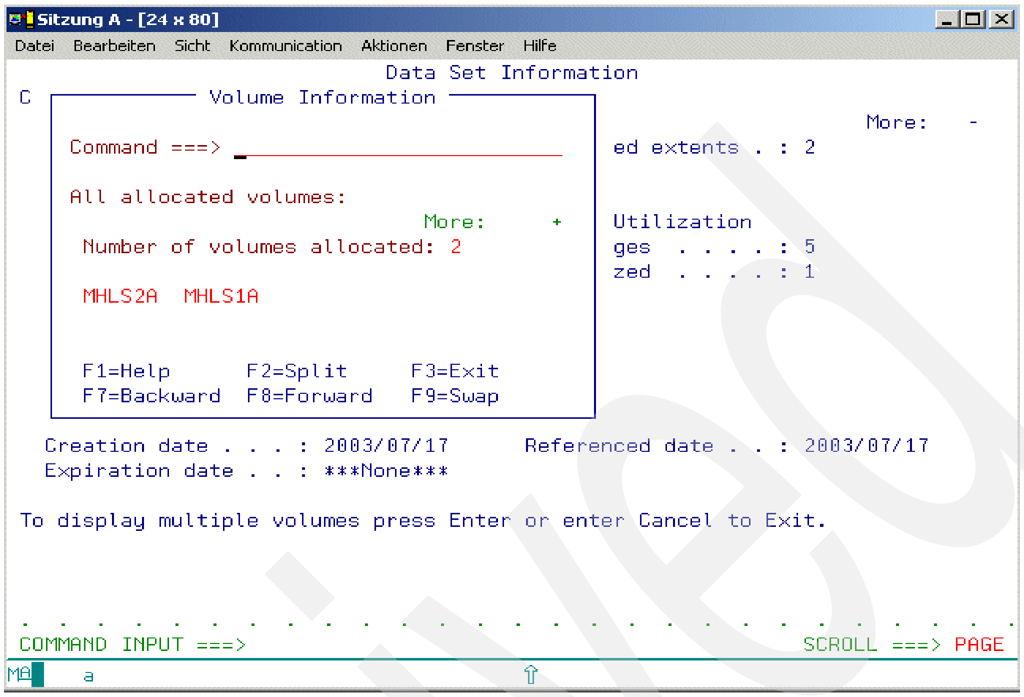
Figure 11-13 File details before extent to another volume

To expand the file system you must use the **confighfs** utility available since OS/390 V2R9. A sample command would be as follows:

```
>confighfs -xn 10M /tmp/lutz
```

The option `-xn` means go to the next volume.

Figure 11-14 on page 450 illustrates the allocation of the second volume.



```
                           Data Set Information
C    ─────────── Volume Information ───────────
                                                                   More:    -
    Command ===> _                              ed extents . : 2

    All allocated volumes:
                                 More:     +    Utilization
     Number of volumes allocated: 2             ges   . . . . : 5
                                                zed   . . . . : 1
     MHLS2A   MHLS1A


     F1=Help      F2=Split     F3=Exit
     F7=Backward  F8=Forward   F9=Swap

   Creation date . . . : 2003/07/17      Referenced date . . : 2003/07/17
   Expiration date . . : ***None***

 To display multiple volumes press Enter or enter Cancel to Exit.


 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 COMMAND INPUT ===>                                              SCROLL ===> PAGE
MA      a
```

Figure 11-14 File details after expanding to the next volume

11.3.3 Logical backup and restoring of file systems using TSM

Logical backup means backup on the file level. You can use Tivoli Storage Manager (TSM). TSM, previously known as ADSTAR Distributed Storage Manager or ADSM, is a client-server product designed to protect and manage a broad range of data, from Notebook PCs to powerful corporate servers. TSM supports more than 35 different operating platforms using a consistent Web-based graphical user interface (GUI). It provides granularity to the storage management of file systems.

DFSMSdss DUMP provides recovery protection of entire HFS/zFS data sets, while TSM allows you to back up and restore individual files. The price paid for this level of granularity is performance. DFSMSdss DUMP works much faster than TSM, because it is not keeping track of each file, but as you will see below, you do not need to back up every file using TSM, just the data critical to your organization.

We recommend that you use both DFSMSdss and TSM to provide your data management. DFSMSdss DUMP allows rapid recovery from a complete loss of HFS/zFS data sets, including system files, and TSM allows you to recover lost or damaged critical files or directories in an otherwise healthy HFS/zFS system. Important features of TSM are:

- TSM works on a “progressive incremental” (also known as “incremental forever”) principle by default, allowing you to back up only those files that have changed since the last backup.
- TSM has a Central Scheduler, which allows incremental or selective backups at defined intervals. For example, you could back up all of your files nightly, and critical files in a specific subdirectory hourly.
- Separate policies can be applied to manage the data within a given HFS file system. These policies, which will be familiar to users of SMS but are not dependent on SMS,

define retention periods and versioning, so that you may keep more versions of critical files and allow point-in-time restoration. These policies, together with an Include/Exclude list, allow you to control what is backed up, how often it is backed up, and how the backup data will be managed.

- According to your security requirements, you may choose to allow end users to recover files they access, or you may allow storage administrators or help desk operators to perform functions on users' behalf.

Setting up the TSM client

The Tivoli Storage Manager (TSM) is a client-server program that helps you protect valuable information on client workstations. Using the TSM backup-archive client, you can maintain backup versions of workstation files that can be restored quickly and easily if the original files are damaged or lost. You can also archive files that are not currently needed on a client workstation, and retrieve them when necessary. In addition to the backup-archive client, TSM provides a Web backup-archive client (Web client) that permits an authorized administrator, help desk person, or end user to perform backup, restore, archive, and retrieve services on any machine. To run the Web client you need the following requirements:

- Netscape Navigator 6.0 or higher
- Netscape Navigator 4.7 with Java Runtime Environment (JRE) 1.3.1 or higher
- Microsoft Internet Explorer 5.0 or higher with JRE 1.3.1_01 or higher

You can perform multiple Web client sessions simultaneously. For example, you can perform a backup, archive, restore, and query on separate Web browsers. Do not perform such operations simultaneously on the same browser, or you may unintentionally destroy data that you need. We assume that the SMP/E software installation of the TSM is already done.

Restriction: TSM client version 5.1 cannot back up files greater than 4 GB.

To set up the TSM client, do the following:

1. Customize TSM client configuration files.
2. Install TSM client configuration files.
3. Register the TSM client at the server.
4. Start the TSM client.

Customizing the TSM client

During the installation of TSM, a sample client system options file called dsm.sys.smp is provided. This file is used to specify one or more TSM servers to contact for services, and communications options for each server. It can also include authorization options, backup and archive processing options, scheduling options, and HSM space management options. If you are a TSM authorized user, you are responsible for copying the dsm.sys.smp file to dsm.sys in your TSM installation directory, and modifying the required options in the new copy according to your needs. Required options are those that provide information the TSM client programs need to establish communication with a TSM server. You can edit your dsm.sys file as appropriate for your system.

Attention: If you are reinstalling TSM, do not copy the dsm.sys.smp file to dsm.sys if you have already modified your dsm.sys file and do not want it overwritten.

If you are a user and want to use different options than those specified in the default client user options file, you can create your own client user options file. There you can set options that determine which formats to use for date, time, numbers, a language option, and options that affect backup, archive, restore, and retrieve processing. Figure 11-15 shows an example of dsm.opt.

```
**********************************************************************
* Tivoli Storage Manager                                             *
*                                                                    *
* Client User Options file for z/OS UNIX                             *
**********************************************************************

SErvername       WTSCMXA
```

Figure 11-15 Sample of dsm.opt

Figure 11-16 shows an example of a client system options file that contains options for a server you want users to be able to contact. You can specify options for more than one server. This file is required for communication.

```
**********************************************************************
* Tivoli Storage Manager                                             *
*                                                                    *
* Client System Options file for z/OS UNIX                           *
**********************************************************************
SErvername       WTSCMXA
   COMMmethod         TCPip
   TCPPort            1500
   TCPServeraddress   wtscmxa.itso.ibm.com
   PasswordAccess     Generate
   PasswordDir        /var/tsm
   NODename           WTSC64OE
   SCHEDLOGName          /var/tsm/dsmsched.log
   SCHEDLOGRETENTION 14 D
   ERRORLOGName          /var/tsm/dsmerror.log
   ERRORLOGRETENTION 14 D
   INCLexcl           //'NIGELR3.TSM.PARMLIB(WTSC64OE)'
   TCPBuffsize 512
   TCPWindowsize 640
```

Figure 11-16 Sample of dsm.sys

You can specify some environment variables to control the location of the TSM configuration files. Table 11-2 lists all available environment variables that you can use with the TSM client.

Table 11-2 TSM client environment variables

Variable	Explanation
DSM_DIR	Points to the executable file dsmtca, the resource files, and the dsm.sys file. You cannot specify the root directory for DSM_DIR. If DSM_DIR is not set, the executables are expected in the installation directory.
DSM_CONFIG	Points to the client user options file for users who create their own personalized options file. You cannot specify the root directory for DSM_CONFIG. If DSM_CONFIG is not set, the options file is expected in the directory pointed to by DSM_DIR. If DSM_DIR is not set, the options file is expected in the installation directory. On Solaris, dsm.sys and dsm.opt are symbolic links to the actual files stored in /usr/bin which prevents the deletion of these files in the event that TSM is uninstalled.

Variable	Explanation
DSM_LOG	Points to the directory where you want the dsmerror.log file to reside. You cannot specify the root directory for DSM_LOG. The error log file contains information about any errors that occur during processing. The error log is intended for IBM service personnel to help you diagnose severe errors.

```
# ======================================================================
#                     TSM environment variables
#                     -------------------------
# Specifies the ADSM (TSM) options locations
# ======================================================================
DSM_DIR=/var/tsm
DSM_CONFIG=/var/tsm/dsm.opt
DSM_LOG=/var/tsm
export DSM_DIR DSM_CONFIG DSM_LOG
```

Figure 11-17 Sample start environment definitions

Register TSM client at the server

Each node must be registered with the TSM server and requires an option file with a pointer to the server. Before a user can request TSM services, the node must be registered with the server. Closed registration is the default at installation. The administrator must register client nodes when registration is set to closed. Open registration allows the client nodes to register their node names, passwords, and compression options. The following is a sample of the **register** command:

```
register node WTSC64OE xxxxxxx
```

Starting the TSM client

We recommend that you start the TSM client in a separate address space, as shown in Figure 11-18. This has many advantages:

- Better performance
- Better control of the address space via OPCs

```
//TSMCLNT PROC
//*********************************************************************
//*  TSM client start procedure                                       *
//*********************************************************************
//TSM      EXEC PGM=BPXBATSL,REGION=128M,TIME=1440,
//         PARM='PGM /usr/lpp/Tivoli/tsm/client/ba/bin/dsmc sched'
//SYSIN    DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//
```

Figure 11-18 Sample stc for the TSM client

Note: It is beyond the scope of this book to provide full details of the TSM product. For further information, see *Tivoli Storage Manager for MVS and OS/390: Quick Start,* GC35-0376. All TSM manuals are available in PDF and HTML format on the World Wide Web via URL:

http://www.tivoli.com/tsm/

Many redbooks have also been written about TSM. Many of them were written when the product was known as ADSM, but these are still relevant to the TSM product. To view or order TSM redbooks, search for ADSM or TSM at:

http://www.redbooks.ibm.com

11.3.4 Physical copying of file systems

Since z/OS V1R3 you have the possibility to copy an HFS file system directly with ADRDSSU. DFDSSsms will automatically send a QUIESCE command before the copy process starts.

```
//LUTZCOPY JOB ,'HFS COPY',NOTIFY=LUTZ,
//          CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),TIME=1440
//DUMP      EXEC PGM=ADRDSSU
//SYSPRINT DD   SYSOUT=*
//SYSIN     DD   *
 COPY                                           -
     DS(INCLUDE( LUTZ.HOME.HFS ))               -
     RENAMEU( LUTZ.HOME.HFS, LUTZ.HOME.COPY.HFS ) -
     FASTREP( PREFERRED )                       -
     CONCURRENT                                 -
     CATALOG
/*
//
```

Figure 11-19 Sample JCL for copying an HFS

11.4 Monitoring z/OS UNIX

In this section we describe some of the monitoring functions that are at your disposal. Although there are some ISV products on the market that can be used to monitor your z/OS UNIX environment, we only describe basic z/OS functionality.

11.4.1 Resource Measurement Facility (RMF)

The Resource Measurement Facility (RMF) collects data used to describe z/OS UNIX performance. It monitors the use of resources in an *OMVS Kernel Activity Report*. But in addition to that, the OMVS Process Data (OPD) report within the RMF Monitor III can be a good tool for problem determination.

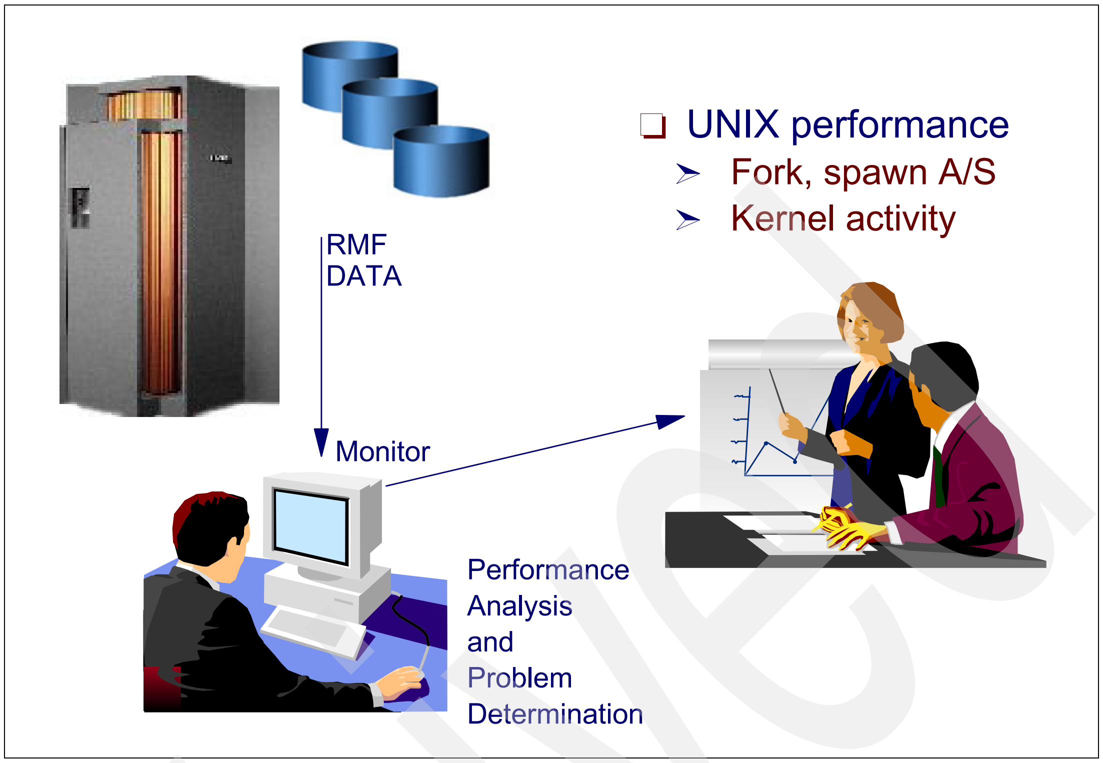


Figure 11-20 Resource Measurement Facility (RMF)

OPD - OMVS Process Data Report

To enter the OPD report, select option 3 “Interactive performance analysis with Monitor III” on the main RMF performance management panel. From there press 1 for the RMF overview report panel. Finally press 7 for the “OMVS process data” report. A more direct way to enter this report is by entering the following command from within the monitor III panel:

```
OPD
```

The report shown in Figure 11-21 on page 456 can help you do a performance analysis on UNIX System Services (USS). Each process in USS is associated with a UNIX command, state information, and CPU consumption. The OPD report can assist you in answering your questions regarding the USS processes, by providing basic performance metrics on the first screen.

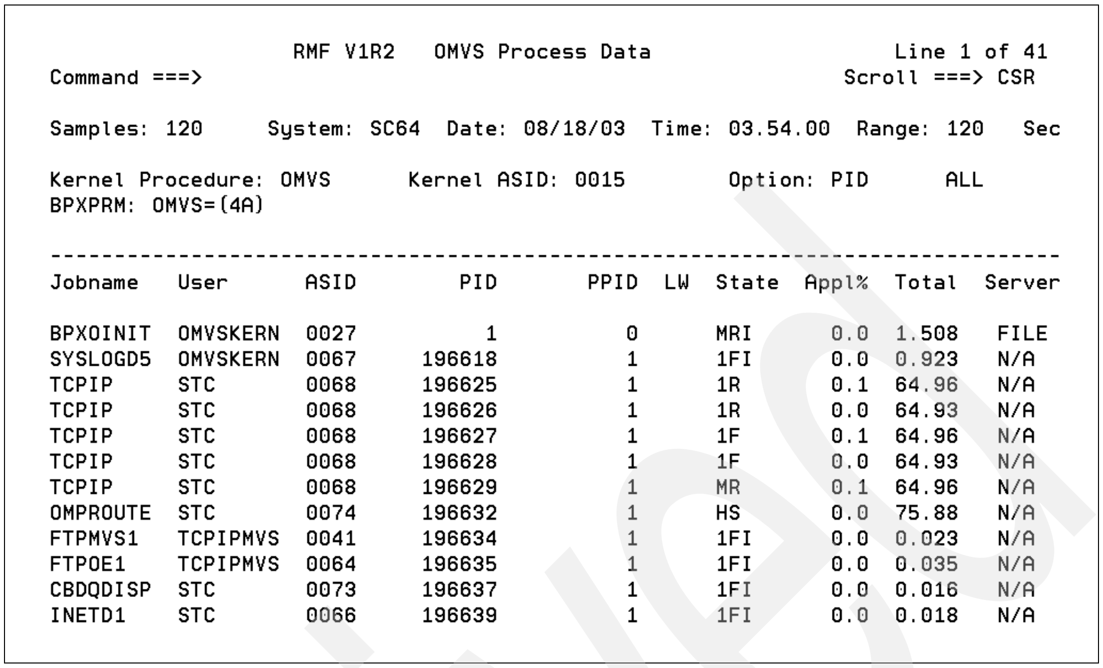

```
                   RMF V1R2   OMVS Process Data                   Line 1 of 41
Command ===>                                                  Scroll ===> CSR

Samples: 120     System: SC64  Date: 08/18/03  Time: 03.54.00  Range: 120   Sec

Kernel Procedure: OMVS      Kernel ASID: 0015       Option: PID      ALL
BPXPRM: OMVS=(4A)

-------------------------------------------------------------------------------
Jobname   User      ASID        PID       PPID  LW  State  Appl%  Total  Server

BPXOINIT  OMVSKERN  0027          1          0      MRI      0.0  1.508   FILE
SYSLOGD5  OMVSKERN  0067     196618          1      1FI      0.0  0.923   N/A
TCPIP     STC       0068     196625          1      1R       0.1  64.96   N/A
TCPIP     STC       0068     196626          1      1R       0.0  64.93   N/A
TCPIP     STC       0068     196627          1      1F       0.1  64.96   N/A
TCPIP     STC       0068     196628          1      1F       0.0  64.93   N/A
TCPIP     STC       0068     196629          1      MR       0.1  64.96   N/A
OMPROUTE  STC       0074     196632          1      HS       0.0  75.88   N/A
FTPMVS1   TCPIPMVS  0041     196634          1      1FI      0.0  0.023   N/A
FTPOE1    TCPIPMVS  0064     196635          1      1FI      0.0  0.035   N/A
CBDQDISP  STC       0073     196637          1      1FI      0.0  0.016   N/A
INETD1    STC       0066     196639          1      1FI      0.0  0.018   N/A
```

Figure 11-21   OPD report

You can get a more detailed overview of the process status by placing your cursor on it and pressing Enter. See Figure 11-22.

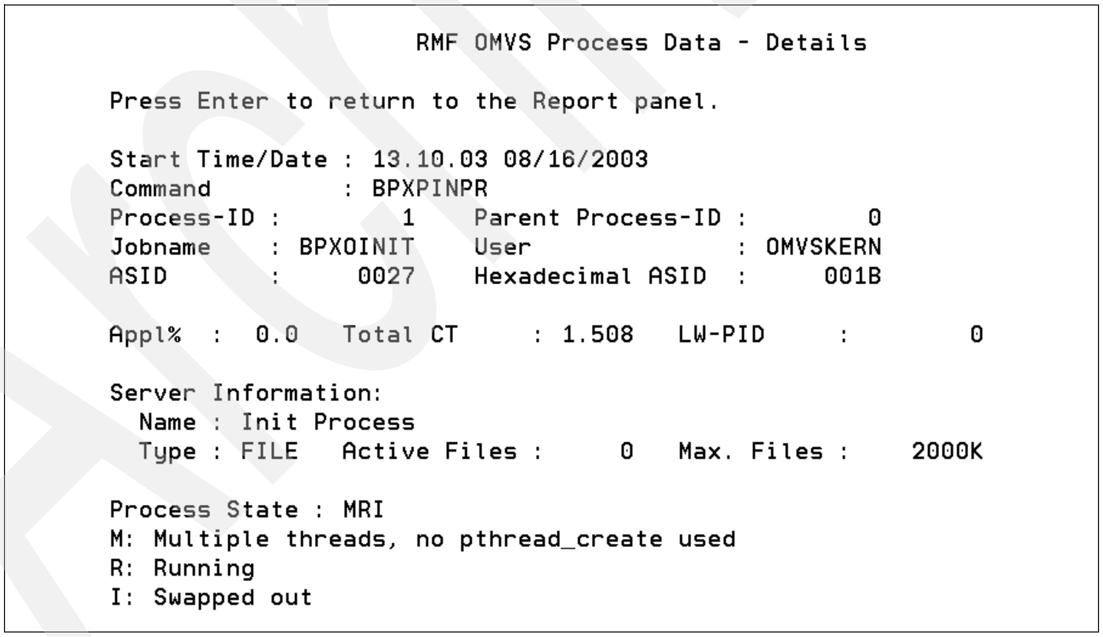
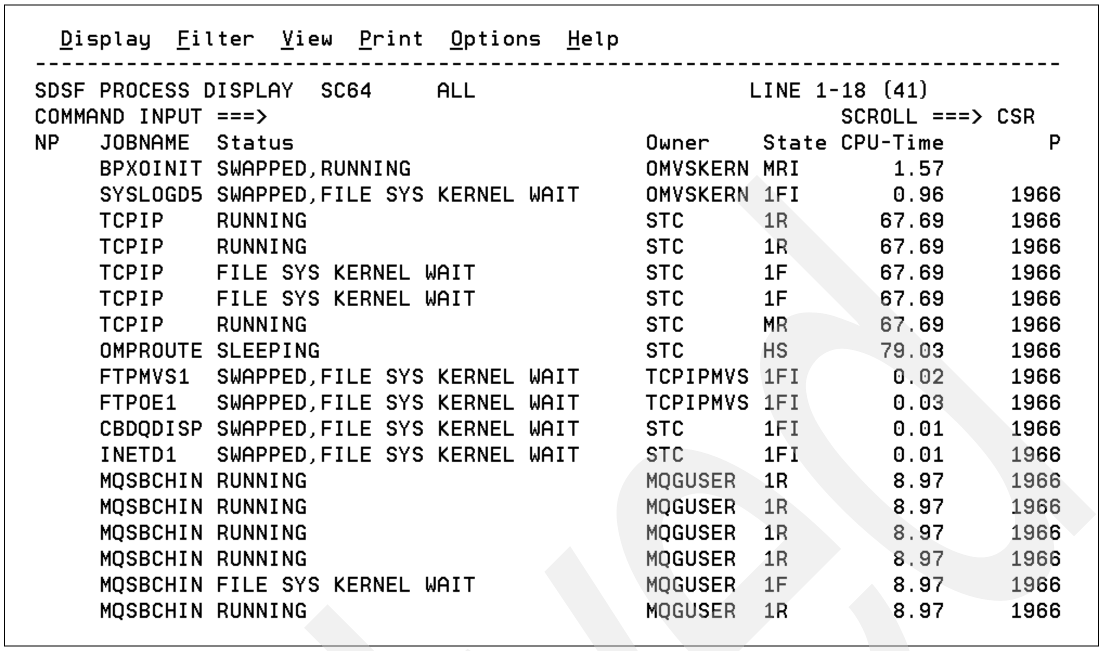

```
                    RMF OMVS Process Data - Details

    Press Enter to return to the Report panel.

    Start Time/Date : 13.10.03 08/16/2003
    Command         : BPXPINPR
    Process-ID :        1    Parent Process-ID :        0
    Jobname    : BPXOINIT    User              : OMVSKERN
    ASID       :     0027    Hexadecimal ASID  :     001B

    Appl%  :  0.0   Total CT     : 1.508   LW-PID     :        0

    Server Information:
      Name : Init Process
      Type : FILE   Active Files :     0   Max. Files :    2000K

    Process State : MRI
    M: Multiple threads, no pthread_create used
    R: Running
    I: Swapped out
```

Figure 11-22   OPD detailed process report

11.4.2 SDSF process panel

The process panel (PS) in SDSF makes it easier to manage z/OS UNIX processes. As shown in Figure 11-23 on page 457, the panel shows information about each process and supports the use of action characters to display (D) and cancel (C) a specific process.



```
  Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF PROCESS DISPLAY  SC64     ALL                    LINE 1-18 (41)
COMMAND INPUT ===>                                           SCROLL ===> CSR
NP   JOBNAME  Status                         Owner    State CPU-Time        P
     BPXOINIT SWAPPED,RUNNING                OMVSKERN MRI       1.57
     SYSLOGD5 SWAPPED,FILE SYS KERNEL WAIT   OMVSKERN 1FI       0.96     1966
     TCPIP    RUNNING                        STC      1R       67.69     1966
     TCPIP    RUNNING                        STC      1R       67.69     1966
     TCPIP    FILE SYS KERNEL WAIT           STC      1F       67.69     1966
     TCPIP    FILE SYS KERNEL WAIT           STC      1F       67.69     1966
     TCPIP    RUNNING                        STC      MR       67.69     1966
     OMPROUTE SLEEPING                       STC      HS       79.03     1966
     FTPMVS1  SWAPPED,FILE SYS KERNEL WAIT   TCPIPMVS 1FI       0.02     1966
     FTPOE1   SWAPPED,FILE SYS KERNEL WAIT   TCPIPMVS 1FI       0.03     1966
     CBDQDISP SWAPPED,FILE SYS KERNEL WAIT   STC      1FI       0.01     1966
     INETD1   SWAPPED,FILE SYS KERNEL WAIT   STC      1FI       0.01     1966
     MQSBCHIN RUNNING                        MQGUSER  1R        8.97     1966
     MQSBCHIN RUNNING                        MQGUSER  1R        8.97     1966
     MQSBCHIN RUNNING                        MQGUSER  1R        8.97     1966
     MQSBCHIN RUNNING                        MQGUSER  1R        8.97     1966
     MQSBCHIN FILE SYS KERNEL WAIT           MQGUSER  1F        8.97     1966
     MQSBCHIN RUNNING                        MQGUSER  1R        8.97     1966
```

Figure 11-23   Process panel in SDSF

You have to be specifically authorized to use the PS command, as well as for the action characters cancel (C) and display (D). The PS panel is very wide and can be best viewed by using the PF10 and PF11 keys to scroll left and right. The usual PRE and OWNER commands are available to set all sorts of filters on your PS panel, and PF7 and PF8 are well known for scrolling up and down.

Using the action character (D) on a process will show you some additional information concerning that process. See Figure 11-24.

```
RESPONSE=SC64
 BPXO040I 06.40.18 DISPLAY OMVS 283
 OMVS     000F ACTIVE          OMVS=(4A)
 USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
 PATRICK  PATRICK1 0026   50528312   67305527 1R---- 06.40.04      1.59
   LATCHWAITPID=        0 CMD=java sample01
  THREAD_ID        TCB@     PRI_JOB  USERNAME   ACC_TIME SC  STATE
  2199160000000000 007E0E88                        1.565 PTQ  RU
```

Figure 11-24   Display process action character in the PS panel

Note: To enable this feature, sysplex-wide WebSphere MQ is required. SDSF uses MQ for communication between SDSF servers. Keep in mind that this support also requires the z/OS V1R2 SDSF server.

12

Tuning and performance

This chapter provides information about tuning and performace of HFS and zFS.

We discuss the following topics:

- A comparison of HFS and zFS file system performance
- A Lotus Domino performance study, HFS versus zFS

12.1 HFS and zFS file system comparison

We created a simple test scenario to compare large file access behavior between HFS and zFS. Following are the steps that were performed to prepare and perform the tests.

12.1.1 zFS cache sizes

zFS was started with the cache sizes shown in Figure 12-1. The results are shown as ZFS1 in Table 12-1 on page 464.

```
*********************************************************************
* zSeries File System (zFS) Sample Parameter File:  ioefsprm
...
*adm_threads=5
*auto_attach=ON
user_cache_size=256M
log_cache_size=64M
sync_interval=60
*vnode_cache_size=5000
nbs=off
*fsfull(85,5)
*aggrfull(90,5)
...
```

Figure 12-1 zFS cache sizes defined in IOEFSPRM

HFS cache size

We started with the HFS cache size value shown in Figure 12-2.

```
$> /usr/lpp/dfsms/bin/confighfs -l
HFS Limits
 Maximum virtual storage: _______751(MB)
 Minimum fixed storage:   _________0(MB)
```

Figure 12-2 Querying the HFS cache limit

zFS cache size increased

We increased the zFS user file cache to 384 MB later, which provided better numbers for zFS. We also increased the cache size value for HFS and found no benefit in our test scenario. Nevertheless, we used this higher cache size during the tests that we discuss here. We did not use fixed storage for HFS and zFS. Figure 12-3 shows the increasd zFS cache size used during the test as ZFS2.

```
#> /usr/lpp/dfsms/bin/confighfs -v 1500
#> /usr/lpp/dfsms/bin/confighfs -l
HFS Limits
 Maximum virtual storage: ______1500(MB)
 Minimum fixed storage:   _________0(MB)
```

Figure 12-3 Enlarging the HFS cache size

Defining the HFS file system

We defined an HFS file system large enough to be able to keep a 500 MB file. The JCL used is shown in Figure 12-4 on page 461.

```
//ZFSJOB   JOB ,'Define HFS File',NOTIFY=&SYSUID.,REGION=0M
//* ------------------------------------------------------------------
//* Define HFS File
//* Property of IBM  (C) Copyright IBM Corp. 1999, 2002
//* ------------------------------------------------------------------
//*
//  SET  HFSNAME=OMVS.LARGE.HFS            <=== HFS data set name
//  SET   VOLSER=TOTZF2                    <=== Volume
//  SET   HFSPRM=750                       <=== HFS primary allocation
//  SET   HFSSEC=50                        <=== HFS secon'y allocation
//*
//* ------------------------------------------------------------------
//CREATE   EXEC PGM=IEFBR14
//HFS      DD DSN=&HFSNAME.,DISP=(NEW,CATLG,DELETE),UNIT=SYSALLDA,
//            DCB=(DSORG=PO),SPACE=(CYL,(&HFSPRM.,&HFSSEC.,0)),
//            DSNTYPE=HFS,VOL=SER=&VOLSER.
//* ------------------------------------------------------------------
```

Figure 12-4 Defining the HFS file system

Defining and formatting a zFS aggregate

Then we created a zFS aggregate on the same disk, as shown in Figure 12-5.

```
zfsadm define -aggregate OMVS.LARGE.ZFS -volume TOTZF2 -megabytes 550 50
IOEZ00248E VSAM linear dataset OMVS.LARGE.ZFS successfully created.
zfsadm format -aggregate OMVS.LARGE.ZFS -compat -owner HERING -perms o755
IOEZ00077I HFS-compatibility aggregate OMVS.LARGE.ZFS has been successfully created
```

Figure 12-5 Creating the zFS aggregate

Mounting the file systems

Then we mounted the file systems at directory location /u/zfs, as shown in Figure 12-6 on page 462.

```
mkdir -m 755 /u/zfs/largehfs
/usr/sbin/mount -o sync(60) -f OMVS.LARGE.HFS -t HFS /u/zfs/largehfs
chmod 755 /u/zfs/largehfs
ls -Ed /u/zfs/largehfs
drwxr-xr-x        2 HERING   SYS1        8192 Apr 19 23:24 /u/zfs/largehfs
df -kvP /u/zfs/largehfs
Filesystem         1024-blocks        Used  Available  Capacity Mounted on
OMVS.LARGE.HFS     540000               20     539908        1% /u/zfs/largehfs
HFS, Read/Write, Device:89, ACLS=Y
Filetag : T=off   codeset=0

mkdir -m 755 /u/zfs/largezfs
/usr/sbin/mount -o noreadahead -f OMVS.LARGE.ZFS -t ZFS /u/zfs/largezfs
ls -Ed /u/zfs/largezfs
drwxr-xr-x        2 HERING   SYS1         256 Apr 19 23:30 /u/zfs/largezfs
df -kvP /u/zfs/largezfs
Filesystem         1024-blocks        Used  Available  Capacity Mounted on
OMVS.LARGE.ZFS     552343                9     552334        1% /u/zfs/largezfs
ZFS, Read/Write, Device:90, ACLS=Y
Filetag : T=off   codeset=0
```

Figure 12-6 Mounting the file systems

Filling the file systems with data

Then we used a REXX procedure in a job to create and fill the file systems with a size of 500 MB in the HFS and the zFS file systems.

It took 85.4 seconds to fill the zFS file system and 155.9 seconds to fill the HFS file system.

Accessing the file systems

Here is a description of how the I/O test was done and what the selection criteria were to test access to the file systems.

- Choose either the zFS or the HFS file system.
- Define the number of processes that run in parallel to read or write 1 MB blocks of data.
- Specify the number of I/Os that each process has to perform.
- Select the percentage of reads among all I/Os. R70, shown in Figure 12-7 on page 463, means that 70% of the I/Os are reads and 30% are writes.
- Provide a seed value. This is used to create predictable random numbers for offsets into the large file when doing an I/O access.

The JCL for the job used, shown in Figure 12-7 on page 463, is where you specify the selection criteria for accessing the file system.

```
//ZFSJOB   JOB ,'LARGEIOS',NOTIFY=&SYSUID.,REGION=0M
//* ------------------------------------------------------------------
//* Run test with doing a specified amount of large random I/Os
//* Property of IBM  (C) Copyright IBM Corp. 2002
//* ------------------------------------------------------------------
//  SET    FDIR='/u/zfs/largezfs'         <=== Directory of large_file
//  SET   PROCS=10        <=== Number of parallel processes to start
//  SET  BLKIOS=200       <=== Number of 1MB block I/Os to perform
//  SET  TYPIOS=R70       <=== Type of IOs, Rxx: xx% Rs, (100-xx)% Ws
//  SET    SEED=88888     <=== Seed value for predictable random nbrs
//  SET TIMEOUT=0         <=== Timeout value in seconds, 0=no timeout
//  SET REXXLIB=HERING.ZFS.REXX.EXEC              <=== SYSEXEC library
//* ------------------------------------------------------------------
//ZFSADM   EXEC PGM=IKJEFT01,
// PARM='LARGESCD &TIMEOUT &REXXLIB &PROCS &BLKIOS &TYPIOS &SEED &FDIR'
//SYSEXEC  DD DSNAME=&REXXLIB.,DISP=SHR
//SYSTSIN  DD DUMMY
//SYSTSPRT DD SYSOUT=*,LRECL=260,RECFM=V
//* ------------------------------------------------------------------
```

Figure 12-7 Job to run a specified amount of large random I/Os

12.1.2 Comparison of results

We always used TYPEIOS=R70 and tested with 10, 20, and 40 processes running in parallel. With the HFS, we did not change the cache at all during the tests. With zFS, the cache still had good data in it after doing a number of I/Os. So we decided to always run a test twice for zFS to demonstrate how this may change response times. Afterwards we stopped and restarted zFS to be sure the cache was invalidated. It turned out that good cache data may help for a second similar test up to about 20 processes involved.

Note: We decided not to show the test results for when the cache was invalidated, in the table.

Random number specification

For the predictable random numbers, we used a seed value of 88888 for the 10 processes tests, a value of 76543 for the 20 processes tests, and a value of 66666 for the 40 processes tests just to produce some different patterns.

During the tests we also had a look at the CPU usage and found the following effects:

- For both HFS and zFS, all processes use up about the same CPU resources. But in case of zFS, the value is about 100% higher.
- As the zFS processes only used 50% of the time that the HFS processes did, both types used up about the same amount of CPU resources. See 12.2, “Domino and zFS performance” on page 464 for details on this.

Results

In Table 12-1 on page 464, ZFS1 indicates the results with a user cache size of 256 MB, which is the default. ZFS2 indicates the results using a cache size of 384 MB. The zFS address space was restarted to clear or invalidate the cache before running with a new cache size.

Table 12-1 Results

	Processes	AVG secs	MIN secs	MAX secs
HFS	10	499.26	504.42	506.04
ZFS1	10	254.21	239.78	261.59
ZFS2	10	193.15	185.33	197.44
HFS	20	992.09	960.84	1003.27
ZFS1	20	486.69	465.67	499.93
ZFS2	20	311.28	284.40	325.54
HFS	40	1973.64	1893.17	2005.07
ZFS1	40	939.01	848.78	979.64
ZFS2	40	590.29	516.23	623.42

12.2 Domino and zFS performance

DFSMS has provided the only local physical file system, HFS, available for OS/390 and z/OS since UNIX System Services was first released, that hardens data to DASD. It was rewritten with the DFSMS 1.5 release that shipped with OS/390 V1R6 to enhance performance, and this provided a substantial performance improvement over previous releases. It did not, however, cache reads or writes for Domino data because the design point for DFSMS 1.5 was to cache only those files that are 1 MB or less in size, and practically all Domino databases are larger than this limit.

zFS has no such restrictions on caching, and has mechanisms that allow administrators to control how the file system caches I/O. It provides the administrator with more flexibility to apply more system resources (primarily storage) to any operations that are performance sensitive. It is the purpose of this study to explore these advantages so that administrators can make informed decisions about zFS in their environments.

The Domino server uses UNIX files for many different purposes, and the characteristics of the I/O for each of these purposes are different. We examine the performance of the two file systems for each of these types of I/O to illustrate the strengths and weaknesses of each. We also show how these differences manifest themselves to the end user.

12.3 The Domino server environment

The test environment for this study simulates an enterprise server domain. The file system structure for the environment is shown in Figure 12-8 on page 465.

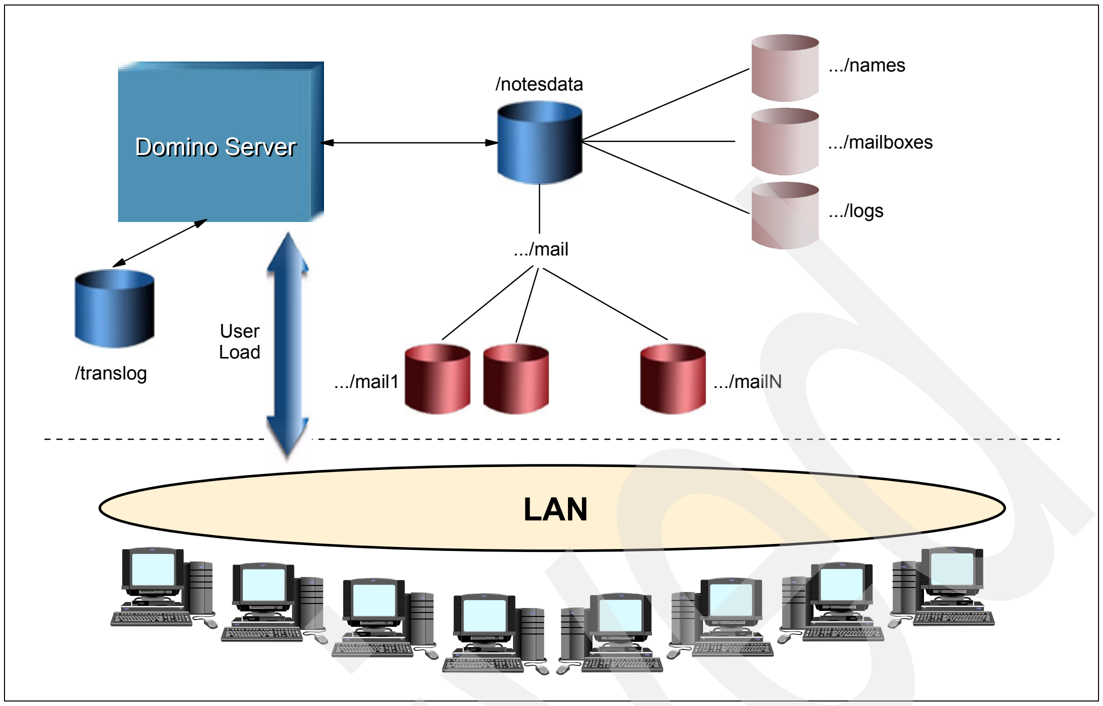


Figure 12-8 File system structure for the test environment

Many of the file systems in a Domino environment have a common purpose and can be grouped together according to the usage patterns that the server drives against them:

/translog The Domino transaction log1. All I/O to a Domino database passes through the transaction log in a two-phased commit that provides some modest performance advantage and better recovery characteristics for the server. I/O to and from the transaction log is sequential, with a small number of logger threads performing all of the I/O. The transaction log resides in a file system as a series of files, all about 64 MB in size, plus a small control file. The logger processes these files using a small number of threads (2 or 3, depending on the configuration of the logger). Transaction logging can run in circular, linear, or archive mode, but these modes have largely the same I/O characteristics—data is written sequentially to one of the logging files until the file fills up, when the logger moves on to the next file. A separate thread reads these logger files and writes the data it reads to the appropriate target database. Although contention for these logger files is low, the I/O rates to these files are very high. We strongly recommend that you put the transaction log in a separate file system, on a separate DASD device, and have plenty of channel bandwidth to the device.

/names The Name and Address Book (NAB, now called the Domino Directory) for the domain. This one database completely describes the Domino domain—from user and server identities to ACLs to data routing information. I/O to this file tends to be random, with a potentially large number of users attempting to perform I/O at the same time. I/O is primarily read, with few write operations. The Domino Directory is used to authenticate every client request made of the server, every server-to-server interaction, and the execution of all Domino applications that run on the server. It is the common resource for all access control requests required of the server. This usually makes the Directory the common resource most contended for by the server. For this reason, it is often advisable to treat it like the transaction log, that is, isolate it from the rest of the /notesdata directory and file system.

/notesdata The notes data directory. This is the home directory for the Domino server, which contains most of the common resources used for Domino server operation. I/O to files in this file system tends to be random, with potentially many threads attempting to perform I/O simultaneously. This file system usually contains the central mail boxes (mail.box*) used as a clearing house for all mail routed to users on that server. It also often contains the Name and Address Book for the server as well. Our tests have the NAB residing in both a separate file system and the Notes data directory, depending on the object of the test.

/logs The server logs. These contain console data from the running server. I/O here tends to be sequential in nature with a small number of writers, and occasional readers. This data is often separated into a discrete file system because of space considerations, and not to avoid contention issues. I/O is primarily write, with relatively few read operations.

/mailx The mail file systems. These hold databases for mail users, the workload most often supported by a Domino server. I/O here is random, generally with a 1-to-1 relationship between thread/user and database. Contention for a mail database is not often an issue. These databases tend to range in size from 50-250 MB, although some power users have mail databases in the gigabyte range. Because of space management issues for these databases, spreading these files over several file systems is the most practical option. Reads and writes generally tend to be about evenly proportioned.

12.3.1 Tasks performed by the Domino server

The Domino server performs various tasks that fall into two broad classifications: offline housekeeping functions, and servicing client requests. Each class of tasks causes a different type of stress on the server.

Offline housekeeping functions

The offline housekeeping functions are used to perform maintenance and migration operations on Domino databases. These functions are usually performed when the server is down, either during scheduled maintenance windows, or during migrations from other servers. They generally require a single thread of execution, and operate on a single database. The functions measured during this study include:

- Compact
  - Copy-style compaction (-c)

    A temporary copy of the database is made during the compact process, and any required structural changes are performed. This flavor of compact recovers space internally within the database, and reduces the size of the file.
  - In-place compaction without file reduction (-b)

    Compact a database “in place”. No temporary copy is required. Space is recovered internally within the database, but no reduction in actual file size occurs.
  - In-place compaction with file reduction (-B)

    Compact the database “in place”. No temporary copy is required. Space is recovered internally within the database, and the size of the file is reduced.
- Updall

  Rebuild all full-text indexes and all used/unused views in the database from scratch (-RC).
- Create a full-text index (FTI)

  Create a new full text index to facilitate keyword searches in a target database.

Note: For more detailed explanations of these functions, see the Domino 5 Administration Help database.

These functions were chosen for this test because they are key migration steps whenever moving from one feature release of the Domino server to another. They often provide the first impression of performance that an administrator has of the platform, and they have historically been areas where performance has been an issue.

We measured the performance of these functions while they were being performed against the IBM US Directory (NAB). When compacted to bring this database to On Disk Structure 41 format (the standard R5 ODS level), this database is approximately 1.24 GB in size, with 191,000 person documents, 1850+ groups, and 1900+ connection documents for other purposes.

Client-driven workloads

These workloads are primarily standard benchmark applications that simulate different flavors of client access to mail. They create many virtual threads of execution to drive client access to many different databases served by Domino. As with the common housekeeping functions, there is largely a 1-to-1 relationship between virtual thread and target database. The difference is that there are large numbers of these virtual threads running simultaneously.

Virtual threads are a Domino abstraction, used to represent each client request being serviced. They provide a context within which the server takes actions on behalf of the client against a specified database. Rather than allocating a UNIX pthread for each virtual Domino thread, the server maintains a relatively small pool of pthreads that are all driven by a small dispatch routine. This routine essentially causes each pthread to sleep until a client request arrives for servicing. When this happens, one of the pthreads is bound to a virtual Domino thread, and the conversation between client and server proceeds.

What all this means from an I/O standpoint is that although the server may appear to be servicing thousands of clients simultaneously, there are generally not more than about 50 physical threads of execution within the core Domino server that are being managed by z/OS at any one instant in time. Still, these physical threads that are running tend to be very active, and there are sufficient numbers of them to drive interesting amounts of I/O, particularly to common files, such as the Domino directory, and the files that make up the transaction log. During our tests here, we allocated 100 pthreads to the Domino thread pool.

The workloads used for this test that can be considered client-driven workloads include:

- R5Mail - This is standard R5 mail clients accessing their mail.

  This is a simulation of what is considered a “light” mail user. Each user performs the following tasks in an average 15-minute interval:

  - Opens the inbox view
  - Reads five documents
  - Categorizes two documents
  - Sends a message of 4000 bytes to six recipients
  - Adds two documents to the inbox
  - Schedules an appointment with a description of 4000 bytes
  - Sends a meeting invitation of 4000 bytes to six recipients
  - Deletes two documents from the inbox

– Responds to a meeting invitation

– Closes the view

Table 12-2 Workload parameters for R5Mail

Parameter	Default value	Value for this test	Meaning
NormalMessageSize	1,000	4,000	Size of a message or invitation, in bytes
NumMessageRecipients	3	6	Number of users to send mail or invitations to others
NthIteration	6	1	Number of loops through test before sending mail or invitations to others

By setting these three parameters to values that are higher than the defaults, as shown in Table 12-2, we substantially increased the amount of I/O required to support a single user, and therefore the stress on the underlying file system.

► Webmail - This is Web users accessing their mail.

This is a simulation of a “light” mail user browsing their mail using a regular Web browser. The first difference to note here is that no calendaring and scheduling functions (meetings) are exploited. Like the R5mail workload, this test works on a 15-minute interval to:

– Send a message of 1000 bytes to 5 recipients.

– Delete a document.

– Read 5 documents.

As with the R5Mail workload, we tuned the parameters of the test to make the workload more I/O intense, as shown in Table 12-3.

Table 12-3 Workload parameters for Webmail

Parameter	Default value	Value for this test	Meaning
NormalMessageSize	1,000	1,000	Size of a message, in bytes
NumMessageRecipients	3	5	Number of users to send mail to others
NthIteration	6	1	Number of loops through the test before sending mail to others

► Inotes - This is Web users accessing their mail via the iNotes™ Web Access interface.

This is an improved interface over the usual Web mail interface. It provides more features as well as a measurably lighter load on the server, both in terms of CPU consumption and I/O required. The functionality performed for this test is identical to the Webmail workload, except that the functions drive a different path through the Domino HTTP server when exploiting the iNotes Web Access interface. Given the lighter I/O workload of this test, we changed the size of the messages read and sent, as shown in Table 12-4.

Table 12-4 Workload parameters for Inotes

Parameter	Default value	Value for this test	Meaning
NormalMessageSize	1,000	4,000	Size of a message, in bytes
NumMessageRecipients	3	5	Number of users to send mail to others

Parameter	Default value	Value for this test	Meaning
NthIteration	6	1	Number of loops through test before sending mail to others

Production Domino applications

During this study, we also tested two custom Domino applications to see how zFS would impact their response times and throughput characteristics. These applications were from two Domino for S/390 customer production environments.

Our interest in the first application was to simply measure the elapsed time of a single thread of execution performing operations on a database. The second application had a requirement to support both a given throughput and response time for a set of concurrent users processing a single database. We generated a workload of 70 concurrent users against the database containing the application using LoadRunner1, and a custom workload script.

12.3.2 Test results

The test results indicate that in virtually all scenarios, zFS outperforms HFS by a substantial margin. We believe that this is due in large part to the fact that zFS is able to cache Domino data where HFS cannot. We understand from the zFS and HFS development teams that the design point for HFS (DFSMS 1.5) was to cache files of up to 1 MB, but larger files would not be cached. Since all Domino databases are larger than 1 MB in size, HFS is at an inherent performance disadvantage relative to zFS.

Offline housekeeping functions

Table 12-5 shows the measurements for all of the offline housekeeping functions, described in “Offline housekeeping functions” on page 466.

Table 12-5 Measurements for the offline housekeeping functions

Function	HFS Elapsed Time (secs)	zFS Elapsed Time (secs)	Difference	HFS:zFS
compact -c	2,631	1,065	-60%	2.47:1
compact -b	405	166	-59%	2.44:1
compact -B	3,179	1,548	-51%	2.05:1
updall -RC	17,553	5,490	-69%	3.20:1
create FTI	4,463	3,300	-26%	1.35:1

Although we don't have any specific numbers to compare in this study, we made measurements showing that Domino with zFS will complete these housekeeping functions about 10% faster than a similarly configured Intel® machine running Windows NT4.

Note: The CPU and storage requirements for the two file systems varied considerably, as shown in Table 12-6 and Table 12-7.

Table 12-6 Measurements for the offline housekeeping functions

Function	HFS CPU usage (percent)	zFS CPU usage (percent)	Difference	HFS:zFS
compact -c	11.38	18.63	+64%	1:1.64

Function	HFS CPU usage (percent)	zFS CPU usage (percent)	Difference	HFS:zFS
compact -b	3.24	5.30	+64%	1:1.64
compact -B	4.23	12.02	+184%	1:2.84
updall -RC	10.07	27.22	+170%	1:2.70
create FTI	28.50	15.83	-44%	1.80:1

Table 12-7 Measurements for the offline housekeeping functions

Function	HFS Central Storage usage (MB)	zFS Central Storage usage (MB)	Difference	HFS:zFS
compact -c	704	1620	+130%	1:2.30
compact -b	623	1599	+157%	1:2.57
compact -B	639	1614	+153%	1:2.53
updall -RC	858	1949	+127%	1:2.27
create FTI	712	1822	+156%	1:2.56

Although zFS resource usage is significantly higher here, it didn't have to be. The purpose of this test was to explore the best performance measurements achievable with both types of file systems. zFS could have been tuned to match the storage resources of HFS, and we could have gathered more CPU and elapsed time measurements for comparison. The key point illustrated by this data is that although HFS and zFS can be configured to use the same resources, zFS is capable of applying more available system resources to the tasks it is asked to perform in order to enhance performance.

The data showing I/O to DASD (see Figure 12-9) indicates just how effective zFS' caching is at preventing I/O requests from traveling all the way out to the hardware.

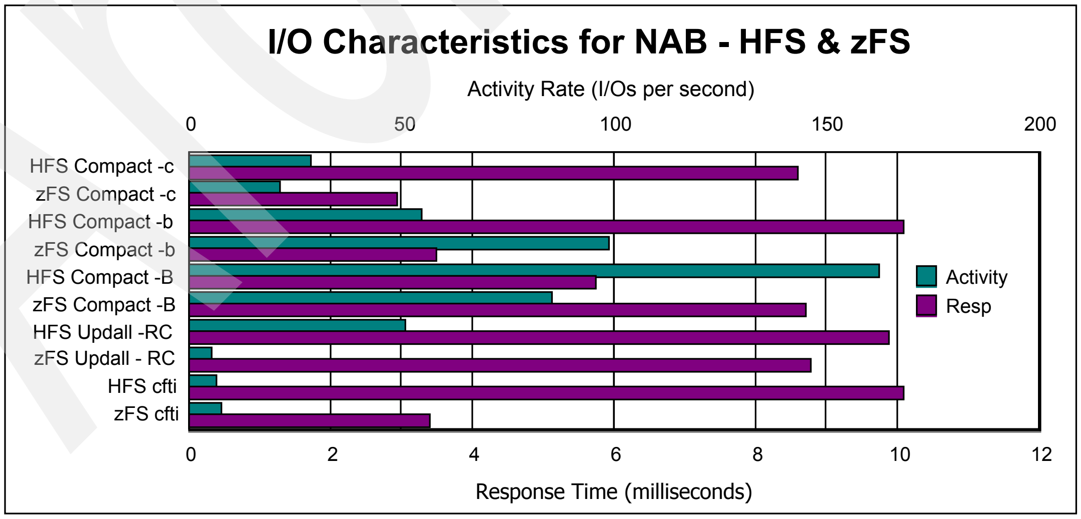


Figure 12-9 I/O characteristics for NAB between HFS and zFS

Each pair of bars indicates the activity rate and response time for the DASD supporting the file system containing the NAB. For instance, one can see from the chart that when running updall to rebuild views and indexes for the IBMUS NAB, the DASD had much less I/O activity (51.1 I/Os per sec. for HFS vs. 5.6 for zFS), and marginally better response time (9.9 msec for HFS vs. 8.8 msec for zFS). What is interesting is that for any given task, either the activity rate or the response time for the DASD associated with the zFS measurement may actually be higher than for zFS; however, in all cases, the overall I/O intensity (activity rate multiplied by response time) is better for zFS than for HFS.

Table 12-8 Measurements for the offline housekeeping functions

Function	HFS I/O Intensity	zFS I/O Intensity	Difference	HFS:zFS
compact -c	281	76	-73%	3.70:1
compact -b	557	347	-38%	1.61:1
compact -B	843	641	-24%	1.32:1
updall -RC	552	52	-91%	10.62:1
create FTI	66	28	-58%	2.36:1

A closer look at the DASD response time reveals some details about how HFS and zFS perform their I/O to the physical device; see Figure 12-10.

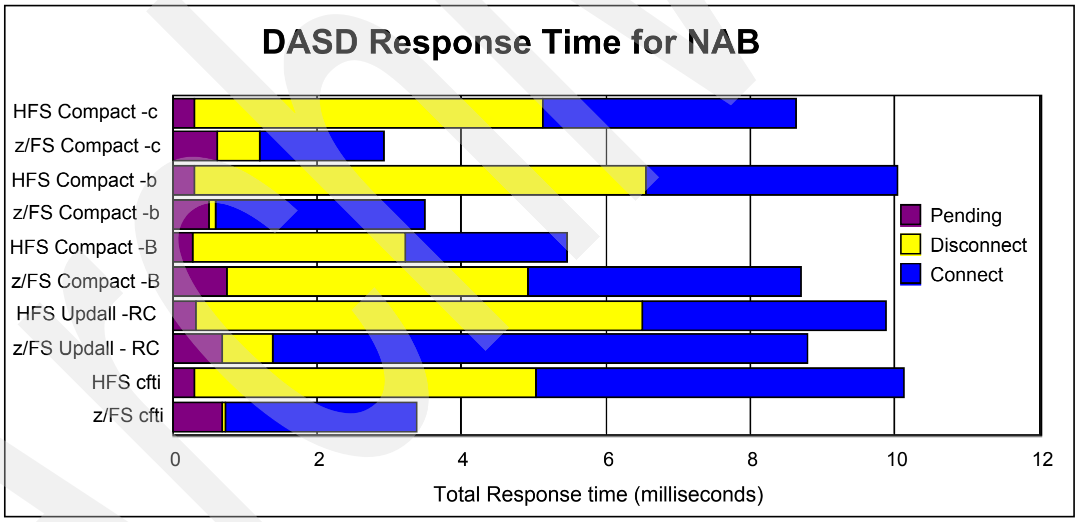


Figure 12-10 DASD response time for NAB

There was no I/O service queue (IOSQ) time component to the response time for either the HFS or zFS measurements, which is to be expected, since there was only one thread of execution trying to use this file and its associated file system at a time. Several general patterns emerge from this chart:

- zFS has a slightly higher pending time in all cases, indicating longer waits in the path out to the device. This is likely due to a configuration reality of the environment, since the HFS, and zFS file systems were defined on different DASD devices, with different channel path characteristics. We don't believe this to be caused by zFS.
- In all cases except for the compact in place with file size reduction (compact -B), disconnect time for zFS is much better than for HFS. Disconnect indicates the amount of time that an I/O request is freed from the channel due to seek, latency, or rotational delays.

Connect time—the portion of an I/O request where data is actually transferred to the device—is a higher portion of the total response time for zFS in all cases. It appears that when zFS does have to do I/O to the device, it has the ability to do so more efficiently than HFS.

12.3.3 Client-driven workloads

As described, we used three different benchmark workloads to simulate users for this portion of the evaluation: R5Mail, Webmail, and iNotes Web Access. R5Mail users drive the server through the traditional Notes RPC interface, while Webmail and iNotes Web Access users pass through Domino's HTTP server. The Notes RPC interface is inherently more efficient, since the server and client cooperate to provide functionality to the end user through a robust set of remote procedure calls. Web access to the server is somewhat more primitive in comparison, because the server is required to perform all formatting and rendering of the Web page before delivering it to the Web browser. As a result, Web-based clients are generally sized to be 3 to 4 times more CPU intensive on the server size than Notes clients.

We included both Webmail and iNotes Web Access workloads here because both drive large amounts of I/O on the server in different and important ways.

The R5Mail workload

We simulated 2,000 users for this test, all running the R5Mail workload. All of our results here were collected on a 5-minute interval for 1 hour after the steady state of the workload had been reached (all users ramped up and driving work on the server). We ran this test using zFS in three different configurations so that we could demonstrate the benefits of zFS when used for different purposes:

- All HFS

  HFS used for the transaction log, Notes data, and mail file systems.
- Mixed

  zFS used for the mail file systems, but the transaction log and Notes data remain on HFS.
- All zFS

  zFS used for the transaction log, Notes data, and mail file systems.

The response times, shown in Table 12-9, are average response times calculated by the benchmark application at the client.

Table 12-9 R5Mail workload with 2000 users

	CPU usage %	Central storage (MB)	Expanded storage (MB)	End user response time (msec)
All HFS	52.1	1,590	9 **	902
Mixed	60.4	2,063	633	308
All zFS	63.4	2,063	657	211
Diff, All HFS vs. All zFS	+22%	+30%	n/a	-77%

Note: ** There is always a small amount of expanded storage in use; 9 MB in use for HFS is effectively zero.

In this particular configuration, we had 2 GB of central storage and 1 GB of expanded storage installed. HFS used about 3/4 of available central storage, and no significant expanded storage. zFS made use of all available central storage, and a substantial amount of expanded storage, presumably for caching purposes. The payoff is a large reduction in end-user response time. As with the offline housekeeping functions above, we could have configured zFS to match the HFS resource usage, and traded off some end-user response time dividend for it.

DASD I/O characteristics for all of the file systems used for this test are as one might expect, with zFS driving much less I/O to the device. Figure 12-11 shows the I/O characteristics for the mail file systems.

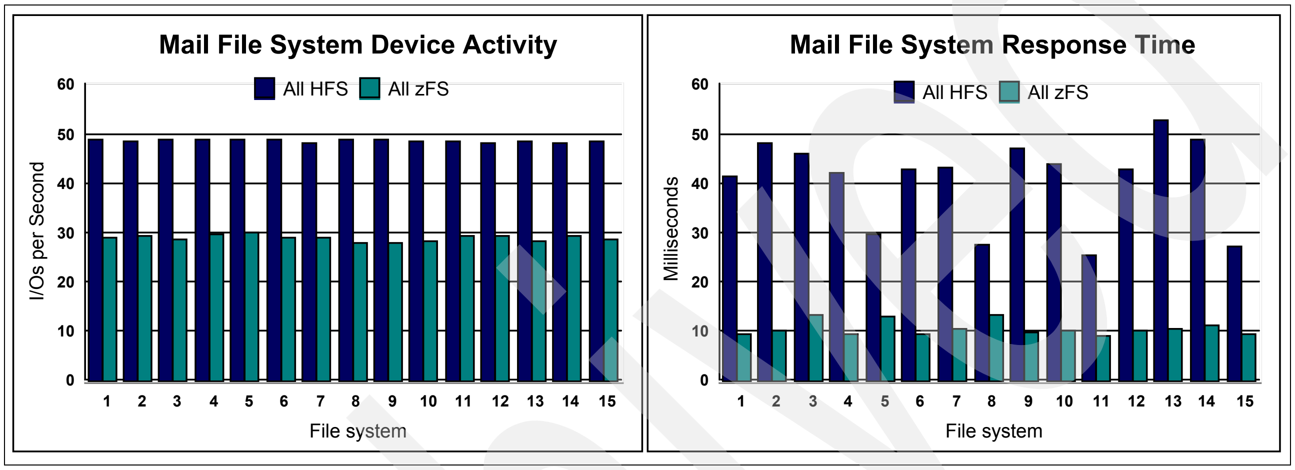


Figure 12-11 Mail file system device activity and response time

For this portion of the test, we spread the 2,000 simulated users across 15 mail file systems for a density of 133 users per file system. With this user density, and the tuning of the workload via the parameters listed above (NormalMessageSize, NumMessageRecipients, NthIteration) we were able to drive the HFS to relatively long average DASD response times, and substantial I/O rates. HFS drove substantially less I/O, and allowed the DASD to respond with a generally acceptable response time. Note that there is no real difference between the Mixed and All zFS configurations, because the mail file systems were zFS file systems in both cases.

Figure 12-12 on page 474 shows the components of the response time for a representative mail file system in both HFS, and zFS configurations. The first thing to note is that there is no I/O service queue time during any time interval for the zFS file system. There was no contention between threads simultaneously trying to reach the DASD with I/O requests. The second thing to note is that although disconnect time is substantial for the zFS case, it is still significantly smaller than that for HFS.

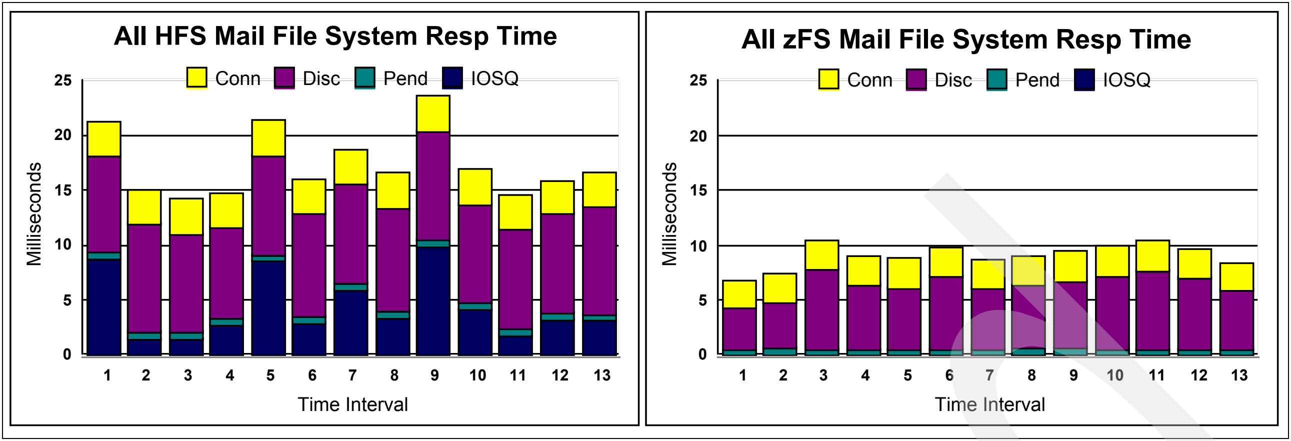


Figure 12-12 Mail response time comparison between HFS and zFS

The I/O characteristics for the Notes data shown in Figure 12-13, and transaction log file systems shown in Figure 12-14 on page 475, are very interesting.

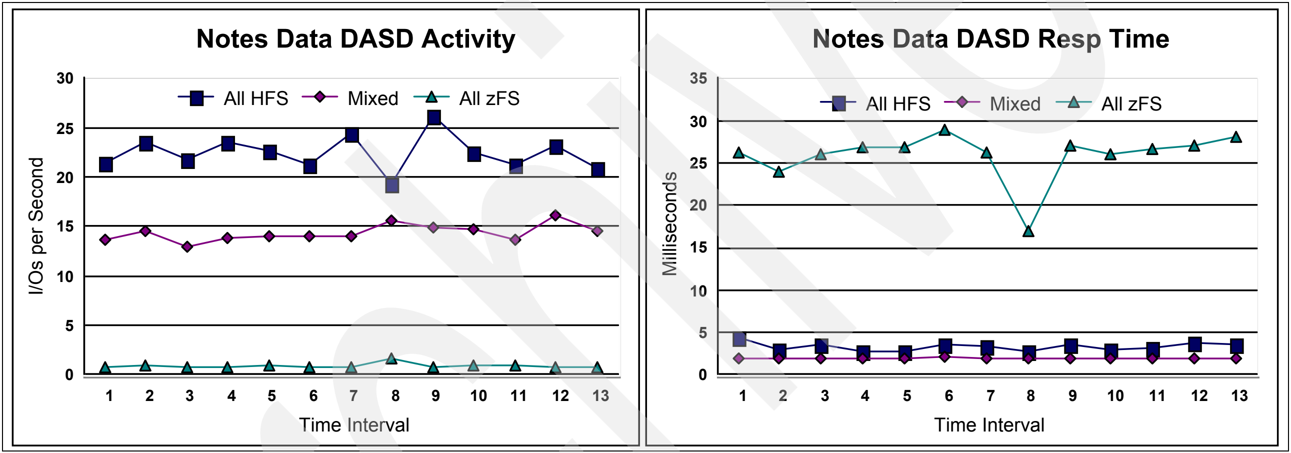


Figure 12-13 I/O characteristics for Notes data

DASD activity for the Notes data directory actually fell when the only change to the configuration was to move the target mail databases for the simulated users to zFS. This is not intuitively obvious, and we don't really have a good explanation for it. The other thing to note is that although the response time for the zFS configuration is several times higher than that for the HFS configuration, zFS was performing almost no I/O to DASD at the time. These long response times are inconsequential, given the very low DASD activity.

One detail not shown here is that the Notes data file system spanned two volumes in the HFS configuration, and four volumes in the zFS configuration. I/O was only performed to the first volume of each file system, so all of the DASD data presented here is for the first volume only.

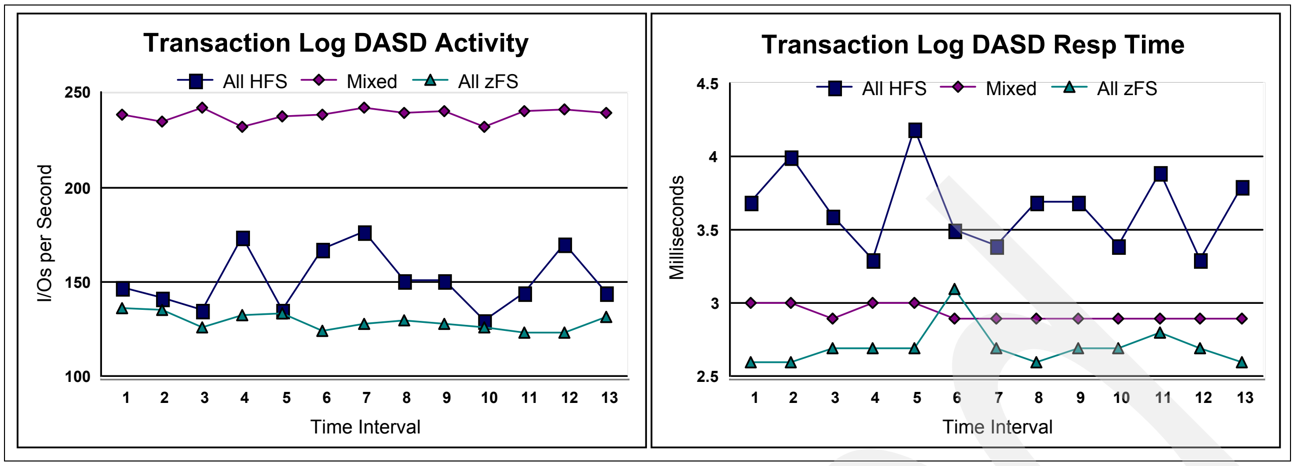


Figure 12-14 I/O characteristics for the transaction log

This data shows the high activity rates inherent in the use of the transaction log, and the relatively low response times for all three configurations. I/O to this file system is highly sequential, so it's not surprising that both file systems were able to handle the load very well. I/O service queue times were either absolutely, or practically, zero for all configurations. Similarly, pending and disconnect times were very low, making the connect time the largest component of the I/O response time in all cases.

As with the Notes data directory, shown in Figure 12-13 on page 474, we can't account for the difference between the All HFS, and Mixed file system configurations. The benchmark workload runs at a constant transaction rate, so differences in the DASD rates cannot be attributed to the workload stressing the server more or less heavily between the different tests.

The Webmail workload

For this workload we simulated 500 users, all running the Webmail workload described above. All of our results here were collected on a 5-minute interval for 1 hour after the steady state of the workload had been reached (all users ramped up, and were driving work on the server). We used only two configurations for this test, All HFS, and All zFS.

From a storage usage and end-user response time standpoint, the results are very similar to those for R5Mail.

Table 12-10 Webmail workload with 500 users

	CPU usage %	**Central Storage (MB)**	**Expanded Storage (MB)**	**End-user response time (Msec)**
All HFS	69.9	1,614	9 **	1,229
All zFS	69.4	2,063	651	544
Difference	-1%	+28%	n/a	-56%

Note: ** There is always a small amount of expanded storage in use; 9 MB in use for HFS is effectively zero.

What is substantially different here is the CPU consumption, which shows essentially no difference between the HFS and zFS configurations.

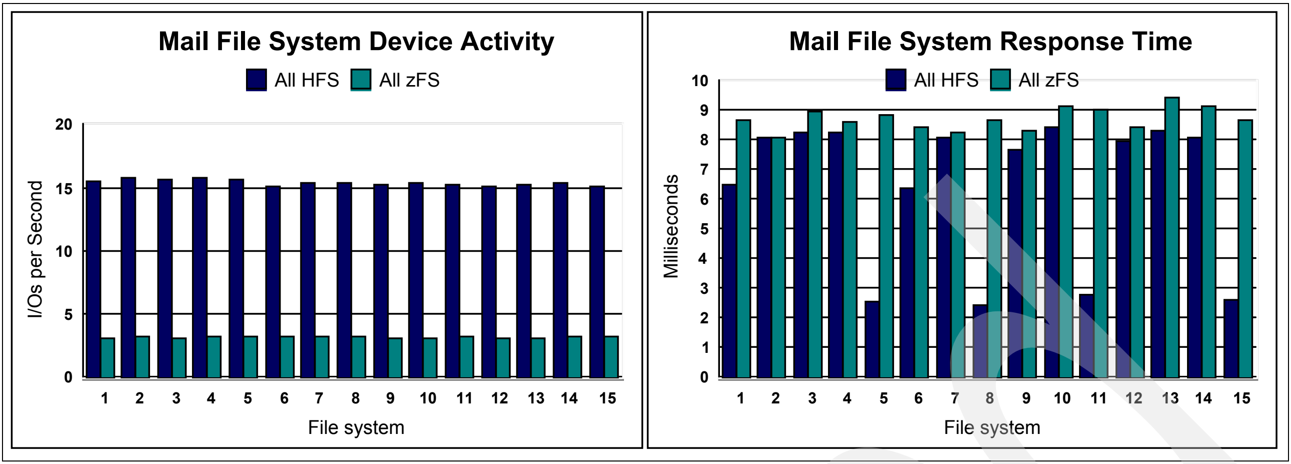


Figure 12-15 Mail file system device activity and response time

The I/O details for the mail file systems, shown in Figure 12-15, demonstrate the differences between the R5Mail and Webmail benchmark workloads in terms of the amount of stress that they drive against the mail file systems. For the Webmail-based workloads, the Domino server spends more time and resource assembling and rendering data than it does actually acquiring that data from the database. This is the fundamental difference between Web-based and Notes client-based access to a database.

As can be seen in Figure 12-15, neither HFS nor zFS were stressed heavily to service I/O requests to the mail file systems for this workload. DASD response times for HFS dropped back down into the acceptable range, and were comparable to those of zFS. What is worth noting is that the DASD activity gap between HFS and zFS grew under this lighter workload. For R5Mail, HFS drove about 1.7 times as much I/O as zFS. For the lighter Webmail workload, this gap was about 4.9.

Where zFS provides its real performance benefit for this workload is in the I/O performed to the Notes data and transaction log file systems, as shown in Figure 12-16.

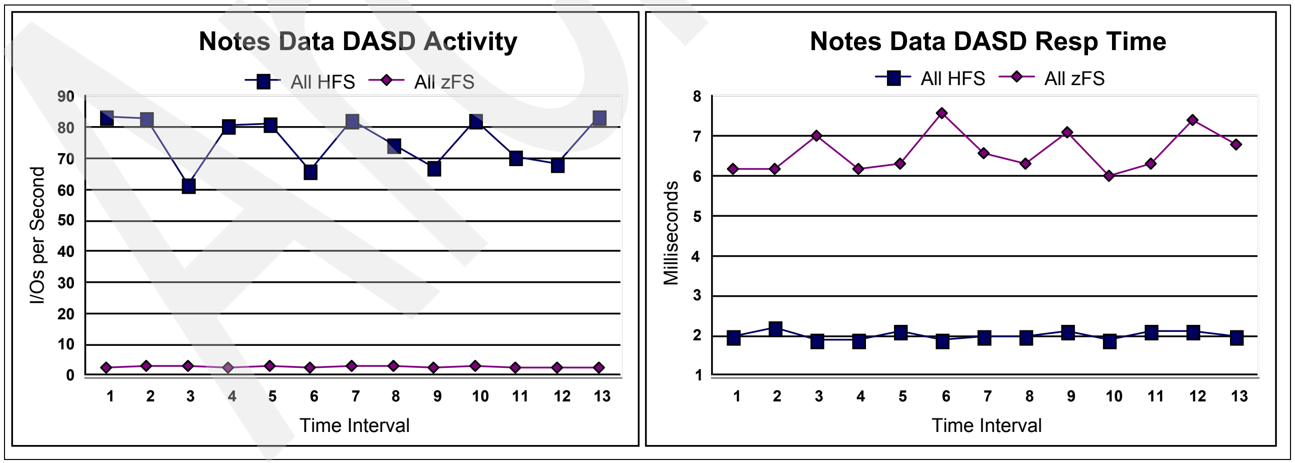


Figure 12-16 I/O characteristics for Notes data and the transaction log

One can see that HFS has to drive much more I/O to the Notes data file system in order to gather the things that it needs to build Web pages from the documents that it has read from the user's mail database. These parts include things like GIF files and HTML templates, and they reside in a subdirectory off the Notes data directory. zFS was able to cache this data, and keep its I/O to DASD low and very flat. The difference in response time between HFS and zFS was not significant.

The difference in traffic to the transaction log also follows the same pattern as for the R5Mail workload, only the absolute amount of I/O required for both file system types is lower than for the R5Mail workload, as shown in Figure 12-17.

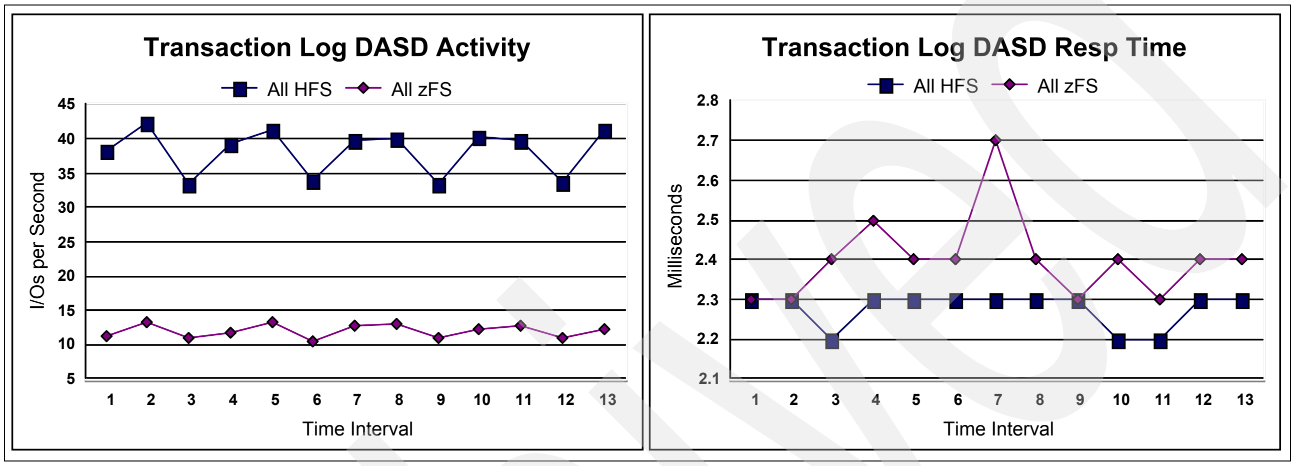


Figure 12-17 Transaction log device activity and response time

What is significant here again is the gap between the I/O performed by HFS and zFS. For the R5Mail workload, where the volume of traffic to the actual mail databases was much higher, the difference in DASD utilization between the two file system types was smaller. As with the other workloads, the higher resource usage of zFS reflects how it is able to apply more available system resource to service the loads required of the file system.

The Inotes workload

This workload was configured to run the same way as the Webmail workload, with two important differences:

- 750 users were simulated instead of 500.
- The size of the message sent was 4,000 bytes instead of 1,000 bytes.

The reason for these differences is to make the CPU and I/O usages, shown in Figure 12-18 on page 478, higher so that the measurements would show differences more clearly.

Table 12-11 Inotes workload with 750 users

	CPU Usage %	**Central Storage (MB)**	**Expanded Storage (MB)**	**End-user response time (Msec)**
All HFS	88.5	1,582	7 **	731
All zFS	91.3	2,063	647	709
Difference	+3%	+30%	n/a	-3%

Note: ** There is always a small amount of expanded storage in use; 7 MB in use for HFS is effectively zero.

One of the primary advantages of Inotes relative to the "old" Webmail interface tested above is that Inotes performs substantially better. One can see from the data in Figure 12-18 that an important reason for this is that Inotes requires much less I/O to be performed to accomplish the same amount of work. For this test, even though we simulated 50% more users sending notes that are 4 times larger, the DASD I/O rates to the mail databases for both HFS and zFS tests were lower for Inotes than for Webmail.

This smaller I/O component within the overall cost to support a given user is the reason why there is so little difference in the CPU usage and response time measurements between the HFS and zFS tests. Enhancing the performance of a small part of the cost yields a small overall cost reduction.

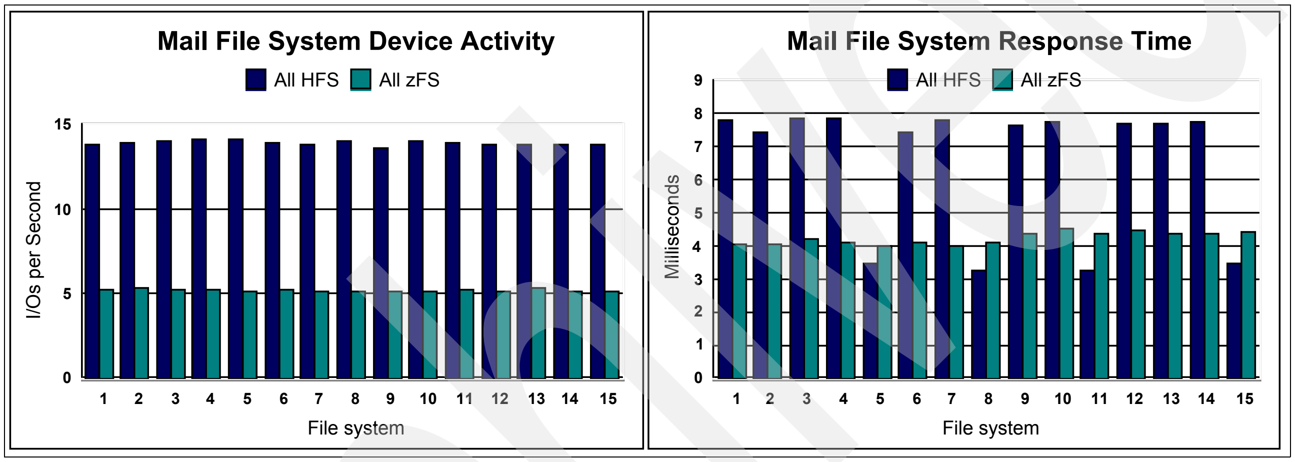


Figure 12-18 I/O usages for the Inotes workload with 750 users

Our efforts to drive the server utilization to higher levels by increasing the parameters of the Inotes workload did cause the Domino server to do substantially more work. Although we saw the I/O to the mail databases above actually drop relative to Webmail, both in terms of the total I/O required and the I/O per simulated user, the heavier workload did manifest itself as higher I/O to the transaction log.

These results, shown in Figure 12-19 on page 479 and Figure 12-20 on page 479, remain consistent with those of the other workloads. The point here though is that one has to understand the components of the overall user cost and performance metrics. zFS usage provides more advantage in those environments where local file system I/O is a bigger part of the cost to support a workload.

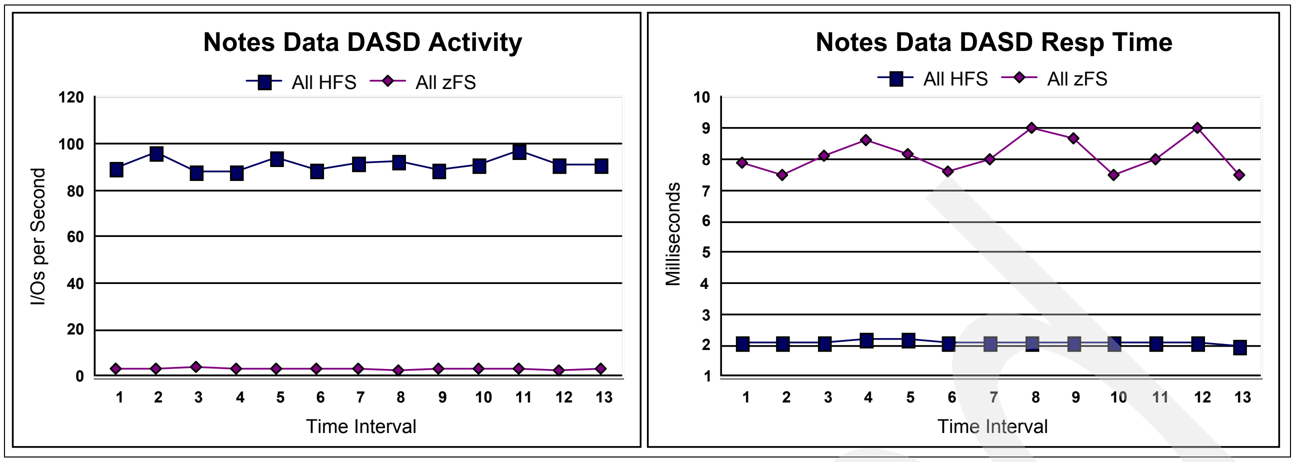


Figure 12-19 I/O usage for Notes data showing device activity and response time

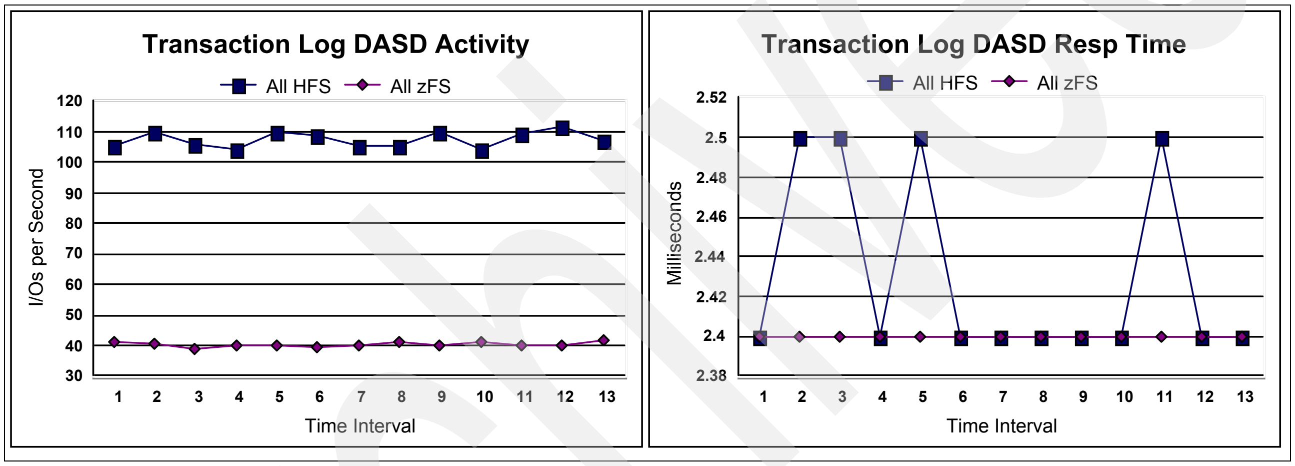


Figure 12-20 I/O usage for the transaction log for device activity and response time

Production Domino applications

We measured two production environments to see what benefits zFS might bring to a Domino application (not mail).

The first application is from a state agency and is used to trace case histories for all types of state business (driver's license applications, taxes, human services affairs, etc.). The application is implemented as a collection of agents that are used to process the data in the database. For the purposes of this test, the customer implemented a simple test agent that performed three types of operations:

- Run a query built from twelve separate conditional clauses, and count the number of matching documents.
- Explicitly loop through all documents, looking for those having a field with a specific value, and updating a count field in that document.
- Loop through all case types, and generate a monthly report.

The database containing this agent is 34 MB in size, and contains more than 37,000 documents. The point of this test is simply to measure the elapsed time required for a single thread of execution to complete running the test agent. Measurements were made with the test database in the Notes data directory.

Table 12-12 Elapsed time differences for the five test runs

Test Run	Elapsed Time HFS (seconds)	Elapsed Time zFS (seconds)	% Difference
1	50	19	-62%
2	22	18	-18%
3	22	18	-18%
4	22	18	-18%
5	22	18	-18%

The second application that we evaluated with zFS was from a different state agency that is interested in the performance of multiple Web users simultaneously performing operations on the same set of databases. For this test, all databases were located in a single file system. The tests were driven through LoadRunner, using test scripts created by the customer who wrote the application. Table 12-13 shows the results generated by LoadRunner.

Table 12-13 LoadRunner test results

	HFS	zFS	% Difference
Number of simulated users	70	70	0%
Total throughput (MB)	212	239	12.7%
Throughput (bytes/second)	78,776	89,121	13.1%
Total hits	40,172	45,539	13.4%
Hits per second	14.27	16.22	13.7%
Total passed	8,979	10,222	13.8%
Total failed	174	162	-6.9%
Total abort	164	157	-4.3%
Minimum (seconds)	165	160.5	-2.7%
Average (seconds)	262.6	233.4	-11.1%
Maximum (seconds)	579.9	555.3	-4.2%
90% (seconds)	416.4	400.6	-3.8%

As can be seen in Table 12-13, throughput is improved significantly, both in terms of the number of bytes passed between server and browser, and the number of Web hits serviced. This had a positive effect on the number of tests that LoadRunner deemed to be successful, presumably because there were fewer timeouts during the test. Average end-user response time was also substantially reduced.

Although the relatively short duration of this test, and the limited number of simulated users, make drawing any conclusions about contention between the multiple threads somewhat questionable, the higher throughput numbers would seem to indicate that zFS allowed the workload to run with greater freedom, and less waiting for the common resources of the application.

12.3.4 Domino performance conclusions

Domino servers perform better when deployed with zFS instead of HFS in all of the operations that we tested. This manifests itself at the client in terms of higher throughput, and lower elapsed times to perform any client-initiated operation that requires a significant amount of I/O to be driven on the server. This is true for both Notes client access, and access via Web browser. Non-client-driven server operations and housekeeping functions are similarly improved.

The caching capabilities of zFS appear to be the key to providing this performance gain. This is supported by the fact that zFS drives only about 1/3 the amount of I/O to DASD that HFS drives when put under an identical load. This reduction in DASD traffic is realized for both highly random and highly sequential I/O patterns. Although we were not able to directly determine the cause for improvement in this test, our end-user throughput measurements indicate that zFS also provides significantly higher throughput rates than HFS.

This performance improvement comes with a higher cost in terms of the amount of storage required for the particular zFS configurations that we tested. It was not a goal of this study to tune caching values so that the storage required by zFS matched that for HFS when the server was under a given workload. The key observation is that zFS gives the administrator as much or as little storage to apply to the file system as needed in order to achieve the performance levels required.

It is also important to note that server CPU consumption does not significantly change for a given workload, regardless of file system type. In fact, when converting an existing Domino environment to zFS, it is possible that one could actually see CPU increase marginally, depending on how much the server is currently bottlenecked by HFS. Users who are converting from HFS to zFS in an environment that has high levels of I/O to DASD should monitor CPU consumption, and be prepared to tune zFS to keep resource consumption—both CPU and storage—within required operating ranges.

zFS has provided us with many performance advantages in the lab—both for our internal benchmarking efforts, and for various customer environments that we have recreated in the lab. To date, zFS has been deployed internally within IBM for limited production over the last several months, and its performance and stability have been very good. zFS is a viable alternative to HFS that anyone should seriously consider for their Domino environment.

12.4 Additional information about zFS

For more information and details on zFS and especially zFS performance see the zFS redbook *z/OS Distributed File Service zSeries File System Implementation*, SG24-6580.

13

Maintenance of z/OS UNIX

This chapter describes what is required to maintain a z/OS UNIX environment.

We discuss the following topics:

- HFS data set backup and recovery
- Increasing the size of an existing HFS data set using DFSMShsm
- Installing service
- Post-installation tasks

13.1 HFS data set backup and recovery

Data in an HFS data set can be backed up and restored with any of the following:

- Hierarchical Storage Manager (DFSMShsm) to back up, migrate, and restore files at the HFS data set level.
- Data Facility Data Set Services (DFSMSdss) to dump and restore files at the HFS data set level.
- Tivoli Storage Manager (TSM) is a client/server storage management product that provides administrator-controlled, highly-automated, centrally-scheduled, network-based backup and archive functions for workstations and LAN file servers.

13.1.1 Backing up and restoring HFS data sets using DFSMShsm

If you use DFSMShsm, you must define a user ID for the DFSMShsm address space. In order for DFSMShsm to be able to access the HFS data sets, it must run under a user ID that is set up for UNIX System Services access:

- The default group for the DFSMShsm user ID must have an OMVS segment defined and a group ID associated with it.
- The home directory should be the root file system.
- The user ID should be defined as a superuser (with a UID of 0), or the DFSMShsm address space should be defined as TRUSTED in the RACF started procedures table.
- If the file system being dumped by DFSMShsm is currently mounted as read/write, then this file system can only be dumped from the system on which it is mounted. If the file system is mounted as read only, then it can be dumped from any system that has access to it.

For more information, see *z/OS DFSMSdss Storage Administration Reference,* SC35-0424.

13.1.2 Backing up and restoring HFS data sets using DFSMSdss

You can also use DFSMSdss to back up and restore files at the HFS data set level. The JCL shown in Figure 13-1 on page 485 can be used to dump an HFS data set, while the JCL shown in Figure 13-2 on page 486 can restore an HFS data set.

```
  //DSSDUMP  JOB (999,POK),'HFS DUMP',MSGLEVEL=(1,1),
  //             CLASS=A,MSGCLASS=T,NOTIFY=SLEKKA
  //****************************************************************
  //*
  //* THIS JOB WILL CREATE A BACKUP OF THE EXISTING ROOT HFS
  //* THIS IS DONE BY USING THE LOGICAL DUMP FUNCTION OF DFSMSDSS
  //*
  //* EXISTING ROOT HFS : SPECIFY THE EXISTING ROOT HFS NAME AT
  //* THE DSN PARM IN THE HFSVOL DD STATEMENT
  //*
  //* HFS DUMP NAME : SPECIFY THE NEW HFS NAME AT THE DSN
  //* PARM OF HFSOUT DD STATEMENT
  //*
  //* STORCLAS : SPECIFY YOUR STORAGE CLASS
  //*
  //* OMVS.SC49.ROOTA : DSN OF THE EXISTING ROOT FILE SYSTEM
  //*
  //* OMVS.SC49.ROOTA.SEQ : DSN OF THE DUMPED ROOT FILE SYSTEM
  //*
  //****************************************************************
  //SU       EXEC PGM=ADRDSSU,REGION=6M
  //SYSPRINT DD SYSOUT=*
  //HFSVOL   DD UNIT=3390,VOL=SER=TMPSMS,DISP=SHR
  //HFSOUT   DD DSN=OMVS.SC49.ROOTA.SEQ,
  //            SPACE=(CYL,(100,100),RLSE),
  //            UNIT=3390,STORCLAS=OPENMVS,
  //            DISP=(NEW,CATLG,DELETE)
  //SYSIN    DD *
    DUMP DATASET(INCLUDE(OMVS.SC49.ROOTA)) -
      COMPRESS TOL(ENQF) -
      LOGINDDNAME(HFSVOL) OUTDDNAME(HFSOUT) -
      ALLDATA(*) ALLEXCP
/*
```

Figure 13-1 JCL used to dump an HFS data set

```
//DSSREST  JOB (999,POK),'HFS RESTORE',NOTIFY=SLEKKA,
//             CLASS=A,MSGCLASS=T,TIME=1439,
//             REGION=5000K,MSGLEVEL=(1,1)
//*****************************************************************
//*
//* THIS JOB WILL RESTORE AN HFS DATA SET THAT WAS PREVIOUSLY
//* DUMPED USING DFDSS. A RENAMEU PARAMETER IS USED
//* TO RENAME THE HFS DATA SET BEING RESTORED AS NOT TO
//* CONFLICT WITH AN EXISTING HFS DATA SET ON THE PACK.
//*
//* HFSSEQ : SPECIFY THE SEQUENTIAL BACKUP DATA SET THAT
//* CONTAINS THE PREVIOUSLY DUMPED HFS DATA SET
//* IN THE HFSSEQ DD STATEMENT.
//*
//* HFSOUT : SPECIFY THE PACK WHERE THE RESTORED HFS DATA
//* SET WILL RESIDE.
//*
//* RENAMEU PARM : SPECIFY THE OLD HFS NAME AND THE NEW HFS
//* NAME, USING THE RENAMEU PARM.
//*
//* STORCLAS : SPECIFY YOUR STORAGE CLASS.
//*
//******************************************************************
//STEP1    EXEC PGM=ADRDSSU,REGION=6M
//SYSPRINT DD SYSOUT=*
//HFSSEQ   DD DSN=OMVS.SC49.ROOTA.SEQ,DISP=SHR,
//            VOL=SER=TMPSMS,UNIT=3390
//HFSOUT   DD UNIT=3390,VOL=SER=TMPSMS,DISP=SHR
//SYSIN    DD *
  RESTORE INDD(HFSSEQ) OUTDD(HFSOUT) TOL(ENQF) -
    DATASET(INCLUDE(OMVS.SC49.ROOTA)) -
    RENAMEU((OMVS.SC49.ROOTA,OMVS.SC49.ROOTB)) -
    STORCLAS(OPENMVS) -
    CANCELERROR
/*
```

Figure 13-2 JCL used to restore an HFS data set

A TSM server backs up and/or archives data from a TSM client and stores the data in the TSM server storage pool for z/OS UNIX clients.

There are two types of backup: incremental, in which all new or changed files are backed up; and selective, in which the user backs up specific files.

Backup can be performed automatically or when the user requests it. The user can initiate a specific type of backup or start the scheduler, which will run whatever action the TSM administrator has scheduled for the user's machine.

As a TSM authorized user, you also have the authority to back up and archive all eligible files in all locally mounted file systems on your workstation, restore and retrieve all backup and archive files for your workstation from TSM storage, within the limits imposed by the UNIX file access permissions. A TSM authorized user can also grant users access to specific files in TSM storage.

Information about using the z/OS UNIX client is documented in:

- *TSM Using the Backup-Archive Clients,* SH26-4105
- *TSM Installing the Clients,* SH26-4102

13.2 Increasing the size of an existing HFS data set

It may be necessary to increase the size of an existing HFS data set when it has run out of extents. The procedure to increase the size of the root HFS data set differs from the procedure to increase the size of any other HFS data set because the root HFS data set is always allocated.

13.2.1 Increasing the size of the root HFS data set

The root HFS data set must be cloned before you can increase the data set size.

To increase the size of the root HFS data set, do the following:

1. DFSMSdss DUMP the root HFS data set to a sequential backup data set.
2. DFSMSdss RESTORE the HFS data set, specifying a new name with the RENAMEU keyword. This creates a clone of the root HFS data set.
3. DFSMSdss DUMP the cloned HFS data set to a sequential backup data set.
4. Delete the cloned HFS data set.
5. Allocate a new, larger HFS data set with the same name as the cloned HFS data set.
6. DFSMSdss RESTORE the cloned HFS data set, specifying the REPLACE keyword. This will RESTORE the source data set into the target data set that was allocated in the previous step.
7. To make the changes effective, do one of the following:
   - Unmount the current root file and mount the new, larger data set.

     Unmounting the root file system: Unmounting and remounting the root file system is very disruptive. You will have to stop all UNIX System Services work and unmount any files that are mounted on the root file system. You will also have to stop any address spaces, for example INETD or IMWEBSRV, which have HFS data sets allocated.Once you have stopped all work, unmount the root file system via the shell or use the following command: UNMOUNT FILESYSTEM('OMVS.SC49.ROOTA') IMMED. You must specify the IMMED keyword when unmounting the root file system. Use the following command to mount the new file system: MOUNT FILESYSTEM('OMVS.SC49.ROOTB') TYPE(HFS) MOUNTPOINT('/').

   - A less disruptive method to make the changes effective is to schedule the change at the next IPL and modify BPXPRMxx to point to the new data set name.

The procedure for increasing the size of the root HFS data set is shown in Figure 13-3 on page 488.

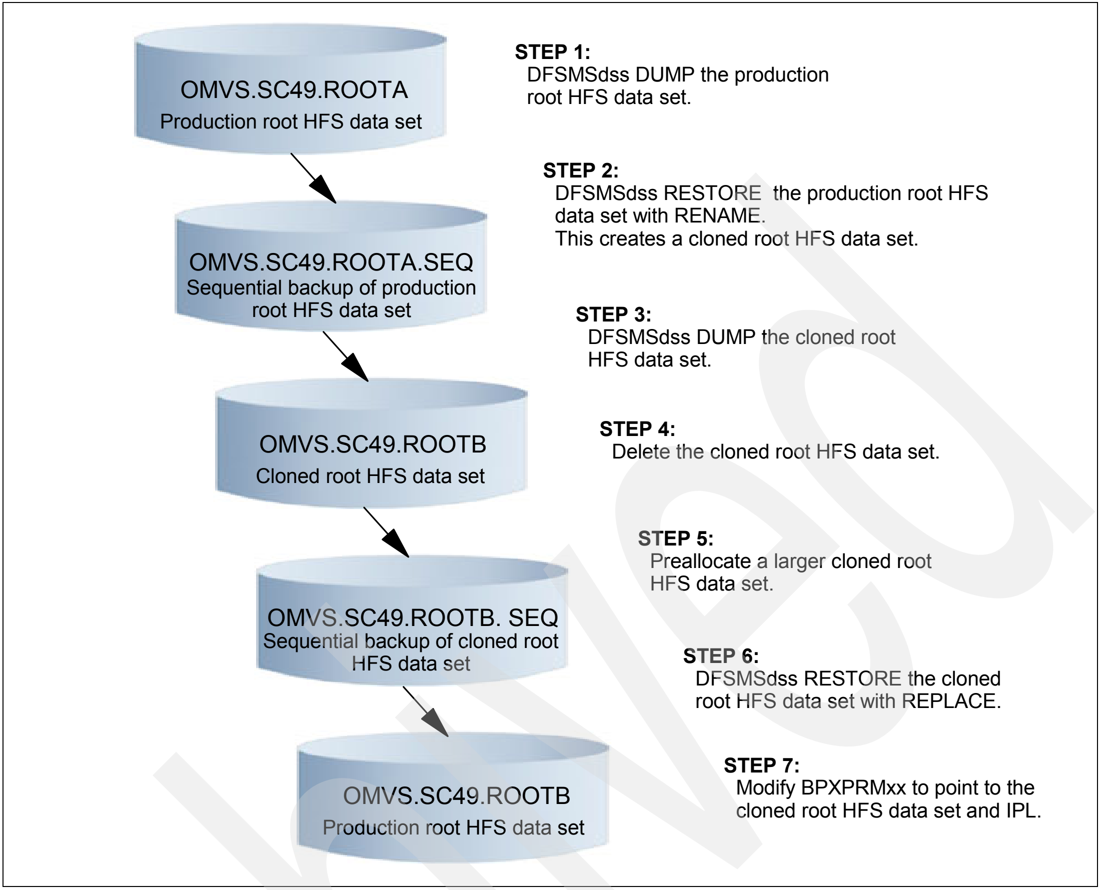

OMVS.SC49.ROOTA
Production root HFS data set
STEP 1:
DFSMSdss DUMP the production root HFS data set.
OMVS.SC49.ROOTA.SEQ
Sequential backup of production root HFS data set
STEP 2:
DFSMSdss RESTORE the production root HFS data set with RENAME.
This creates a cloned root HFS data set.
STEP 3:
DFSMSdss DUMP the cloned root HFS data set.
OMVS.SC49.ROOTB
Cloned root HFS data set
STEP 4:
Delete the cloned root HFS data set.
STEP 5:
Preallocate a larger cloned root HFS data set.
OMVS.SC49.ROOTB. SEQ
Sequential backup of cloned root HFS data set
STEP 6:
DFSMSdss RESTORE the cloned root HFS data set with REPLACE.
STEP 7:
Modify BPXPRMxx to point to the cloned root HFS data set and IPL.
OMVS.SC49.ROOTB
Production root HFS data set

Figure 13-3 Increasing the size of the root HFS data set

13.2.2 Increasing the size of other HFS data sets

Increasing the size of other HFS data sets is simpler because they can be unmounted and unallocated. To increase the size of other HFS data sets:

1. Unmount the HFS data set and any HFS data sets that are mounted at mount points lower in the tree. The data set will need to be unallocated before it can be unmounted. If an application, for example the WebSphere is accessing the HFS, it may need to be stopped until the dump and restore have been completed.
2. Use the JCL in Figure 13-1 on page 485 to dump the HFS data set, OMVS.SC49.INTERNET.A in the following example.

```
DUMP DATASET(INCLUDE(OMVS.SC49.INTERNET.A)) -
   COMPRESS TOL(ENQF) -
   LOGINDDNAME(HFSVOL) OUTDDNAME(HFSOUT) -
   ALLDATA(*) ALLEXCP
```

3. Rename the HFS data set. Once the data set is no longer mounted, you can rename it from ISPF Option 3.4 by using the R - Rename Data Set line command.
4. Preallocate a larger HFS data set with the same name as the old smaller HFS data set.
5. Use the JCL in Figure 13-2 on page 486 to restore the HFS data set, replacing the RENAMEU keyword with the REPLACE keyword as follows:

```
RESTORE INDD(HFSSEQ) OUTDD(HFSOUT) TOL(ENQF) -
   DATASET(INCLUDE(OMVS.SC49.INTERNET.A)) -
   STORCLAS(OPENMVS) -
```

```
        REPLACE -
        CANCELERROR
```

6. Mount the new file system. This must be done by an authorized user with either the TSO MOUNT command, or an authorized user running the REXX exec /samples/mountx from the shell. If necessary, restart the application.

The procedure for increasing the size of HFS data sets is shown in Figure 13-4.

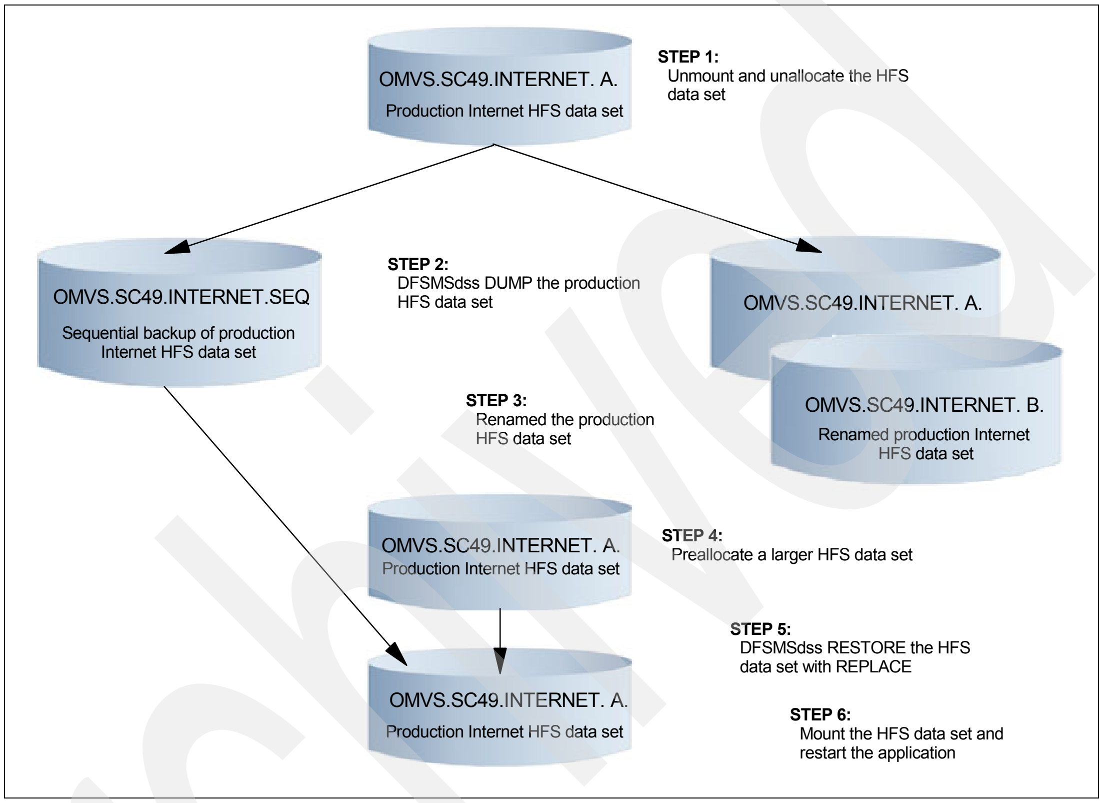

OMVS.SC49.INTERNET. A.
Production Internet HFS data set
STEP 1:
Unmount and unallocate the HFS data set
OMVS.SC49.INTERNET.SEQ
Sequential backup of production Internet HFS data set
STEP 2:
DFSMSdss DUMP the production HFS data set
OMVS.SC49.INTERNET. A.
OMVS.SC49.INTERNET. B.
Renamed production Internet HFS data set
STEP 3:
Renamed the production HFS data set
OMVS.SC49.INTERNET. A.
Production Internet HFS data set
STEP 4:
Preallocate a larger HFS data set
STEP 5:
DFSMSdss RESTORE the HFS data set with REPLACE
OMVS.SC49.INTERNET. A.
Production Internet HFS data set
STEP 6:
Mount the HFS data set and restart the application

Figure 13-4 Increasing the size of other HFS data sets

For more information on the RESTORE parameter keywords, see *z/OS DFSMSdss Storage Administration Reference,* SC35-0424.

13.3 Installing service using SMP/E

There is no single recommended way of updating a system with service. MVS users have different configurations, product sets, business requirements, support personnel, and so on. In other words, every user is unique, and a recommendation that would work well for one user may be unworkable for another. It is beyond the scope of this book to explore all aspects of managing software on a z/OS system, but a general methodology of how one might manage maintenance when the UNIX System Services product is involved is discussed.

z/OS now includes UNIX components as a standard part of the operating system. Therefore, it is now mandatory to have z/OS UNIX System Services enabled on any system where SMP/E maintenance is performed.

Other traditional MVS products also now contain UNIX components; DB2 and NetView are two such examples.

Figure 13-5 shows a typical PTF for TCP/IP (a component of z/OS). You will notice that it contains typical MCS statements such as `++ PTF`, `++ VER`, and `++ MOD`. You will also notice the `++ HFS` MCS statement, which directs element EZAFTPSM as a file into the HFS path pointed to by DDDEF SEZAMMSC. In addition, a *hard link* (alias name) will be created to EZAFTPSM in the directory one higher in the hierarchy (indicated by the '..' in the LINK parameter), called ftpdmsg.cat.

In other words, if the path in DDDEF SEZAMMSC points to:

```
/usr/lpp/tcpip/lib/nls/msg/C/IBM
```

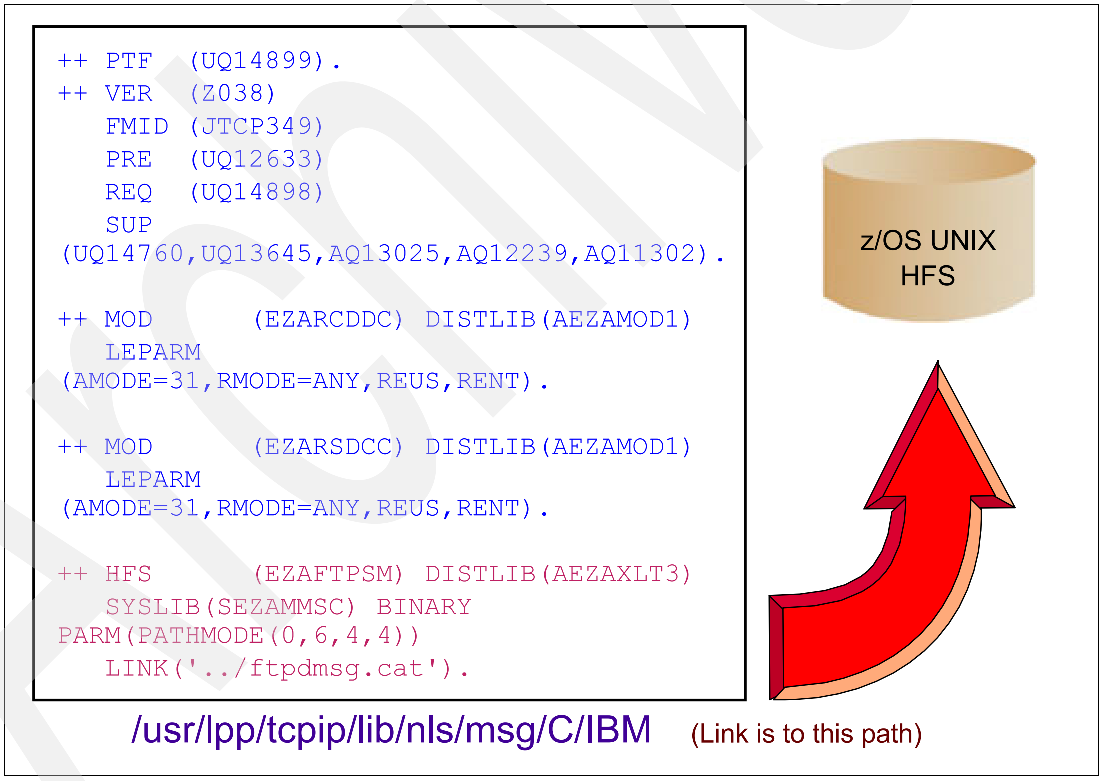
Then the element will be written as file:

```
/usr/lpp/tcpip/lib/nls/msg/C/IBM/EZAFTPSM
```

With a hard link defined as:

```
/usr/lpp/tcpip/lib/nls/msg/C/ftpdmsg.cat
```

Note: See how the original file (EZAFTPSM) is written in the /C/IBM subdirectory, while the hard link (ftpdmsg.cat) is created in the /C subdirectory, one directory higher because of the '..' specification in the LINK parameter. This type of packaging allows SMP/E unique element names while also meeting the requirements of UNIX naming conventions.



```
++ PTF   (UQ14899).
++ VER   (Z038)
   FMID  (JTCP349)
   PRE   (UQ12633)
   REQ   (UQ14898)
   SUP
(UQ14760,UQ13645,AQ13025,AQ12239,AQ11302).

++ MOD       (EZARCDDC) DISTLIB(AEZAMOD1)
   LEPARM
(AMODE=31,RMODE=ANY,REUS,RENT).

++ MOD       (EZARSDCC) DISTLIB(AEZAMOD1)
   LEPARM
(AMODE=31,RMODE=ANY,REUS,RENT).

++ HFS       (EZAFTPSM) DISTLIB(AEZAXLT3)
   SYSLIB(SEZAMMSC) BINARY
PARM(PATHMODE(0,6,4,4))
   LINK('../ftpdmsg.cat').
```

Figure 13-5 A typical PTF

13.3.1 Applying service to an active root

Figure 13-6 on page 491 shows an example of an active file system. The file system is in use by the active system (the system you are logged on to), so applying maintenance directly onto the active file system is undesirable because:

- It could introduce a change that impacts work already running.
- If a problem is encountered during the SMP/E APPLY causing the APPLY to fail, it could damage the active file system and impact work already running.
- If you need to fall back to the point before the service was applied, the process is greatly complicated when it is the active file system.

The problems are similar to those concerning z/OS system residence (SYSRES) volumes, where typically a cloned copy of the active SYSRES is used to receive maintenance, then it is IPLed. This way, application of maintenance does not affect the active system until an IPL is performed. If there is a problem with the new maintenance level, fallback is to reIPL from the old SYSRES.

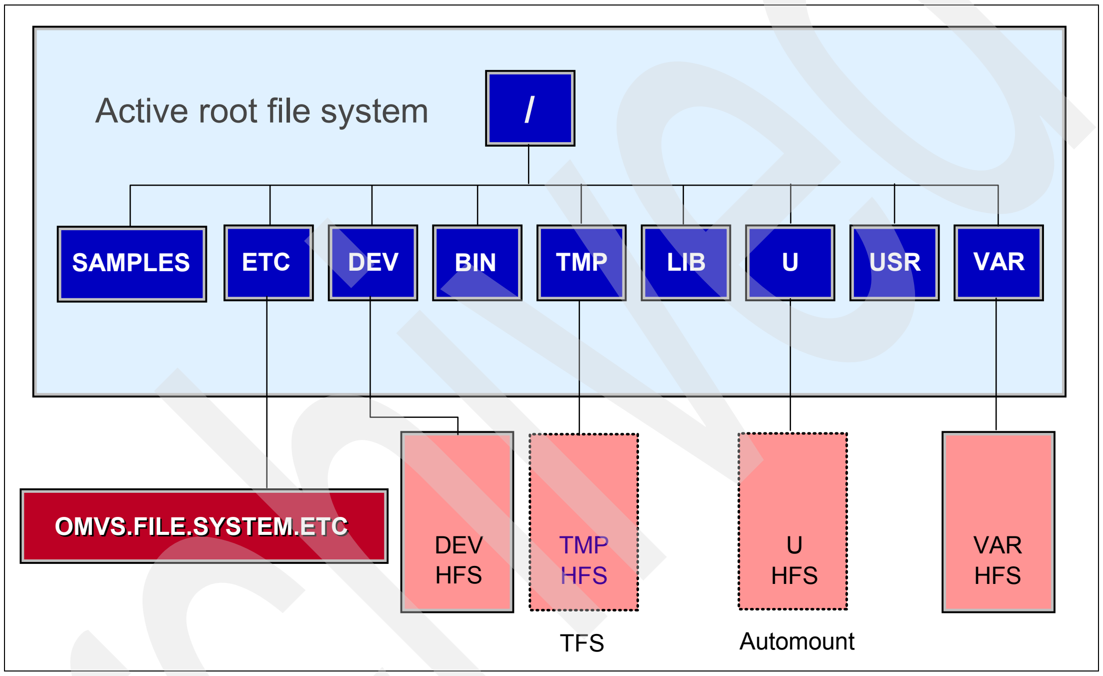


Figure 13-6 Active root file system

Clone the active root file system

To apply maintenance safely without impact to an active system, we need to clone the z/OS UNIX file system, and work with the inactive copy of the file system. This is the same technique used for z/OS system residence (SYSRES) volumes, only the method has to vary because we are only dealing with HFS data sets and not full volumes.

To create a cloned copy of a HFS, DFSMSdss (ADRDSSU) must be used to first DUMP, then RESTORE with the RENAMEU (RENUNC) parameter, to result in a copied file with a different name, as shown in Figure 13-7 on page 492. See the JCL in Figure 13-1 on page 485 and Figure 13-2 on page 486 to dump/restore the HFS.

Note: The DFSMSdss COPY function is supported for HFS data sets beginning in z/OS V1R3.

z/OS UNIX can only access files if the HFS that contains them is mounted into the active file system. So, the newly created clone file system needs to be somehow connected into the active file system structure so SMP/E processes can access the data.

The file system could be connected anywhere in the active file system, but the recommended place to do it is under a directory called /SERVICE. Using this directory ensures no impact to other UNIX processing.

If this directory does not exist on your system, you could create it via a TSO command, a shell command, or using the ISHELL. As an example, the TSO command to do this is:

```
MKDIR '/SERVICE' MODE(7,5,5)
```

Once the /SERVICE directory has been created, the cloned file system needs to be mounted there. The mount can happen by specifying the HFS in BPXPRMxx or issuing a TSO MOUNT command.

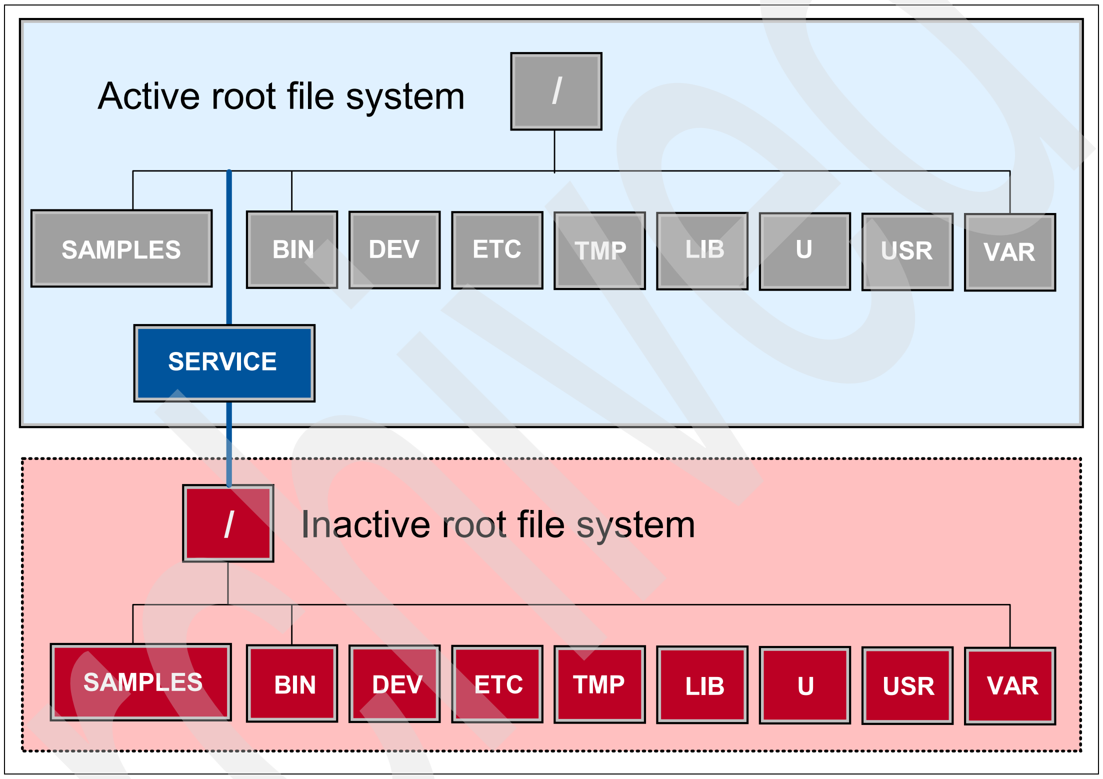


Figure 13-7 A clone of the active root file system

Once the HFS data sets are cloned into the /SERVICE directory, some decisions need to be made about the SMP/E configuration that will be used to support the cloned environment. You could either change the DDDEFs of the old SMP/E configuration (meaning that the active environment is no longer maintainable), or you could clone the SMP/E CSIs and change the new CSIs to point to the new HFS.

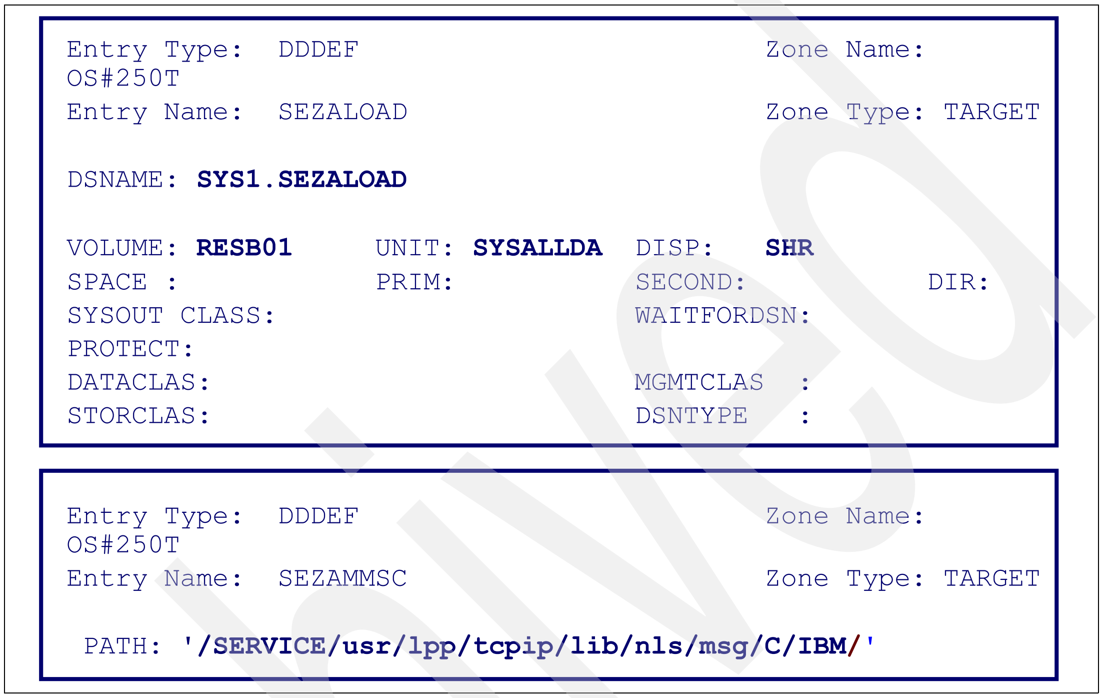
Although we are mainly talking about HFS data sets in this topic, you must of course ensure the integrity of the SMP/E configuration and make sure that all DDDEFs in the CSIs point to data sets with contents that match what is recorded in SMP/E. How this is handled may be different depending on the product and how it is implemented on your system. More about this on the next few visuals.

Regardless of how you choose to solve the problem of SMP/E integrity, you will still have to adjust the SMP/E DDDEFs to point to the cloned file system mounted at /SERVICE. The SMP/E ZONEEDIT command can be used to do this.

Figure 13-8 shows a traditional DDDEF (top box) and an HFS DDDEF (bottom box). The traditional DDDEF SEZALOAD points to a PDS called SYS1.SEZALOAD on DASD volume RESB01, while the HFS DDDEF SEZAMMSC points to a path called:

```
/SERVICE/usr/lpp/tcpip/lib/nls/msg/C/IBM/
```

(Note the /SERVICE directory.)



```
Entry Type:  DDDEF                                  Zone Name:
OS#250T
Entry Name:  SEZALOAD                               Zone Type: TARGET

DSNAME: SYS1.SEZALOAD

VOLUME: RESB01      UNIT: SYSALLDA  DISP:    SHR
SPACE :             PRIM:           SECOND:           DIR:
SYSOUT CLASS:                       WAITFORDSN:
PROTECT:
DATACLAS:                           MGMTCLAS   :
STORCLAS:                           DSNTYPE    :
```

```
Entry Type:  DDDEF                                  Zone Name:
OS#250T
Entry Name:  SEZAMMSC                               Zone Type: TARGET

 PATH: '/SERVICE/usr/lpp/tcpip/lib/nls/msg/C/IBM/'
```

Figure 13-8 Example of SMP/E DDDEFs

13.3.2 Installing service to products in the HFS

Service for UNIX System Services is installed in much the same way as for traditional z/OS products. Service is applied using System Modification Program Extended (SMP/E). Differences include:

- The target libraries for UNIX System Services are mostly HFS data sets.
- The UNIX System Services kernel must be running on the driving system in order to apply service to HFS data sets.
- The user ID that is running the SMP/E jobs must have an OMVS security segment, and it must be running as a superuser (UID=0).

In a typical z/OS installation, an active, production-level system is called a *driving system*. This system could be used to run business-critical applications, or it could be a system dedicated for system build and testing activity.

Note: You should never directly upgrade your driving system. Rather, you should install changes on a copy or clone of your production system, called a *target system*, test the changes, and then migrate the changes into your production environment. This process minimizes the risk of new code causing an outage to your production system. Figure 13-9 on page 494 shows the process of cloning the OMVS.SC43.RESA01.ROOT.HFS and OMVS.SC49.INTERNET.A data sets.

13.3.3 Prepare for SMP/E APPLY

The preparation for system maintenance should include the following steps:

- Clone the data sets that the SMP/E DDDEFs point to:
  - For non-HFS data sets, you would DFSMSdss (ADRDSSU) COPY the active files to create inactive equivalents. You should consider that:
    - SYSRES data sets would have the entire volume copied, such that the inactive data sets have the same names but are uncataloged. Figure 13-9 shows active volume RESA01 being copied to RESB01.
    - Non-SYSRES data sets may either be copied to another volume with the same name but are uncataloged, or they may be copied with other names allowing them to be cataloged if you choose.
  - For HFS data sets, you would DFSMSdss (ADRDSSU) DUMP/RESTORE the active files to create inactive equivalents (as described in visuals prior to this). You should consider that:
    - HFS data sets that belong to SYSRES products need something in their name to associate them to the correct SYSRES. For example, say you clone SYSRES volume RESA01 onto volume RESB01, the HFS data sets that relate to RESB01 should have RESB01 in their name. This allows the use of the &SYSR1. symbol in the BPXPRMxx member so that the HFS data sets that relate to the IPLed SYSRES are always correctly selected.
    - For example, BPXPRMxx contained a reference to HFS data set OMVS.&SYSNAME..&SYSR1..ROOT.HFS.
    - If you IPLed from volume RESA01, symbol &SYSR1. would be substituted with string RESA01 resulting in a data set name of OMVS.SC43.RESA01.ROOT.HFS.
    - If you IPLed from volume RESB01, symbol &SYSR1. would be substituted with string RESB01 resulting in a data set name of OMVS.SC43.RESB01.ROOT.HFS.
  - This technique simplifies moving back and forth between maintenance levels (test sessions, implementations, and so on).

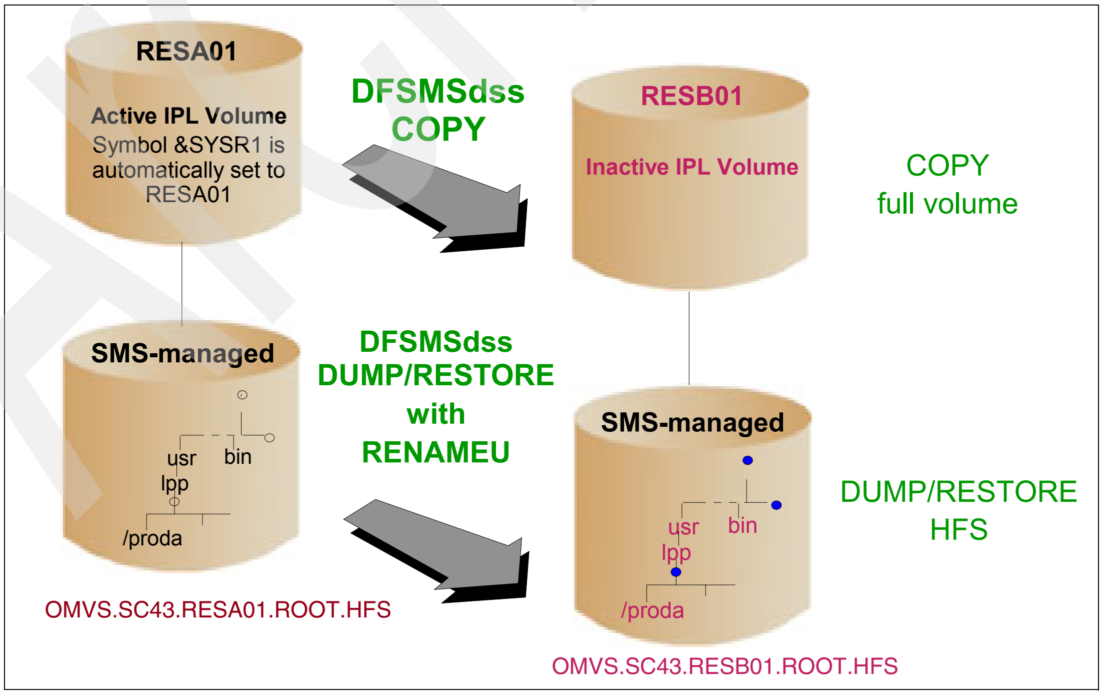


Figure 13-9 Make a clone of your system using DFSMSdss

- Make SMP/E point to the inactive data. You could choose to:
  - Change the DDDEFs in the existing SMP/E CSIs to point to the inactive data. This would result in the active data having no SMP/E CSIs pointing to them, which may be a problem if an emergency fix needs to be applied to the live system.
  - Clone the SMP/E CSIs so the old CSIs continue pointing to the active data, while the DDDEFs in the new CSIs can be ZONEEDITed to point to the inactive data.

The following steps could be followed to apply service to UNIX System Services.

Install service with SMP/E

For the SMP/E APPLY process to work as desired, you should have inactive clones of the active data sets.

- HFS data sets should be mounted into the active file system off the /SERVICE directory. The data sets could be mounted there permanently via the BPXPRMxx member, or dynamically via a TSO MOUNT command. For the Figure 13-10 on page 496, the following command could be used:

```
MOUNT FILESYSTEM('OMVS.SC43.RESB01.ROOT.HFS') +
    TYPE(HFS) MODE(RDWR) MOUNTPOINT('/SERVICE')
```

- If you wanted to dynamically remove the file system after maintenance work has been performed, the following command could be used:

```
UNMOUNT FILESYSTEM('OMVS.SC43.RESB01.ROOT.HFS') +
     IMMEDIATE
```

For SMP/E CSIs that point to the inactive data sets, no matter how you choose to set up your SMP/E CSIs as described previously, you will probably need to use an SMP/E ZONEEDIT command such as the following to get the DDDEFs set correctly:

```
ZONEEDIT DDDEF.
     CHANGE VOLUME(RESA01,
                   RESB01).
     CHANGE PATH('/'*,
                 '/SERVICE/'*).
     ENDZONEEDIT.
```

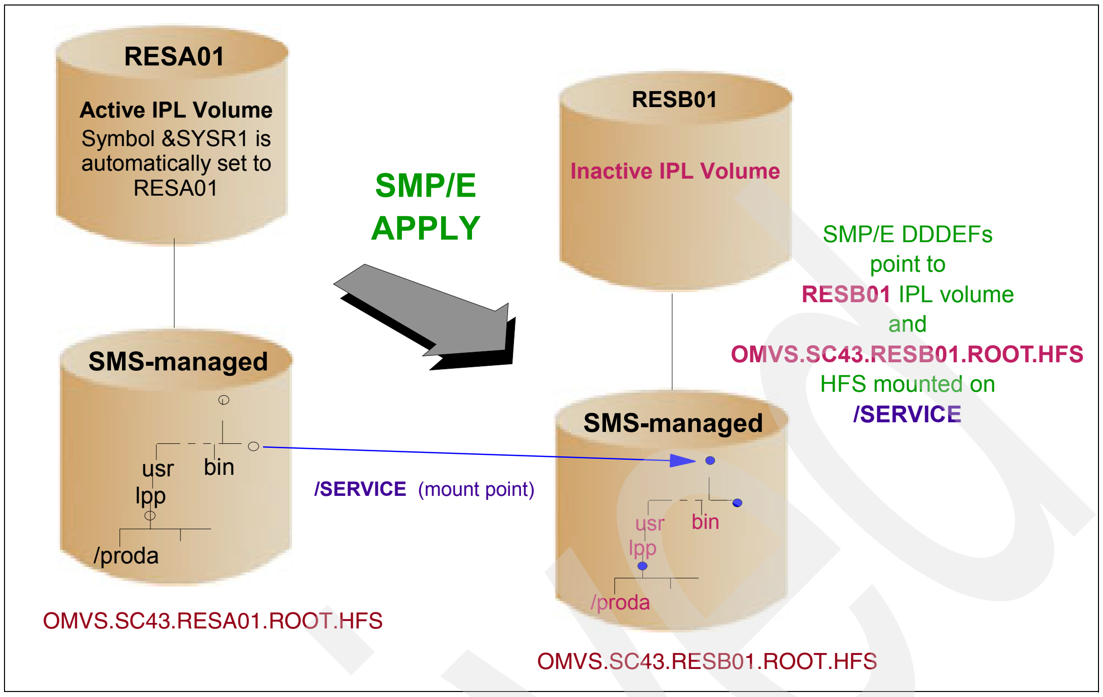

RESA01
Active IPL Volume
Symbol &SYSR1 is automatically set to RESA01
SMP/E APPLY
RESB01
Inactive IPL Volume
SMP/E DDDEFs point to RESB01 IPL volume and OMVS.SC43.RESB01.ROOT.HFS HFS mounted on /SERVICE
SMS-managed
usr
bin
lpp
/proda
/SERVICE (mount point)
SMS-managed
usr
bin
lpp
/proda
OMVS.SC43.RESA01.ROOT.HFS
OMVS.SC43.RESB01.ROOT.HFS

Figure 13-10 SMP/E APPLY of UNIX System Services service

Testing the new root

After installing the service, the new UNIX System Services target libraries should be tested. The level of testing depends on the amount of service and what UNIX System Services components were hit with service. Check the DDDEF list in your SMP/E APPLY listing for the data sets that were changed when service was applied. If service hits SYS1.LPALIB, you have to IPL the SYSRES.

If service hits paths in the root HFS file system, you should change the BPXPRMxx member to point to the new cloned HFS data set, and then IPL. You are able to unmount the current root file system and mount the new root file data set but this is very disruptive.

If service hits paths other than the root HFS file, you only have to:

1. Stop the affected application.
2. Unmount the old HFS data set.
3. Mount the new cloned HFS data set.
4. Start the affected application.

If in doubt, change the BPXPRMxx member to point to the new cloned HFS data sets and IPL the SYSRES with a Clear Link Pack Area (CLPA).

Note: There can only be one UNIX System Services running at one time on any one system. If you are going to test the newly serviced UNIX System Services system on the same processor or LPAR as the driving system, you must IPL off the SYSRES. It is not possible to have UNIX System Services use STEPLIB to get to any files in the new HFS data sets. The BPXPRMxx member of parmlib must be updated with the names of the new HFS data sets and the system must be IPLed.

If the test has been successfully completed, this new cloned system can now be propagated to other systems in your enterprise using the same DFSMSdss methods as were discussed earlier in this chapter. Remember that the HFS data sets are now part of your overall cloning and system propagation scheme and must be copied to other systems along with the traditional target libraries if you plan to run UNIX System Services on any other systems in your enterprise.

13.4 Post-installation tasks

Many products require post-installation tasks to be completed after maintenance has been applied. You may experience problems with symbolic links if you run the tasks while mounted at the /service directory mount point.

A symbolic link is a file that contains the pathname for another file or directory. Only the original pathname is the real name of the original file. An external link is a type of symbolic link, a link to an object outside of the HFS. Typically, it contains the name of an MVS data set.

To ensure that all links are resolved, we recommend that you check the task to determine which pathnames are being modified. You can then decide whether to complete the task after the HFS data set has been mounted at the original mount point, or to modify the tasks to point to the /service mount point.

If the tasks are run while the HFS is mounted at the /service mount point, ensure that any references to pathnames are resolved to the /service mount point. When testing has been completed and the file has been mounted at the root file, the pathnames and any symbolic links will have to be modified to reflect the production environment.

14

Problem determination

This chapter discusses a few means and commands on how to display and collect information in case of troubleshooting, as follows:

- Failures and messages in the z/OS UNIX environment
- Slip trap settings and OMVS component trace
- z/OS USS sysplex sharing diagnosis

14.1 Failures and messages in the z/OS UNIX environment

Hard failures end with an abend or message from the failing function including return and reason codes. For a description of abend codes refer to *z/OS MVS System Codes,* SA22-7626.

14.1.1 z/OS USS messages and codes

In Table 14-1 we list message identifiers of components related to UNIX System Services.

Table 14-1 Component message indentifiers

Identifier	Component or message type
BPX	z/OS USS MVS system messages
FSUM	USS shell and utilities messages
FDBX	USS debugger (DBX) messages
FOM	USS application services messages
EDC	LE C/C++ runtime library (RTL) messages
CEE	LE base messages
CBC	C/C++ compiler messages
EZx	TCP/IP messages
ICH/IRR	RACF messages
IMW	WebSphere messages

In order to list messages online you can use the LOOKAT message tool on the Web at:

```
http://www-1.ibm.com/servers/s390/os390/bkserv/lookat/lookat.html
```

14.1.2 Messages from failing z/OS UNIX functions

A description of the UNIX return and reason codes is available in *z/OS UNIX System Services Messages and Codes,* SA22-7807.

z/OS UNIX return codes generally correspond to standard POSIX errors, for example EAGAIN (`The resource is temporarily unavailable.`), EACCESS (`Permission is denied.`), and EBUSY (`The resource is busy.`).

z/OS UNIX Reason Codes, also referenced as “errnojr”, are made up of 4 bytes in the following format:

```
cccc rrrr
```

The first halfword `cccc` is a reason code qualifier. This is used to identify the issuing module and represents a module ID. The second 2 bytes are the reason codes that are described in the messages books.

If this value is between 0000 and 20FF and the return code is not A3 or A4, then this is a USS reason code. In this situation you can use the BPXMTEXT procedure to get more information about this reason code.

You can use BPXMTEXT from TSO or in a UNIX shell environment. Figure 14-1 on page 501 shows an example taken from a USS shell session.

```
$> bpxmtext 055B005B
BPXFSMNT 04/30/03
JRIsMounted: The file system is already mounted

Action: If the file system must be mounted on the specified mountpoint, first
unmount it, and then reissue the request.
$>
```

Figure 14-1 Using BPXMTEXT in a UNIX shell session

For more information about reason codes and where to find a description for a specific one, see *z/OS UNIX System Services Messages and Codes,* SA22-7807.

14.1.3 z/OS UNIX latches

Latches are used to serialize critical operations to USS resources when this is necessary. They are system-specific, and each system has its own set of latches.

However, latch hangs on a single system can lead to sysplex-wide hangs in a USS sysplex sharing environment.

Kernel latch set (LSET) control blocks reside in the OMVS kernel private storage. The types of latches that are maintained are the following:

- SYS.BPX.A000.FSLIT.FILESYS.LSN Latch#=xxxx
  - Latch#1 = LFS latch
  - Latch#2 = Mount latch
  - Latch#3 = used by pipes
  - Latch#4 = used by character special files
  - Latch#5 = used by osi_sleep and osi_wakeup
  - Latch#6 = used by LFS cache
  - Latch#7 = RFI (Register File Interest) lock
  - Latch#8...n = USS file system latch
- SYS.BPX.A000.FSLIT.FILESYS.LSN.01 (02,03,...,0n) Latch#=xxxx
  - Latch#n = File latch
- SYS.BPX.AP00.PRTB1.PPRA.LSN Latch#=xxxx
  - Latch#n = Process latch

You can use the **D GRS,C** command or the **D GRS,LATCH,C** command to show latch contentions. An example is provided in Figure 14-2 on page 502.

```
D GRS,LATCH,C
ISG343I 15.28.43 GRS STATUS 066
LATCH SET NAME:  SYS.BPX.A000.FSLIT.FILESYS.LSN
CREATOR JOBNAME: OMVS      CREATOR ASID: 000E
  LATCH NUMBER:  12
    REQUESTOR  ASID  EXC/SHR    OWN/WAIT
    OMVS      000E  EXCLUSIVE  OWN  (2)
    FTPD      003E  EXCLUSIVE  WAIT (1)
  LATCH NUMBER:  93
   REQUESTOR  ASID  EXC/SHR    OWN/WAIT
    WELLIE3   00C2  EXCLUSIVE  OWN  (4)
    OMVS      000E  SHARED     WAIT (3)
```

Figure 14-2 Latch contentions shown with D GRS,LATCH,C

Figure 14-2 notes:

1. FTPD hangs, waiting for latch #12.
2. Latch #12 is owned by OMVS.
3. OMVS hangs, waiting for latch #93.
4. Latch #93 owned by WELLIE3.

Tip: Since WELLIE3 is at the root of the contention, the address space C2 should be included when taking a dump of this hang situation.

RAS

One of the keystones of IBM's research policy is *autonomic computing*. The vision of self-managing systems includes four key goals. Autonomic systems will be:

– Self-configuring: Able to adapt dynamically to changing environments
– Self-healing: Able to discover, diagnose, and prevent disruptions
– Self-optimizing: Able to tune resources and balance workloads to make maximum use of available IT resources
– Self-protecting: Able to anticipate, detect, identify, and protect against attacks

Part of the zSeries strategy for autonomic computing is *RAS*, which stands for *Reliability, Availability, Serviceability*. In the past it could happen that customers experienced hangups in z/OS UNIX for internal processing. These hangups even led to system outages in some cases. An enhancement was made to z/OS V1R6 to address this by providing additional latch cleanup and identification for z/OS UNIX hang conditions.

When z/OS UNIX is unable to resolve a latch contention over several minutes, a new message will appear:

```
BPXM056E UNIX SYSTEM SERVICES LATCH CONTENTION DETECTED
```

What it means is that at least one unit of work in the system is holding on to a z/OS UNIX GRS latch for several minutes. To determine which z/OS UNIX latches are the cause of the problem, you can use the command shown in Figure 14-2. A new command can be used to try to abend (422-1A5) the user address space tasks that are causing the contention:

```
F BPXOINIT,RECOVER=LATCHES
```

If it works, the following message is displayed:

```
BPXM067I UNIX SYSTEM SERVICES LATCH CONTENTION RESOLVED
```

In the unfortunate circumstance that the latch contention cannot be resolved, the following message is displayed:

```
BPXM057E UNIX SYSTEM SERVICES LATCH CONTENTION NOT RESOLVED
```

Note: With **`F BPXOINIT,RECOVER=LATCHES`**, only individual tasks are ended and not entire processes. Also, it may not be able to solve latch hangs in the kernel address space.

Latch contention analysis

Physical File Systems (PFS) also use latches and problems with mount latch contention could, for example, prevent an **`F OMVS,SHUTDOWN`** from completing successfully. This could happen when a task like zFS is waiting for a reply to a cross-system message, or for another latch. With z/OS V1R7 some display capability was added to **`DISPLAY OMVS`** that enables you to analyze the latch contention. As you can see in Figure 14-3, there are no latch holders or waiters to be displayed. But if there were any, it would tell you the age of the latch, and it would also tell you the user, ASID, TCB, the reason, and the file system involved.

```
D OMVS,W
BPXO063I 14.59.20 DISPLAY OMVS 421
OMVS     0011 ACTIVE          OMVS=(7A)
MOUNT LATCH ACTIVITY: NONE
OUTSTANDING CROSS SYSTEM MESSAGES: NONE
```

Figure 14-3 Display latch activity

Another parameter was introduced with z/OS V1R7. Figure 14-4 below shows the command to display the file systems, with two newly added fields.

```
D OMVS,F
BPXO045I 15.16.02 DISPLAY OMVS 452
OMVS     0011 ACTIVE          OMVS=(7A)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
ZFS           836 ACTIVE                      RDWR  05/01/2005  L=112
  NAME=ZFSFR.ZFSG.ZFS                               20.43.49    Q=0
  PATH=/SC65/z/zfsg
  AGGREGATE NAME=ZFSFR.ZFSG.ZFS
  OWNER=SC65     AUTOMOVE=U CLIENT=N
ZFS           835 ACTIVE                      RDWR  05/01/2005  L=111
  NAME=ZFSFR.ZFSF.ZFS                               20.43.48    Q=0
  PATH=/SC65/z/zfsf
  AGGREGATE NAME=ZFSFR.ZFSF.ZFS
  OWNER=SC65     AUTOMOVE=U CLIENT=N
```

Figure 14-4 Display file systems

14.1.4 Getting a console dump for a hang

Generally, when analyzing USS hang situations, a dump of OMVS, OMVS data spaces, and the hanging job address space is required.

In order to easily take a dump later, you can use member IEADMCxx in SYS1.PARMLIB to store the dump command information. A sample entry is shown in Figure 14-5 on page 504.

In this example you can vary the comment information provided with the dump according to the actual situation. Instead of `TSONAME=WELLIE3`, you can also specify `ASID=(yy)` if no

TSO-related process is involved; yy here is the number in hex of the address space that suffers the hang.

```
COMM=(TSO OMVS HANG)
JOBNAME=(OMVS),TSONAME=WELLIE3,DSPNAME=('OMVS'.*),
SDATA=(ALLNUC,PSA,CSA,LPA,TRT,SQA,RGN,GRSQ,SUM)
```

Figure 14-5 Parmlib member IEADMCxx

Note: According to the D GRS output shown in Figure 14-2, using ASID=(C2) would provide the same results.

Most likely you will receive the parameters for the IEADMCxx member from your IBM support representative. But for those of you who need a good starting point there are some examples available in the SYS1.SAMPLIB, including members IEADMCOE, IEADMCTO and IEADMCWT.

You can initiate the dump processing now using the following MVS system command:

```
DUMP PARMLIB=xx
```

As soon as the dump has been taken successfully, you see the following message:

```
IEA911E COMPLETE DUMP ON SYS1.DUMPxx
```

If a dump is available, you can use the following MVS Interactive Problem Control System (IPCS) command to get the desired information for the latch hang situation:

```
IPCS ANALYZE RESOURCE
```

The output shows which TCB and address space is holding the latch and so causes the contention. Let us assume the decision can be taken that this TCB is no longer needed or is not essential for further system processing. In this situation you can get rid of the TCB with a CALLRTM program using the CALLRTM macro. See *z/OS MVS Programming Authorized Assembler Services Guide,* SA22-7608 for more information on CALLRTM.

See also informational APAR II08038 for a description of how to request USS MVS data to diagnose problems.

14.2 Slip trap settings and OMVS component trace

Here we provide information on how to take a take a dump based on a UNIX reason code or a message written to the syslog. We also show how to provide an OMVS component trace.

14.2.1 Setting a slip for SVCDUMP based on a UNIX reason code

A generic slip trap can be used to get a dump on the issuance of an z/OS UNIX reason code, also referenced as "errnojr". This is supported if your system is running at least at level z/OS V1.2.

You can use member IEASLPxx in SYS1.PARMLIB to store the slip setting, as shown in Figure 14-6.

```
SLIP SET,IF,A=SYNCSVCD,RANGE=(10?+8C?+F0?+1F4?),
DATA=(13R??+B0,EQ,12CA00B6),DSPNAME=('OMVS'.*),
SDATA=ALLNUC,PSA,CSA,LPA,TRT,SQA,RGN,SUM),
J=WELLIE3*,END
```

Figure 14-6 Parmlib member IEASLPxx

In this example, `12CA00B6` is the 8-digit (4-byte) reason code that is to be trapped. The expression `J=WELLIE3*` is the optional jobname expected to see the error. The asterisk (*) is a wildcard character that is useful for forked address spaces, if you do not know the ending number that the failing job will have.

Using this technique of putting the slip command into a IEASLPxx parmlib member makes it possible to activate the slip simply by using the following command:

```
SET SLIP=xx
```

For more information on setting slips to obtain new diagnostic data, see *z/OS UNIX System Services Messages and Codes,* SA22-7807.

14.2.2 Slip for SVDUMP on FSUM shell and utilities message

We use the following message as an example to show how to do that:

```
ls: FSUM6785 File or directory "xxxx" is not found
```

Figure 14-7 shows the appropriate slip command for message FSUM6785.

```
SLIP SET,IF,N=(BOBJCST,9D0),DATA=(H.1R?+4??+9,EQ,F6F7F8F5), AL=(0,E),DN=('OMVS'.*),
ACTION=SYNCSVCD,SDATA=(ALLNUC,PSA,CSA,LPA,TRT,SQA,RGN,SUM,GRSQ,LSQA),END
```

Figure 14-7 Slip command for message FSUM6785

The number `9` is the position where the numeric part of the message starts. `F6F7F8F5` is the EBCDIC code of the numeric part of the message. And `AL=(0,E)` dumps the current address space and OMVS, assuming the address space number of OMVS is `E`.

14.2.3 OMVS component trace

The OMVS component trace (CTRACE) is especially valuable in slip dumps because it provides a history of the OMVS Syscalls used up to the moment that the slip event matched and a dump was taken.

Turn on the OMVS component trace with the MVS system command:

```
TRACE CT,4M,COMP=SYSOMVS
R xx, OPTIONS=(ALL),END
```

Replace `xx` with reply number displayed after you entered the first line listed with the TRACE command.

To verify the current OMVS CTRACE status, use:

```
D TRACE,COMP=SYSOMVS
```

The following commands turn the OMVS ctrace off again after taking the slip or console dump:

```
TRACE CT,128K,COMP=SYSOMVS
TRACE CT,OFF,COMP=SYSOMVS
```

14.2.4 General message slip trap

A new parameter, MSGID, is available for slip trap settings. This support was introduced with z/OS V1.2. We show this with the following message:

```
BPXP009I THREAD threadid, IN PROCESS pid, ENDED ABNORMALLY WITH COMPLETION CODE compcode,
REASON CODE reasoncode.
```

The following shows the slip command with the MSGID parameter:

```
SLIP SET,MSGID=BPXP009I,AL=(H,P,S,CU),JL=OMVS,DN=('OMVS'.*),SDATA=(ALLNUC,PSA,CSA,LPA,
TRT,SQA,RGN,SUM,GRSQ,LSQA),END
```

The advantage is that this is no PER trap. This means that more than one can be set active concurrently, for example together with an SA trap without blocking it. But the disadvantage is that ACTION=SYNCSVCD cannot be used.

14.3 USS sysplex sharing diagnosis

This section discusses the changes to the MODIFY command to control UNIX System Services. These functions are very useful when a problem related to the UNIX System Services sysplex sharing support appears and needs to be analyzed. The interface is available on all OS/390 and z/OS systems that support USS sysplex sharing.

14.3.1 Shared USS diagnostic and repair functions

UNIX System Services (USS) currently processes MODIFY commands that are targeted for BPXOINIT, the job name of the USS Init process. The command syntax is being enhanced to support a new FILESYS= keyword, along with associated parameters.

Figure 14-8 shows the MODIFY command syntax to support file system shared USS diagnostic and repair functions.

```
F BPXOINIT,{FILESYS={DISPLAY[,FILESYSTEM=filesystemname]}[,OVERRIDE]}
                            [,ALL]
                            [,EXCEPTION]
                            [,GLOBAL]
                    {DUMP                               }
                    {FIX                                }
                    {REINIT                             }
                    {RESYNC                             }
                    {UNMOUNT,FILESYSTEM=filesystemname  }
                    {UNMOUNTALL                         }
```

Figure 14-8 Modify BPXOINIT FILESYS= command syntax

For a complete description of the command and its parameters, see *z/OS MVS System Commands,* SA22-7627.

Normally only one MODIFY command for a FILESYS= function can be active on each system. Additionally, only one instance of the MODIFY command in the sysplex can be active for the FIX, UNMOUNT, UNMOUNTALL, and REINIT functions.

By specifying the OVERRIDE parameter, multiple invocations of this command are accepted on each system for the DISPLAY, DUMP, and RESYNC functions.

Note: Nevertheless, a second invocation using this OVERRIDE parameter may be delayed.

The primary intent of the OVERRIDE parameter is to allow DISPLAY functions to be issued while there is still a MODIFY in progress and the MODIFY appears to be delayed.

14.3.2 USS sysplex sharing diagnostic procedures

In *z/OS MVS Diagnosis Reference,* GA22-7588 you can find a detailed description about diagnostic procedures for a UNIX sysplex sharing environment. The types of problems that are addressed basically relate to file system availability on one or multiple systems in a parallel sysplex environment.

The intent of the recovery procedures described there is as follows:

- To prevent a sysplex-wide restart by either correcting the problem or limiting the scope of the restart to one or a subset of systems
- To collect sufficient documentation about the problem to provide to the IBM Support Center so that the root cause of the problem can be identified and resolved expediently

A

Managing z/OS UNIX user IDs and groups

This appendix provides information about the management of RACF-defined user IDs and groups.

We provide examples of tools that can be used to understand the current environment of your z/OS UNIX system.

- Listing GIDs
  - The RACFICE utility
  - Using UNIXMAP profiles
  - Using the RACF `search` command
- Defining OMVS segments using JCL

A.1 Managing RACF user and group profiles

RACF provides utilities such as the RACFICE reporting tool that allow you to see all of the groups currently defined in the RACF database.

A.1.1 Listing GIDs

You can list the RACF groups having an OMVS segment with a GID using the following options:

- The RACFICE utility
- Using UNIXMAP profiles
- Using the RACF `SEARCH` command

The RACFICE utility

The JCL for the RACFICE utility and the output from a report to identify all RACF groups having a GID are shown in Figure A-1, "RACFICE JCL and output to list RACF groups" on page 511.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//STEP1    EXEC PGM=ICETOOL,REGION=1M
//TOOLMSG  DD SYSOUT=*
//PRINT    DD SYSOUT=*
//DFSMSG   DD DUMMY
//INDD     DD DSN=ANTOFF.IRRDBU00,DISP=SHR
//TEMP0001 DD DISP=(NEW,DELETE,DELETE),SPACE=(CYL,(2,5,0)),UNIT=3390
//TOOLIN   DD *
**********************************************************************
 SORT    FROM(INDD) TO(TEMP0001) USING(GIDS)
 DISPLAY FROM(TEMP0001) LIST(PRINT) -
         PAGE -
         TITLE('ALL GROUPS HAVING A GID AND THE GID') -
         DATE(YMD/) -
         TIME(12:)  -
         BLANK -
         ON(10,8,CH)  HEADER('GROUP WITH GID') -
         ON(19,10,CH) HEADER('GID')
**********************************************************************
//GIDSCNTL DD *
SORT    FIELDS=(10,8,CH,A,19,10,CH,A)
 INCLUDE COND=(5,4,CH,EQ,C'0120')
 OPTION  VLSHRT
- 1 -       ALL GROUPS HAVING A GID AND THE GID        03/06/24        01:31:52

GROUP WITH GID   GID
--------------   ----------
AOPOPER          0000000100
BBPAAWG          0000002270
BBPAAWP          0000002272
BBTESTR          0000002271
CBADMGP          0000002203
CBASR1           0000002205
CBCLGP           0000002202
BBPRODR          0000002271
BBTESTR          0000002271
CBADMGP          0000002203
CBASR1           0000002205
CBASR2           0000002216
IMWEB            0000000205
LDAPGRP          0000000002
LOTUSGRP         0000006789
AOPADMIN         0000000101
MISC             0000000106
NOGROUP          0000000000
NWGROUP          0000000000
PKIADM           0000000105
PKIGRP           0000000655
PKI3GRP          0000002003
STG              0000000104
SYS1             0000000002
TTY              0000000000
TWS810           0009876789
USSTEST          0000001047
```

Figure A-1   RACFICE JCL and output to list RACF groups

RLIST UNIXMAP

If you still use UNIXMAP profiles to map RACF groups with GIDs (and you have not done AIM stage 2 yet), you can use the **RLIST** command. For example, to list the RACF groups that are associated with GID 237, issue the command:

```
RLIST UNIXMAP G237 ALL
```

The access list of this command may contain more than one RACF group. Therefore, `G237` is shared between the groups on the access list.

To list all RACF groups with GIDs, issue the command:

```
RLIST UNIXMAP * ALL
```

Note: The output from this command contains all UNIXMAP profiles, namely all profiles for UIDs and all profiles for GIDs. The output is not very useful and depending on the number of user IDS, is very large.

The following command lists all the group names defined in RACF class UNIXMAP:

```
SEARCH CLASS(UNIXMAP)
```

Attention: From OS/390 V2R10 onwards, if you have done the AIM migration to stage 3, you have lost the ability to use RLIST and SEARCH for UNIXMAP.

SEARCH command

For installations at AIM stage 2 or higher, you can use the RACF SEARCH command to determine which groups are assigned a specified GID. For example, to find all groups having a GID of 2270, issue the following command; the results are shown in Figure A-2:

```
SEARCH CLASS(GROUP) GID(2270)
```

```
BBTESTG
BBPRODG
BBOLTRG
BBPAAWG
BBJ2MQG
```

Figure A-2 Groups found with the SEARCH command

A.2 JCL example to define a user OMVS segment

In Figure A-3 on page 513, there is a job to be used by authorized administrators (members of group SECADM) for creation of the following:

- The user's HFS data set
- The OMVS segment for an existing RACF user ID to become a z/OS UNIX user

Do the following:

- Issue the CHOWN command to make the user owner of his directory.
- Issue the CHGRP command to make his default group the owning group of his directory.
- Issue the CHMOD command to change the permission bits for the user's directory to 700

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//*******************************************************************
//* STEP01 -  ALLOCATES THE USER'S HFS DATA SET
//* NOTE: TO EXECUTE THIS JOB, THE AUTOMOUNT FACILITY MUST BE ACTIVE
//* STEP02 -  ADDS AN OMVS SEGMENT TO THE USER ID
//* STEP03 -  ASSIGNS OWNERSHIP OF USER HOME DIRECTORY TO USERID
//*           (OTHERWISE THE USER WILL BE UNABLE TO LOGON TO OMVS)
//* STEP04    ASSIGNS OWNING GROUP TO DEFAULT GROUP OF USER
//* STEP05 -  SETS PERMISSION BITS TO RWX------  (700)
//*******************************************************************
//* 1.  MUST BE RUN:-
//*     - BY A USERID WHO IS PERMITTED TO PROFILE BPX.SUPERUSER IN
//*       CLASS FACILITY
//*     - WITH CAPS OFF .... STEP03,04 AND 05 ARE CASE-SENSITIVE
//* 2.  CHECK THAT THE USERID'S DEFAULT GROUP ALREADY HAS OMVS SEGMENT
//*     - uuuuuuu TO USER ID(LOWER CASE)
//*     - YYYYYYY TO USER ID(UPPER CASE)
//* 4.  CHANGE ggggggg TO USER'S DEFAULT GROUP
//*******************************************************************
//STEP01  EXEC PGM=IEFBR14
//HFS1     DD  DSN=OMVS.YYYYYYY.HFS,SPACE=(CYL,(1,1,1)),
//          DSNTYPE=HFS,DCB=(DSORG=PO),
//          DISP=(NEW,CATLG,DELETE)
//*
//STEP02  EXEC PGM=IKJEFT01,REGION=1M
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
  ALU UUUUUUU OMVS(AUTOUID HOME('/u/uuuuuuu') PROG('/bin/sh'))
//STEP03     EXEC PGM=BPXBATCH,
//   PARM='SH echo chown uuuuuuu /u/uuuuuuu | su'
//STEP04     EXEC PGM=BPXBATCH,
//   PARM='SH echo chgrp ggggggg /u/uuuuuuu | su'
//STEP05     EXEC PGM=BPXBATCH,
//   PARM='SH echo chmod 700 /u/uuuuuuu | su'
```

Figure A-3 JCL to allocate a user HFS and define user ownership

A.3 Methods to list UIDs

You can list the RACF user IDs having an OMVS segment using the following options:

ISHELL command

You can list all user IDs and their OMVS segments by positioning the cursor under Setup, pressing Enter and selecting option 7 (to become a superuser). Now go again to Setup and select option 2 (User list...).

ICETOOL utility

The JCL for the ICETOOL utility is shown in Figure A-4 on page 514, and a partial output from a report to identify all RACF user IDs having an OMVS segment and their UIDs, HOMEs and PROGs, are shown in Figure A-5 on page 515. All users having UID=0 are excluded.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//STEP1    EXEC PGM=ICETOOL,REGION=1M
//TOOLMSG  DD SYSOUT=*
//PRINT    DD SYSOUT=*
//DFSMSG   DD DUMMY
//INDD     DD DSN=ANTOFF.IRRDBU00,DISP=SHR
//TEMP0001 DD DISP=(NEW,DELETE,DELETE),SPACE=(CYL,(2,5,0)),UNIT=3390
//TOOLIN   DD *
***********************************************************************
 SORT    FROM(INDD) TO(TEMP0001) USING(UIDS)
 DISPLAY FROM(TEMP0001) LIST(PRINT) -
         PAGE -
TITLE('USERIDS AND THEIR OMVS SEGMENTS') -
         DATE(YMD/) -
         TIME(12:)  -
         BLANK -
         ON(10,8,CH)  HEADER('USERID') -
         ON(19,10,CH) HEADER('UID') -
         ON(30,14,CH) HEADER('HOME') -
         ON(1054,10,CH) HEADER('PROG')
***********************************************************************
//UIDSCNTL DD *
 SORT    FIELDS=(10,8,CH,A,19,10,CH,A,30,14,CH,A,1054,10,CH,A)
 INCLUDE COND=(5,4,CH,EQ,C'0270',AND,19,10,CH,NE,C'0000000000')
```

Figure A-4   JCL for the ICETOOL report

Figure A-5 on page 515 is the output of ICETOOL.

```
OPTION  VLSHRT
- 1 -        USERIDS AND THEIR OMVS SEGMENTS        03/06/24        04:08:17 pm

USERID     UID          HOME              PROG
--------   ----------   --------------    ----------
-------------------------------------------------
ANTOFF     0000000340   /u/antoff         /bin/sh
ARTG       0000000482   /u/artg           /bin/sh
BARTR7     0000087658   /                 /bin/sh
BBJ2MQC    0000002170   /tmp              /bin/sh
BBJ2MQD    0000002172   /tmp              /bin/sh
BBJ2MQI    0000002173   /tmp              /bin/sh
BBJ2MQS    0000002171   /tmp              /bin/sh
BBOLTRC    0000002170   /tmp              /bin/sh
BBOLTRD    0000002172   /tmp              /bin/sh
BBOLTRI    0000002173   /tmp              /bin/sh
BBPRODD    0000002172   /tmp              /bin/sh
BBPRODI    0000002173   /tmp              /bin/sh
BBPRODS    0000002171   /tmp              /bin/sh
BBTESTC    0000002170   /tmp              /bin/sh
BBTESTD    0000002172   /tmp              /bin/sh
BBTESTI    0000002173   /tmp              /bin/sh
BBTESTS    0000002171   /tmp              /bin/sh
BOCHE      0000000339   /u/boche          /bin/sh
BRADY      0000000012
BUCZAK     0000010004   /                 /bin/sh
CAMILLA    0000010005   /                 /bin/sh
-------------------------------------------------
CHARTEH    0000010069   /                 /bin/sh
CHRISR     0000000337   /u/chrisr         /bin/sh
CHUCKK     0000010006   /                 /bin/sh
-------------------------------------------------
HERING     0000000888   /u/hering         /bin/sh
HERITST    0000000088   /u/heritst        /bin/tcsh
HERZOG     0000010074   /                 /bin/sh
HGT        0000010028   /                 /bin/sh
HIR        0000010126   /                 /bin/sh
HOERNER    0000010029   /                 /bin/sh
HOLL       0000000487   /u/holl           /bin/sh
-------------------------------------------------
KAPPELE    0000000336   /u/kappele        /bin/sh
-------------------------------------------------
PKISERV    0000000555
PKISRV     0000001016   /                 /bin/sh
PKISRVD    0000000554
PKISRV3    0020030422
PKISTU     0000001015   /web/pki1         /bin/sh
PUBLIC     0000000998   /                 /bin/sh
RALPHB     0000000477   /u/ralphb         /bin/sh
RAMA       0000087674   /                 /bin/sh
RANIERI    0000000338   /u/ranieri        /bin/sh
-------------------------------------------------
TRAUNER    0000000335   /u/trauner        /bin/sh
TROWELL    0000010110   /                 /bin/sh
-------------------------------------------------
WUNDERL    0000000484   /u/wunderl        /bin/sh
```

Figure A-5   Output from the ICETOOL utility

RLIST UNIXMAP

If you still use UNIXMAP profiles to map RACF user IDs with UIDs (you have not done AIM stage 2 yet), you can use the RLIST command. For example:

- To list the RACF user IDs that are associated with UID 2170, issue the command:

  ```
  RLIST UNIXMAP U2170 ALL
  ```

- But from OS/390 V2R10 onwards, if you have done the AIM migration to stage 3, you have lost this option.

SEARCH command

For installations at AIM stage 2 or higher, you can use the RACF SEARCH command to determine which user IDs are assigned a specified UID. For example, to find all user IDs having a UID of 2170, issue the command:

```
SEARCH CLASS(USER) GID(2170)
```

The result of the SEARCH command is as follows:

```
BBTESTC
BBPRODC
BBOLTRC
BBPAAWC
BBJ2MQC
```

Occasionally, if your selection of UID picks a nonexistent UID, such as in the following SEARCH command:

```
SEARCH CLASS(USER) UID(1234567)
```

The expected RACF message appears:

```
ICH31005I NO ENTRIES MEET SEARCH CRITERIA
```

This support requires at least application identity mapping stage 2.

A.4 The ICETOOL utility

The IBM DFSORT product provides a reporting facility called ICETOOL. You can create ICETOOL reports based on output files from the RACF database unload utility (IRRDBU00) or the SMF data unload utility (IRRADU00). The SYS1.SAMPLIB member IRRICE contains DFSORT statements for record selection and ICETOOL statements for report formatting for a wide variety of reports. The IEBUPDTE utility processes the IRRICE member and creates a partitioned data set (PDS) that contains two PDS members for each report. The two members contain:

- The report format
- The record selection criteria

The use of ICETOOL is described in the following sections.

A.4.1 Unload the RACF database

Unload the RACF database using the RACF database unload utility (IRRDBU00) for input to the IRRICE reports. Figure A-6 on page 517 shows sample JCL for the IRRDBU00 utility.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//IRRDBU00 EXEC PGM=IRRDBU00,PARM='NOLOCK'
//SYSPRINT DD SYSOUT=*
//INDD1 DD DISP=SHR,DSN=SYS1.RACFESA
//OUTDD DD DSN=ANTOFF.IRRDBU00,DISP=(,CATLG),
//*         VOL=SER=SBOX20,
//          SPACE=(CYL,(15,5)),
//          DCB=(LRECL=4096,RECFM=VB),
//          UNIT=3390
```

Figure A-6 Sample JCL for the IRRDBU00 utility

A.4.2 Run a UIDs report using ICETOOL

Use ICETOOL to run the UIDs report from member IRRICE using the JCL shown in Figure A-7. The sort statements identify all shared UIDs.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//STEP1    EXEC PGM=ICETOOL,REGION=1M
//TOOLMSG  DD SYSOUT=*
//PRINT    DD SYSOUT=*
//DFSMSG   DD DUMMY
//INDD     DD DSN=ANTOFF.IRRDBU00,DISP=SHR
//TEMP0001 DD DISP=(NEW,DELETE,DELETE),SPACE=(CYL,(2,5,0)),UNIT=3390
//TOOLIN   DD *
**********************************************************************
 COPY    FROM(INDD) TO(TEMP0001) USING(UIDS)
 OCCURS  FROM(TEMP0001) LIST(PRINT) -
         PAGE -
         TITLE('SHARED UNIX UIDS AND NUMBER OF SHARING USERS') -
         DATE(YMD/) -
         TIME(12:)  -
         BLANK -
         ON(19,10,CH) HEADER('UNIX UID') -
         ON(VALCNT)   HEADER('NUMBER OF SHARING USERS') -
         HIGHER(1)
**********************************************************************
//UIDSCNTL DD *
 SORT    FIELDS=(19,10,CH,A)
 INCLUDE COND=(5,4,CH,EQ,C'0270')
 OPTION  VLSHRT
```

Figure A-7 JCL for ICETOOL UIDs report

Figure A-8 on page 518 shows output from the UIDs JCL.

```
ICE601I 0 DFSORT ICETOOL UTILITY RUN ENDED - RETURN CODE:  00
- 1 -        SHARED UNIX UIDS AND NUMBER OF SHARING USERS        03/06/20

UNIX UID     NUMBER OF SHARING USERS
----------   -----------------------
0000000000                       142
0000000001                         2
0000000012                         2
0000002170                         5
0000002171                         5
0000002172                         5
0000002173                         5
```

Figure A-8 Output from UIDs report

Assign unique UIDs using ICETOOL

If you wish to assign unique UIDs to user IDs who share a UID, you can run an ICETOOL report with the JCL shown in Figure A-9 using all nonzero shared UIDs as input to the INCLUDE COND statements.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//STEP1    EXEC PGM=ICETOOL,REGION=1M
//TOOLMSG  DD SYSOUT=*
//PRINT    DD SYSOUT=*
//DFSMSG   DD DUMMY
//INDD     DD DSN=ANTOFF.IRRDBU00,DISP=SHR
//TEMP0001 DD DISP=(NEW,DELETE,DELETE),SPACE=(CYL,(2,5,0)),UNIT=3390
//TOOLIN   DD *
***********************************************************************
 SORT    FROM(INDD) TO(TEMP0001) USING(UIDS)
 DISPLAY FROM(TEMP0001) LIST(PRINT) -
         PAGE -
         TITLE('SHARED UNIX UIDS AND SHARING USERIDS') -
         DATE(YMD/) -
         TIME(12:)  -
         BLANK -
         ON(19,10,CH) HEADER('SHARED UNIX UID') -
         ON(10,8,CH)  HEADER('SHARING USERIDS')
***********************************************************************
//UIDSCNTL DD *
 SORT    FIELDS=(19,10,CH,A,10,8,CH,A)
 INCLUDE COND=(5,4,CH,EQ,C'0270',AND,19,10,CH,EQ,C'0000000001',
               OR,19,10,CH,EQ,C'0000000012',
               OR,19,10,CH,EQ,C'0000002170',
               OR,19,10,CH,EQ,C'0000002171',
               OR,19,10,CH,EQ,C'0000002172',
               OR,19,10,CH,EQ,C'0000002173')
 OPTION  VLSHRT
```

Figure A-9 Report identifying user IDs sharing a UID

The output from the above report is shown in Figure A-10 on page 519.

```
ICE601I 0 DFSORT ICETOOL UTILITY RUN ENDED - RETURN CODE:  00
- 1 -       SHARED UNIX UIDS AND SHARING USERIDS       03/06/20       02:27:4

SHARED UNIX UID   SHARING USERIDS
---------------   ---------------
0000000001        LDAPSRV
0000000001        MSYSLDAP
0000000012        BRADY
0000000012        MARCY
0000002170        BBJ2MQC
0000002170        BBOLTRC
0000002170        BBPAAWC
0000002170        BBPRODC
0000002170        BBTESTC
0000002171        BBJ2MQS
0000002171        BBOLTRS
0000002171        BBPAAWS
0000002171        BBPRODS
0000002171        BBTESTS
0000002172        BBJ2MQD
0000002172        BBOLTRD
0000002172        BBPAAWD
0000002172        BBPRODD
0000002172        BBTESTD
0000002173        BBJ2MQI
0000002173        BBOLTRI
0000002173        BBPAAWI
0000002173        BBPRODI
0000002173        BBTESTI
```

Figure A-10 List of user IDs sharing a UID

Note: If your installation has more than 130 matching UIDs, see “More than 130 user IDs with the same GID/UID” on page 526.

After assigning a unique UID to each of the sharing user IDs (manually or using AUTOUID), you have to go to the UNIX shell and find all occurrences of the old UIDs in files and directories as owner and permission bits and change them to the newly assigned UIDs. For more information on how to achieve this, see Appendix C, “Access control list (ACL) support considerations” on page 559.

A.4.3 GID reports using ICETOOL

You can use ICETOOL to run the GIDS report from member IRRICE using the JCL shown in Figure A-11 on page 520. The sort statements identify all shared GIDs.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//STEP1    EXEC PGM=ICETOOL,REGION=1M
//TOOLMSG  DD SYSOUT=*
//PRINT    DD SYSOUT=*
//DFSMSG   DD DUMMY
//INDD     DD DSN=ANTOFF.IRRDBU00,DISP=SHR
//TEMP0001 DD DISP=(NEW,DELETE,DELETE),SPACE=(CYL,(2,5,0)),UNIT=3390
//TOOLIN   DD *
**********************************************************************
 COPY    FROM(INDD) TO(TEMP0001) USING(GIDS)
 OCCURS  FROM(TEMP0001) LIST(PRINT) -
         PAGE -
         TITLE('SHARED UNIX GIDS AND NUMBER OF SHARING GROUPS') -
         DATE(YMD/) -
         TIME(12:)  -
         BLANK -
         ON(19,10,CH) HEADER('UNIX GID') -
         ON(VALCNT)   HEADER('NUMBER OF SHARING GROUPS') -
         HIGHER(1)
**********************************************************************
//GIDSCNTL DD *
 SORT    FIELDS=(19,10,CH,A)
 INCLUDE COND=(5,4,CH,EQ,C'0120')
 OPTION  VLSHRT
```

Figure A-11   JCL to run GID reports

The output from report GIDS is shown in Figure A-12.

```
ICE601I 0 DFSORT ICETOOL UTILITY RUN ENDED - RETURN CODE:  00
- 1 -        SHARED UNIX GIDS AND NUMBER OF SHARING GROUPS        03/06/20

UNIX GID     NUMBER OF SHARING GROUPS
----------   ------------------------
0000000000                          3
0000000002                          4
0000002270                          5
0000002271                          5
0000002272                          5
```

Figure A-12   Output from GID report

Assign unique GIDs to groups

If you wish to assign unique GIDs to groups which share a GID, you can run an ICETOOL report with the JCL shown in Figure A-13 on page 521 using the shared GIDs as input to the INCLUDE COND statements.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//STEP1    EXEC PGM=ICETOOL,REGION=1M
//TOOLMSG  DD SYSOUT=*
//PRINT    DD SYSOUT=*
//DFSMSG   DD DUMMY
//INDD     DD DSN=ANTOFF.IRRDBU00,DISP=SHR
//TEMP0001 DD DISP=(NEW,DELETE,DELETE),SPACE=(CYL,(2,5,0)),UNIT=3390
//TOOLIN   DD *
***********************************************************************
 SORT    FROM(INDD) TO(TEMP0001) USING(GIDS)
 DISPLAY FROM(TEMP0001) LIST(PRINT) -
         PAGE -
         TITLE('SHARED UNIX GIDS AND SHARING GROUPS') -
         DATE(YMD/) -
         TIME(12:)  -
BLANK -
         ON(19,10,CH) HEADER('SHARED UNIX GID') -
         ON(10,8,CH)  HEADER('SHARING GROUPS')
***********************************************************************
//GIDSCNTL DD *
 SORT    FIELDS=(19,10,CH,A,10,8,CH,A)
 INCLUDE COND=(5,4,CH,EQ,C'0120',AND,19,10,CH,EQ,C'0000000000',
               OR,19,10,CH,EQ,C'0000000002',
               OR,19,10,CH,EQ,C'0000002270',
               OR,19,10,CH,EQ,C'0000002271',
               OR,19,10,CH,EQ,C'0000002272')
 OPTION  VLSHRT
```

Figure A-13 Report identifying all groups sharing a GID

The output from the above report is shown in Figure A-14 on page 522.

```
ICE601I 0 DFSORT ICETOOL UTILITY RUN ENDED - RETURN CODE:  00
- 1 -        SHARED UNIX GIDS AND SHARING GROUPS        03/06/20        02:39:11

SHARED UNIX GID   SHARING GROUPS
---------------   --------------
0000000000        NOGROUP
0000000000        NWGROUP
0000000000        TTY
0000000002        FWGRP
0000000002        GLDGRP
0000000002        ITSOLDP
0000000002        LDAPGRP
0000000002        SYS1
0000002270        BBJ2MQG
0000002270        BBOLTRG
0000002270        BBPAAWG
0000002270        BBPRODG
0000002270        BBTESTG
0000002271        BBJ2MQR
0000002271        BBOLTRR
0000002271        BBPAAWR
0000002271        BBPRODR
0000002271        BBTESTR
0000002272        BBJ2MQP
0000002272        BBOLTRP
0000002272        BBPAAWP
0000002272        BBPRODP
0000002272        BBTESTP
```

Figure A-14 List of groups sharing a GID

After assigning a unique GID to each of the sharing groups (manually or using AUTOGID) you have to go to the UNIX shell and find all occurrences of the old GIDs in files and directories as owning group and permission bits and change them to the newly assigned GIDs.

A.4.4 Backing up the primary RACF database

You can run the IRRUT400 utility to back up the primary RACF database. Figure A-15 shows sample JCL for IRRUT400.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//STEP1   EXEC PGM=IRRUT400,PARM='NOLOCKINPUT,FREESPACE(30)'
//SYSPRINT DD SYSOUT=*
//INDD1  DD DSN=SYS1.RACFESA,DISP=SHR
//OUTDD1 DD DSN=SYS1.RACF.ANTOFF.B062003,DISP=(,CATLG),
//          VOL=SER=SBOX02,
//          SPACE=(CYL,25,,CONTIG),
//          DCB=DSORG=PSU,
//          UNIT=3390
```

Figure A-15 Sample JCL for the IRRUT400 utility

A.4.5 Statistics on the UNIXMAP class

After the backup has run successfully, it is essential to run the IRRUT200 utility to obtain the statistics on the UNIXMAP class, and the LNOTES and NDS segments. Figure A-16 shows sample JCL for the IRRUT200 utility.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//STEP1   EXEC PGM=IRRUT200
//SYSRACF  DD DSN=SYS1.RACFESA,DISP=SHR
//SYSUT1   DD UNIT=3390,SPACE=(CYL,25,,CONTIG),
//            DCB=DSORG=PSU
//SYSUT2   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
     INDEX FORMAT
     MAP ALL
     END
```

Figure A-16 Sample JCL for the IRRUT200 utility

Check the output from the IRRUT200 job, in particular the statistics at the bottom of the output as shown in Figure A-17.

```
NUMBER OF NOTELINK ENTRIES - 0000009
NUMBER OF NDSLINK ENTRIES - 0000007
NUMBER OF UNIXMAP ENTRIES - 0000083
```

Figure A-17 Output from the IRRUT200 utility

A.4.6 Backing up the RACF database using the IRRIRA00 utility

We recommend that before you start the IRRIRA00 utility for application identity mapping (AIM), you inactivate your RACF backup database. Check to see what RACF databases are online, using the RVARY LIST command, as follows:

```
RVARY LIST
ICH15013I RACF DATABASE STATUS:
ACTIVE   USE    NUMBER   VOLUME     DATASET
------   ---    ------   ------     -------
 YES     PRIM      1     SBOX02     SYS1.RACFESA
 YES     BACK      1     SBOX01     SYS1.RACF.BKUP1
MEMBER SC64     IS SYSPLEX COMMUNICATIONS ENABLED & IN NON-DATA SHARING MODE.
ICH15020I RVARY COMMAND HAS FINISHED PROCESSING.
```

Next, inactivate the RACF backup database by using the RVARY INACTIVE command, as follows:

```
RVARY INACTIVE,DATASET(SYS1.RACF.BKUP1)
```

After entering this command you must respond to the WTO message with the correct STATUS password. You can use the SDSF command ACTION ALL and respond to the message from your SDSF LOG screen.

```
ICH15013I RACF DATABASE STATUS:
ACTIVE   USE    NUMBER   VOLUME     DATASET
------   ---    ------   ------     -------
 YES     PRIM      1     SBOX02     SYS1.RACFESA
 NO BACK      1     *DEALLOC SBOX01 SYS1.RACF.BKUP1
MEMBER SC64     IS SYSPLEX COMMUNICATIONS ENABLED & IN NON-DATA SHARING MODE.
```

```
ICH15020I RVARY COMMAND HAS FINISHED PROCESSING.
```

A.4.7 IRRIRA00 utility - stage 0 to stage 1

You can now run the first stage of the IRRIRA00 utility, to go from stage 0 to stage 1, as shown in Figure A-18.

```
//AIMSTAGE JOB
//STEP EXEC PGM=IRRIRA00,PARM=STAGE(1)
//SYSPRINT DD SYSOUT=A

IRR66017I The system is currently operating in stage 0.
IRR66018I Stage 1 requested. Database now at requested stage 1.
IRR66007I Backup RACF database not converted to stage 1. It is not active.
```

Figure A-18 Sample JCL and output for IRRIRA00 utility stage 1

You get the warning message IRR66007I stating that the RACF backup database is not active. Since you inactivated the RACF backup database, this message is acceptable.

IRRIRA00 utility - stage 1 to stage 2

You now move to the next step of the stage enablement by executing the utility IRRIRA00 again. The only modification required is to change the STAGE parameter from 1 to 2, as shown in Figure A-19.

```
//AIMSTAGE JOB
//STEP EXEC PGM=IRRIRA00,PARM=STAGE(2)
//SYSPRINT DD SYSOUT=A

IRR66017I The system is currently operating in stage 1.
IRR66018I Stage 2 requested. Database now at requested stage 2.
IRR66007I Backup RACF database not converted to stage 2. It is not active.
```

Figure A-19 Sample JCL and output for IRRIRA00 utility stage 2

IRRIRA00 utility - stage 2 to stage 3

To complete the migration, we run the utility IRRIRA00 to move from stage 2 to stage 3.

Again, we must emphasize that you do not go to stage 3 until you have run to stage 2, without getting any LOGRECs indicating an AIM failure. Additionally, all systems sharing the database must be at the same code level supporting AIM.

Note: It is not possible to go back after you have reached stage 3 unless you back out using the backup copy of the RACF primary database taken in step 4.

Sample JCL for executing the IRRIRA00 utility to change from stage 2 to stage 3 is shown in Figure A-20 on page 525.

```
//AIMSTAGE JOB
//STEP EXEC PGM=IRRIRA00,PARM=STAGE(3)
//SYSPRINT DD SYSOUT=A

IRR66017I The system is currently operating in stage 2.
IRR66018I Stage 3 requested. Database now at requested stage 3.
IRR66007I Backup RACF database not converted to stage 3. It is not active.
```

Figure A-20 Sample JCL and output for IRRIRA00 utility stage 3

A.4.8 Inactivate UNIXMAP class

After the successful migration to stage 3, the class UNIXMAP has to be inactivated from RACF with the command:

```
SETR NOCLASSACT(UNIXMAP)
```

Note: You have to inactivate classes NOTELINK and NDSLINK if they were in use at your installation.

A.4.9 COFVLF00 parmlib member

In the COFVLF00 member of SYS1.PARMLIB, delete references to the IRRGMAP and IRRUMAP definitions, because they are related to the UNIXMAP class. In our installation, member COFVLF00, after deleting these references, is shown in Figure A-21.

```
CLASS NAME(CSVLLA)          /* Class name for Library Lookaside @P2C*/
      EMAJ(LLA)             /* Major name for Library Lookaside @P2C*/
CLASS NAME(IRRGTS)          /* RACF GTS Table                       */
      EMAJ(GTS)             /* Enable caching of RACF GTS           */
CLASS NAME(IRRACEE)         /* RACF saved ACEEs                     */
      EMAJ(ACEE)            /* Enable caching of RACF ACEE          */
CLASS NAME(IRRSMAP)         /* OpenEdition security packet          */
      EMAJ(SMAP)            /* Major Name = SMAP                    */
```

Figure A-21 Member COFVLF00 of SYS1.PARMLIB

A.4.10 Rerun the IRRUT200 utility

Run the IRRUT200 utility again to check for the codes that have replaced OMVS UID/GID, LNOTES SNAME, and NDS UNAME, as shown in Figure A-22 on page 526. Also, at the bottom of the statistics section of the IRRUT200 job, there will be no references anymore to the UNIXMAP, NOTELINK and NDSLINK classes.

010302	OMVS GID alias index key prefix
020802	OMVS UID alias index key prefix
020C02	LNOTES SNAME alias index key prefix
020D02	NDS UNAME alias index key prefix

```
06F 0006 0103020076AE3B
088 0000 02080200000000
411 0000 020C02aims01
5C1 0000 020D02AIMS06
```

Figure A-22 Rerun of IRRUT200 to check for codes

A.4.11 Replace the RACF backup database

Replace the RACF backup database with a copy of the RACF primary database, using the IRRUT400 utility, as shown in Figure A-23.

```
//ANTOFFZ  JOB MSGCLASS=H,CLASS=A,NOTIFY=ANTOFF
//STEP1   EXEC PGM=IRRUT400,PARM='NOLOCKINPUT,FREESPACE(30)'
//SYSPRINT DD SYSOUT=*
//INDD1  DD DSN=SYS1.RACFESA,DISP=SHR
//OUTDD1 DD DSN=SYS1.RACF.BKUP1,DISP=(,CATLG),
//          VOL=SER=SBOX04,
//          SPACE=(CYL,25,,CONTIG),
//          DCB=DSORG=PSU,
//          UNIT=3390
```

Figure A-23 JCL to replace the RACF primary database

A.4.12 Activate the backup database

After successful completion of the IRRUT400 utility, activate the backup RACF again, using the RVARY ACTIVE command, as follows:

```
RVARY ACTIVE,DATASET(SYS1.RACF.BKUP1)
```

You must respond to the WTO message with the correct RVARY SWITCH password.

For an explanation of the space savings in the RACF database achieved with AIM, refer to *Putting the Latest z/OS Security Features to Work,* SG24-6540.

More than 130 user IDs with the same GID/UID

The IRRIRA00 utility fails when it is used to move a database from stage 0 to stage 1 and the database contains more than 130 8-character user IDs mapping to the same GID/UID. The utility will fail after some higher number if your user IDs have fewer characters. Before IRRIRA00 can run again against the database successfully, the number of mappings must be reduced and the BLKUPD utility must be used to correct some information in the ICB for the database.

Figure A-24 shows the messages IRRIRA00 issues when it detects that the database maps more user IDs to a single UID than it can handle. IRRIRA00 does not proceed to stage 1.

```
IRR66017I The system is currently operating in stage 0.
IRR66016I Unexpected RACF manager return code updating entry JP in class USER. Return
code 132. Reason code 0.
IRR66009I Last entry processed successfully was HOERNER in class USER.
```

Figure A-24 Error messages issued by IRRIRA00 when more than 130 IDs share a GID/UID

Altering or deleting the user IDs that share the same UID will not eliminate these messages the next time IRRIRA00 is run.

The following steps must be performed before IRRIRA00 can successfully move a database to stage 1 again:

1. Familiarize yourself with the use of the BLKUPD command. It is documented in *z/OS Security Server RACF Diagnosis Guide,* GA22-7689.
2. Issue the BLKUPD command from the READY prompt as follows:

```
READY
BLKUPD 'SYS1.RACFESA'
BLKUPD:
```

3. Read in the block at relative byte address 0, indicating that it will be updated as shown in the following command

```
BLKUPD:
READ 0 UPDATE
BLKUPD:
```

4. List the byte value at offset x'16F'. If it is not zero, then issue the replace command to set it to zero. Issue the LIST NEW keyword again to verify that the change is correct. The LIST and REP keywords of the BLKUPD command are as follows:

```
BLKUPD:
LIST NEW RANGE(X'16F',1)
RBA=000000000000
016F 01
*.
BLKUPD:
REP x'00' OFFSET(x'16F') VER(x'01')
IRR63004I REPLACE complete.
BLKUPD:
```

5. List the 12 bytes at offset x'3E0' and replace them with zeros. Issue the LIST NEW keyword to verify the change, as follows:

```
LIST RANGE(X'3E0',12)
RBA=000000000000
03E0 00000005 20000000 00052000
*............
BLKUPD:
REP X'000000000000000000000000' OFFSET(X'3E0') VER(X'000000052000000000052000')
IRR63004I REPLACE complete.
BLKUPD:
LIST NEW RANGE(X'3E0',12)
RBA=000000000000
03E0 00000000 00000000 00000000
*............
BLKUPD:
```

6. Save the updates using the END SAVE keyword of the BLKUPD command, as follows:

```
BLKUPD:
END SAVE
IRR63013I READ ended. Block saved.
BLKUPD:
```

7. End the BLKUPD session using the END subcommand, as follows:

```
BLKUPD:
END
READY
```

These steps will leave some residual storage in the database. Use the IRRUT400 utility to clean up the unused storage. You can now restart the AIM migration process.

Installation files

This appendix contains sample files that can be used during the installation process of z/OS UNIX.

We show examples of the following files:

- SYS1.PROCLIB(OMVS)
- SYS1.SAMPLIB(BPXPMxx)
- /sample/inetd.conf
- /samples/init.options
- /samples/profile
- /samples/rc

SYS1.PROCLIB(BPXAS)

Example B-1 shows SYS1.PROCLIB(BPXAS) at the RMID=HBB7707 level.

Example: B-1   BPXAS

```
//*-------------------------------------------------------------------- 00050000
//IEFPROC   EXEC   PGM=IEFIIC,DPRTY=12,PARM=',,&GETWORK,BPXPRJRW'       00100000
//                                                                      00150000
//*******************************************************************/ 00200000
//*                                                                 */ 00250000
//*01* PROCEDURE NAME : BPXAS                                       */ 00300000
//*                                                                 */ 00550000
//*01* FUNCTION:                                                    */ 00600000
//*                                                                 */ 00650000
//*       The BPXAS procedure is used to start the MVS Initiator,   */ 00700000
//*       it invokes the module IEFIIC.                             */ 00750000
//*                                                                 */ 00800000
//*       DPRTY=12  -  Sets the dispatching priority for the        */ 00850000
//*                    initiator.                                   */ 00900000
//*                                                                 */ 00950000
//*01* METHOD OF ACCESS:                                            */ 01000000
//*                                                                 */ 01050000
//* Do not start this procedure externally.  This procedure is      */ 01100000
//* started internally whenever an OE address space is needed to    */ 01150000
//* process a fork or spawn request.                                */ 01200000
//*                                                                 */ 01250000
//*01* COMPONENT:                                                   */ 01300000
//*                                                                 */ 01350000
//*       SC1B6 (Initiator)                                         */ 01400000
//*                                                                 */ 01450000
//*01* DISTRIBUTION LIBRARY:                                        */ 01500000
//*                                                                 */ 01550000
//*       SYS1.PROCLIB                                              */ 01600000
//*                                                                 */ 01650000
//*01* CHANGE ACTIVITY:                                             */ 01700000
//*                                                                 */ 01750000
//*                                                                 */ 01800000
//*******************************************************************/ 01850000
```

SYS1.PROCLIB(BPXOINIT)

Example B-2 shows SYS1.PROCLIB(BPXOINIT) at the RMID=HBB7707 level.

Example: B-2   BPXOINIT

```
//********************************************************/           00050000
//*                                                      */           00100000
//*     $MAC(BPXOINIT) COMP(SCPX1) PROD(BPX):            */           00150000
//*                                                      */           00200000
//* INIT PROCESS STARTUP PROCEDURE                       */           00250000
//*                                                      */           00253800
//********************************************************/           00500000
//BPXOINIT PROC                                                        00550000
//BPXOINIT EXEC PGM=BPXPINPR,REGION=OK,TIME=NOLIMIT                    00600000
```

SYS1.PROCLIB(OMVS)

Example B-3 shows SYS1.PROCLIB(OMVS) at the RMID=HBB7707 level.

Example: B-3 OMVS

```
//*******************************************************/         00050000
//*                                                     */         00100000
//*     $MAC(OMVS) COMP(SCPX1) PROD(BPX):               */         00150000
//*                                                     */         00200000
//* OPENMVS STARTUP PROC                                */         00250000
//*                                                     */         00300000
//* THE OPENMVS KERNEL ADDRESS SPACE IS STARTED AS PART */         00350000
//* OF SYSTEM INITIALIZATION.                           */         00400000
//*                                                     */         00450000
//* CHANGE-ACTIVITY:                                    */         00500000
//*                                                     */         00550000
//*  $00=OW06506  HOM1130 940912 PDJI: FREE IEFPARM     */         00600000
//*  $D0=DRGA112  HOM1150 950918 PDI6: PERMANENT KERNEL */         00650000
//*  $P0=PUX0571  HBB6605 970925 PDI6: REMOVE S OMVS CMT */         00700000
//*  $D1=DWKA335  HBB6608 980810 PDJC: A335.00 LIMITS   */         00750000
//*******************************************************/         00800000
//OMVS PROC                                                        00850000
//OMVS EXEC     PGM=BPXINIT,REGION=0K,TIME=NOLIMIT                 00900000
```

SYS1.SAMPLIB(BPXPRMXX)

Example B-4 shows SYS1.SAMPLIB(BPXPRMXX) at the RMID=HBB7707 level.

Example: B-4 BPXPRMXX

```
/*01* PROPRIETARY STATEMENT=                                         */ 00050000
/***PROPRIETARY_STATEMENT********************************************/ 00100000
/*                                                                   */ 00150000
/*                                                                   */ 00200000
/* LICENSED MATERIALS - PROPERTY OF IBM                              */ 00250000
/* THIS MACRO IS "RESTRICTED MATERIALS OF IBM"                       */ 00300000
/* 5694-A01 (C) COPYRIGHT IBM CORP. 1993, 2002                       */ 00350000
/*                                                                   */ 00400000
/* STATUS= HBB7707                                                   */ 00450000
/*                                                                   */ 00500000
/* $02=OW19605   HOM1130  960419  PDJM: Provide working BPXPRMxx    */ 00550000
/* $P0=PSP0066   HOM1150  960501  PDAE: MAXFILESIZE NOLIMIT         */ 00600000
/* $P1=PSP0072   HOM1150  960501  PDAE: PRIORITYPG & GOAL NONE      */ 00650000
/* $D0=DSYA199   JBB6604  960916  PDAE: A199.01 RUNOPTS() support   */ 00700000
/* $D1=DSYA211   JBB6604  961210  PDAE: A211.01 SYSCALL_COUNTS      */ 00750000
/* $P2=PUX0164   HBB6605  970430  PDJV: Add commented filesystems   */ 00800000
/* $D2=PVTA242   HBB6606  970922  PDJM: Add NOARGS                  */ 00850000
/* $P3=PWA0116   JBB6607  980428  PDJM: remove PVTA242 (NOARGS)     */ 00900000
/* $P4=PWA0292   JBB6607  980716  PDJV: remove BPXTIINT stuff       */ 00950000
/* $P5=PWA0309   JBB6607  980723  PDPD: placed filenames under /etc */ 01000000
/* $D3=DWKA335   HBB6608  980810  PDJC: A335.00 max limits changes  */ 01050000
/*                                                                   */ 01100000
/* $P6=PWK0423   HBB6608  981123  PDOC: OE->OS/390 UNIX Name Change */ 01150000
/* $P7=PWK0789   HBB6608  990407  PDJQ: Add MAXQUEUEDSIGS           */ 01200000
/* $P8=PWK0900   HBB6608  990408  PDJT: R8 Editorial Changes        */ 01250000
/*                                                                   */ 01300000
/* $D4=DWYA309   JBB6609  980508  PDJI: Add SYSPLEX=xx              */ 01350000
/* $D5=DWYA315   JBB6609  990104  PDAE: Add SHRLIBRGNSIZE          */ 01400000
/* $D6=DWYA315   JBB6609  990115  PDAE: Add SHRLIBMAXPAGES         */ 01450000
/* $P8=PWY0316   JBB6609  990409  PDJT: Roll up R8 Changes         */ 01500000
/* $01=OW40336   JBB6609  990802  PDNU: New MaxFileProc (PWY0737)  */ 01550000
/*                                      (PXD0418)                  */ 01600000
/* $D7=DXDA402   HBB7703  990831  PDJN: Remove MAXRTYS (A402.00)   */ 01650000
/* $P9=PXD1415   HBB7703  000228  PDAE: Change Defaults to match   */ 01700000
/*                                      Wizard's suggestions       */ 01750000
/* $D8=DXVA405   HBB7704  000127  PDUA: Added LIMMSG   (A405.00)   */ 01800000
/* $D9=DXOA412   HBB7705  000424  PDAE: Added AUTOCVT and TAG      */ 01850000
/* $DA=DXOA514   HBB7705  000915  PDNU: New Resolver               */ 01900000
/* $PA=PXO0588   HBB7705  000915  PD2R: Wait/Nowait option deleted */ 01925000
/* $DB=DYMA522   HBB7706  001107  PDAE: DCR A522.00 UNMOUNT        */ 01937500
/* $DC=DYMA565   HBB7706  010529  PDAE: DCR A565.01 asname parms   */ 01943700
/*                                PDJT: & OS/390->z/OS Name Change */ 01946800
/* $DD=DYMA579   HBB7707  011109  PDFO: DCR A579.00 AUTOMOVE SYSLIST*/ 01948400
/* $PB=PYV0412   HBB7707  020214  PDJV: Remove reference to BPXTLINT*/ 01949200
/* $PC=PYV0529   HBB7707  020410  PDJT: Updates for AF_INET6       */ 01949600
/*                                                                 */ 01950000
/***END_OF_PROPRIETARY_STATEMENT***********************************/ 02050000
                                                                      02100000
                                                                      02150000
/******************************************************************/ 02200000
/*                                                                 */ 02250000
/*      $MAC(BPXPRMXX) COMP(SCPX1) PROD(BPX):                      */ 02300000
/*                                                                 */ 02350000
/*   This is a sample BPXPRMxx member of SYS1.PARMLIB.             */ 02400000
/*                                                                 */ 02450000
/*   BPXPRMxx parmlib members contain customization values         */ 02500000
/*   for z/OS UNIX System Services. They also contain file         */ 02550000
/*   system information required for its start up.                 */ 02600000
/*                                                                 */ 02650000
/*   This member illustrates the syntax of the following statement  */ 02700000
/*   types:                                                        */ 02750000
/*                                                                 */ 02800000
/*   MAXPROCSYS          MAXPROCUSER (P)       MAXUIDS             */ 02850000
/*   MAXFILEPROC (P)     MAXPTYS                                   */ 02900000
/*   CTRACE              STEPLIBLIST           FILESYSTYPE         */ 02950000
/*   ROOT                MOUNT                 NETWORK             */ 03000000
/*   SUBFILESYSTYPE      MAXTHREADTASKS (P)    MAXTHREADS (P)      */ 03050000
/*   PRIORITYPG          PRIORITYGOAL          IPCMSGNIDS          */ 03100000
/*   IPCMSGQBYTES        IPCMSGQMNUM           IPCSHMNIDS          */ 03150000
/*   IPCSHMSPAGES        IPCSHMMPAGES          IPCSHMNSEGS (P)     */ 03200000
/*   IPCSEMNIDS          IPCSEMNSEMS           IPCSEMNOPS          */ 03250000
/*   MAXMMAPAREA         MAXFILESIZE (P)       MAXCORESIZE (P)     */ 03300000
/*   MAXASSIZE           MAXCPUTIME            MAXSHAREPAGES       */ 03350000
/*   FORKCOPY            SUPERUSER             RUNOPTS             */ 03400000
/*   SYSCALL_COUNTS      USERIDALIASTABLE      TTYGROUP            */ 03450000
/*   STARTUP_PROC        STARTUP_EXEC          MAXQUEUEDSIGS (P)   */ 03500000
/*   SYSPLEX             SHRLIBRGNSIZE         SHRLIBMAXPAGES      */ 03550000
/*   LIMMSG              AUTOCVT                                   */ 03600000
/*                                                                 */ 03650000
/*   NOTE:                                                         */ 03700000
/*                                                                 */ 03750000
/*   This SAMPLIB member is only an example. The value             */ 03800000
/*   represented on each statement is not necessarily an           */ 03850000
/*   IBM-recommended value.                                        */ 03900000
/*   Installations may use this member as a sample, and            */ 03950000
/*   modify it according to their needs.                           */ 04000000
/*                                                                 */ 04050000
/*   NOTE:  Parameter values that require additional setup in order */ 04100000
/*   to be used are commented out.  Before removing the comment     */ 04150000
/*   delimiters, be sure to first perform the required setup.       */ 04200000
/*                                                                  */ 04250000
/*   (P): A (P) behind a parameter indicates, that this parameter   */ 04300000
/*        is changeable individualy for each process, by using      */ 04350000
/*        the SETOMVS PID=<pid>,<limitname>=<newvalue) command.     */ 04400000
/*                                                                  */ 04450000
/*        In addition it is still changeable system-wide for all    */ 04500000
/*        running and future processes, by omitting the PID=        */ 04550000
/*        parameter.                                                */ 04600000
/*                                                                  */ 04650000
/********************************************************************/ 04700000
                                                                       04750000
/********************************************************************/ 04800000
/*                                                                  */ 04850000
/*   Specify the maximum number of processes that z/OS UNIX         */ 04900000
/*   will allow to be active concurrently.                          */ 04950000
/*                                                                  */ 05000000
/*   Notes:                                                         */ 05050000
/*                                                                  */ 05100000
/*   1. Minimum allowable value is 5.                               */ 05150000
/*   2. Maximum allowable value is 32767.                           */ 05200000
/*   3. If this parameter is not provided, the system default       */ 05250000
/*      value for this parameter is 900.                            */ 05300000
/*                                                                  */ 05350000
/********************************************************************/ 05400000
  MAXPROCSYS(900)                  /* System will allow at most 900    05450000
                                      processes to be active           05500000
                                      concurrently              @P9C*/ 05550000
                                                                       05600000
/********************************************************************/ 05650000
/*                                                                  */ 05700000
/*   Specify the maximum number of processes that a single user     */ 05750000
/*   (that is, with the same UID) is allowed to have concurrently   */ 05800000
/*   active regardless of origin.                                   */ 05850000
/*                                                                  */ 05900000
/*   Notes:                                                         */ 05950000
/*                                                                  */ 06000000
/*   1. This parameter is the same as the Child_Max variable        */ 06050000
/*      defined in POSIX 1003.1.                                    */ 06100000
/*   2. Minimum allowable value is 3.                               */ 06150000
/*   3. Maximum allowable value is 32767.                           */ 06200000
/*   4. If this parameter is not provided, the system default       */ 06250000
/*      value for this parameter is 25.                             */ 06300000
/*                                                                  */ 06350000
/********************************************************************/ 06400000
  MAXPROCUSER(25)                  /* Allow each user (same UID) to    06450000
                                      have at most 25 concurrent       06500000
                                      processes active              */ 06550000
                                                                       06600000
/********************************************************************/ 06650000
/*                                                                  */ 06700000
/*   Specify the maximum number of unique UID's that can be using   */ 06750000
/*   z/OS UNIX services at a given time.                            */ 06800000
/*                                                                  */ 06850000
/*   Notes:                                                         */ 06900000
/*                                                                  */ 06950000
/*   1. Minimum recommended value is 1.                             */ 07000000
/*   2. Maximum recommended value is 32767.                         */ 07050000
/*   3. If this parameter is not provided, the system default       */ 07100000
/*      value for this parameter is 200.                            */ 07150000
/*                                                                  */ 07200000
/********************************************************************/ 07250000
  MAXUIDS(200)                  /* Allow at most 200 z/OS UNIX         07300000
                                   users to be active concurrently  */ 07350000
                                                                       07400000
/********************************************************************/ 07450000
/*                                                                  */ 07500000
/*   Specify the maximum number of file descriptors that a single   */ 07550000
/*   user is allowed to have concurrently active or allocated.      */ 07600000
/*                                                                  */ 07650000
/*   Notes:                                                         */ 07700000
/*                                                                  */ 07750000
/*   1. This parameter is the same as the Open_Max variable         */ 07800000
/*      defined in POSIX 1003.1.                                    */ 07850000
/*   2. Minimum recommended value is 3.                             */ 07900000
/*   3. Maximum recommended value is 65535.                         */ 07950000
/*   4. If this parameter is not provided, the system default       */ 08000000
/*      value for this parameter is 2000.                           */ 08050000
/*                                                                  */ 08100000
/* Note: There is no system wide limit on total active descriptors. */ 08150000
/*                                                                  */ 08200000
/********************************************************************/ 08250000
  MAXFILEPROC(2000)                /* Allow at most 2000 open files    08300000
                                      per user                @P9C*/ 08350000
                                                                       08400000
/********************************************************************/ 08450000
/*                                                                  */ 08500000
/*   Specify the maximum number of pseudo-terminal sessions         */ 08550000
/*   that can be active concurrently.                               */ 08600000
/*                                                                  */ 08650000
/*   Notes:                                                         */ 08700000
/*                                                                  */ 08750000
/*   1. Minimum recommended value is 1.                             */ 08800000
/*   2. Maximum recommended value is 10,000.                        */ 08850000
/*   3. If this parameter is not provided, the system default       */ 08900000
/*      value for this parameter is 800.                            */ 08950000
/*                                                                  */ 09000000
/********************************************************************/ 09050000
  MAXPTYS(800)                     /* Allow up to 800 pseudo-terminal  09100000
                                      sessions                @P9C*/ 09150000
                                          /* 1@D7D - Remove MAXRTYS */ 09200000
                                                                       09250000
/********************************************************************/ 09300000
/*                                                                  */ 09350000
/*   Specify the parmlib member containing the initial tracing      */ 09400000
/*   options to be used.                                            */ 09450000
/*                                                                  */ 09500000
/*   Notes:                                                         */ 09550000
/*                                                                  */ 09600000
/*   1. If this parameter is not provided, the system default       */ 09650000
/*      value for this parameter is 'CTIBPX00'.                     */ 09700000
/*   2. Suppose CTIBPX01 is named as in this parameter, then        */ 09750000
/*      the data set SYS1.PARMLIB(CTIBPX01) should already be       */ 09800000
/*      defined to the system.                                      */ 09850000
/*                                                                  */ 09900000
/********************************************************************/ 09950000
  CTRACE(CTIBPX00)                 /* Parmlib member 'CTIBPX00' will   10000000
                                      contain the initial tracing      10050000
                                      options to be used            */

/********************************************************************/
/*                                                                  */
/*   Specify the HFS file that contains the list of data sets that  */
/*   are sanctioned for use as step libraries during the running of */
/*   set-user-ID and set-group-ID executable files.                 */
/*                                                                  */
/*   In this sample:                                                */
/*     o o The HFS file '/etc/steplib' is to contain the list of    */
/*       sanctioned data sets.                                      */
/*                                                                  */
/*   Notes:                                                         */
/*                                                                  */
/*   1. If this parameter is not provided, step libraries will not  */
/*      be set up for set-user-ID and set-group-ID executable files.*/
/*                                                                  */
/********************************************************************/
/*STEPLIBLIST('/etc/steplib') */   /* HFS file /etc/steplib will
                                      contain the list of sanctioned
                                      step libraries for set-user-ID
                                      and set-group-ID executables. */

/********************************************************************/
/*                                                                  */
/*   Specify the HFS file that contains the list of MVS userids     */
/*   that will use the specified alias name for z/OS UNIX           */
/*   functions.                                                     */
/*                                                                  */
/*   In this sample:                                                */
/*     o The HFS file '/etc/tablename' is to contain a list of      */
/*       valid MVS user IDs, each with a corresponding valid XPG4   */
/*       alias name.                                                */
/*                                                                  */
/*   Notes:                                                         */
/*                                                                  */
/*   1. If this parameter is not provided, alias names will not be  */
/*      used.                                                       */
/*                                                                  */
/********************************************************************/
/*USERIDALIASTABLE('/etc/tablename') */  /* HFS file /etc/tablename
                                      contain the list of MVS userids
                                      and their corresponding XPG4
                                      compliant alias names.        */

/********************************************************************/
/*                                                                  */
/*   The FILESYSTYPE statement defines a Physical File System (PFS).*/
/*                                                                  */
/*   In this sample:                                                */
/*     o 'HFS' is the TYPE of the Physical File System.             */
/*     o The ENTRYPOINT 'GFUAINIT' is the name of the load module   */
/*       for the DFSMS/MVS Hierarchical File System.                */
/*     o PARM is not specified.  For information about values to    */
/*       specify for this operand, either refer to 'z/OS MVS        */
/*       Initialization and Tuning Reference' or to the             */
/*       documentation for the specific PFS.                        */
/*                                                                  */
/*     o 'AUTOMNT' is the TYPE of the automount facility.           */
/*     o The ENTRYPOINT 'BPXTAMD' is the name of the load module    */
/*       for the automounter.                                      */ 13100000
/*     o This entire entry is commented out.  It is included to     */ 13150000
/*       show the syntax for using the automount facility.          */ 13200000
/*                                                                  */ 13250000
/*     o 'TFS' is the TYPE of the Temporary File System.            */ 13300000
/*       TFS file data only resides in memory, not on disk.         */ 13350000
/*     o The ENTRYPOINT 'BPXTFS' is the name of the load module     */ 13400000
/*       for the Temporary File System.                             */ 13450000
/*     o This entire entry is commented out.  It is included to     */ 13500000
/*       show the syntax for the temporary file system.             */ 13550000
/*                                                                  */ 13600000
/*     o 'NFS' is the TYPE of the Network File System Client.       */ 13650000
/*     o The ENTRYPOINT 'GFSCINIT' is the name of the load module   */ 13700000
/*       for the Client component of the z/OS Network File System.  */ 13750000
/*     o The ASNAME 'MVSNFSC' is the name of the address space for  */ 13800000
/*       the physical file system.  In this case, this is the name  */ 13850000
/*       of the address space as well as the procedure member name  */ 13900000
/*       in SYS1.PROCLIB.                                           */ 13950000
/*     o PARM is 'biod(6)'.  For valid parameters for the NFS file  */ 14000000
/*       system, refer to 'z/OS Network File System Customization   */ 14050000
/*       and Operation'.                                            */ 14100000
/*     o This entire entry is commented out.  It is included to     */ 14150000
/*       show the syntax for the network file system.               */ 14200000
/*                                                                  */ 14250000
/*                                                                  */ 14300000
/*   Notes:                                                         */ 14350000
/*                                                                  */ 14400000
/*   1. TYPE and ENTRYPOINT are required parameters.                */ 14450000
/*   2. The TYPE of the file system can be up to 8 characters       */ 14500000
/*      and must be unique among FILESYSTYPE statements.            */ 14550000
/*   3. ENTRYPOINT can be up to 8 characters.                       */ 14600000
/*   4. There can be multiple FILESYSTYPE statements.               */ 14650000
/*   5. PARM can be up to 1024 characters.                          */ 14700000
/*      It must be entered as a quoted string. It can be entered    */ 14750000
/*      in mixed case, as required by the physical file system,     */ 14800000
/*      e.g. PARM ('/u').                                           */ 14850000
/*      A null PARM can be omitted, or optionally be specified      */ 14900000
/*      as PARM(' ').                                               */ 14950000
/*      For specific information about values to specify for PARM   */ 15000000
/*      refer to either 'z/OS MVS Initialization and Tuning         */ 15040000
/*      Reference' or to the documentation for the specific PFS.    */ 15080000
/*   6. ASNAME ia a 1 to 8 character procedure name in SYS1.PROCLIB */ 15120000
/*   7. start_parms is optional and is a quoted string that is to   */ 15140000
/*      be appended to the procname when the address space is       */ 15160000
/*      started. The string may be up to 100 characters long.       */ 15180000
/*      The start_parms are not validated, they are just passed to  */ 15200000
/*      the system when the address space is started with an        */ 15220000
/*      internal start command.                                 @DCA*/ 15240000
/*   8. The specific parameters and values for the parameters       */ 15260000
/*      are file system dependent.  Refer to "UNIX System Services  */ 15280000
/*      Planning" for the file system that is to be started.        */ 15300000
/*                                                                  */ 15350000
/********************************************************************/ 15400000
  FILESYSTYPE TYPE(HFS)           /* Type of file system to start  */ 15450000
              ENTRYPOINT(GFUAINIT) /* Entry Point of load module    */ 15500000
              PARM(' ')           /* Null PARM for physical file      15550000
                                     system                         */ 15600000
                                                                      15650000
/*FILESYSTYPE TYPE(AUTOMNT)      *//* Type of file system to start  */ 15700000
/*            ENTRYPOINT(BPXTAMD)*//* Entry Point of load module    */ 15750000
```

```
/*FILESYSTYPE TYPE(TFS)          *//* Type of file system to start  */
/*            ENTRYPOINT(BPXTFS) *//* Entry Point of load module    */

/*FILESYSTYPE TYPE(NFS)          *//* Type of file system to start  */
/*            ENTRYPOINT(GFSCINIT)*//* Entry Point of load module   */
/*            ASNAME(MVSNFSC,'start_parms')*/ /*               @DCC*/
/*            PARM('biod(6)')     *//* Parameter to pass in         */

/*******************************************************************/
/*                                                                 */
/*   The ROOT statement defines and mounts the root file system.   */
/*                                                                 */
/*   In this example:                                              */
/*     o 'OMVS.ROOT' is the FILE system which is the name of an    */
/*       already defined HFS data set.                             */
/*     o 'HFS' is the TYPE of the file system.                     */
/*     o The root file system is in read/write mode.               */
/*                                                                 */
/*                                                                 */
/*   Notes:                                                        */
/*                                                                 */
/*   1. There can only be one ROOT statement.                      */
/*   2. FILESYSTEM can be up to 44 characters.                     */
/*      It must be entered as a quoted string.                     */
/*   3. DDNAME is the name of a DD statement in a UNIX System      */
/*      Services PROC. It can be up to 8 characters.               */
/*   4. FILESYSTEM and DDNAME are mutually exclusive.              */
/*      Exactly one of them must be specified.                     */
/*   5. TYPE is required and can be up to 8 characters.            */
/*      This matches the TYPE specified in a FILESYSTYPE statement. */
/*   6. PARM can be up to 1024 characters.                         */
/*      It must be entered as a quoted string. It can be entered   */
/*      in mixed case, as required by the physical file system,    */
/*      e.g. PARM ('80').                                          */
/*      For specific information about values to specify for PARM  */
/*      refer to either 'z/OS MVS Initialization and Tuning        */
/*      Reference' or to documentation for the specific PFS.       */
/*   7. MODE is either 'READ' or 'RDWR'. Default for MODE is       */
/*      'RDWR' (read/write).                                       */
/*   8. The specific parameters and values for the parameters are  */
/*      file system dependent.  Refer to 'Z/OS UNIX System Services */
/*      Planning' for the file system that owns the root file set. */
/*   9. SETUID or NOSETUID can be specified to support or ignore   */
/*      the setuid() and setgid() mode bits on an executable file. */
/*      The default is SETUID.                                     */
/*  10. If no ROOT statement is found, then the temporary file     */
/*      system will be started if it has not been specified, and   */
/*      a TFS root will be mounted.  This root is empty initially. */
/*                                                                 */
/*******************************************************************/
  ROOT     FILESYSTEM('OMVS.ROOT') /* Either FILESYSTEM or DDNAME must
                                      be specified, but not both.
                                      FILESYSTEM must be entered in
                                      quotes.                      */
           TYPE(HFS)               /* Type of File system          */
           MODE(RDWR)              /* (Optional) Can be READ or RDWR.
                                      Default = RDWR               */

/*******************************************************************/
```

```
/*                                                                   */ 18800000
/*   The MOUNT statement defines which file systems will be mounted */ 18850000
/*   at initialization and where in the file hierarchy they will be */ 18900000
/*   mounted.                                                        */ 18950000
/*   The syntax of the MOUNT statement is similar to the ROOT       */ 19000000
/*   statement, except that there is an additional keyword         */ 19050000
/*   'MOUNTPOINT'.  MOUNTPOINT is the path to the mountpoint        */ 19100000
/*   directory.                                                      */ 19150000
/*                                                                   */ 19200000
/*   In this example:                                                */ 19250000
/*     o 'OMVS.USER.JOE' is the FILESYSTEM which is the name of an  */ 19300000
/*       already defined HFS data set.                               */ 19350000
/*     o 'HFS' is the TYPE of the file system.                       */ 19400000
/*     o The file system is in read/write mode.                      */ 19450000
/*     o MOUNTPOINT is '/u/joe'.                                     */ 19500000
/*     o NOSETUID specifies to ignore setuid() and setgid() mode    */ 19550000
/*       bits on an executable file.                                 */ 19600000
/*                                                               @PAD*/ 19650000
/*     o SECURITY checks are to be enforced for files in this file  */ 19750000
/*       system.                                                     */ 19800000
/*                                                                   */ 19850000
/*                                                                   */ 19900000
/*   Notes:                                                          */ 19950000
/*                                                                   */ 20000000
/*   1. There can be multiple MOUNT statements.                      */ 20050000
/*   2. FILESYSTEM can be up to 44 characters long.                  */ 20100000
/*      It must be entered as a quoted string.                       */ 20150000
/*   3. DDNAME is the name of the DD statement in a UNIX System     */ 20200000
/*      Services PROC. It can be up to 8 characters.                 */ 20250000
/*   4. FILESYSTEM and DDNAME are mutually exclusive.                */ 20300000
/*      Exactly one of them must be specified.                       */ 20350000
/*   5. TYPE is required and can be up to 8 characters.              */ 20400000
/*      This matches the TYPE specified in a FILESYSTYPE statement. */ 20450000
/*   6. PARM is optional and can be up to 1024 characters.           */ 20500000
/*      It must be entered as a quoted string. It can be entered    */ 20550000
/*      in mixed case, as required by the physical file system,     */ 20600000
/*      e.g. PARM ('80').                                            */ 20650000
/*      For specific information about values to specify for PARM   */ 20700000
/*      refer to either 'z/OS MVS Initialization and Tuning         */ 20750000
/*      Reference' or to documentation for the specific PFS.        */ 20800000
/*   7. MODE is either 'READ' or 'RDWR'. Default for MODE is        */ 20850000
/*      'RDWR' (read/write).                                         */ 20900000
/*   8. MOUNTPOINT is required and can be up to 1023 characters.    */ 20950000
/*      It specifies the path to the mount point directory.         */ 21000000
/*      It must be entered as a quoted string. It can be entered    */ 21050000
/*      in mixed case, as required by the physical file system.     */ 21100000
/*   9. The specific parameters and values for the parameters are   */ 21150000
/*      file system dependent.  Refer to the installation guide for */ 21200000
/*      the file system that owns the file set being mounted.       */ 21250000
/*  10. SETUID or NOSETUID can be specified to support or ignore    */ 21300000
/*      the setuid and setgid mode bits on an executable file.  The */ 21350000
/*      default is SETUID.                                           */ 21400000
/*                                                              @PAD */ 21450000
/*  11. SECURITY or NOSECURITY can be specified to indicate whether */ 21500000
/*      security checks should be enforced for files in this file   */ 21650000
/*      system.                                                      */ 21700000
/*  12. TAG(TEXT|NOTEXT,ccsid)                                       */ 21750000
/*      TEXT - specifies each untagged file is implicitly marked as */ 21800000
/*             containing pure text data that can be converted.     */ 21850000
/*      NOTEXT - specifies none of the untagged files in the file   */ 21900000
/*             system are implicitly tagged.                        */ 21950000
/*      ccsid - identifies the coded character set identifier to be */ 22000000
/*             implicitly set for the untagged file. ccsid is       */ 22050000
/*             specified as a 2 byte decimal value from 0 to 65535. */ 22100000
/*             ccsid is only used when TEXT is spcified. Both the   */ 22150000
/*             Mount and Root statements can use the Tag statement. */ 22200000
/*             TAG(NOTEXT,0) is the Default.                        */ 22250000
/*  13. AUTOMOVE or NOAUTOMOVE can be specified to indicate if this */ 22258300
/*      filesystem can be moved to another server.                  */ 22266600
/*      AUTOMOVE is the default.                                    */ 22274900
/*  14. UNMOUNT this filesystem is unmounted when the system leaves */ 22283200
/*      the sysplex.                                            @DBA*/ 22291500
/*  15. AUTOMOVE(I,sy1,..,syn) to specify the target systems for    */ 22294300
/*      AUTOMOVE.  AUTOMOVE(E,sy1,..,syn) to exclude systems.   @DDA*/ 22297100
/*                                                                  */ 22300000
/********************************************************************/ 22350000
/*MOUNT FILESYSTEM('OMVS.USER.JOE')*/ /* Either FILESYSTEM or DDNAME   22400000
                                       must be specified, but not both. 22450000
                                       FILESYSTEM must be entered in    22500000
                                       quotes.                      */ 22550000
/*      TYPE(HFS)            */    /* Type of FileSystem            */ 22600000
/*      MODE(RDWR)           */    /* Can be READ or RDWR           */ 22650000
/*      MOUNTPOINT('/u/joe') */    /* Must be entered in quotes.    */ 22700000
/*      NOSETUID             */    /* ignore setuid/gid mode bits   */ 22750000
/*                                                             @PAD */ 22800000
/*      SECURITY             */    /* enforce security checks       */ 22850000
/*      TAG(NOTEXT,0)        */    /*                           @D9A*/ 22900000
                                                                       22950000
/********************************************************************/ 23000000
/*                                                                  */ 23050000
/*   The NETWORK statement defines which domain the specified       */ 23100000
/*   file system supports and some socket and port limits in that   */ 23150000
/*   domain by specifying:                                          */ 23200000
/*     o The address family domain name.                            */ 23250000
/*     o Its associated domain number.                              */ 23300000
/*     o The maximum number of sockets the address family will      */ 23350000
/*       support.                                                   */ 23400000
/*     o The ports to be reserved for use with port zero,           */ 23450000
/*       INADDR_ANY binds. This is for Common INET only.            */ 23500000
/*   There must be a previous FILESYSTYPE statement that has a TYPE */ 23550000
/*   operand that matches the TYPE operand on the NETWORK           */ 23600000
/*   statement.                                                     */ 23650000
/*                                                                  */ 23700000
/*   Currently, only three domains are supported:                   */ 23750000
/*    AF_UNIX, domain number 1, and entry point (BPXTUINT)          */ 23800000
/*    AF_INET, domain number 2, and, for example:                   */ 23850000
/*             - entry point (EZBPFINI) for CS390 TCP/IP            */ 23900000
/*          OR - entry point (ISTOEPIT) for CS390 AnyNet (SNA)      */ 23950000
/*          OR - entry point (BPXTCINT) for Common Inet.            */ 24000000
/*    AF_INET6, domain number 19, and, for example:             @PCA*/ 24030000
/*             - entry point (EZBPFINI) for CS390 TCP/IP            */ 24060000
/*          OR - entry point (BPXTCINT) for Common Inet.            */ 24090000
/*                                                                  */ 24120000
/*      Common Inet Sockets is intended to be used only if          */ 24150000
/*      multiple network socket file systems (such as 2             */ 24180000
/*      TCP/IP's) are to be active at one time.  There is a        */ 24210000
/*      performance degradation with using Common Inet Sockets      */ 24240000
/*      with just a single sockets physical file system.)           */ 24270000
/*                                                                  */ 24300000
/*   Port reservation information for port zero, INADDR_ANY binds   */ 24350000
/*   is only required for the AF_INET NETWORK in a Common INET      */ 24400000
/*   configuration. It is specified with the INADDRANYPORT and      */ 24450000
/*   INADDRANYCOUNT parameters. If both of these parameters are     */ 24500000
/*   omitted, then default port values are reserved.                */ 24550000
/*   The port range reserved for AF_INET is also used by AF_INET6   */ 24566600
/*   for IN6ADDR_ANY binds with port 0.                             */ 24583200
/*                                                                  */ 24600000
/*   INADDRANYPORT specifies the starting port number to be         */ 24650000
/*   reserved for use by applications that issue portzero,          */ 24700000
/*   INADDR_ANY binds. INADDRANYCOUNT specifies how many ports      */ 24750000
/*   to reserve.                                                    */ 24800000
/*                                                                  */ 24850000
/*   If you are running a Common INET configuration and you         */ 24900000
/*   specify the INADDRANYPORT and INADDRANYCOUNT parameters then   */ 24950000
/*   you must specify the same values to each transport provider    */ 25000000
/*   that is specified with the SUBFILESYSTYPE statement. Refer     */ 25050000
/*   to the documentation for that transport provider to determine  */ 25100000
/*   how the port reservation information is specified.             */ 25150000
/*                                                                  */ 25200000
/*   In this example --                                             */ 25250000
/*                                                                  */ 25300000
/*   For TYPE(UDS):                                                 */ 25350000
/*     o ENTRYPOINT is BPXTUINT,                                    */ 25400000
/*     o DOMAINNAME is 'AF_UNIX'.                                   */ 25450000
/*     o DOMAINNUMBER is 1.                                         */ 25500000
/*     o MAXSOCKETS is 200.                                         */ 25550000
/*     o The TYPE of the file system is 'UDS'.                      */ 25600000
/*     o No port reservations are required for AF_UNIX.             */ 25650000
/*                                                                  */ 25700000
/*   For TYPE(INET):                                                */ 25750000
/*     o ENTRYPOINT is EZBPFINI,                                    */ 25800000
/*     o DOMAINNAME is 'AF_INET'.                                   */ 25850000
/*     o DOMAINNUMBER is 2.                                         */ 25900000
/*     o MAXSOCKETS is 64000.                                       */ 25950000
/*     o The TYPE of the file system is 'INET'                      */ 26000000
/*     o No port reservations are required.                         */ 26050000
/*     o Optionally, IPv6 is activated too.                         */ 26075000
/*                                                                  */ 26100000
/*   Notes:                                                         */ 26150000
/*                                                                  */ 26200000
/*   1. The name specified for DOMAINNAME is the name that will     */ 26250000
/*      appear in messages referring to this address family.  The  */ 26300000
/*      name specified can be any name up to 16 bytes in length.    */ 26350000
/*   2. The value specified for DOMAINNUMBER is the numerical       */ 26400000
/*      representation of the address family.  For a list of valid */ 26450000
/*      values for this operand, refer to BPXYSOCK.                 */ 26500000
/*   3. MAXSOCKETS refers to the maximum number of sockets that can */ 26550000
/*      be active at one time.  This value will depend on how many  */ 26600000
/*      socket programs will be run and, for servers, how many      */ 26650000
/*      remote clients will connect to the local servers.           */ 26700000
/*   4. The name specified for the TYPE operand must match the name */ 26750000
/*      of a previous FILESYSTYPE statement.                        */ 26800000
/*                                                                  */ 26850000
/*                                                                  */ 26900000
/*******************************************************************/ 26950000
  FILESYSTYPE TYPE(UDS) ENTRYPOINT(BPXTUINT)                           27000000
  NETWORK DOMAINNAME(AF_UNIX)                                          27050000
          DOMAINNUMBER(1)                                              27100000
          MAXSOCKETS(200)                                              27150000
          TYPE(UDS)                                                    27200000

  FILESYSTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)
  NETWORK DOMAINNAME(AF_INET)
          DOMAINNUMBER(2)
          MAXSOCKETS(64000)
          TYPE(INET)

/* NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) */   /* For IPv6   */
/*         TYPE(INET) */

/********************************************************************/
/*                                                                  */
/*   The SUBFILESYSTYPE statement specifies a socket file system    */
/*   that the z/OS UNIX Common Inet Sockets physical file system    */
/*   is to start.  This is used if more than one AF_INET/AF_INET6   */
/*   physical file system (TCP/IP) is active at the same time.      */
/*                                                                  */
/*   The SUBFILESYSTYPE statement is associated with the            */
/*   FILESYSTYPE statement that describes the z/OS UNIX Common      */
/*   Inet Sockets physical file system by matching the value        */
/*   specified in the TYPE operand.                                 */
/*                                                                  */
/*   In this sample:                                                */
/*     o 'CINET' is the TYPE specified for this file system         */
/*     o The ENTRYPOINT 'BPXTCINT' is the name of the load module   */
/*       of the Common Inet Sockets PFS.                            */
/*     o The ENTRYPOINT 'EZBPFINI' is the name of the load module   */
/*       of the CS/390 TCP/IP Sockets PFS.  In this example we      */
/*       start two separate TCP/IP stacks.                          */
/*     o NAME is the name that the transport provider (such as      */
/*       TCP/IP) will provide when it initializes to identify       */
/*       itself.                                                    */
/*     o PARM is not specified.                                     */
/*                                                                  */
/*   Notes:                                                         */
/*                                                                  */
/*   1. NAME, TYPE and ENTRYPOINT are required parameters.          */
/*   2. NAME can be up to 8 characters.  It specifies the           */
/*      name by which this file system will be known to the Common  */
/*      Inet Sockets physical file system.  In the case of          */
/*      TCP/IP, this is the procname or TCPIPJOBNAME.               */
/*   3. TYPE can be up to 8 characters.  It specifies the name of   */
/*      the Common Inet Sockets physical file system type           */
/*      identified in a FILESYSTYPE statement TYPE parameter.       */
/*   4. ENTRYPOINT can be up to 8 characters.  It specifies the     */
/*      name of the load module containing the entry point into     */
/*      the file system type.                                       */
/*   5. There can be up to 32 SUBFILESYSTYPE statements.            */
/*   6. PARM can be up to 1024 characters.                          */
/*      It must be entered as a quoted string. It can be entered    */
/*      in mixed case, as required by the physical file system,     */
/*      e.g. PARM ('/u').                                           */
/*      The value specified here is the same as you would specify   */
/*      on a FILESYSTYPE statement, and is dependent on the physical*/
/*      file system.                                                */
/*      A null PARM can be omitted, or optionally be specified      */
/*      as PARM(' ').                                               */
/*   7. DEFAULT has no parameters.                                  */
/*      The sockets physical file system designated as the DEFAULT  */
/*      will be used by Common Inet Sockets to set default routes   */
/*      and to supply the local hostname and hostid.                */ 30250000
/*   8. The specific parameters and values for the parameters are   */ 30300000
/*      file system dependent.  Refer to the installation guide for */ 30350000
/*      file system that is to be started.                          */ 30400000
/*                                                                  */ 30450000
/********************************************************************/ 30500000
/*FILESYSTYPE TYPE(CINET) ENTRYPOINT(BPXTCINT) */                      30550000
/*NETWORK DOMAINNAME(AF_INET)                  */                      30600000
/*        DOMAINNUMBER(2)                      */                      30650000
/*        MAXSOCKETS(64000)                    */                      30700000
/*        TYPE(CINET)                          */                      30750000
/*        INADDRANYPORT(2000)                  */                      30800000
/*        INADDRANYCOUNT(325)                  */                      30850000
/*NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) */   /* For IPv6   */  30900000
/*        TYPE(CINET)                           */                     30950000
                                                                       31150000
/*SUBFILESYSTYPE NAME(TCPIP)     */  /* Name of file system         */ 31200000
/*               TYPE(CINET)     */  /* Type matching Cinet's TYPE  */ 31250000
/*           ENTRYPOINT(EZBPFINI)*/  /* Entry point of load module  */ 31300000
/*               DEFAULT         */  /* <- The Default Socket PFS   */ 31350000
                                                                       31400000
/*SUBFILESYSTYPE NAME(TCPIP2)    */  /* Name of file system         */ 31450000
/*               TYPE(CINET)     */  /* Type matching Cinet's TYPE  */ 31500000
/*           ENTRYPOINT(EZBPFINI)*/  /* Entry point of load module  */ 31550000
                                                                       31600000
/********************************************************************/ 31650000
/*                                                                  */ 31700000
/*   Specify the maximum number of thread tasks that z/OS UNIX      */ 31750000
/*   will allow to be active concurrently in a single process.      */ 31800000
/*                                                                  */ 31850000
/*   Notes:                                                         */ 31900000
/*                                                                  */ 31950000
/*   1. Minimum allowable value is 0.                               */ 32000000
/*   2. Maximum allowable value is 32768.                           */ 32050000
/*   3. If this parameter is not provided, the system default       */ 32100000
/*      value is 1000.                                              */ 32150000
/*                                                                  */ 32200000
/********************************************************************/ 32250000
  MAXTHREADTASKS(1000)             /* System will allow at most 1000   32300000
                                      threads tasks to be active       32350000
                                      concurrently in a single process 32400000
                                                                @P9C*/ 32450000
                                                                       32500000
/********************************************************************/ 32550000
/*                                                                  */ 32600000
/*   MAXTHREADS is the maximum number of threads that               */ 32650000
/*   z/OS UNIX will allow to be active concurrently in a            */ 32700000
/*   single process.                                                */ 32750000
/*                                                                  */ 32800000
/*   Notes:                                                         */ 32850000
/*                                                                  */ 32900000
/*   1. Minimum allowable value is 0.                               */ 32950000
/*   2. Maximum allowable value is 100,000                          */ 33000000
/*   3. If this parameter is not provided, the system default       */ 33050000
/*      value is 200.                                               */ 33100000
/*                                                                  */ 33150000
/********************************************************************/ 33200000
  MAXTHREADS(200)                  /* System will allow at most 200    33250000
                                      threads to be active             33300000
                                      concurrently in a single process 33350000
```

```
                                                                     */

/*******************************************************************/
/*                                                                  */
/*   PRIORITYPG  allows the user to specify 1 to 40 performance     */
/*               group numbers.                                     */
/*   PRIORITYGOAL allows the user to specify 1 to 40 service        */
/*               classes.                                           */
/*                                                                  */
/*   These values result in an array that is accessed via an index */
/*   value.                                                         */
/*                                                                  */
/*   Keyword          Value Range     Number of Values              */
/*   -------          -----------     ----------------              */
/*   prioritypg         1 - 999             40                      */
/*                                                                  */
/*   prioritygoal     1-8 characters       40                       */
/*                                                                  */
/*   Notes:                                                         */
/*                                                                  */
/*   1. Generally, it is not recommended that these values be       */
/*      set.  They are required however if the nice, setpriority or */
/*      chpriority system services has been enabled.                */
/*                                                                  */
/*   2. All performance groups specified on the PRIORITYPG          */
/*      statement must also be specified in the IEAIPSxx member     */
/*      of parmlib.                                                 */
/*                                                                  */
/*   3. All service classes specified on the PRIORITYGOAL           */
/*      statement must also be specified in your WLM Service        */
/*      Policy.                                                     */
/*                                                                  */
/*   4. If fewer than 40 values are specified, the last value       */
/*      specified is propogated to the end of the array.            */
/*                                                                  */
/*   5. The default value is NONE. For example PRIORITYPG(NONE)     */
/*      or PRIORITYGOAL(NONE) means there are no values. If the     */
/*      PRIORITYPG or PRIORITYGOAL are not specified then this      */
/*      also means that there are no values.                    @P1C*/
/*                                                                  */
/*******************************************************************/
/*PRIORITYPG (n,...,n)*/   /* Do not use this value unless the nice,
                              setpriority or chpriority services
                              has been enabled                      */

/*PRIORITYGOAL (n,...,n)*/ /* Do not use this value unless the nice,
                              setpriority or chpriority services
                              has been enabled                      */

/*******************************************************************/
/*                                                                  */
/*   XPG4 Interprocess Communications: the following keywords and   */
/*   associated values allow the user to define the IPC values.     */
/*                                                                  */
/*   Keyword       Value Range  Default   Description               */
/*   -------       -----------  -------   -----------------------   */
/*   IPCMSGNIDS    1 -   20000      500   Maximum number of unique  */
/*                                        message queues, systemwide*/
/*   IPCMSGQBYTES  0 -      2147483647   Maximum number of bytes   */
/*                                        in a single message       */
/*                                        queue.                @P9C*/
/*   IPCMSGQMNUM   0 -           10000   Maximum number of messages*/
/*                                        per queue, systemwide.@D3C*/
/*   IPCSHMNIDS    1 -   20000      500   Maximum number of unique  */
/*                                        shared memory segments,   */
/*                                        systemwide.               */
/*   IPCSHMSPAGES  0 - 2621440   262144   Maximum number of pages   */
/*                                        for shared memory segments*/
/*                                        systemwide.               */
/*   IPCSHMMPAGES  1 -  524287    25600   Maximum number of pages   */
/*                                        for a shared memory       */
/*                                        segment.              @P9C*/
/*   IPCSHMNSEGS   0 -    1000      500   Maximum number of shared  */
/*                                        memory segments attached  */
/*                                        per address space.    @P9C*/
/*   IPCSEMNIDS    1 -   20000      500   Maximum number of unique  */
/*                                        semaphore sets, systemwide*/
/*   IPCSEMNSEMS   0 -   32767     1000   Maximum number of         */
/*                                        semaphores per semaphor   */
/*                                        set.                  @P9C*/
/*   IPCSEMNOPS    0 -   32767       25   Maximum number of         */
/*                                        operations per semop call.*/
/*                                                                  */
/*   Notes:  None                                                   */
/*                                                                  */
/********************************************************************/
  IPCMSGNIDS    (500)
  IPCMSGQBYTES  (2147483647)                                /* @P9C*/
  IPCMSGQMNUM   (10000)
  IPCSHMNIDS    (500)
  IPCSHMSPAGES  (262144)
  IPCSHMMPAGES  (25600)                                     /* @P9C*/
  IPCSHMNSEGS   (500)                                       /* @P9C*/
  IPCSEMNIDS    (500)
  IPCSEMNSEMS   (1000)                                      /* @P9C*/
  IPCSEMNOPS    (25)

/********************************************************************/
/*                                                                  */
/*   MaxMMapArea is the maximum amount of data space storage (in    */
/*   pages) that can be allocated for memory mappings of            */
/*   HFS files. Storage is not allocated until memory mapping is    */
/*   active.                                                        */
/*                                                                  */
/*   Notes:                                                         */
/*                                                                  */
/*   1. Minimum allowable value is 1 (page).                        */
/*   2. Maximum allowable value is 16777216.                        */
/*   3. If this parameter is not provided, the system default       */
/*      value is 40960.                                             */
/*                                                                  */
/********************************************************************/
  MAXMMAPAREA(40960)              /* System will allow at most 40960
                                     pages to be used for memory
                                     mapping.
                                                                    */

/********************************************************************/
/*                                                                  */
/*   The MAXFILESIZE statement specifies the RLIMIT_FSIZE soft and  */
/*   hard limit resource values that processes receive when they    */ 39400000
/*   are dubbed a process.  Also when they are initiated by a       */ 39450000
/*   daemon process using an exec after setuid().                   */ 39500000
/*                                                                  */ 39550000
/*   RLIMIT_FSIZE indicates the maximum file size (in 4 KB          */ 39600000
/*   increments) that a process can create.                         */ 39650000
/*                                                                  */ 39700000
/*   Refer to the definition of setrlimit() in 'z/OS UNIX System    */ 39750000
/*   Services Programming: Assembler Callable Services Reference'   */ 39800000
/*   for more information about RLIMIT_FSIZE.                       */ 39850000
/*                                                                  */ 39900000
/*     Value Range     Default                                      */ 39950000
/*     --------------  ---------                                    */ 40000000
/*     0 - 2147483647  NOLIMIT                                  @P0C*/ 40050000
/*                                                                  */ 40100000
/*   Notes:                                                         */ 40150000
/*                                                                  */ 40200000
/*   1. OMITTING this statement indicates an unlimited file size.   */ 40250000
/*      NOLIMIT also indicates an unlimited file size.          @P0A*/ 40300000
/*                                                                  */ 40350000
/********************************************************************/ 40400000
MAXFILESIZE(NOLIMIT)         /* unlimited file size                 */ 40450000
                                                                       40500000
/********************************************************************/ 40550000
/*                                                                  */ 40600000
/*   The MAXCORESIZE statement specifies the RLIMIT_CORE soft and   */ 40650000
/*   hard limit resource values that processes receive when they    */ 40700000
/*   are dubbed a process, and when they are initiated by a daemon  */ 40750000
/*   process using an exec after setuid().                          */ 40800000
/*                                                                  */ 40850000
/*   RLIMIT_CORE indicates the maximum core dump file size (in      */ 40900000
/*   bytes) that a process can create.                              */ 40950000
/*                                                                  */ 41000000
/*   Refer to the definition of setrlimit() in 'z/OS UNIX System    */ 41050000
/*   Services Programming: Assembler Callable Services Reference'   */ 41100000
/*   for more information about RLIMIT_CORE.                        */ 41150000
/*                                                                  */ 41200000
/*     Value Range     Default                                      */ 41250000
/*     --------------  -------                                      */ 41300000
/*     0 - 2147483647  4194304 (4 MB)                               */ 41350000
/*                                                                  */ 41400000
/*   Notes:                                                         */ 41450000
/*                                                                  */ 41500000
/*   1. Specifying a value of 2147483647 indicates unlimited core   */ 41550000
/*      file size.                                                  */ 41600000
/*                                                                  */ 41650000
/********************************************************************/ 41700000
  MAXCORESIZE(4194304)                                                 41750000
                                                                       41800000
/********************************************************************/ 41850000
/*                                                                  */ 41900000
/*   The MAXASSIZE statement specifies the RLIMIT_AS hard limit     */ 41950000
/*   resource value that processes receive when they are dubbed a   */ 42000000
/*   process.  The soft limit is obtained from MVS.  If the soft    */ 42050000
/*   limit value from MVS is greater than the MAXASSIZE value, the  */ 42100000
/*   hard limit is set to the soft limit.                           */ 42150000
/*                                                                  */ 42200000
/*   This value is also used when processes are initiated by a      */ 42250000
/*   a daemon process using an exec after setuid().  In this case,  */ 42300000
/*   both the RLIMIT_AS hard and soft limit values are set to the   */ 42350000
/*   MAXASSIZE specified value.                                    */
/*                                                                 */
/*   RLIMIT_AS indicates the address space region size.            */
/*                                                                 */
/*   Refer to the definition of setrlimit() in 'z/OS UNIX System    */
/*   Services Programming: Assembler Callable Services Reference'   */
/*   for more information about RLIMIT_AS.                         */
/*                                                                 */
/*     Value Range            Default                              */
/*     ---------------------  --------                             */
/*     10485760 - 2147483647  209715200 (200 MB)                   */
/*                                                                 */
/*******************************************************************/
  MAXASSIZE(209715200)                                    /* @P9C*/

/*******************************************************************/
/*                                                                 */
/*   The MAXCPUTIME statement specifies the RLIMIT_CPU hard limit   */
/*   resource value processes receive when they are dubbed a        */
/*   process.  The soft limit is obtained from MVS.  If the soft    */
/*   limit value from MVS is greater than the MAXCPUTIME value, the */
/*   hard limit is set to the soft limit.                          */
/*                                                                 */
/*   This value is also used when processes are initiated by a      */
/*   a daemon process using an exec after setuid().  In this case,  */
/*   both the RLIMIT_AS hard and soft limit values are set to the   */
/*   MAXASSIZE specified value.                                    */
/*                                                                 */
/*   RLIMIT_CPU indicates the CPU time, in seconds, that a process  */
/*   can use.                                                      */
/*                                                                 */
/*   Refer to the definition of setrlimit() in 'z/OS UNIX System    */
/*   Services Programming: Assembler Callable Services Reference'   */
/*   for more information about RLIMIT_CPU.                        */
/*                                                                 */
/*     Value Range     Default                                     */
/*     --------------  -------                                     */
/*     7 - 2147483647  1000                                        */
/*                                                                 */
/*   Notes:                                                        */
/*                                                                 */
/*   1. Specifying a value of 2147483647 indicates unlimited CPU    */
/*      time.                                                      */
/*                                                                 */
/*******************************************************************/
  MAXCPUTIME(1000)

/*******************************************************************/
/*                                                                 */
/*   MAXSHAREPAGES is the maximum number of system shared storage   */
/*   pages that can concurrently be active using the fork(), ptrace,*/
/*   shmat, and mmap services.  The fork service uses shared storage*/
/*   only when the parmlib statement FORKCOPY(COW) is specified.    */
/*   The other services use shared storage as part of their normal  */
/*   operation.  Since the fork() and ptrace services use shared    */
/*   storage for performance improvements, these services continue  */
/*   to function when the shared storage limit is reached by no     */
/*   longer using shared storage to perform their function.  The    */
/*   shmat and mmap services, however, no longer function when the  */
/*   shared storage limit has been reached, because these functions */
/*   require shared storage to operate successfully.                */ 45400000
/*                                                                  */ 45450000
/*   By controlling the number of shared storage pages in use, an   */ 45500000
/*   installation can control the amount of System Queue Area (SQA) */ 45550000
/*   storage consumed to support these pages.                       */ 45600000
/*   Approximately 48 bytes of SQA storage are consumed to support  */ 45650000
/*   each page of shared storage.                                   */ 45700000
/*                                                                  */ 45750000
/*                                                                  */ 45800000
/*   Notes:                                                         */ 45850000
/*                                                                  */ 45900000
/*   1. Minimum allowable value is 0 (page).                        */ 45950000
/*   2. Maximum allowable value is 32768000.                        */ 46000000
/*   3. If this parameter is not provided, the system default       */ 46050000
/*      value is 131072.                                            */ 46100000
/*                                                                  */ 46150000
/********************************************************************/ 46200000
  MAXSHAREPAGES(131072)            /* System will allow at most 131072 46250000
                                      pages of shared storage to be    46300000
                                      concurrently in use           */ 46350000
                                                                       46400000
/********************************************************************/ 46450000
/*                                                                  */ 46500000
/*   FORKCOPY specifies how user storage is to be copied from       */ 46550000
/*   the parent process to the child process during a fork()        */ 46600000
/*   system call.                                                   */ 46650000
/*                                                                  */ 46700000
/*   If FORKCOPY(COW) is specified, all fork() calls are processed  */ 46750000
/*   with the copy-on-write mode if the suppression-on-protection   */ 46800000
/*   hardware feature is available.  Before the storage is modified,*/ 46850000
/*   both the parent process and child process refer to the same    */ 46900000
/*   view of the data.  The parent storage is copied to the child   */ 46950000
/*   as soon as the storage is modified by either the parent or     */ 47000000
/*   the child.  Use of copy-on-write causes the system to used the */ 47050000
/*   system queue area (SQA) to manage page sharing.                */ 47100000
/*                                                                  */ 47150000
/*   If FORKCOPY(COPY) is specified, fork() immediately copies      */ 47200000
/*   the parent storage to the child, regardless of whether the     */ 47250000
/*   suppression-on-protection feature is available.                */ 47300000
/*                                                                  */ 47350000
/*   Notes:                                                         */ 47400000
/*                                                                  */ 47450000
/*   1. If the FORKCOPY statement is not provided, the default is   */ 47500000
/*      the same as if FORKCOPY(COW) had been specified.            */ 47550000
/*                                                                  */ 47600000
/********************************************************************/ 47650000
  FORKCOPY(COW)                    /* System will use copy-on-write    47700000
                                      for fork system calls if the     47750000
                                      suppression-on-protection        47800000
                                      hardware feature is available    47850000
                                                                    */ 47900000
                                                                       47950000
/********************************************************************/ 48000000
/*                                                                  */ 48050000
/*   SYSPLEX specifies if this system should join the               */ 48100000
/*   S390 Unix System Services kernel group to share resources      */ 48150000
/*   across the sysplex.                                            */ 48200000
/*                                                                  */ 48250000
/*   If SYSPLEX(YES) is specified, the S390 Unix System Services    */ 48300000
/*   will join the sysplex group.                                   */ 48350000
/*                                                                  */
/*   If SYSPLEX(NO) is specified, the S390 Unix System Services     */
/*   will operate in local mode.                                    */
/*                                                                  */
/*                                                                  */
/*   Notes:                                                         */
/*                                                                  */
/*   1. If the SYSPLEX statement is not provided, the default is    */
/*      the same as if SYSPLEX(NO) had been specified.              */
/*                                                                  */
/********************************************************************/
  SYSPLEX(NO)                       /* This system will not join the
                                       Unix System Services kernel
                                       sysplex group to share
                                       resources across the sysplex;
                                       it will operate in local mode.*/

/********************************************************************/
/*                                                                  */
/*   SUPERUSER is a 1 to 8-character name that must conform to the  */
/*   restrictions for an MVS user ID.  This user ID is assigned     */
/*   to shell users when they enter the su command.  This user ID   */
/*   should be defined to the security product and have a UID of 0 */
/*   assigned to it.                                                */
/*   The default is SUPERUSER(BPXROOT).                             */
/*                                                                  */
/********************************************************************/
  SUPERUSER(BPXROOT)

/********************************************************************/
/*                                                                  */
/*   TTYGROUP is a 1 to 8-character name that must conform to the   */
/*   restrictions for an MVS group name.  Slave pseudoterminals     */
/*   (ptys) and OCS rtys are given this group name when they are    */
/*   first opened.  The name is used by certain setgid programs,    */
/*   such as talk and write, when attempting to write to another    */
/*   user's pty or rty.  This group name should be defined to the   */
/*   security product and have a unique GID.  No users should be    */
/*   connected to this group.                                       */
/*                                                                  */
/*   The default is TTYGROUP(TTY).                                  */
/*                                                                  */
/********************************************************************/
  TTYGROUP(TTY)

/********************************************************************/
/*                                                                  */
/*   A 1-8 character name of started procedure JCL initializing     */
/*   the z/OS UNIX kernel.                                          */
/*   Default: OMVS                                                  */
/*                                                                  */
/********************************************************************/
  STARTUP_PROC(OMVS)

/********************************************************************/
/*                                                                  */
/*   STARTUP_EXEC is the name of the REXX exec that performs        */
/*   application environment initialization for z/OS UNIX.          */
/*   There is no default.                                           */
/*                                                                  */
/*   'Dsname(Memname)' must be a quoted string. Memname is a REXX   */
/*      exec member in the PDS Dsname.                               */
/*      Dsname is a 1-44 character valid dataset name.               */
/*      Memname is a 1-8 character valid member name.                */
/*   SysoutClass is 1 character and alphanumeric. It is the sysout  */
/*      class that the REXX exec will be running under. SysoutClass */
/*      is optional.                                                 */
/*                                                                   */
/*******************************************************************/
/* STARTUP_EXEC('Dsname(Memname)',SysoutClass) */

/*******************************************************************/
/*                                                                   */
/*   RUNOPTS is a List of LE runtime options. This is 1-250         */
/*   characters long and must be enclosed in single quotes.         */
/*                                                               @D0A*/
/*******************************************************************/
/* RUNOPTS('runtime options') */

/*******************************************************************/
/*                                                                   */
/*   SYSCALL_COUNT(YES|NO) YES -the syscall layer will accumulate   */
/*                              the count of the number of          */
/*                              syscalls. This is for accounting    */
/*                              purposes. The RMF data gatherer     */
/*                              collects this information.          */
/*                         NO  -the syscall layer will NOT          */
/*                              accumulate the count of the number  */
/*                              of syscalls. The RMF data gatherer  */
/*                              will NOT collect this information.  */
/*                         DEFAULT is NO.                        @D1A*/
/*******************************************************************/
   SYSCALL_COUNTS(NO)
                                                         /* @D2A @P3D */
/*******************************************************************/
/*                                                                   */
/*   Specify the maximum number of signals that z/OS UNIX will      */
/*   allow to be concurrently queued within a single process.       */
/*                                                                   */
/*   Notes:                                                          */
/*                                                                   */
/*   1. Minimum allowable value is 1.                                */
/*   2. Maximum allowable value is 100,000.                          */
/*   3. If this parameter is not provided, the system default       */
/*      value for this parameter is 1000.                            */
/*                                                                   */
/*******************************************************************/
  MAXQUEUEDSIGS(1000)              /* Allow up to 1000 queued signals
                                      in a single process       @P7A*/

/*******************************************************************/
/*                                                                   */
/*   SHRLIBRGNSIZE(shrlibrgnsize) Limits the size of the system     */
/*                         shared library region. This is where the */
/*                         system library modules are loaded.       */
/*                         Default    67108864  (64MB)              */
/*                         Minimum    16777216  (16MB)              */
/*                         Maximum  1610612736  (1.5G)          @D5A*/
/*******************************************************************/
   SHRLIBRGNSIZE(67108864)
```

```
/********************************************************************/
/*                                                                  */
/*   SHRLIBMAXPAGES(shrlibmaxpages) Amount of data space storage    */
/*                        pages that can be allocated for non-      */
/*                        system shared library modules.            */
/*                        Default     4096  (4K)                    */
/*                        Minimum        1  (1 page)                */
/*                        Maximum 16777216  (16M)               @D6A*/
/********************************************************************/
   SHRLIBMAXPAGES(4096)

/********************************************************************/
/*                                                                  */
/*   LIMMSG(NONE|SYSTEM|ALL)                                        */
/*        Specify which level of Warning Messages the system will   */
/*        write to the system console whenever the actual usage     */
/*        of the limits in here reached 85%/90%/95% or 100% of      */
/*        it current limit value.                                   */
/*                                                                  */
/*        NONE   - No messages will be sent to the concole.         */
/*                                                                  */
/*        SYSTEM - Warning Messages for system-wide defined values  */
/*                 written to the console.                          */
/*                 System-wide values are:                          */
/*                 MAXPROCSYS      MAXUIDS        MAXPTYS           */
/*                 MAXMMAPAREA     MAXSHAREPAGES  IPCSMSGNIDS       */
/*                 IPCSEMNIDS      IPCSHMNIDS     IPCSHMSPAGES      */
/*                 SHRLIBRGNSIZE   SHRLIBMAXPAGES IPCMSGQBYTES      */
/*                 IPCMSGQMNUM     IPCSHMMPAGES                     */
/*                 In addition messages for Process-wide limits     */
/*                 are displayed, whenever the process has its      */
/*                 own User Profile, or the limit for a process     */
/*                 has been changed by a previous SETOMVS command.  */
/*                                                                  */
/*        ALL    - In addition to the System-wide values above,     */
/*                 messages on process level for all processes      */
/*                 are displayed, whenever one process reaches      */
/*                 85%/90%/95%/ or 100% of its actual limit value   */
/*                 (which can be different to those of another      */
/*                 process).                                        */
/*                 Process-wide values are:                         */
/*                 MAXFILEPROC     MAXPROCUSER    MAXQUEUEDSIGS     */
/*                 MAXTHREADS      MAXTHREADTASKS IPCSHMNSEGS       */
/*                                                                  */
/*                                                                  */
/*                                                                  */
/*   Default is: NONE                                               */
/*                                                              @D8A*/
/********************************************************************/
   LIMMSG(NONE)
/********************************************************************/
/*                                                                  */
/*   AUTOCVT(ON|OFF)                                                */
/*        ON     - Activates Automatic Conversion of I/O data using */
/*                 coded character sets for the program and its     */
/*                 associated files.                                */
/*        OFF    - AutoCVT not activated. This is the Default.      */
/*                                                              @D9A*/
/********************************************************************/
   AUTOCVT(OFF)
```

```
/********************************************************************/ 57400000
/*                                                                  */ 57450000
/* RESOLVER_PROC is used to specify how the resolver address space  */ 57500000
/* is processed during Unix System Services initialization.         */ 57550000
/* The resolver address space is used by Tcp/Ip applications        */ 57600000
/* for name-to-address or address-to-name resolution.               */ 57650000
/* In order to create a resolver address space, a system must be    */ 57700000
/* configured with an AF_INET or AF_INET6 domain.                   */ 57750000
/*                                                                  */ 57800000
/*  RESOLVER_PROC(procname|DEFAULT|NONE)                            */ 57850000
/*                                                                  */ 57900000
/*     procname - The name of the address space for the resolver.   */ 57950000
/*                In this case, this is the name of the address     */ 58000000
/*                space as well as the procedure member name        */ 58050000
/*                in SYS1.PROCLIB. procname is 1 to 8 characters    */ 58100000
/*                long.                                             */ 58150000
/*                                                                  */ 58200000
/*     DEFAULT -  An address space with the name RESOLVER will      */ 58250000
/*                be started. This is the same result that will     */ 58300000
/*                occur if the RESOLVER_PROC statement is not       */ 58350000
/*                specified in the BPXPRMxx profile.                */ 58400000
/*                                                                  */ 58450000
/*     NONE    -  Specifies that a RESOLVER address space is        */ 58500000
/*                not to be started.                                */ 58550000
/*                                                              @DAA*/ 58600000
/********************************************************************/ 58650000
RESOLVER_PROC(DEFAULT)                                                 58700000
```

/samples/inetd.conf

Example B-5 shows /samples/inetd.conf (element FOMRCONF) at the RMID=HOT7707 level.

Example: B-5 /samples/inetd.conf

```
###
# Internet server configuration database
#
# (C) COPYRIGHT International Business Machines Corp. 1985, 2001
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# /etc/inetd.conf
#
#          Internet server configuration database
#
#  $01=PYQ0049,  HOT7705, 010130, PDJP: Correct paths and remove
#       unsupported services (FIN APAR OW45915
#
# Services can be added and deleted by deleting or inserting a
# comment character (ie. #) at the beginning of a line
#
#======================================================================
# service | socket | protocol | wait/ | user | server  | server program
# name    | type   |          | nowait|      | program |   arguments
```

```
#=====================================================================
#
#otelnet  stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l
#shell    stream tcp nowait OMVSKERN /usr/sbin/orshd orshd -LV
login     stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
#exec     stream tcp nowait OMVSKERN /usr/sbin/orexecd orexecd -LV
```

/samples/init.options

Example B-6 shows /samples/init.options (element FSUMIOPT) at the RMID=HOT7707 level.

Example: B-6 /samples/init.options

```
/*   /etc/init options file, pathname = /etc/init.options
*
*      -a nnnn        =  maximum time to allow for shell script
*                        to complete (seconds):
*                           o default =  180
*                           o minimum =   10
*                           o maximum = 9999
*
*      -t n           =  terminate shell process group after
*                        timeout:
*                           o 0       =   no
*                           o >0      =  yes
*                           o default =  1 (yes)
*
*      -e string      =  environment variable string of form
*                        name=value:
*                          o maximum length = 255 characters
*                          o \ interpreted at continuation
*                            character
*
*      -sc pathname   =  pathname of init shell script:
*                          o default = /etc/rc
*                          o maximum length = 255 characters
*                          o \ interpreted as continuation
*                          o if -s <name> specified and <name>
*                            is blank or not found in /etc
*                            directory, /etc/init exits
*                            without invoking shell.
*
*      -sh pathname   =  pathname of init shell:
*                          o default = /bin/sh
*                          o maximum length = 255 characters
*                          o -sh pppp\ interpreted as continuation,
*                            where pppp = pathname characters
*                          o -sh <blanks>\ interpreted as
*                            "do not exec shell" option.
*                          o if -s <name> specified and <name>
*                            is blank or not found in /etc
*                            directory, /etc/init exits
*                            without invoking shell.
*
*    One option per line
*    First character of option line must be hyphen (-).
*
*/
```

```
-a  9999                           timeout = 9999 seconds
-t  1                              terminate shell = yes
-sc /etc/rc                        shell script = /etc/rc
-e  TZ=EST5EDT                     TZ environment variable
*e  LANG=C                         LANG environment variable
*e  NLSPATH=/usr/lib/nls/msg/%L/%N  NLSPATH environment variable
*sh /bin/sh                        shell = /bin/sh
*e  PATH=/bin                      PATH environment variable
*e  SHELL=/bin/sh                  SHELL environment variable
*e  LOGNAME=ROOT                   LOGNAME environment variable
```

/samples/profile

Example B-7 shows /samples/profile (element FSUMUPRF) at the RMID=HOT7707 level.

Example: B-7 /samples/profile

```
#*********************************************************************
#       Licensed Materials - Property of IBM                         *
#       5647-A01                                                     *
#       (C) Copyright IBM Corp. 1992, 2000                           *
#                                                                    *
#*********************************************************************
# This is a sample profile defining system wide variables. The
# variables set here may be overridden by a user's personal .profile
# in their $HOME directory.
#
# In order to customize this profile, you must first copy this sample
# to /etc/profile. More information on this profile may be found in
# OS/390 UNIX System Services Planning book and the OS/390 UNIX System
# Services User's Guide. This sample does not contain an exhaustive
# list of all variables, but only lists commonly used ones.
#
# To enable and disable lines in this profile you may remove or
# add '#' to uncomment or comment the desired lines.
#
# Variables must be 'exported' in order for the variables to be
# available for subsequent commands.
#
# Some environment variables allow you to concatenate data set names
# or directories. Most use the colon character (':') as a delimiter
# between these names.
#
# Example -   PATH=/bin:/usr/lpp/xxxxxx
#             export PATH
#
# Another method to concatenate and also allow for easier
# management is by using the previous setting of that environment
# variable.
#
# Example -   PATH=/bin
#             PATH=$PATH:/usr/lpp/xxxxxx
#             export PATH
#
#*********************************************************************

# ===================================================================
#                     STEPLIB environment variable
#                     ----------------------------
# Specifies a list of data sets to be searched ahead of the normal
# search order when executing a program. To improve the shell's
# performance for users from ISPF or users with data sets allocated to
# STEPLIB DD statements, specify "STEPLIB=none" .
# This performance improvement is not applicable to non-interactive
# shells, for example those started with the BPXBATCH and OSHELL
# utilities.
# ====================================================================
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=none
    exec sh -L
fi

# ====================================================================
#                       TZ environment variable
#                       -----------------------
# Specifies the local time zone.
# ====================================================================
TZ=EST5EDT
export TZ

# ====================================================================
#                      LANG environment variable
#                      -------------------------
# Specifies the language you want the messages to displayed in.
# For Japanese: LANG=Ja_JP
# ====================================================================
LANG=C
export LANG

# ====================================================================
#                    LOGNAME environment variable
#                    ----------------------------
# This environment variable is set when 'logging' into the shell
# environment. You can avoid accidental modification to this variable
# by making the LOGNAME variable read-only.
# ====================================================================
readonly LOGNAME

# ====================================================================
#                      PATH environment variable
#                      -------------------------
# Specifies the list of directories that the system searches for an
# executable command. If you want to include the current working
# directory in your search order, then the enviroment variable would
# be
#   PATH=/bin:.
#
# The current working directory is represented by dot ('.') .
# ====================================================================
PATH=/bin
export PATH

# ====================================================================
#                     LIBPATH environment variable
#                     ----------------------------
# Specifies the list of directories that the system searches for a DLL
# (Dynamic Link Library) filename. If not set, the current working
# directory is searched.
# ========================================================================
LIBPATH=/lib:/usr/lib:.
export LIBPATH

# ========================================================================
#                    NLSPATH environment variable
#                    ----------------------------
# Specifies the list of directories that the system searches for
# message catalogs (NLS files). The %L represents the language currently
# set by the LANG environment variable, and %N represents the name
# of the message catalog.
# ========================================================================
NLSPATH=/usr/lib/nls/msg/%L/%N
export NLSPATH

# ========================================================================
#                    MANPATH environment variable
#                    ----------------------------
# Specifies the list of directories that the system searches for man
# pages (help files). The %L represents the language currently set by
# the LANG environment variable.
# ========================================================================
MANPATH=/usr/man/%L
export MANPATH

# ========================================================================
#                     MAIL environment variable
#                     -------------------------
# Sets the name of the user's mailbox file and enables mail
# notification.
# ========================================================================
MAIL=/usr/mail/$LOGNAME
export MAIL

# ========================================================================
#                           umask variable
#                           --------------
# Sets the default file creation mask - reference umask in the OS/390
# UNIX System Services Command Reference
# ========================================================================
umask 022

# ========================================================================
# Start of c89/cc/c++ customization section
# ========================================================================
#
#   The following environment variables are used to provide information
#   to the c89/cc/c++ utilities, such as (parts of) data set names which
#   are dynamically allocated.
#
#   If installation of the compiler and/or runtime library elements use
#   different values, then the appropriate "export" lines should be
#   set to the correct value (and uncommented).  Note that since a
#   VOL=SER= paramater is not supported by c89/cc/c++, all named data
#   sets used by c89/cc/c++ must be cataloged.
#
#   It may be necessary to override the default esoteric unit for
#   (unnamed) work data sets, if the c89/cc/c++ default (SYSDA) is not
#   defined for the installed system. A NULL ("") value may be specified
#   in order to allow c89/cc/c++ to use an installation defined default.
#
#   For the _INCDIRS and _LIBDIRS environment variables, use the
#   blank character (' ') as a delimiter when concatenating directories.
#
#   To enable exporting c89/cc/c++ environment variables, uncomment the
#   "for" statement, the "done" statement, and whichever "export"
#   statements need to be customized.  Normally c89, cc and c++ all use
#   the same values, so this "for" loop will cause all of them to be
#   set. To set any particular c89, cc or c++ variable differently,
#   just code the necessary "export" statement (using the appropriate
#   prefix), following the "for" loop.
#
#   Note: This is not an exhaustive list of the environment
#   variables that affect the behavior of c89/cc/c++.  It is however all
#   those that will normally might require customization by the system
#   programmer.  For ease of migration, it is recommended that of these
#   only the variables necessary for correct operation of cc/c89/c++ be
#   set.  Consult the "Environment Variables" section of the c89/cc/c++
#   command in the OS/390 UNIX System Services Command Reference for
#   complete information about these environment variables.
#
# #######################################################################
#
# for _CMP in _C89 _CC _CXX; do
#
#
# High-Level Qualifier "prefixes" for data sets used by c89/cc/c++:
# =======================================================================
#
#
#   C/C++ Compiler:
#   ----------------------------------------
#   export ${_CMP}_CLIB_PREFIX="CBC"
#
#
#   Prelinker and runtime library:
#   ----------------------------------------
#   export ${_CMP}_PLIB_PREFIX="CEE"
#
#
#   OS/390 system data sets:
#   ----------------------------------------
#   export ${_CMP}_SLIB_PREFIX="SYS1"
#
#
# Compile and link-edit search paths:
# =======================================================================
#
#
#   Compiler include file directories:
#   ----------------------------------------
#   export ${_CMP}_INCDIRS="/usr/include /usr/lpp/ioclib/include"
#
#
#   Link-edit archive library directories:
#   ----------------------------------------
#   export ${_CMP}_LIBDIRS="/lib /usr/lib"
#
```

```
#
# Esoteric unit for data sets:
# ========================================================================
#
#
#   Unit for (unnamed) work data sets:
#   ----------------------------------------
#   export ${_CMP}_WORK_UNIT="SYSDA"
#
#
# done; unset _CMP
#
# ########################################################################
#
# ========================================================================
# End of c89/cc/c++ customization section
# ========================================================================
```

/samples/rc

Example B-8 shows /samples/rc (element FSUMIRC) at the RMID=HOT7707 level.

Example: B-8 /samples/rc

```
# Initialization shell script, pathname = /etc/rc
#
#    LICENSED MATERIALS - PROPERTY OF IBM
#    5694-A01 (C) COPYRIGHT IBM CORP. 1993, 2001
#

# Initial setup for z/OS UNIX
export _BPX_JOBNAME='ETCRC'

# Provide z/OS UNIX Startup Diagnostics
set -v -x

# Setup utmpx file
>/etc/utmpx
chmod 644 /etc/utmpx

# Reset all slave tty files
chmod 666 /dev/tty*
chown 0 /dev/tty*

# Allow only file owner to remove files from /tmp
chmod 1777 /tmp
# Allow only file owner to remove files from /var
chmod 1777 /var
# Allow only file owner to remove files from /dev
chmod 1755 /dev

# Setup write, talk, mesg utilities
# chgrp TTY   /bin/write
# chgrp TTY   /bin/mesg
# chgrp TTY   /bin/talk
# chmod 2755  /bin/write
# chmod 2755  /bin/mesg
# chmod 2755  /bin/talk
# Performed at install in HOT7707
# Commented out in HOT6609 and performed in SAMPLIB job FOMISCHO

# Setup mailx utility
# No need to CHGRP /usr/mail directory
# No need to CHGRP mailx utility
# No need to CHMOD mailx to turn on SETGID

# Setup uucp utility
# chown uucp:uucpg /usr/lib/uucp
# chown uucp:uucpg /usr/lib/uucp/IBM
# chown uucp:uucpg /usr/spool/uucp
# chown uucp:uucpg /usr/spool/locks
# chown uucp:uucpg /usr/spool/uucppublic
# chown uucp:uucpg /usr/spool/uucp/.Xqtdir
# chown uucp:uucpg /usr/spool/uucp/.Sequence
# chown uucp:uucpg /usr/spool/uucp/.Status
# chown uucp:uucpg /bin/uucp
# chown uucp:uucpg /bin/uuname
# chown uucp:uucpg /bin/uustat
# chown uucp:uucpg /bin/uux
# chown uucp:uucpg /usr/lib/uucp/uucico
# chown uucp:uucpg /usr/lib/uucp/uuxqt
# chown uucp:uucpg /usr/lib/uucp/uucc
# chmod 4755 /bin/uucp
# chmod 4755 /bin/uuname
# chmod 4755 /bin/uustat
# chmod 4755 /bin/uux
# chmod 4754 /usr/lib/uucp/uucico
# chmod 4754 /usr/lib/uucp/uuxqt
# chmod 4754 /usr/lib/uucp/uucc
# Performed at install in HOT7707
# Commented out in HOT6609 and performed in SAMPLIB job FOMISCHO

# Invoke vi recovery
#
#
# mkdir -m 777 /var/tmp
# export TMP_VI="/var/tmp"
mkdir -m 777 /etc/recover
/usr/lib/exrecover

# Create TERMINFO database
# tic /usr/share/lib/terminfo/ibm.ti
# tic /usr/share/lib/terminfo/dec.ti
# tic /usr/share/lib/terminfo/wyse.ti
# tic /usr/share/lib/terminfo/dtterm.ti
# commented tic out in HOT1180 - all TERMINFO files are shipped

# Start the INET daemon for remote login activity
#_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf &

sleep 5
echo /etc/rc script executed, `date`
```

C

Access control list (ACL) support considerations

This appendix describes some of the programming changes for system programmers and z/OS UNIX users that may be used in support of the changes made in z/OS V1R3 to support ACLs with z/OS UNIX.

The changes to support ACLs are:

- Using the **`setfacl`** and **`getfacl`** z/OS UNIX shell commands
- Working with default ACLs
- Callable services ACL support
- z/OS UNIX REXX support for ACLs
- LE callable services support for ACLs

C.1 Examples of the setfacl and getfacl commands

The **-m option** modifies ACL entries, or adds them if they do not exist. The command to grant read/write permissions to user ID PKISTU and group PKIADM to file /web/pki1/httpd.conf is:

```
setfacl -m user:PKISTU:rw-,group:PKIADM:rw- /web/pki1/httpd.conf
```

Figure C-1 shows the result of the **getfacl** command.

```
ANTOFF:/u/antoff: >getfacl /web/pki1/httpd.conf
#file:  /web/pki1/httpd.conf
#owner: HAIMO
#group: SYS1
user::rwx
group::r-x
other::r-x
user:PKISTU:rw-
group:PKIADM:rw-
```

Figure C-1 Output from the getfacl command for file httpd.conf

The output from the **ls -al** command, shown in Figure C-2, shows file httpd.conf having a plus (+) sign following the permission bits, which indicates that an extended ACL exists.

```
ANTOFF:/u/antoff: >ls -la /web/pki1/
total 768
drwxr-xr-x   8 HAIMO    SYS1        8192 Apr 16 15:37 .
drwxr-xr-x  14 HAIMO    SYS1        8192 Apr 21 13:45 ..
-rw-r--r--   1 HAIMO    IMWEB          9 Apr 21 13:26 httpd-pid
-rwxr-xr-x+  1 HAIMO    SYS1      146400 Apr 15 11:51 httpd.conf
-rw-r--r--   1 HAIMO    SYS1         483 Apr 18 17:28 httpd.envvars
drw-------   2 HAIMO    SYS1        8192 Apr 16 15:37 pkiserv
drw-------   2 HAIMO    SYS1        8192 Apr 14 15:11 sec
```

Figure C-2 Output from ls -la showing the plus sign for file httpd.conf

C.1.1 Change access level for user/group in an extended ACL

With the **-m option** you can change the access level for a user/group in an extended ACL. For example, if you wish to change the access for group PKIADM from rw- to r-- for file /we/pki1/httpd.conf, issue the command:

```
setfacl -m group:PKIADM:r-- /web/pki1/httpd.conf
```

Delete a user/group from an extended ACL

Option **-x** removes a user/group from an extended ACL, as shown in Figure C-3 on page 561 when we removed user PKISTU (but group PKIADM is left):

```
setfacl -x user:PKISTU  /web/pki1/httpd.conf
```

```
getfacl /web/pki1/httpd.conf
#file:  /web/pki1/httpd.conf
#owner: HAIMO
#group: SYS1
user::rwx
group::r-x
other::r-x
group:PKIADM:rw-
```

Figure C-3 Removing user ID PKISTU from ACL for file httpd.conf

C.1.2 Set an entire ACL (base and extended)

The **-s** option sets an entire ACL, which includes the base ACLs and extended ACLs. The base ACL (permission bits) must be indicated by omitting the user or group qualifiers and using column (:) instead. Extended ACLs may not be defined at all. For example, for file /we/pki1/httpd.conf, if you want to disallow access to anybody but the file owner, who will have read only, issue the command:

```
setfacl -s user::r--,group::---,other::--- /web/pki1/httpd.conf
```

The result from the **getfacl** command is in Figure C-4.

```
getfacl /web/pki1/httpd.conf
#file:  /web/pki1/httpd.conf
#owner: HAIMO
#group: SYS1
user::r--
group::---
other::---
```

Figure C-4 Setting base permissions

The **-s** option can be used to simultaneously change the base ACL and to add an extended ACL:

```
setfacl -s user::rwx,group::r-x,other::---,user:PKISTU:r-- /web/pki1/httpd.conf
```

The result of the **getfacl** command is shown in Figure C-5.

```
getfacl /web/pki1/httpd.conf
#file:  /web/pki1/httpd.conf
#owner: HAIMO
#group: SYS1
user::rwx
group::r-x
other::---
user:PKISTU:r--
```

Figure C-5 Simultaneously setting base and extended permissions

Delete an entire access ACL

The **-D a** option specifies that the access ACL is to be deleted. The base ACL remains unchanged. When a file is deleted, its ACL is automatically deleted; no extra administrative effort is required.

```
setfacl -D a /web/pki1/httpd.conf
```

The result is shown in Figure C-6.

```
getfacl /web/pki1/httpd.conf
#file:  /web/pki1/httpd.conf
#owner: HAIMO
#group: SYS1
user::r--
group::---
other::---
```

Figure C-6 Deleting the whole access ACL

The options for the **setfacl** command work exactly in the same manner for directories as for files, for example:

```
setfacl -m user:PKISTU:r-- /web/pki1/
```

The result is shown in Figure C-7.

```
getfacl /web/pki1/
#file:  /web/pki1//
#owner: HAIMO
#group: SYS1
user::rwx
group::r-x
other::r-x
user:PKISTU:r--
```

Figure C-7 Allow user ID PKISTU to have r-- for directory /web/pki1/

As we can see, the creation and deletion of access ACLs resembles the creation and deletion of RACF discrete data set profiles.

C.1.3 Pipe output from getfacl to setfacl

You may wish to take the ACL from FileA in the current directory and apply it to FileB, also in the current directory:

```
getfacl FileA | setfacl -S - FileB
```

The shell pipes the output of **getfacl** to the input of **setfacl**. The **-S** option of **setfacl** says to replace the contents of the file's ACL with ACL entries specified within a file, and the "-" is a special case of the file name designating **stdin**. Thus, you can maintain a list of ACL entries within a file, and use that file as input to a **setfacl** command. You might use this ability to implement a "named ACL" for a given project, such as in the next paragraph.

Output from the find command as input to setfacl

The file /u/joeadmn/Admins contains a list of ACL entries for users and groups who need to support some administrative work. The file contains ACL entries, one per line, in the format that **setfacl** expects and which **getfacl** displays.

These people must be granted access to all of the directories within the file system subtree starting and including /admin/work.

```
setfacl -S /u/joeadmn/Admins $(find /admin/work -type d)
```

This example uses shell command substitution to use the output of the **find** command as input to the **setfacl** command. The /u/joeadmn/Admins file may contain, for example:

```
user::rwx
group::---
other::---
u:user1:rwx
u:user2:rwx
g:group1:rwx
```

Suppose you wish to give Lucy rw- to every file within Fred's home directory for which Ricky has rw-. Issue the following command:

```
setfacl -m user:lucy:rw- $(find ~fred -acl_entry user:ricky:+rw)
```

You can use an access ACL on the parent directory to grant search access only to those users and groups who should have file access. The access ACL of the parent directory can have been automatically created as the result of a directory default ACL on its parent. Make sure that the "other" and perhaps the "group" search permission bits are off for the parent directory.

When creating ACLs, consider the following:

- To minimize the impact on performance, keep ACLs as small as possible, and permit groups to files instead of individual users. The path length of the access check will increase with the size of an ACL, but will be smaller than the associated checking would be for a RACF profile with the same number of entries in its access list.
- Do not disable ACLs after you have used them for a while and have created many entries. Only consider disabling ACLs if you have not used them for very long. If you have been using ACLs to grant, rather than deny, access to particular users and groups, then disabling ACLs will likely result in a loss of file access authority rather than a gain.

The file default ACL and the directory default ACL may be used to facilitate and streamline the security administrator's effort by providing models for access for newly created files or directories.

C.2 Working with default ACLs

The command **setfacl** uses a second keyword in order to manage file and directory default ACLs:

d	For directory
f	For file

To create a directory default ACL for directory /u/antoff/ that will allow access r-x (read content of directory and search directory) to group PKIADM every time when a subdirectory of /u/antoff/ is created, use the command:

```
setfacl -s u::rwx,g::---,o::---,d:g:PKIADM:r-x /u/antoff/
```

To display the result, **getfacl** must be used with the **-d** keyword as shown in Figure C-8 on page 564.

```
getfacl -d /u/antoff/
#file:  /u/antoff//
#owner: ANTOFF
#group: SYS1
default:group:PKIADM:r-x
```

Figure C-8 Creating a directory default ACL for directory /u/antoff/

If we now try to create a *file* default ACL for directory /u/antoff/ that will allow access READ to user ID TRAUNER every time a new file in /u/antoff/ is created with:

```
setfacl -s u::rwx,g::---,o::---,f:u:trauner:r-- /u/antoff/
```

Then our just created *directory* default ACL will be deleted.

More flexible is the **-m option** when we do not want to change the basic ACL. To test this, we deleted the just created file default ACL with:

```
setfacl -D f /u/antoff/
```

And issued in succession:

```
setfacl -m d:g:PKIADM:r-x /u/antoff/
```

```
setfacl -m f:u:trauner:r-- /u/antoff/
```

This time **getfacl** is used with the **-df** keywords and the result is shown in Figure C-9.

```
getfacl -df /u/antoff/
#file:  /u/antoff//
#owner: ANTOFF
#group: SYS1
user::rwx
group::---
other::---
fdefault:user:TRAUNER:r--
default:group:PKIADM:r-x
```

Figure C-9 Creating directory and file default ACLs for directory /u/antoff/

C.2.1 Use output of find as input to setfacl

If you wish to define a file default ACL for the directory named /u/ProjectX, and also for all of its subdirectories, specifying that group *admins* has r-- and group *dirgrp* has rw-, issue the command:

```
setfacl -m f:group:admins:r--,f:group:dirgrp:rw- $(find /u/ProjectX -type d)
```

Groups admins and dirgrp will automatically get access to any new files created within the /u/ProjectX subtree.

Note that the creation of a default ACL will not grant access to files/directories that already exist.

Delete default ACLs

The following command removes both file and default ACLs:

```
setfacl -D df /u/antoff
```

To remove all extended ACL entries (access, directory, and file defaults) for all files and directories if the current directory is /u/antoff, issue the command:

```
setfacl -D e *
```

Attention: Analyze your HFS space utilization before implementing default ACLs in your file system. If you use both file and directory default ACLs in every directory in the file system, a separate physical ACL is created for every new file and directory. Using an access ACL for every directory will probably not cause concerns about space utilization. However, the same cannot be said of files, especially if the inherited ACLs are large.

Tip: ACLs are not inherited across mount points. Suppose that you have a default ACL defined on the directory /dir1/dir2. You decide to create another directory, /dir1/dir2/dir3, and use it as a mount point on which to mount another file system.However, if you do so, the root directory of the mounted file system will not inherit the default ACL that had been established at /dir1/dir2. If you want the default ACLs of dir2 to apply to dir3, you must copy them to dir3 after dir3 has been mounted.

C.3 Callable services for ACL support

Several callable services have been modified in order to support ACL entries:

- BPX1IOC(w_ioctl)/BPX1PIO(w_pioctl)

  These callable services (called, for example, by **setfacl** and **getfacl** commands) perform a device-specific command.

  ```
  Call
  BPX1IOC,(file_descriptor,command,argument_length,argument,return_value,return_c
  ode,reason_code)
  ```

  Support has been added in order to send and receive directory or file ACL structures to/from Physical File Systems (PFS) by:

  – Expanding interface to allow larger *argument_length* from 1024 to 2,147,483,647.

  – Adding new command codes for processing ACLs

    - SetfACL sets ACL for file or directory.
    - GetfACL gets ACL for file or directory.

- BPX1STA(stat/fstat) returns flags indicating the presence and type of any ACLs on the specified file or directory. It is used by the shell **ls** command.

  ```
  Call
  BPX1STA,(pathname_length,pathname,Status_Area_Length,Status_Area,return_value,r
  eturn_code,reason_code)
  ```

  BPXYSTAT mapping macro used adds information indicating ID access ACL exists, default directory model exists, or default file model exists.

- BPX1PCF(pathconf/fpathconf) returns configurable variables associated with the file at the specified path name. It is used by the shell *getconf* command.

  ```
  Call
  BPX1PCF,(pathname_length,pathname,name,return_value,return_code,reason_code)
  ```

  New parm values have been added:

  _ACL - Indicates whether an access control mechanism is supported by the file system owning the file specified by “pathname”. Values can be TRUE or FALSE.

_ACL_ENTRIES_MAX - Specifies the maximum number of entries in an ACL for file or directory.

BPXYPCF mapping macro changes to support.

- BPX1GMN(w_getmntent) returns information on a mounted file system. It is used by the **df** shell command.

  `Call BPX1GMN,(`*`Buffer_Length,Buffer,return_value,return_code,reason_code`*`)`

  BPXYMNTE mapping macro changes to support.

- Other mapping macro changes:

  BPXYVFSI - VFS Callable Service Interface includes ACL fields on the ATTR structure.

For more information about the changes made to the Callable Services, refer to *z/OS UNIX System Services Programming Assembler Callable Services Reference,* SA22-7803.

C.4 z/OS UNIX REXX support for ACLs

New services to get, create, update, replace, or delete an ACL for a file or directory have been added:

- **aclinit** *variable.* Obtain resources necessary to process ACLs and associate those resources with *variable*. *Variable* is the name of a REXX variable that contains a token to access an ACL.
- **aclfree** *variable***.** Releases resources associated with the ACL represented by *variable* that were obtained by the **aclinit** service.
- **aclget** *variable pathname acltype*. Read an ACL of specified *acltype* associated by the file identified by *pathname.* The ACL is associated with the specified *variable*. *Pathname* is the pathname of the file or directory the ACL is associated with*. Acltype* indicates the type of ACL (access, default file or default directory).
- **acldelete** *pathname acltype.* Deletes an ACL of specified *acltype* associated by the file identified by *pathname*.
- **aclset** *variable pathname acltype*. Replace the ACL associated by the file identified by *pathname* with the ACL represented by *variable*.
- **aclgetentry** *variable stem[index]*. Reads the ACL entry from the ACL represented by *variable.* The entry is identified by *index* if specified, otherwise the entry is identified by the type and ID specified by *stem.* If index is specified, stem is purely an output variable. Otherwise, stem is used for both input and output. The ACL entry is placed in stem.

  stem is the name of a stem variable that contains an ACL entry. STEM.0 contains a count of the number of variables set in the stem. The following variables may be used to access the stem variables:

  – **acl_entry_user** indicates user ACL.

  – **acl_entry_group** indicates group ACL.

  – **acl_id is** the UID or UID of the entry.

  – **acl read** indicates read access.

  – **acl_write** indicates write access.

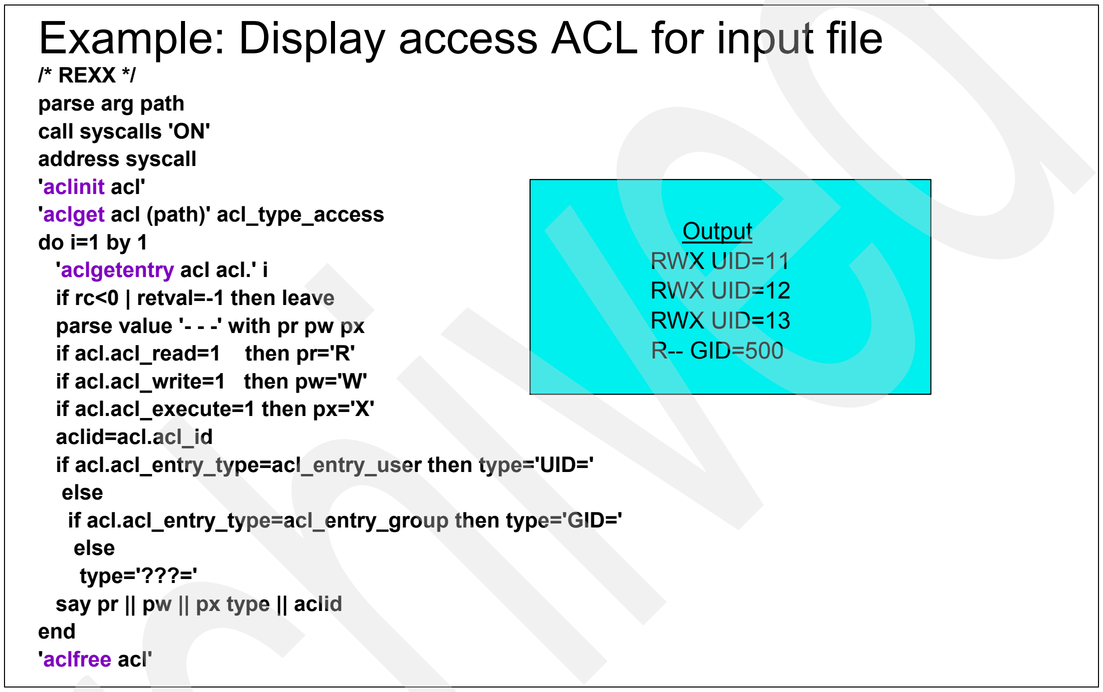
  – **acl_execute** indicates execute/search access.

  – **acl_delete** indicates the ACL entry as deleted.

  index specifies the relative ACL entry to process. Indexing begins at 1.

- `aclupdateentry` *variable stem[index]*. Updates (or creates, if it does not already exist) an ACL entry from the ACL represented by *variable*. The entry is identified by *index* if specified; otherwise, the entry is identified by the type and ID specified by *stem.* If *index* is specified, *stem* is purely an output variable. Otherwise, *stem* is used for both input and output.
- `acldeleteentry` *variable stem*. Deletes an ACL entry from the ACL represented by *variable*. The entry is identified by the entry type and ID specified by *stem.*

The example in Figure C-10 shows a REXX program used for displaying ACL for a requested input file.



Example: Display access ACL for input file

```
/* REXX */
parse arg path
call syscalls 'ON'
address syscall
'aclinit acl'
'aclget acl (path)' acl_type_access
do i=1 by 1
  'aclgetentry acl acl.' i
  if rc<0 | retval=-1 then leave
  parse value '- - -' with pr pw px
  if acl.acl_read=1    then pr='R'
  if acl.acl_write=1   then pw='W'
  if acl.acl_execute=1 then px='X'
  aclid=acl.acl_id
  if acl.acl_entry_type=acl_entry_user then type='UID='
   else
    if acl.acl_entry_type=acl_entry_group then type='GID='
     else
      type='???='
  say pr || pw || px type || aclid
end
'aclfree acl'
```

```
Output
RWX UID=11
RWX UID=12
RWX UID=13
R-- GID=500
```

Figure C-10 Sample REXX program

C.4.1 Other interface changes

New variables for ACL support have been added to the STAT, FSTAT, LSTAT syscall commands:

- ST_ACCESSACL- 1 if access ACL exits
- ST_DMODELACL- 1 if directory model ACL exists
- ST_FMODELACL- 1 if file model ACL exists.

New variables for ACL support have been added to the PATHCONF syscall command:

- PC_ACL to test if ACLs are supported for the resource
- PC_ACL_MAX to query max number of allowed entries in an ACL

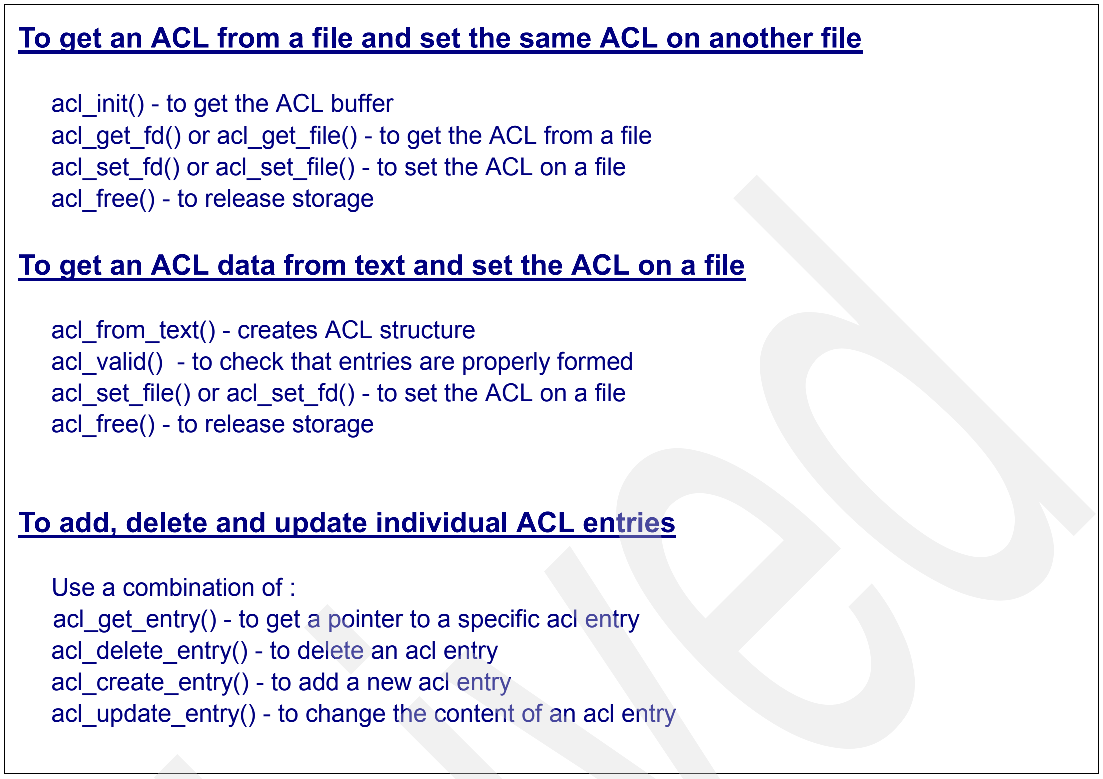
For more information about REXX support for ACL, refer to *z/OS Using REXX and z/OS UNIX System Services,* SA22-7806.

C.4.2 LE Callable Services support for ACLs

LE Callable Services has been modified in order to support ACL entries, as follows:

- ACL Storage Management functions:
  - Initialize ACL working storage
    `lacl_t acl_init(init count);`
  - Release memory allocated to an ACL Data Object
    `int acl_free(acl_t obj_pp);`
- Functions that manipulate complete entries in an ACL:
  - Get an ACL entry
    `int acl_get_entry(lacl_t acl_d,acl_entry_t*entry_p);`
  - Return to beginning of ACL Working Storage
    `int acl_first_entry(lacl_t acl_d);`
  - Validate an ACL
    `int acl_valid(lacl_t acl_d,acl_entry_t*entry_p)`
  - Add a new extended ACL entry to the ACL
    `int acl_create_entry(lacl_t*acl_p,acl_entry_t entry_p,int version);`
  - Delete the specified extended ACL entry from the ACL
    `int acl_delete_entry(lacl_t acl_d,acl_entry_t entry_d):`
  - Update the extended ACL entry
    `int acl_update_entry(lacl_t acl_d,acl_entry_t entry_s,acl_entry_t entry_d,int version);`
- Functions that manipulate the whole ACL object:
  - Delete an ACL by File Descriptor
    `int acl_delete_fd(int fd,acl_type_t type_d):`
  - Delete an ACL by Filename
    `int acl_delete_file(const char *path_p,acl_type_t type_d);`
  - Get an ACL by File Descriptor
  - *int* **`acl_get_fd`**(*int fd,acl_type_t type_d,lacl_t acl_d,int *num)*;
  - Get ACL by filename
  - *int* **`acl_get_file(`***const char *path_p,acl_type_t type_d,lacl_t acl_d,int *num*);
  - Set an ACL by file descriptor
  - *int* **`acl_set_fd`**(*int fd,acl_type_t type_d,lacl_t acl_d,short OpType,acl_entry_t *entry_p);*
  - Set an ACL by filename
  - *int* **`acl_set_file`**(*const char *path_p,acl_type_t type_d,lacl_t acl_d,short OpType,acl_entry_t*entry_p)*;
- Functions that convert between formats of ACL:
  - Convert an ACL to Text
  - *char ** **`acl_to_text`**(*const lacl_t acl_d,ssize_t*len_p,acl_type_t type_d,char delim*);
  - Create an ACL from text
  - *int* **`acl_form_text`**(*const char *buf_p,short OpType,acl_all_t ptr,char **ret*);
  - Sort the extended ACL entries (USER, GROUP low to high)
  - *int* **`acl_sort`**(*lacl_t acl_d*);

Figure C-11 on page 569 shows examples of using the functions.



To get an ACL from a file and set the same ACL on another file

acl_init() - to get the ACL buffer
acl_get_fd() or acl_get_file() - to get the ACL from a file
acl_set_fd() or acl_set_file() - to set the ACL on a file
acl_free() - to release storage

To get an ACL data from text and set the ACL on a file

acl_from_text() - creates ACL structure
acl_valid() - to check that entries are properly formed
acl_set_file() or acl_set_fd() - to set the ACL on a file
acl_free() - to release storage

To add, delete and update individual ACL entries

Use a combination of :
acl_get_entry() - to get a pointer to a specific acl entry
acl_delete_entry() - to delete an acl entry
acl_create_entry() - to add a new acl entry
acl_update_entry() - to change the content of an acl entry

Figure C-11 Examples of using ACL functions

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 573. Note that some of the documents referenced here may be available in softcopy only.

- *Putting the Latest z/OS Security Features to Work,* SG24-6540
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 1: Base and TN3270 Configuration,* SG24-5227
- *OS/390 Security Server 1999 Updates: Installation Guide,* SG24-5629
- *Implementing DFSMSdss SnapShot and Virtual Concurrent Copy,* SG24-5268
- *Hierarchical File System Usage Guide,* SG24-5482
- *z/OS Distributed File Service zSeries File System Implementation*, SG24-6580

Other publications

These publications are also relevant as further information sources:

- *z/OS UNIX System Services Planning,* GA22-7800
- *z/OS Using REXX and z/OS UNIX System Services,* SA22-7806
- *z/OS UNIX System Services Command Reference,* SA22-7802
- *z/OS UNIX System Services Messages and Codes,* SA22-7807
- *z/OS UNIX System Services File System Interface Reference,* SA22-7808
- *z/OS UNIX System Services User's Guide,* SA22-7801
- *z/OS UNIX System Services Programming Assembler Callable Services Reference,* SA22-7803
- *z/OS DFSMS Using Data Sets,* SC26-7410
- *z/OS Communications Server IP Configuration Guide,* SC31-8775
- *z/OS Communications Server IP Configuration Reference,* SC31-8776
- *z/OS Communications Server IP Application Programming Interface Guide,* SC31-8788
- *z/OS DFSMSdss Storage Administration Reference,* SC35-0424
- *z/OS MVS Diagnosis Reference,* GA22-7588
- *z/OS MVS Initialization and Tuning Reference,* SA22-7592
- *z/OS MVS JCL Reference,* SA22-7597
- *z/OS MVS Setting Up a Sysplex,* SA22-7625
- *z/OS MVS System Codes,* SA22-7626
- *z/OS MVS System Commands,* SA22-7627

- *z/OS MVS Programming Authorized Assembler Services Guide,* SA22-7608
- *z/OS Security Server RACF System Programmer's Guide,* SA22-7681
- *z/OS Security Server RACF Macros and Interfaces,* SA22-7682
- *z/OS V1R4.0 Security Server RACF Security Administrator's Guide,* SA22-7683
- *z/OS Distributed File Service SMB Administration,* SC24-5918
- *z/OS Distributed File Service zSeries File System Administration,* SC24-5989
- *z/OS Program Directory,* GI10-0670
- *TCP/IP Tutorial and Technical Overview,* GG24-3376
- *Implementing Concurrent Copy,* GG24-3990
- *TSM Using the Backup-Archive Clients,* SH26-4105
- *TSM Installing the Clients,* SH26-4102
- *Tivoli Storage Manager for MVS and OS/390: Quick Start,* GC35-0376
- *Tivoli Storage Manager for MVS and OS/390: Administrator's Guide,* GC35-0377
- *z/OS Security Server RACF Diagnosis Guide,* GA22-7689
- *z/OS DFSMS/MVS DFSMSdss Storage Administration Reference,* SC26-4929
- *z/OS DFSMS/MVS V1R5 DFSMSdss Storage Administration Guide,* SC26-4930
- *BM BookManager BookServer for World Wide Web for z/OS: Getting Started,* SC31-8814
- *z/OS HTTP Server Planning, Installing, and Using,* SC34-4826
- *New IBM Technology Features Persistent Reusable Java Virtual Machines,* SC34-6201
- *z/OS Language Environment Customization,* SA22-7564
- *z/OS Language Environment Programming Guide,* SA22-7561
- *XPLink: OS/390 Extra Performance Linkage,* SG24-5991
- *Network File System Customization and Operation,* SC26-7417
- *OS/390 UNIX System Services Planning,* SC28-1890
- *Text Search - Programming the Text Search Engine,* SH12-6717

Online resources

These Web sites and URLs are also relevant as further information sources:

- Adobe Acrobat PDF versions of publications available on the June 2003 z/OS V1R4.0 elements and features CD Collection Kit (SK3T-4269-08) plus additional related PDFs:

  `http://www-1.ibm.com/servers/eserver/zseries/zos/bkserv/r4pdf/`

- IBM interactive wizards assistants that ask you a series of questions about the task you want to perform (for example, IP Configuration Wizard). The wizards simplify your planning and configuration needs by exploiting recommended values and by building customized checklists for you to use. For configuration tasks, our wizards also generate outputs like jobs, policies, or parmlib members that you can upload to z/OS and use.

  `http://www-1.ibm.com/servers/eserver/zseries/zos/wizards/`

- The RACF Web site:

  `http://www-1.ibm.com/servers/eserver/zseries/zos/racf/goodies.html`

- The z/OS UNIX home page on the World Wide Web has the latest technical news, customer stories, tools, and FAQs (frequently asked questions). You can visit it at:

  `http://www.ibm.com/servers/eserver/zseries/zos/unix/`

- Customizing z/OS UNIX, with the Web-based wizard. You can access it at:

  `http://www.ibm.com/servers/eserver/zseries/zos/wizards/`

- The full-function Software Development Kit (SDK) at the Java 2 technology level, compliant with the Sun SDK 1.4 APIs. For more information on the actual list of supported APIs, check the following URL:

  `http://java.sun.com/j2se/1.4/docs/api/`

- The complete Java 2 Technology Development Kit SDK at 1.3 level for the zSeries and S/390 platforms. For the complete API functions list, check the following URL:

  `http://java.sun.com/j2se/1.3/docs/api/`

- Internet address for the latest information on Java release level incompatibilities:

  `http://java.sun.com/j2se/1.4/compatibility.html`

- The non-SMP/E Java 2 V1.4 can be downloaded at URL:

  `http://www-1.ibm.com/servers/eserver/zseries/software/java/getsdk14.html`

- Java 2 Technology Edition version 1.4 has its list of prerequisites. Check for the latest APARs that need to be applied at URL:

  `http://www-1.ibm.com/servers/eserver/zseries/software/java/prereqs14.html`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols
$HOME/.profile 75, 313, 364, 367
$HOME/.sh_history 362
.profile 312
/bin/sh 73
/etc/auto.master 303
/etc/ftp.data 213
/etc/inetd.conf 211
/etc/init.options 73, 75, 312
/etc/log 73
/etc/profile 75, 78, 312
/etc/rc 73, 75–77, 304, 312
/etc/services 210, 213
/etc/suid_profile 86
/etc/syslog.conf 207
/etc/u.map 303
/notesdata directory 465
/samples/rc 76
/tmp directory 310
/usr/sbin/automount 304
/usr/sbin/init 73, 75
_BPX_JOBNAME 405
_BPX_JOBNAME environment variable 174
_BPX_SHAREAS 370, 378
_BPX_SPAWN_SCRIPT 370–371, 378
_BPXK_SETIBMOPT_TRANSPORT 200
_CEE_RUNOPTS 331
_POSIX_CHOWN_RESTRICTED 127

Numerics
3DES 387

A
access ACL 140
access control lists 121, 135, 232
accessor environment element 102
ACEE 102
ACEE support 170
ACL 232
ACL inheritance 141
ACL support 232
ACLs 121, 229
AF_INET 193
aggregate
  compatibility mode 233
  definition 233, 296
AIM 92, 523
  migration 528
ALLOCxx parmlib member 67
analyzing USS hang situations 503
APPLDATA keyword 116
Application heap 336
Application Identity Mapping 92
application identity mapping 92, 96, 523
ASCII functionality 418
ASCII to EBCDIC 420
ASNAME keyword 438
ASSIZEMAX 328
audit records 179
autoconversion 420
AUTOCVT 429
AUTOCVT statement 422
AUTOGID keyword 115
AUTOID keyword 115
automatic codeset conversion 423
automatic conversion 422
automatic UID/GID assignment 115
automount 303
automount command 304
automount facility 302
  customizing 309
automount policy 304, 308–309
  dynamic allocation od hfs 305
automount policy keywords
  lowercase 307
AUTOMOVE 265–266
  keyword 436
AUTOMOVE EXCLUDE 266
AUTOMOVE INCLUDE 266
AUTOMOVE(INCLUDE,SC64,*) 266
Autonomic Computing 502
AUTOUID operand 293

B
backup file system 236
base ACL entries 134
Berkeley UNIX C shell 10
BLKUPD command 527
BookManager READ 371
Bourne shell 10
BPX.DAEMON 154, 202
BPX.DAEMON RACF facility class 212
BPX.DAEMON.HFSCTL 157
BPX.DEBUG 164
BPX.DEBUG resource 172
BPX.FILEATTR.APF 164
BPX.FILEATTR.PROGCTL 164
BPX.FILEATTR.SHARELIB 164
BPX.MAINCHECK 154, 168
BPX.SERVER 168
BPX.SERVER profile 171
BPX.SUPERUSER 84
BPX.SUPERUSER profile 82
BPX1SEC1 175
BPXAS 11
BPXAS address spaces 22
BPXAS procedure 55

BPXBATCH 326–327, 376, 379
BPXBATCH REGION 327
BPXBATCH sample job 376
BPXBATSL 334, 379
BPXFX111 translation table 312
BPXFX450 conversion table 312
BPXINIT program 54
BPXISMKD 209
BPXMCDS 273
BPXMCDS couple data set 269
BPXOINIT 54
BPXOINIT address space 22
BPXPINPR program 54
BPXPRMxx 58
BPXPRMxx member 231, 302
    MOUNT statements 298
BPXPRMxx MKDIR() parameter 62
BPXPRMxx parmlib member
    FILESYSTYPE 59
    LIMMSG parameter 270
    STARTUP_PROC 54
    TAG parameter 424
bpxrexx sample 380
BPXROOT 153
bridge 187
BRLM 272
BSAM 337
byte-range lock manager 272
byte-range locking 273

C
CCSID 420–424, 428–429
CEE.SCEERUN 202
CEEDUMP 439
centralized BRLM 272
character special file 18
chaudit command 179–180, 182
chcp command 312
chgrp command 127
child process 3
chkfs program 404
chmod command 128
    authorization to use 128
chmount command 172, 259, 267
chown command 127
chtag command 425
CINET sockets 438
class descriptor table 181
clone 235
clone file system 235
code page 311
    IBM-037 312
    IBM-1047 312
coded character set identifier 420
codeset 423
COFVLF00 member 525
COFVLFxx member 94
colony address space 22, 232, 438
command prompt layout 367
commands 362
COMMNDxx 196
compatibility mode aggregate 233
concurrent copy 443
confighfs command 272
confighfs utility 449
console dump 503
copytree utility 444
cp command 150, 425
cron daemon 152–153, 162
CTRACE statement 68

D
daemon 91, 151
daemon security 162
Data Facility Data Set Services (DFSMSdss) 484
DATASET class 161
DATASETPREFIX 195
DDDEF SEZAMMSC 490
df command 151
DFSMS 233, 296
DFSMSdss 484
    functions 441
    logical dump 442
DFSMSdss DUMP function 441
DFSMSdss RESTORE function 441
DFSMShsm 484
directory default ACL 141
Directory List enhancements 353
dirty address space 157
display latch activity 503
distributed BRLM 273
Domino applications 479
Domino environment 481
Domino performance 464, 481
Domino server 466
dsm.opt 452
dsm.sys 452
dub 10
dumping HFS data sets 484

E
EBCDIC format 419
EBCDIC programs 424
Edit Entry Panel 366
effective user ID (EUID) 86, 354
END SAVE keyword 527
END subcommand 527
enhanced ASCII 418, 420
    support 422
enhanced program security 158, 160
ENHANCED-WARNING mode 158
environment file 365
environment variable
    _ BPXK_AUTOCVT 421
environment variables 311
Ephemeral 187
ETC.SERVICES 197, 213
etc/profile 364
EUID 86

EXCLUDE 266
exec() 204
export command 368
EXPORTS 223
extattr bits 163
extended ACL entries 134
extended attributes bits 162
external link 19, 21
EZAZSSI 194

F
F OMVS,RESTART command 436
FACILITY class
  BPX.DAEMON.HFSCTL 157
  BPX.DEBUG 164
  BPX.FILEATTR.APF 164
  BPX.FILEATTR.PROGCTL 156, 164
  BPX.FILEATTR.SHARELIB 164
  BPX.MAINCHECK 160
  BPX.NEXT.USER 116
  BPX.SUPERUSER 137
  IRR.PGMSECURITY 158–159
FACILITY class profile
  BPX.SAFFASTPATH 173
  BPX.SERVER 169
  BPX.STOR.SWAP 174
  BPX.WLMSERVER 174
FIELD class 114
file default ACL 141
file mode 122
file security packet 121, 232
file system
  backup 236
  clone 235
file tagging 422, 424
FILESYSTYPE 59
FILESYSTYPE statements 437
FILETAG run-time option 421
find command 150
FOMISCHO job 175
FSP 121, 131, 162, 232
FSSEC class 137–138, 182
FTP.DATA 213
FTPD 212
FTPD procedure 200
full function mode 48
FwdRetr 363

G
garbage collection 328
gateway 187
getfacl command 134, 560
GFSCPROC STEPLIB 218
global variables 311
GLOBALTCPIPDATA 193
GRS 502

H
hardlink 409
heap 326
  application 336
  middleware 336
  pools 333–334
  size 328
  system 336
  transient 336
HeapPools storage option 334
HEAPPOOLS(ON) 334
HFS data set recovery 484
HFS ROOT 60
Hierarchical Storage Manager (DFSMShsm) 484
high-level qualifier 195
history file 362
hlq.SEZAHELP 194
hlq.SEZAMENU 194
hlq.SEZAPENU 194
host 187
host address 188

I
IBM-1047 421, 423
ICETOOL 101
ICETOOL utility 513
ICHRIN03 definitions 55
ICHRRCDX 181
iconv command 426
IDCAMS 294
identity mapping 92
IEAAPFxx 195
IEASYMxx 196
IEASYSxx 196
IEASYSxx member
  omvs=default 51
IECIOSxx parmlib member 196
IEFPDSI 194
IEFSSNxx 196
IEFUSI exit 328
IFAPRDxx 196
INCLUDE 266
INET sockets 438
inetd 201
inetd daemon 374
inetd.conf 203
Infoprint Server 319
internetwork 186
ioeagslv command 236
IP router 186
ipcrm command 173
IRR.PGMSECURITY 158
IRRDBU00 94
IRRDBU00 utility 101
IRRGMAP class 92
IRRHFSU utility 149
IRRIRA00 conversion utility 93
IRRIRA00 utility 96, 98, 523
IRRPFACL macro 135

IRRUMAP 525
IRRUMAP class 92
IRRUT200 utility 98, 101, 523, 525
IRRUT400 utility 101, 526
ISHELL command 301
ISHELL mount panel 301
ISHELL panel
  mounts 257
ISO8859-1 421, 423
ISPF editor 112
ISPF shell 23
IXCL1DSU utility 269

J
Java Native Interface (JNI) 333
Java program 371
Java Virtual Machine 323, 328
JWT value 68

K
kernel 3, 11
korn shel 10

L
Language Environment 333–334
LATCH 502
latch contention 502
Latch contention analysis 503
latches 501
LFS (logical file system) 231
LIMMSG parameter 270
LIST NEW keyword 527
LNKLSTxx 194, 196
local_spawn() 204
logical file system (LFS) 231
LOGREC 524
lp command 320
LPALSTxx parmlib member 196
lpstat command 322
ls command 150, 425

M
mailx utility 129
maintaining UNIX System Services
  backup and restoring HFS data sets 484
  installing service 493
malloc() 330
man command 372
man pages 408
map file 306
  system symbols 308
MapName file 303
MAXASSIZE 411
MAXFILEPROC 411
MAXFILESIZE 411
MAXPTYS 209–210
metadata 235
metadata cache 235
Middleware heap 336
minimum mode 48
mkdir command 298
MKDIR() 62
MOUNT 61
MOUNT MKDIR() 62
mount shell command 267
MOUNT statement 61, 440
MSTJCLxx 194
multi-file mode aggregate 234
multi-file system aggregates 234
mv command 150
mvslogin command 223
mvslogout command 223
MVSNFS 222

N
NDSLINK class 525
network address 188
NETWORK statements 63
NFS 217, 337
  customize the procedure 222
  customizing and starting 217
  EXPORTS security 224
  impact on HFS 225
  mvslogin command 223
  mvslogout command 223
  NFS mount command from OS/2 223
  recommendations for using NFS with UNIX System Services 226
  security exchange between client and server 224
  security levels 224
  security settings 223
NFS client 217, 438
NFS server 337
NFSCLNT 438
NOAUTOMOVE 265
NOAUTOMOVE keyword 436
NOPADCHK 156
NOTELINK class 525

O
OBEYFILE command 199
oedit command 112, 365
OHELP 371
OHELP command 372
OMVS 361
  address space 22
  command 374
  restart support 432
  segment shared keyword 120
  shutdown support 432–433
OMVSGRP 51
OMVSKERN 51, 202
online help facility 371
ONLYAT keyword 118
OPD report 455
OpenEdition 81
openssh 387

otelnetd 208
other licensed programs
  customizing and starting NFS 217
  customizing and starting Telnet and FTP daemons 208

P
PADCHK 156
PADS 156, 159
parent process 3
PARMLIB members
  BPXPRMxx 198
PassTicket 171
pax command 150
pax utility 30
permission bits 121, 135
Persistent Reusable Java Virtual Machine 335
PFS 230
physical file system 230
pipe 16
Port 187
PORTMAP 219
POSIX program 231
POSIX standard 125, 134, 136
Printer Inventory 319
proc_nam 218
process 3
PROCLIB 194
PROFILE.TCPIP 190
program access to data sets 156, 158, 161
PROGRAM class 155
PROGxx 195
protocol suite, TCP/IP 186
PS1 shell environment variable 367
pseudoterminal 209
putty 389

Q
QSAM 337
QUIESCE command 454
quiesce serialization mechanism 442
quota 233

R
RACF 196
  BPX.DAEMON facility class 212
RACF BPX.DAEMON 202
RACF group structure 90
RACF list-of-groups checking 108
RACF OMVS 204
RACF SPECIAL authority 175
RACFICE reporting tool 101
RACFICE utility 510
RAS 502
Redbooks Web site 573
  Contact us xvii
refid 372
region size 326
register command 453
registration support
  OMVS shutdown 432
remount 286
RESOLVER_CONFIG 193
RESOLVER_PROC 193
restoring HFS data sets 484
RESTRICTED attribute 132
Retrieve 363
REXX functions 393
REXX procedures
  LARGEFIL 462
REXX program bpxrexx 379
rhosts authentication 388
RLIST command 512
rlogin client support 374
rlogin daemon 201
rlogind 201
RMF 454
ROOT statement 60
Router 187
RPTSTG(ON) 331
RRGMAP 525
RRSF environment 118
runtime option
  FILETAG 429
RVARY INACTIVE command 523
RVARY LIST command 523
RVARY SWITCH 526

S
sanction lists 165
SDSF 456
sendmail
  Mail Delivery Agent (MDA) 215
  Mail Transfer Agent (MTA) 215
  Mail User Agent (MUA) 215
SERVSTU server authority 171
set command 367
seteuid() 204
setfacl command 128–129, 134, 139–140, 561
setfacl shell command 134
setgid bit 129
SETOMVS command 267
setomvs command 259
SETOMVS SYNTAXCHECK command 410
SETROPTS AUDIT 180
SETROPTS GRPLIST 108, 130
SETROPTS LOGOPTIONS 180
setuid bit 129
setuid() 204
SHARED keyword 117, 120
SHARED.IDS profile 116, 119
shell 4
shell script 371
shell variables 368
SIGDANGER signal 432
SIGKILL signal 435
SIGTERM signal 435
SMF 13

SMP/E maintenance 489
Socket 187
socket 16
Socket address 187
Socket interface 187
Software Development Kit 323
space sharing 234
SPECIAL attribute 292
ssh daemon 387
sshd 387
STAGE parameter 524
STARTED class profiles 55
started tasks
  define users 310
STEPLIB=none 78
sticky bit 21, 130, 158
storage contraints 63
SU Command 84
su -s 85
SUB=MSTR parameter 438
Subcommand function key 369
SUBFILESYSTYPE statements 65
subnet mask 188
superuser 82
  daemons 152
superuser mode 84
SUPERUSER.FILESYS.MOUNT profile 172
SUPERUSER.FILESYS.PFSCTL profile 172
SUPERUSER.FILESYS.QUIESCE profile 172
SUPERUSER.FILESYS.VREGISTER profile 173
SUPERUSER.FILESYS.VREGISTER resource 172
SUPERUSER.IPC.RMID profile 173
SUPERUSER.PROCESS.GETPSENT profile 173
SUPERUSER.PROCESS.KILL profile 173
SUPERUSER.PROCESS.PTRACE profile 173
SUPERUSER.SETPRIORITY profile 173
SURROGAT authority 85
SURROGAT class profile 171
SWA above 63
SWA(ABOVE) 63
SWA(BELOW) 63
symbolic link 19, 21, 239
SYS1.LINKLIB(INETD) 202
SYS1.LINKLIB(RLOGIND) 205
SYS1.PARMLIB
  COFVLFxx member 94
SYSIN 438
syslogd 207
SYSNAME 196
SYSOUT 439
sysplex environment
  using ACls 151
SYSPLEX(YES) 395, 437
sysplex-wide hangs 501
SYSPRINT 439
SYSTCPD 195
SYSTCPD DD 191
System heap 336
system symbols
  &SYSNAME 308
  &SYSPLEX 308

T
TAG parameter 424
tar command 151
TCP/IP 185
TCP/IP DATA 197
TCP/IP PROFILE 190
TCP/IP resolver address space 193
TCP/IP stack 396
TCPDATA 191
TCPIP.DATA 190
TCPIP.ETC.SERVICES 205
tcsh shell 10
Telnet
  client 375
  client session 375
  protocol 375
  session 375
  support 374
telnet 208
telnet server
  character-at-a-time mode 208
  line mode 208
  raw mode 208
TelnetD
  setting up 208
telnetd 208
temporary file system 440
TFS 439
time zone 312
Tivoli Storage Manager 441
Tivoli Storage Manager (TSM) 484
Transient heap 336
TSM 484
TSM client 453
TSM configuration files 452
TSO/E MOUNT command 267
TSO/E OEDIT command 112
TSOEXEC program 160
TZ value 312

U
umask 79, 123–124
umask command 123–124
undub 10
UNIX System Services 230
UNIX System Services for general users
  help facilities 371
UNIXMAP class 92
UNIXMAP profiles 512
UNIXPRIV class 82, 115, 119, 137, 232
  _POSIX_CHOWN_RESTRICTED 127
  CHOWN.UNRESTRICTED 127
  RESTRICTED.FILESYS.ACCESS 133
  SHARED.IDS 116, 119
  SUPERUSER.FILESYS 83, 130, 137
  SUPERUSER.FILESYS.ACLOVERRIDE 137
  SUPERUSER.FILESYS.CHANGEPERMS 128, 137

SUPERUSER.FILESYS.CHOWN 127
SUPERUSER.FILESYS.MOUNT 83
Unknown RefID_tcp1
BPXPRMxx parmlib member 198
connecting to UNIX System Services 199
customizing inetd and rlogin 201
UNMOUNT keyword 436
UNMOUNT option 266
USS 230

V
verbosegc 329
VFS server (virtual file system server) 231
virtual file system server (VFS server) 231
Virtual Lookaside Facility 92
VLF 92
VSAM LDS 233, 296
VSAM linear data sets 233

W
WebNFS protocol 341
Well-known 187
WHEN(PROGRAM) 156
wildcard 266
WLM 11
workload manager 11

X
XPLINK 333
XPLINK environment 335

Z
z/OS Communication Server 189
z/OS NFS server 219
z/OS UNIX ACLs 232
z/OS UNIX Configuration Wizard 190
z/OS UNIX couple data set 273
zFS
log files 236

UNIX System Services z/OS Version 1 Release 7 Implementation

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

UNIX System Services z/OS Version 1 Release 7 Implementation

z/OS UNIX overview

z/OS UNIX setup

z/OS UNIX usage

This IBM Redbook presents the information you need to plan for and run an IBM z/OS system with support for z/OS UNIX System Services (z/OS UNIX) and z/OS.e. It provides information to facilitate the installation and use of z/OS Version 1 Release 4 UNIX System Services, and step-by-step instructions on how to install, customize, and use the z/OS UNIX System Services product set.

This redbook is written for MVS systems programmers who install and customize the z/OS UNIX System Services product set.

Practical examples are presented to demonstrate the installation and customization of UNIX System Services. This includes examples of the customization of DFSMS, RACF, TCP/IP, and NFS required to set up a z/OS UNIX System Services environment.

Some knowledge of UNIX System Services is assumed.

SG24-7035-01 ISBN 073849609X