AMSE JOURNALS-AMSE IIETA publication-2017-Series: Advances B; Vol. 60; N°3; pp 613-629
Submitted May 23, 2017; Revised Dec. 26, 2017; Accepted Jan. 08, 2018

A Comparative Study of Technologies Developed in Perspective of Distributed Operating Systems

*Ahmed Bin Shafaat, **Shuxiang Xu

*Department of Information Technology, University of the Punjab, Jhelum Campus, Jhelum, Pakistan (bcs.f12.04@gmail.com)
**School of Engineering and ICT, Faculty of Science, Engineering and Technology, University of Tasmania, Australia (Shuxiang.Xu@utas.edu.au)

ABSTRACT

The collection of processors that do not share memory and clock is called distributed operating system. It is an infrastructure which is programmed so that it allows the use of multiple workstations as a single integrated system. In distributed OS users can access remote resources as they access local ones. The advantages of distributed operating systems are performance through parallelism, reliability and availability through replication and in addition scalability, expansion and flexibility of resources. This research paper includes a detailed and comparative introduction of different state-of-art technologies and operating systems, their internal structure, design and key features which are developed in perspective of distributed operating systems. These operating systems includes Chorus, V-systems, Amoeba and Mach.

Key words

Distributed operating systems, Chorus, Mach, Amoeba, V-systems.

1. Introduction

A gathering of free PCs that appears to its clients as a solitary framework or a solitary coherent framework is called a distributed system [1]. One can differentiate between a distributed system and a parallel or a network system. A collection of machines that are just capable of communicating to each other is called a network. In a parallel system, multiple machines can also be visible. But distributed systems can share information and resources among scattered users.

Distributed systems can be of both types, wide area and local. But the thing which makes them a distributed system as that software which makes the whole collection act like a single system [2]. Distributed operating systems do not share clock and memory. So a shared memory multiprocessor system is not a distributed operating system [3]. Now we will discuss technologies and operating systems developed in perspective of distributed operating systems.

2. Chorus

Chorus was a distributed system research project in France at INRIA in the time period between 1979 and 1986. There were three following versions of Chorus introduced named as Chorus VO, ChorusV1, and ChorusV2. The fundamental point for taking care of appropriated registering inside of the Chorus, for framework and applications too, is the administration of trade of "Messages" between "Ports" appended to "Performers" by a "Core". Dissimilar to past variants, Chorus-V2 was perfect with UNIX2 System. Also, it had been utilized for supporting about six examination dispersed applications as the premise. The present adaptation Chorus-V3 coordinates numerous ideas from best in class conveyed frameworks. The innovation of ChorusV3 has been utilized to execute an appropriated UNIX framework as an arrangement of servers utilizing the nonspecific administrations that were given by the Chorus Nucleus.

2.1 The Chorus Architecture

Chorus system is made out of a little Nucleus and various System Servers. These servers coordinate in the setting of Subsystems which provides a single set of interfaces and services to their “users”. The overall organization of a Chorus system is a logical presentation of an open operating system. Unlike UNIX, Chorus Nucleus facilitates the operating system by providing generic tools developed for supporting a mixed bag of host subsystems, which can exist together on the highest point of the Nucleus.

2.2 The Chorus Nucleus

The Chorus Nucleus deals with the nearby and physical figuring and processing assets of a "PC" which is known as a Site with the assistance of three parts which are obviously characterized:

- **Virtual Memory Manager** deals with the neighbourhood memory which is finished by organizing virtual memory location spaces and controlling memory space designation.

•**Real-time multi-tasking Executive** controls the portion of neighbourhood processor(s). This executive also provides the priority-based pre-emptive scheduling and fine grain synchronization.

•The communication services and message delivery regardless of their destination area inside of a Chorus system is provided by the **IPC Manager**.

The structure of Chorus Nucleus is logical and distribution is almost hidden within it. The only tool for communication between different site's managers is "standard" Chorus IPC. Site managers use Chorus IPC rather than the dedicated protocols. The nucleus is highly portable, even sometimes this design may prevent it by using specific features of the underlying hardware [4].

2.3 The Subsystems

Implementation of high-level system services is done by system servers. These system servers also cooperate to provide a single coherent interface of the operating system.

2.4 System Interfaces

There are multiple levels within a Chorus System.

•**Nucleus Interface:** A set of procedures provides access to the administrations of the Chorus Nucleus. The Nucleus interface is made out of this arrangement of principles. The Nucleus communicates with a distant Nucleus via the Chorus IPC if it cannot change the service directly.

•**Subsystem Interface:** Subsystem interface is composed of a set of procedures that access some Subsystem-specific protected data and Nucleus interface. If a service can't be changed directly from this information, these procedures "call" (RPC) the services which are provided by System Servers.

2.5 Physical Support: Sites

The Chorus system's physical support is made up of a group of sites ("boards" or "machines"), which are interconnected with the help of a BUS (or communication network). The tightly interconnected collection of one or more processors appended I/O gadgets and focal memory is known as a **Site**. System administrators also called site servers control and supervise the Site management.

2.6 Virtual Machines: Actors

The logical “unit of the collection of resources” and "unit of distribution" of a Chorus system is called **Actor**. Every onscreen character's "system" location space is same and it is only accessible by special levels of execution on every given site. A site can support more than one actors at a time. Because every site has its own protected "virtual machines" defined by actors to the user and "user" address space. Each on-screen character is attached to a site and the strings that are supported by any given actor are always executed on the site to which that actor is tied. The memory which is utilized by the information and code of a string is dependable of the performing artist's site. Performers can't relocate starting with one site then onto the next. Following the set of resources are encapsulated by an actor:

- A virtual memory setting which is isolated into Regions and is combined with far off or nearby section.
- A correspondence context which is made up of a set of sections.
- An execution connection which is made out of an arrangement of string4. More details about Paper title and Author information

2.7 Processes: Threads

A thread is also called "unit of execution" in the Chorus.

A string in Chorus framework can be characterized as the consecutive stream of control and is distinguished by string setting which compares to the condition of the processor. A string is attached to one and only performing artist, which characterizes the address space that can be operated by that thread. In one actor, multiple threads can be formed and can run in parallel. Resources of actors can be shared by threads but only those actors which are related to that thread. Multi-programming servers are allowed by threads, which give a good match to "client-server" style of programming. An intense instrument for programming I/O drivers is provided by threads. Threads synchronize and communicate by exchanging messages by using the Chorus IPC even if the threads are situated on the same site. However, because the same address space is shared by an actor’s threads, the mechanism of synchronization and communication is based on shared memory can be used inside one actor [5].

Chorus system has a goal which includes user level servers, encapsulated resource management and support of network transparency. In the chorus, kernel manages some objects and others objects are managed by user-level servers. In Chorus, user-level servers can be stacked rapidly in the bit location space. Chorus provides separate abstractions of threads and processes.

The chorus can enjoy the benefit of a multiprocessor. The kernel of Chorus offers more calls because of the desire to emulate UNIX and more complex communication models.

2.8 Naming and Protecting Resources

Chorus capabilities can name and protect its resources. Assets distinguished by capacities in Chorus can be gotten to by making an impression on the proper server port. What's more, the server can get to the specific asset named in the capacity.

The Chorus piece gives insurance identifiers to empowering client level administrations that are used for authentication of the on-screen character that communicated something specific and the port utilized by that performing artist. The capacities of Chorus holds on past the execution of any procedure that employments Chorus capabilities. Groups of servers in Chorus system are allowed to manage resources. In Chorus processes can become group member by making procedures ports join port gatherings.

2.9 Inter-process Communication

Chorus kernel provides asynchronous message passing and synchronous request-reply protocol as well. It also provides alternative calls and group communication. An unreliable multicast is provided in Chorus to all group members of ports or to simply choose individuals. Ensemble servers can reconfigure administrations by including or expelling ports from gathering.

2.10 Messages

Chorus employ its techniques of virtual memory management to the large message passing between processes in the same computer. Messages in Chorus are simply back to back groupings of bytes.

2.11 Network Communication

In Chorus, correspondence between distinctive procedures in diverse PCs is performed by client level system servers that keep running on each PC. The system servers are additionally in charge of finding ports. Chorus system first uses location hints for finding the locations of ports. But if that location hint fails then they fall back on TV. For a mixed bag of conventions, client level system servers ought to be allowed to use. Chorus is stuck to International standards, which provide OSI and Internet protocols. However on LAN when request-reply interactions predominate, these standards are not necessarily suitable.

2.12 Memory Management and UNIX Emulation

Chorus system uses local caches and external pages which allow sharing of virtual memory between procedures, regardless of the possibility that they keep running in diverse PCs. Chorus system desires to emulate with UNIX as subsystems [6].

3. Amoeba

Amoeba, which is a more powerful distributed operating system based on micro kernel has been in use in research, industry and academics for some time. Amoeba was introduced by Prof. Andrew S. Tanenbaum and his group members after years of research at Vrije University [7]. Amoeba can convert a group of workstations into a transparent distributed system. In the design of Amoeba, transparency is a key issue. Usually users of Amoeba do not know the location and number of file servers that store their files or location and numbers of processors that process their commands. So Amoeba is a combination of networked machines that can be potentially expanded to many countries. Amoeba is being used on industrial level for more than ten years. Amoeba has been implemented on Sun's workstations that are named as Sun 3/50 and 3/60 workstation, Micro SPARC SPARCstation, Intel 386/486/Pentium and on many other platforms. Amoeba is free for educators and researchers. Government, corporate and other users can avail it on special commercial prices.

3.1 System Architecture

Amoeba can be run on a single microcomputer but a configuration for running Amoeba is recommended by its software developers. An Amoeba system should consist of following four fully functioned classes of machines:

i. Workstations
ii. Specialized Servers
iii. Gateway
iv. Processor Pool

Users use workstations for accessing the system. Workstations have limited power of processing. Workstations do not perform heavy duty processing. The heavy processing is done by set of pool processors. These processors are assigned to user dynamically. Several megabytes of private memory are occupied by these processors. So they do not require any common memory yet it is not prohibited.

3.2 Design Issues

The authors of Amoeba have four major goals in their minds while developing this operating system:

i. Parallelism

ii. Distribution

iii. Performance

iv. Transparency

Tanenbaum and members of his research group wanted to associate an immense number of PCs over a disseminated system in a straightforward manner by creating a distributed operating system. They want their operating system to have the capability of parallel computation. As we are concerned to performance, developers of amoeba kept things as simple as they can while maximizing performance. Main design issues of Amoeba are presented below:

3.2.1 Communication primitives

In Amoeba, communication between servers and clients is done by using a remote procedure call model. A request is send to the server by the client. And client blocks that demand until it gets an answer from server. While the server forms the solicitation and sends an answer to the customer with the result of operation. REQUEST and REPLY are two types of messaged involved in communication in Amoeba. In RPC mechanism, the client has to indicate the item with the solicitation on which operation must be performed, sort of operation and related parameters moreover. In Amoeba every remote strategy call includes the unmarshalling and marshaling of parameters. For taking care of this issue a dialect Amoeba Interface Language (AIL) was composed. The FLIP system convention is utilized for imparting inside the Amoeba framework to expand the execution [8].

3.2.2 Naming and protection

Naming and security plan is subject to two ideas: capacities and items. Capacity is the handle on object provided in Amoeba. It is just a number that a server generates when it gets a request from client to create an object. The client has to use that capability to handle that specific object for all future operations after its creation. On the other hand, an article is a dynamic information sort on which some particular operations are characterized. Amoeba supports software objects on primary basis but hardware objects also exist.

3.2.3 Resource management

Multiple pool processors or dedicated servers apply many different techniques for managing the resources in an Amoeba system. There is a resource manager for every machine in the system, a process that controls the resources and keeps track. There is a Process Server that keeps track of which processors are not in the process pool and which processors are free. It is the job of that

process server to allocate processor(s) for a specific task. The processor allocation for any particular task is kept transparent to the user for the sake of transparency. For handling allocation of shared resources, Amoeba uses a special server which is called Bank Server. It provides a mechanism for managing resources by using accounting techniques.

3.2.4 Fault Tolerance

In Amoeba system, fault tolerance is handled by boot services. All registered servers are polled by boot services. Server is declared broken if it does not respond on time. Then boot server sends request to process server to initialize new server copy. In RPC mechanism if client does not receive reply from server then it resends the request [9].

3.3 Software Outside the Kernel of Amoeba

The real occupation of the microkernel of Amoeba is to bolster memory administration, RPC, strings and I/O.

3.3.1 Directory server

Not at all like other working frameworks, are document naming and record administration isolated in Amoeba working framework. Shot server don't oversee document naming rather it simply handle records. Projectile server just peruses and composes the documents as characterized by abilities. A catalog server composes ASCII strings on abilities. Registries comprises of (ASCII string. ability) matches: these capacities are utilized for registries, documents and diverse different articles. An index passage may have an arrangement of capacities or only a solitary ability, which are utilized for mapping a document name onto an arrangement of reproduced records. For administration of duplicated documents, there are diverse operations gave in Amoeba.

3.3.2 Bullet file server

Shot document server is the standard record server intended for superior in Amoeba. It continuously stores records on circle, and gets complete documents sequentially in center. For establishment of Bullet Server, one needs a devoted machine with least of 16 MB RAM. RAM can be expanded for better execution. By measuring the physical memory which is available for Bullet Server, it confines the greatest size of record.

3.3.3 Utilities

There is a large number of utilities provided in Amoeba. These utilities are modeled after programs that are used in UNIX. These utilities include *basename, cal, chmod, awk, cdiff, cmp, cat, compress, echo, comm, dd, cpdir, diff, cp, sleep, pwd, mv, more, printenv, size, shar, prep, od, mkdir, quote, rm, sed, sh, rmdir, pr, rev, wc, tr, treecmp, strings, split, touch, tset, uniq, uue,*

uud, vi, true, sum, test, termcap, tty, tail, time, tsort, spell, tar, tee, and many others. In addition, new projects are additionally given, for example, a make which is an exceptionally parallel design director.

3.3.4 Parallel programming

A new language is developed which is called ORCA. Programmers can make client characterized information sorts which handle on diverse machines in ORCA. These information sorts can be partaken in a controlled manner and an article based circulated shared memory can be recreated over a LAN. Amoeba IPC facilities are used by Orca run-time system for making sharing of objects of programming over the system profoundly productive. One can avail ORCA from Vrije University.

3.3.5 Compilers

Amoeba operating system has compilers for ANSI standard C, Fortran 77, Modula 2, Pascal and BASIC. Each of these compilers has suitable libraries. One-celled critter too has third-party software collection which also includes the GNU C compiler.

3.3.6 TCP/IP

The communication in Amoeba is done by essential correspondence instrument which is called Amoeba FLIP convention. Another exceptional server is likewise given in Amoeba which permits TCP/IP correspondence, RPCs to the TCP/IP server. So one can get to machine through web using this mechanism.

3.3.7 UNIX emulation

To run UNIX programs in Amoeba environment, Ajax which is an imitating library offers Major POSIX P1003.1 similarity. A considerable lot of POSIX conformant projects keep running without changing them. Single adaptable cell client simply need to aggregate and connection them to Amoeba.

3.3.8 Connection to UNIX

Amoeba has an extraordinary UNIX driver which is able to be connected with Sun OS 4.1.1 (or more advance version) UNIX part. It permits UNIX projects to speak with projects of Amoeba framework. Utilization of TCP/IP is likewise feasible for this correspondence (e.g., for non-Sun machines). Be that as it may, if Sun workstations can be accessible then the element portrayed above can be much quicker, strong and less perplexing. There are utilities accessible in Amoeba framework to exchange documents in the middle of UNIX and Bullet File Server.

3.3.9 X Windows

User interface in Amoeba operating system is the X Window System (X11R6) which is considered to be industry standard system. A unique variant of X is accessible for X servers that

keep running on workstations. Amoeba RPC is being used by these workstations for superior correspondence. The time when hard-wired X terminals are being used as a part of Amoeba, these terminals are able to be interacted by utilizing the TCP/IP server.

3.4 Pricing

Single adaptable cell is free for colleges that have WWW or FTP access. Also, for those colleges that don't have FTP or WWW access, it is accessible for $US 500 on Exabyte or DAT tape [10].

4. Mach

Mach is a multiprocessor working framework part which gives ability based between procedure correspondences. Mach provides following mentioned facilities:

- Protected object capabilities
- Uniform object based reference mechanism
- Efficient cross-domain object operation invocations
- Efficient cross-domain object communication

Mach do not provide pure object oriented environment and in Mach not every bit of information is thought to be an item. Ports are just genuine items in Mach world. Right now scientists are utilizing Mach at CMU as well as it is being used to keep running on machines that range from individual workstations to multiprocessors [11].

4.1 Mach: An Extensible Object-oriented Kernel

Key features of Mach design are described below:

- In Mach, all framework administrations are being given through ability based between procedure correspondence instrument
- Kernel objects in Mach system are referenced through object capabilities
- Support for protection and network transparency is being provided by underlying mechanism for communication
- For an extensive variety of approximately coupled and firmly coupled multiprocessors support for parallelism is permitted in Mach.

Mach likewise furnishes similarity with existing situations at CMU. Mach is source good with existing Accent code and it is double perfect with Berkeley 4.3 bsd. Mach was concocted in 1985. It was thought to be an Accent-like working framework which can furnish complete similarity with UNIX.

4.2 Kernel Abstractions and Operations on Objects

The bit of Mach framework underpins five essential reflections:

i. Typed gathering of information articles which can be utilized as a part of correspondence between strings is called Message. Messages can be of any size and may comprises of pointers and wrote capacities for ports. All message information other than ports is gone by quality.

ii. A port can be characterized as a channel for correspondence. It is a line for messages that are ensured by the portion. Ports are being utilized as abilities that are ensured for all items inside of the Mach environment. Ports are characterized as the reference objects of the Mach plan.

iii. A string is alluded as the fundamental unit of CPU use. It is more or less identical to a free program counter which works in an undertaking. All strings working inside of an undertaking offer access to all errand assets.

iv. The unit of virtual memory, in Mach is known as a memory object. It can be mapped into the location space of an errand.

v. An execution environment in which strings can be run is known as an undertaking. Assignment is the essential unit of asset portion. It comprises of secured access to framework assets, (for example, port abilities, virtual memory and processors) and a paged virtual location space. In Mach, an errand with a solitary string of control is utilized to speak to UNIX thought of a procedure.

4.3 Extending Object Primitives to a Network Environment

The majority of sharing of message correspondence, in Mach, happens between items that are being on the same machine. In spite of the aforementioned point, messages can be sent to procedures utilizing remote machines as a part of the Mach environment. A straightforward middle person procedure can be embedded between a couples of conveying procedures. That middle person procedure forward messages between the procedures. The message servers sends messages for ports that are remote straightforwardly. For every remote port which is known on the neighborhood machine, the nearby message server keeps a nearby port, which is utilized by regional standards for the remote port as a surrogate. At the point when a message server gets a message on a port, that message is epitomized into a representation of system message, and after that it is transmitted to the remote message server. The remote server then reconstitutes the message and sends it to its destination in the interest of the first machine [12].

4.4 Memory Management

In Mach, for the cache of memory objects, physical memory is being used. File server manages some common files which are called memory objects. But an object can also implement a memory object. These objects can be used to handle requests for reading and writing the data. All the pages that are in physical memory cache, they are tracked by the Mach kernel. And kernel also allows tasks that are present in the physical memory to use those pages which are in physical memory. For giving out memory object ports, a protocol is defined by the memory manager. So we can say that a general service port is being registered by memory manager somewhere where client can find it. Then the client can export an object lookup or an object create call which can return a memory object port. The virtual address space of any task can be represented with the help of a configured collection of mapping from consecutive virtual address ranges to these memory objects. A memory object can be also defined as secondary storage object which can be mapped into the virtual memory of a task. When the cache of physical pages is full, the Mack kernel must have to first extract and replace pages from cache afterwards. This can be done by writing the material of edited pages to the relevant memory objects. Currently the users of Mach system are allowed to create memory objects which can be managed by user-defined processes. This feature is allowed by Mach virtual memory manager. A process that can provide data in action to page fault and back storage to clean the page requests is called an external pager. Mach comes with the feature of multiprocessor scheduling. And now a days, this feature is used on both uniprocessor and multiprocessor systems. To increase the system performance, Mach kernel may be allowed by memory manager to maintain its cache for a memory object, when all references of virtual address space to this object are gone by assertion of caching parameter to call the memory_object_set_attribute. But to allow the process of caching cannot stop the kernel from destroying that object. Size of virtual page, in Mach, can be altered and can be set on per machine basis. Conversion of the scheduler of size of a single page into scheduler of size of multiple pages is very quick in Mach system. An ordinary size of page is used by scheduler of multiple size of pages for the solution of problems in two ways:

- That requests whose size is less than size of scheduler page, the size of request is rounded to be equivalent to size of scheduler page.
- That requests whose size is greater than size of scheduler page, the request is completed after making suitable similarity with the help of scheduler of multiple pages.

4.5 Transparent Shared libraries

Mach system comes with a feature which is called transparent shared library. It is actually a code library and this library is located in a program’s address space without the knowing of that

program. Now, when that program will make a system call, this library will intercept it. To make Mach system live and work successfully with non-Mach operating systems environment is the main and basic goal of these libraries. Other goals of this feature includes:

- Network redirection of traps of operating systems
- Monitoring
- Debugging
- Supporting OS environments like UNIX V.4 and UNIX 4.3 and many others [13].

5. V-distributed System

The V-system was developed at Stanford University in 1980s. In fact it was a part of a research project which was being run at Stanford University. The primary group leader was Professor David Cheriton. Verax and Thoth were predecessors of V System. These two operating systems were also developed by Prof. David Cheriton. The purpose of that project was to find the issues and problems in distributed operating systems. V distributed System is almost forgotten but it left its footprints on the sands of time. V was a complete self-hosting distributed system. It is designed for a collection of computers that are connected with a network of larger performance. This system was developed like a collection of collection of service modules, collection of commands, comparably small distributed kernel and many run time libraries. The presence of interconnection of networks and many machines is transparent at process level. Synchronized message passing and multithreading are basic concepts in V System. The different value-added services are implemented by service modules by accessing hardware resources. The V-System kernel provides that resources. The idea of V-System was derived by the functionality of comparably high-performance and low cost computer systems and networks. An abstraction of inter-process communication, address space and lightweight processes is provided by V-System kernel. These features are similar to those facilities which are given by a hardware backplane.

5.1 Key features of V Distributed System

i. The approach of small kernel is suitable, In V System for the implementation of services and protocols in a modern way. Services are given in runtime libraries in V system.

ii. Software do not define the operating system. It is the Protocol which describes and defines the system. The most important point is the capability of supporting the groups in the protocols.

iii. In V System, I/O mechanism is developed in a way so that a standard structure can be enforced on messages which travel between different devices.

iv. For the identification and detection of different objects, V System comes with object identifiers. These objects may include address space, directories and files. Objects can be found by clients by applying caching and multi-casting techniques. There is a name server which is used to record the services so that nonexistence can be differentiated by failure.

v. In different kernel modules, features like process, time, communication, device management and memory are provided by V System kernel. Each of these features are connected to V System IPC facilities. All the services can be accessed in the same way regardless of their remoteness or locality. Fine tuning for separate and individual components is provided by modularity.

vi. The consistency and amount of caching for the files and device management and memory is implemented by V System kernel. Implementation of these features is necessary for the protection of integrity of kernel. Scheduling of processes is done outside the kernel by manipulating priorities.

vii. In distributed operating systems, high performance communication is the key factor. Distributed systems work to structure and optimize the kernel for efficient communication. In V System, optimization is used to lower the multi-cast cost. Which was necessary to handle the communication between different groups. It was also necessary to handle the shared state [14].

6. A Comparison of Chorus, Amoeba and Mach

6.1 Chorus Operating System

- Microkernel Based RTOS
- Flexible Virtual Memory Implementation
- Binary Level OS Emulation
- A. Sync. Communication
- Page Based DSM
- Optimized for Local Case

➢ Chorus Chief Design Features

- Dynamically Loadable Servers
- Enhancement of Unix
- Server Group & Reconfiguration
- Distributed Memory Multiprocessor Operations
- Real-time Operations

6.2 Amoeba Operating System

- Times Sharing DOS
- Based on Microkernel
- Execution Model: Pool Processor
- Automatic Load Balancing
- Automatic File Replication
- Object Based DSM Used
- Main Objectives:
  - Distribution, Parallelism, Transparency, Performance

- *Key Concepts of Amoeba*

- Microkernel
- Remote Procedure Calls (RPC)
- Threads
- FLIP
- Objects
- Capability
- Various Servers

6.3 Mach Operating System

- Designed for 1 CUP/Multiprocessor
- Extensive Multiprocessor Support
- Maximum No. of Kernel Calls: 153
- Memory Mapped Objects
- Integrated Memory Mgmt.
- Page Based DSM
- No Group Communication
  - MACH Principal Abstractions
- Processes
- Threads
- Memory Objects
- Ports
- Messages

7. Conclusion

In this paper we have presented a survey of different technologies developed in perspective of distributed operating systems which includes Amoeba, Chorus, Mach and V System. We have discussed internal structure, kernel design and other key features of each of above mentioned operating systems. In future, any researcher can also include Locus and X-kernel and other state-of-art technologies developed in perspective of distributed operating systems which are being currently used.

Table 1. Tabular Comparison of Different DOS [15]

ITEM	AMOEBA	MACH	CHORUS
Designed for:	Distributed system	1 CPU, multiprocessor	1 CPU, multiprocessor
Execution model	Pool processor	Workstation	Workstation
Microkernel?	Yes	Yes	Yes
Number of kernel calls	30	153	112
Automatic load balancing?	Yes	No	No
Capabilities	General	Only ports	General
Capabilities in:	User space	Kernel	User space
Threads managed by:	Kernel	Kernel	Kernel
Transparent heterogenity?	Yes	No	No
User-settable priorities?	No	Yes	Yes
Multiprocessor support	Minimal	Extensive	Moderate
Mapped object	Segment	Memory object	Segment
Demand paging?	No	Yes	Yes
Copy on write?	No	Yes	Yes
External pagers?	No	Yes	Yes
Distributed shared memory	Object based	Page based	Page based
RPC?	Yes	Yes	Yes
Group communication	Reliable, ordered	None	Unreliable
Asynchronous communication?	No	Yes	Yes
Intermachine messages	Kernel	User space/kernel	Kernel
Messages address to:	Process	Port	Port
UNIX emulation	Source	Binary	Binary
UNIX compatibility	POSIX (partial)	BSD	System V
Single-server UNIX?	No	Yes	No
Multiserver UNIX?	Yes	No	Yes
Optimized for:	Remote case	Local case	Local case
Automatic file replication?	Yes	No	No

References

1. A.S. Tanenbaum, M.V. Steen, Distributed systems: Principles and paradigms, 2007, Prentice-Hall.

2. D. Partha, R.J. LeBlanc, W.F. Appelbe, The clouds distributed operating system: Functional description, implementation details and related work, 1988, In Distributed Computing Systems, 8th International Conference on, pp. 2-9.
3. A.S. Tanenbaum, Distributed operating systems, 1996, Cern European Organization for Nuclear Research. Egmond aan Zee, The Netherlands.
4. R. Marc, Chorus distributed operating system, 1988, Computing Systems Journal.
5. R. Marc, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann et al., Overview of the chorus distributed operating systems, 1991, In Computing Systems.
6. G. Coulouris, J. Dollimore, T. Kindberg, A comparison of mach, amoeba and chorus, distributed systems: concepts and design, 2nd ed. 1994.
7. A.S. Tanenbaum, R.V. Renesse, H.V. Staveren, G.J. Sharp, S.J. Mullender, Experiences with the Amoeba distributed operating system, 1990, Communications of the ACM, vol. 33, no. 12, pp. 46-63.
8. M.F. Kaashoek, A.S. Tanenbaum, Group communication in the Amoeba distributed operating system, 1991, In Distributed Computing Systems, 11th International Conference on, pp. 222- 230.
9. A. Qasem, An Introduction to the amoeba distributed operating system, Computer Science Department, Florida State University.
10. A.S. Tanenbaum, G.J. Sharp, The amoeba distributed operating system, 1996, Cern European Organization for Nuclear Research-Reports-Cern, pp. 109-122.
11. M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, M. Young, Mach: A new kernel foundation for UNIX development, 1986, pp. 93-112.
12. M.B. Jones, R.F. Rashid, Mach and matchmaker: Kernel and language support for object-oriented distributed systems, Technical Report CMU-CS-87-150, 1986, Pittsburgh, Pennsylvania,.
13. Viswanath Veerappan, Mach micro kernel–A case study, University of Texas, Arlington
14. D.R. Cheriton, The V distributed system, 1988, Communications of the ACM, no. 3.
15. Er. Shiva K. Shrestha, Comparison of Amoeba, Mach & Chorus: DOS", ME Computer Nepal College of Information Technology.