Unix et les communications

vincent.roca@lip6.fr - http://www-rp.lip6.fr/~roca

version 1.5b, mai 1999

COPYRIGHT

- **Copyright © 1999 Vincent Roca; all rights reserved**
- **Ce document est copyrighté et n 'est pas dans le domaine public. Sa reproduction est cependant autorisée à condition de respecter les conditions suivantes:**
  - Si ce document est reproduit pour les besoins personnels du reproducteur, toute forme de reproduction (totale ou partielle) est autorisée.
  - Si ce document est reproduit dans le but d 'être distribué à des tierces personnes il devra être reproduit dans son intégralité sans aucune modification. Cette notice de copyright devra donc être présente. De plus il ne devra pas être vendu. Cependant, dans le seul cas d 'un enseignement gratuit, une participation aux frais de reproduction pourra être demandée, mails elle ne pourra être supérieure au prix du papier et de l 'encre composant le document.
- **Toute reproduction sortant du cadre précisé ci-dessus est interdite sans accord préalable écrit de l 'auteur.**

Plan du cours

- **Chapitre 1 : la structure interne d'Unix**
  - Comment ça marche, et comment programmer...
- **Chapitre 2 : les communications locales**
  - Comment faire communiquer les différents processus d'une machine...
    - **Les signaux**
    - **Les tubes ordinaires et nommés**
    - **les mécanismes d'IPC : mémoire partagée, sémaphores, et files de messages**
- **Chapitre 3 : les communications inter-machines**
  - Comment faire communiquer des processus appartenant à des machines distinctes...
    - **Les sockets**
    - **Il reste bien d'autres choses encore : TLI/XTI, RPC, CORBA, etc.**

Pour en savoir plus...

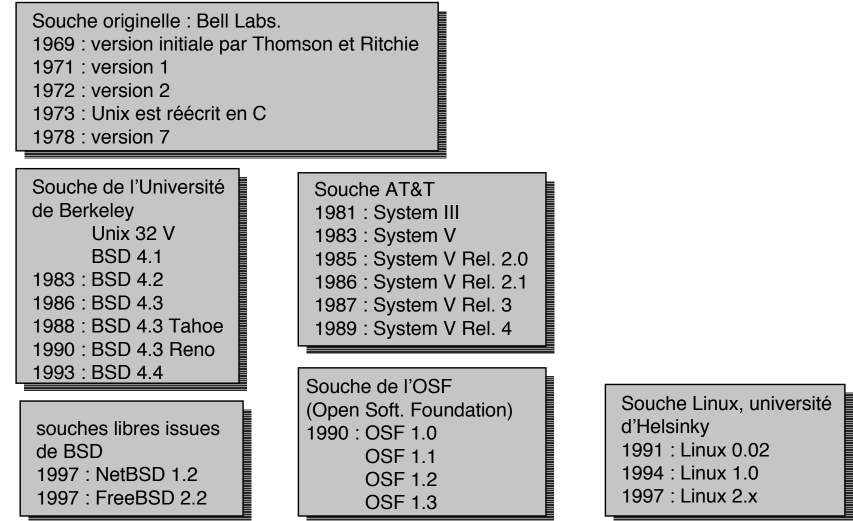
- **La référence en la matière (en anglais)...**
  - W. Richard Stevens, «UNIX network programming», Prentice-Hall, 1990.
- **Un autre ouvrage assez complémentaire (toujours en anglais)...**
  - W. Richard Stevens, «Advanced programming in the UNIX environment», Prentice-Hall, 1992.
- **Enfin un très bon ouvrage en français...**
  - Jean-Marie Rifflet, «La communication sous UNIX : applications réparties», Ediscience international, 1996.

Chapitre 1 : la structure interne d'Unix

- **1.1 Historique**
- **1.2 Les différents composants**
  - Noyau
  - Processus
  - Mode utilisateur, mode noyau
  - Fichiers
  - Fichiers spéciaux
  - Drivers
- **1.3 Programmation**
  - Passage de paramètres
  - Gestion des entrées/sorties standards
  - Les deux bibliothèques de manipulation de fichiers
  - Gestion des processus

copyright © 1999 vincent roca; all rights reserved

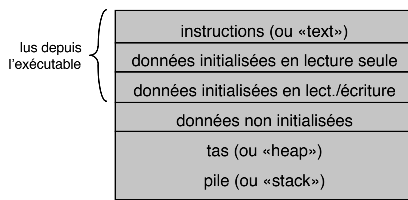
1.1- Historique



copyright © 1999 vincent roca; all rights reserved

1.2- Les différents composants

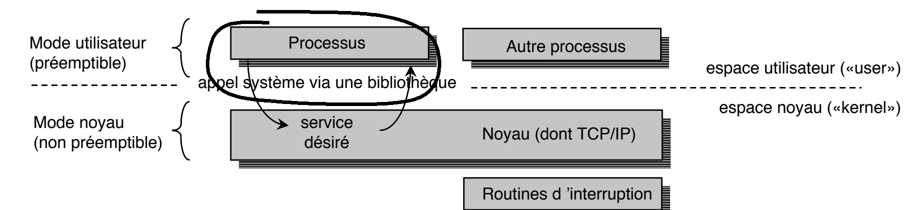
- **Le noyau (kernel)**
  - **cœur du système**
  - **sur les OS traditionnels (BSD) il contient tous les drivers nécessaires**
    ➩ *édition de liens générale*
  - **sur les nouveaux systèmes (AIX, Solaris, bientôt Linux), il contient seulement les drivers indispensables ➩ extensions noyau ajoutées à la demande**
    ➩ *édition de liens dynamique*
- **Processus**
  - **environnement d'exécution complet**
  - **modèle d'exécution :**



copyright © 1999 vincent roca; all rights reserved

Les différents composants... suite

- **Mode utilisateur, mode noyau**
  - **un proccessus est dans un mode ou l'autre (exclusifs)**
  - **basculement lors de l'appel/retour d'un appel système**
  - ***préemptions* (➩ pseudo-parallélisme) en mode utilisateur**
  - ***pas de préemptions* en mode noyau. S'exécute jusqu'à :**
    - retour en mode utilisateur, ou
    - libération volontaire (sleep), ou
    - interruption par une IT non masquée



copyright © 1999 vincent roca; all rights reserved

Les différents composants... suite

- **Les fichiers**
  - **tous les éléments sont vus comme des fichiers :**
    - clavier/écran
      stdin/0 ➩ clavier
      stdout/1 ➩ sortie standard, par défaut l'écran
      stderr/2 ➩ sortie erreur, par défaut l'écran
    - fichiers habituels
    - périphériques matériels (ou «*device*»)
      (liaison série, disque, streamer, etc.)
    - périphériques logiques (ou «*pseudo-device*»)
      (protocoles de communication, etc.)
    - *certains mécanismes de communication* (cf. chap. 2)

copyright © 1999 vincent roca; all rights reserved

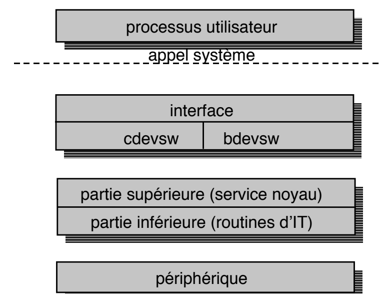
Les différents composants... suite

- **Les fichiers spéciaux : "devices" ou "pseudo-devices"**
  - **permettent l'accès aux périphériques matériels ou logiques**
  - **sous /dev**
  - **l'accès peut se faire :**
    - en mode *caractères* ➩ prise en compte immédiate
      exemple : terminaux, réseau, mémoire virtuelle du noyau (/dev/kmem)
    - en mode *blocs* ➩ prise en compte différée
      exemple : disque, bandes magnétiques
  - **on leur associe :**
    - un numéro de majeur ➩ permet de connaître le point d'entrée dans le driver
    - un numéro de mineur ➩ information supplémentaire passée au driver

copyright © 1999 vincent roca; all rights reserved

Les différents composants... suite

- **Drivers**
  - **interface logicielle entre le noyau et le périphérique**
  - **organisation :**



  - **l'ajout d'un driver nécessite l'ajout d'une entrée dans cdevsw ou bdevsw, puis une édition de liens (dynamique ou statique)**

copyright © 1999 vincent roca; all rights reserved

1.3- Programmation
1.3.1 Passage de paramètres

- **Paramètres passés à main()**
  - prototype :
    `int main (int argc, char *argv[],char *envp[]);`
  - rôle de argc/argv[] :
    - on tape : `commande -c 2 -v`
    - argc vaut 4,
      argv[0] vaut «commande», argv[1] vaut «-c»,
      argv[2] vaut «2» (la chaîne), et argv[3] vaut «-v»
  - rôle de envp[] :
    - permet d'accéder aux variables d'environnement
    - le dernier pointeur de envp[] est nul (pas de compteur)
- **Autres techniques d'accès à l'environnement**
  - variable externe :
    - `extern char *environ;`
  - Fonction
    - `char *getenv (char *nom_variable);`

copyright © 1999 vincent roca; all rights reserved

1.3.2- Manipulation des E/S standards

- **Au niveau du shell...**
  - **redirections possibles au niveau du shell**
  - **exemple (sh) :**

```
ls -lR | more
make > fichier_log 2>&1
make 2>&1 >> fichier_log
```

- **Au sein d'un programme...**
  - **ouverts par défaut dans chaque processus**
  - **se manipule comme un fichier standard par les bibliothèques ANSI et Unix**
  - **... on verra des techniques de redirection (fonction dup, notion de tube)**

copyright © 1999 vincent roca; all rights reserved

1.3.3- Les deux bibliothèques de manipulation de fichiers

- **Bibliothèque ANSI**
  - fichiers identifiés par un pointeur :
    `FILE *fp;`
  - générique, aucune hypothèse sur l'OS
  - reste de haut niveau
  - normalisé
- **Bibliothèque Unix**
  - fichiers identifiés par un descripteur :
    `int fd;`
  - spécifique à Unix
  - permet le contrôle des caractéristiques des fichiers (droits, etc.)
  - fonctions de «bas» niveau, pas de formatage de chaînes à la printf
  - indispensable pour accéder aux moyens de communication (tubes, Sockets, etc.)
  - standard de fait

copyright © 1999 vincent roca; all rights reserved

Les deux bibliothèques... suite

Bibliothèque ANSI	Bibliothèque Unix
fopen, fdopen, freopen	open
fclose	creat
fseek	close
getc, fgetc, getchar	dup, dup2
putc, fputc, putchar	lseek
fgets, gets	read
fputs, puts	write
fscanf, scanf	fcntl, ioctl
fprintf, printf	link, unlink
fread	stat, fstat
fwrite	access
fflush	chmod, fchmod
	chown, fchown

copyright © 1999 vincent roca; all rights reserved

Les deux bibliothèques... suite

- **Passage entre les deux bibliothèques**
  - Problème : fichier ouvert par `open()`, et on veut utiliser `fprintf()`
  - Solution : on utilise

```
FILE   *fdopen (int fd, char *type);
```

- **Manipulation d'enregistrements ou de chaînes**
  - Fonctions ANSI :

```
char   *fgets (char *buf, int lg_buf, FILE *fp);
int     fputs (char *buf, FILE *fp);
```

  (`gets`/`puts` pour la version E/S standard)
  - Permettent de structurer les données sous forme d'enregistrements
  - Utile pour les communications en mode «flux d'octets»

copyright © 1999 vincent roca; all rights reserved

Les deux bibliothèques... suite

- **Se positionner**
  - se fait par :
```
int  fseek (FILE *fp, long offset, int origine);    (ANSI)
long lseek (int fd, long offset, int origine);      (Unix)
```
  - on spécifie un déplacement par rapport au début du fichier, position courante, ou fin du fichier
- **Caractères ou entiers ?**
  - `getc/getchar, putc/putchar` permettent de lire ou écrire un entier
  - erreur courante : `c = getchar();` avec c de type char ! pourquoi est-ce faux ?
- **Dupliquer**
  - se fait par les fonctions (Unix) :
```
int     dup (int fd);                                (Unix)
FILE *  freopen (char *chemin, char *mode, FILE *fp);
                                                     (ANSI)
```
(variante dup2 à dup)

copyright © 1999 vincent roca; all rights reserved

Les deux bibliothèques... suite

- **Dupliquer... suite**
  - le descripteur retourné est le plus petit descripteur libre
  - exemple : programme comptant les caractères d'un fichier donné en paramètre ou lu sur l'entrée standard

```
int fd, n, nb = 0;
char buf[1];

if (argc == 2) {
            /* l'utilisateur a donné un nom de fichier */
            fd = open (argv[1], «r»);
            close (0);  /* on libère l'entrée std */
            dup (fd);   /* l'entrée std est le fichier */
  }
  /* on s'est ramené au cas où le fichier est en entrée std */
  while ((n = read (0, buf, 1)) != 0)
            nb++;
  printf («%d caractères\n», nb);
```

copyright © 1999 vincent roca; all rights reserved

Les deux bibliothèques... suite

- **Obtenir des informations sur un fichier**
  - se fait par les fonctions (Unix) :
```
int     stat (char *nom_fichier, struct stat *buf);
int     fstat (int fd, struct stat *buf);
int     access (char *nom_fichier, int mode);
```
  - On peut modifier certaines caractéristiques par *chmod et *chown
- **Contrôler les propriétés d'un fichier**
  - *1ère possibilité* (pour n'importe quel fichier) :
```
int     fcntl (int fd, int cmd, long arg);
```

cmd	arg (System V)	arg (BSD)	commentaire
F_SETFL	O_NDELAY	FNDELAY	E/S non bloquants
	O_APPEND	FAPPEND	écriture en fin de fichier
	O_SYNC	-	écriture synchrone
	-	FASYNC	un signal est envoyé si des E/S sont possibles (Sockets)

  - *2ème possibilité* (surtout pour les périphériques) :
```
int     ioctl (int fd, ulong requête, char *arg);
```

copyright © 1999 vincent roca; all rights reserved

1.3.4- Gestion des processus

- **Duplication de processus**
  - très fréquent car seul moyen de créer un processus
  - appel système :
```
int fork (void);
```
  - exemple :
```
if ((pid = fork(void)) == 0) {
        /* code du processus fils */
} else if (pid > 0) {
        /* code du processus père */
        /* pid est ici l'identificateur du fils créé */
} else {
        /* erreur */
}
```
  - deux utilisations :
    - déléguer le travail à un autre processus (exemple : inetd)
    - lancer un autre programme ⇨ `fork` puis `exec` (exemple : shell)

copyright © 1999 vincent roca; all rights reserved

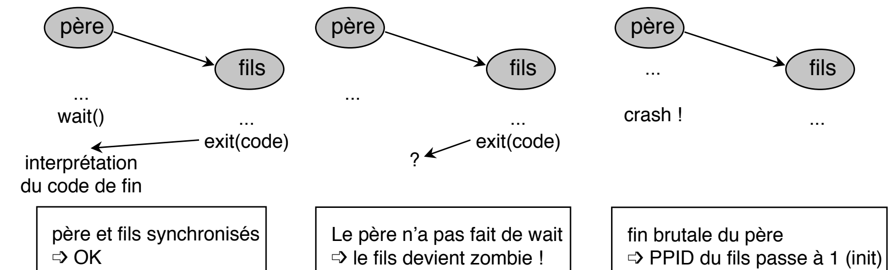
Gestion des processus... suite

- **Duplication de processus... suite**
  - Héritage de certains attributs
  - Nouveau PID («Process Identifier»)
  - PPID («Parent PID») est celui du père
- **Fin de processus**
  - `void exit (int code);`
  - seul le dernier octet de `code` est utilisé
  - peut être intercepté par le père
- **Attente de la fin d'un fils**
  - `int wait (int *code);`
  - père reste bloqué jusqu'à la fin d'un fils
  - permet de savoir si tout s'est bien passé

copyright © 1999 vincent roca; all rights reserved

Gestion des processus... suite

- **Attente de la fin d'un fils... suite**
  - 3 possibilités :



  - signal (SIGCHLD ou SIGCLD) généré chez le père, par défaut ignoré

copyright © 1999 vincent roca; all rights reserved

Gestion des processus... suite

- **Remplacement de processus**
  - appel système :
    `int execv (char *chemin_commande, char *argv[]);`
  - recouvrement des zones text/données_initialisées
  - héritage des :
    - PID, PPID, etc.
    - descripteurs de fichiers
    etc.
  - variante :
    `int execvp (char *nom_commande, char *argv[]);`
    recherche du chemin vers la commande grâce à variable externe `environ`

copyright © 1999 vincent roca; all rights reserved

Chapitre 2 : les communications locales

- **2.1- Introduction**
- **2.2- Les signaux**
- **2.3- Les tubes ordinaires et tubes nommés**
- **2.4- Généralités sur les mécanismes d'IPC**
- **2.5- Mémoire partagée**
- **2.6- Sémaphores**
- **2.7- Files de messages**

copyright © 1999 vincent roca; all rights reserved

2.2- Les signaux

2.2.1- Introduction

- Notification d'un événement
- «Interruption logicielle»
- Liste de signaux

SIGHUP
SIGINT
SIGQUIT
SIGTERM
SIGKILL
SIGTRAP

SIGILL
SIGSEGV
SIGFPE
SIGPIPE
SIGBUS
SIGIOT

SIGIO
SIGURG

SIGCHLD ou SIGCLD
SIGALARM

SIGUSR1
SIGUSR2

SIGPROF
SIGSTKFLT
SIGCONT
SIGSTOP
SIGTSTP
SIGTTIN
SIGTTOU
SIGXFSZ
SIGVTALRM
SIGWINCH
SIGPWR

copyright © 1999 vincent roca; all rights reserved

2.2.2- Génération des signaux

Cinq possibilités...

1- commande kill (shell)

`kill pid` ➪ signal SIGTERM
`kill -9 pid` ➪ signal SIGKILL non masquable

2- fonction kill (programme)

```
int kill (int pid, int signal);
```

Règles :
- même UID ou bien root
- pid > 0 ➪ indique le récepteur
- pid == -1 et non root ➪ à tous les proc. de même UID
- pid == 0 ➪ à tous les proc. du «proc. groupe»
- etc.

copyright © 1999 vincent roca; all rights reserved

Génération des signaux... suite

3- séquences de clavier

Exemples :
- Ctrl-C ou interruption ➪ SIGINT
- Ctrl-\ ou arrêt avec création de core ➪ SIGQUIT
- Ctrl-Z ou suspension de processus ➪ SIGTSTP

4- par des erreurs matérielles

Exemples :
- erreur arithmétique flottante ➪ SIGFPE
- adresse hors espace d'adressage ➪ SIGSEGV

5- sous certaines conditions détectées par le noyau

Exemples :
- arrivée de données urgentes sur une socket ➪ SIGURG
- fin d'un fils ➪ SIGCLD

copyright © 1999 vincent roca; all rights reserved

2.2.3- Utilisation des signaux

Traitements possibles à chaque occurrence :

1- conserver le traitement par défaut

2- ignorer (ou masquer)
- **... sauf SIGKILL / SIGSTOP**

3- Association d'une fonction de traitement ("handler")

- **appel système :**
```
int signal (int signal, void (*fonction)());
```
- **Valeurs possibles du paramètre fonction :**
  - SIG_IGN
  - SIG_DFL
  - fonction utilisateur
- **Contrainte :**
  à chaque occurrence du signal, appeler à nouveau signal()
  est-ce bien nécessaire ???

copyright © 1999 vincent roca; all rights reserved

Utilisation des signaux... suite

- **Exemple 1:**
  - fonction alarm :
    #include <unistd.h>
    long alarm (long secondes);
  - programme un signal SIGALARM après le délai donné
  - utilisation en tant que chien de garde

```
if ((pid = fork()) > 0)
    /* le père arme un chien de garde... */
    signal(SIGALARM, trt_alarm());
    alarm(MAX_EXEC_TIME);
    wait(0);
} else if ( pid == 0) {
    /*... et le fils exécute la commande */
    /* sous son contrôle */
    execv(...);
}
```

- **Exemple 2:**
  - exécution de wait lors de la fin d'un fils

```
signal(SIGCHLD, DeathChild);
...
if ((pid = fork()) > 0)
...

void DeathChild(void)
{
    wait(0);
    signal(SIGCHLD, DeathChild);
}
```


copyright © 1999 vincent roca; all rights reserved

2.2.4- Fiabilisation des signaux

- **Le problème**
  - **premières bibliothèques (pré BSD4.3/SysVR3) non fiables**
  - **exemple : attente de la mise à TRUE d'un flag par une routine de traitement**

```
int flag = 0;         /* var globale mise à 1 par handler */

trt_sigint (void)
{ flag = 1;
}

void main (void)
{ signal (SIGINT, trt_sigint);
  while (1) {
    while (flag == 0)
            pause ();        /* attente arrivée signal */
    <traitement du signal SIGINT>
  }
}
```


copyright © 1999 vincent roca; all rights reserved

Fiabilisation des signaux... suite

- **Désormais...**
  - **les routines de traitement restent installées**
  - **possibilité de *bloquer* des signaux un certain temps (!= ignorer)**
  - **un signal en cours de traitement est automatiquement bloqué jusqu'au retour de la routine**
- **Nouvelles fonctions :**
  - BSD

```
int sigblock (int mask);          /* ajoute les signaux donnés
                                  par mask à ceux bloqués */
int sigsetmask (int mask);        /* bloque les signaux de
                                  mask, débloque les autres */
```

  - System V

```
int sighold (int signal);         /* bloque le signal */
int sirgelse (int signal);        /* débloque le signal */
```


copyright © 1999 vincent roca; all rights reserved

Fiabilisation des signaux... suite

- **Exemple 1 : zone critique où les signaux SIGUSR1|2 sont interdits**

```
int mask_prec;
mask_prec = sigblock(sigmask(SIGUSR1) | sigmask(SIGUSR2));
<zone critique...>
sigsetmask(mask_prec);  /* on rétablit le mask précédant */
```

- **Exemple 2 : Attente de la mise à TRUE d'un flag par une routine de traitement**

```
int flag = 0;        /* variable globale */
void main (void)
{
    int     mask_prec;
    mask_prec = sigblock(sigmask(SIGUSR1));
    while ( !flag)
            sigpause (mask_prec);
    <traitements>
}
```


copyright © 1999 vincent roca; all rights reserved

2.2.5- La bibliothèque POSIX

- **Les principales fonctions**

```
int sigaction (int sig, struct sigaction *action,
               struct sigaction *old_action);
int sigprocmask (int how, sigset_t *set, sigset_t *old_set);
int sigpending (sigset_t *set);
int sigsuspend (sigset_t *mask);
```

- sigaction :
  - change le comportement du processus face à un signal
  - remplace `signal()`
  - avec :

action	nouveau comportement
old_action	si non NULL, on y stocke l'ancien comportement

```
struct sigaction {
    void (*sa_handler)(int); /* SIG_DFL, SIG_IGN, routine */
    sigset_t sa_mask; /* masque des signaux à bloquer pendant*/
                      /* l'execution du handler */
    int      flags;
}
```


copyright © 1999 vincent roca; all rights reserved

La bibliothèque POSIX… suite

- sigprocmask
  - change la liste des signaux actuellement bloqués
  - remplace `sigblock()`
  - avec :

how	SIG_BLOCK	ajoute "set" à liste de signaux déjà bloqués
	SIG_UNBLOCK	retire "set" de la liste de signaux bloqués
	SIG_SETMASK	initialise les signaux bloqués avec "set"

- sigpending
  - examen des signaux de "set" pouvant être survenus alors qu'ils étaient bloqués

- sigsuspend
  - remplace temporairement le masque de signaux avec "mask" et suspend le processus jusqu'à ce qu'un signal autre que ceux de "mask" soit reçu
  - retourne après le retour de la fonction de traitement et restaure l'ancien masque
  - remplace `pause()/sigpause()`

copyright © 1999 vincent roca; all rights reserved

La bibliothèque POSIX… suite

- **Les autres fonctions**

```
int sigemptyset (sigset_t *set);            // init le jeu "set"
int sigfillset (sigset_t *set);             // met à 1 le jeu
int sigaddset (sigset_t *set, int sig);    // ajoute sig au jeu
int sigdelset (sigset_t *set, int sig);    // retire…
int sigismember (sigset_t *set, int sig); // teste…
```

- **Exemple : attente de la mise à TRUE d'un flag par une routine de traitement**

```
sigset_t        mask, old_mask;

sigsetempty (&mask);
sigaddset (&maks, SIGUSR1);

sigprocmask (SIG_BLOCK, &mask, &old_mask);
while ( ! flag )
    sigsuspend (&old_mask);
sigprocmask (SIG_UNBLOCK, &mask, NULL);
<traitements>
```


copyright © 1999 vincent roca; all rights reserved

2.3- Les tubes (ou pipes)

- **2.3.1- Les tubes (pipes) ordinaires**

- canal de communication *unidirectionnel*
- orienté *flux d'octets*

- manipulé comme un fichier (read/write)...
- ... mais *pas de nom* dans l'arborescence Unix
- conséquences :
  - limité à des processus ayant même filiation (héritage)

- création :

```
int pipe (int tube[2]);
```

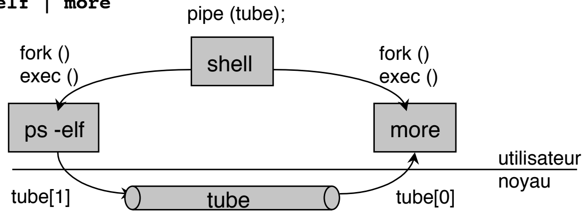
avec :

- tube[0] ⇨ lecture
- tube[1] ⇨ écriture

copyright © 1999 vincent roca; all rights reserved

Les tubes (pipes) ordinaires... suite

- **exemple : `ps -elf | more`**



squelette de programme

```
pipe (tube);
if ((pid = fork ()) == 0) {   /* 1er fils (ps dans exemple) */
  close(1); dup (tube[1]);
  execv («/bin/ps»);
}
if ((pid2 = fork ()) == 0) {  /* 2ème fils (more ds exemple)*/
  close (0); dup (tube[0]);
  execv («/bin/more»);
}
```

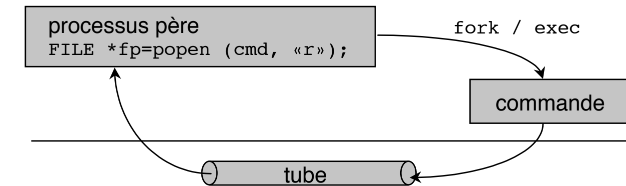

copyright © 1999 vincent roca; all rights reserved

Les tubes (pipes) ordinaires... suite

- **Les fonctions popen / pclose**
  - **association de pipe et de exec**
  - **simplifie l'échange de données avec un processus fils**
  - **prototype :**

```
FILE *popen (char *cmd, char *mode);
int pclose (FILE *tube_p);
```

avec mode=«r» ou «w»



copyright © 1999 vincent roca; all rights reserved

Les tubes (pipes) ordinaires... suite

- **exemple : affichage du répertoire courant**

```
#include <stdio.h>
#define LG   256

void main (void)
{
  FILE *fp;
  char buf[LG];

  if ((fp = popen(«/bin/pwd», «r»)) == NULL)
    /* erreur... */
  if (fgets(buf, LG, fp) == NULL)
    /* erreur... */
  printf(«%s», buf);
  pclose(fp);
}
```


copyright © 1999 vincent roca; all rights reserved

2.3.2- Les tubes (pipe) nommés

- similaires à un tube ordinaire
- flux d'octets unidirectionnel

- ... mais possèdent un nom dans l'arborescence
- conséquence :
  - utilisable par des processus étrangers

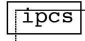
- création :
  `int mknod (char *chemin, int mode, int dev);`

  avec :
  - mode = droits_Unix | S_IFIFO
  - dev = 0 (non utilisé)
  - retourne un `int fd;`
- ouverture :
  - fait par chaque processus avec `open()`
  - ouverture soit en O_RD, soit en O_WR

copyright © 1999 vincent roca; all rights reserved

2.4- Généralité sur les mécanismes d'IPC System V

- Comprend :
  - **mémoire partagée**
  - **sémaphores**
  - **files de messages**
- Nombreux points communs
- Commande Unix pour connaître état :



2.4.1- Format des appels système

	mémoire partagée	*sémaphores*	*file de messages*
include	**<sys/shm.h>**	**<sys/sem.h>**	**<sys/msg.h>**
créer ou ouvrir	**shmget**	**semget**	**msgget**
opérations de contrôle	**shmctl** **shmat** **shmdt**	**semctl**	**msgctl**
communication	**-**	**semop**	**msgsnd** **msgrcv**

copyright © 1999 vincent roca; all rights reserved

2.4.2- La structure ipc_term

- présente pour chaque IPC

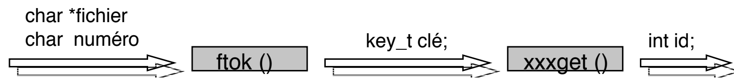
```
struct ipc_perm {
    ushort      uid;     /* UID du propriétaire */
    ushort      gid;     /* GID... */
    ushort      cuid;    /* UID du créateur */
    ushort      cgid     /* GID... */
    ushort      mode;    /* droits lecture/écriture pour */
                         /*  user/group/others */
    ushort      seq;     /* */
    key_t       key;     /* clé d'identification */
};
```

- UID/GID du créateur/propriétaire initialisés lors de la création
- possibilité de changer propriétaire ensuite par `xxxctl`
- droits ➩ initialisés lors de création (premier `xxxget`)
  (les suivants utilisent 0)

copyright © 1999 vincent roca; all rights reserved

2.4.3- Désignation d'un canal IPC; notion de clé

- principe



- génération d'une clé unique au sein du système
  key_t ftok (char *fichier, char num);
  avec :
  num = idf du canal si plusieurs sont nécessaires
- technique :
  combinaison de {num. d'i-node; num. de partition; numéro passé en paramètre}
  ➩ fichier existe et est stable (géré par le serveur)
- remarques :
  - **on peut se passer de ftok mais...**
  - **on peut utiliser IPC_PRIVATE au lieu d'une clé dans xxxget()**
    obtient un numéro unique
    quand l'utiliser ?

copyright © 1999 vincent roca; all rights reserved

2.4.4- Règles de création et d'ouverture d'un canal IPC

- les fonctions `xxxget` prennent un flag

```
flag =     droits_d_accès (9 bits de poids faible) |
           comportement_d_ouverture
```

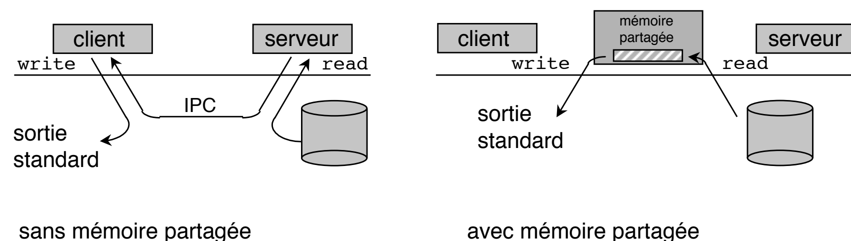
	canal IPC non existant	*canal IPC existant*
rien	**erreur, errno = ENOENT**	**OK**
IPC_CREAT	**OK, on crée le canal**	**OK**
IPC_CREAT I IPC_EXCL	**OK, on crée le canal**	**erreur, errno = EEXIST**
IPC_EXCL	**non sens**	**non sens**

copyright © 1999 vincent roca; all rights reserved

2.5- Mémoire partagée

2.5.1- Principe

- espace d'adressage commun à plusieurs processus
- se manipule normalement
- intérêt : limiter les copies mémoire...



sans mémoire partagée — avec mémoire partagée

copyright © 1999 vincent roca; all rights reserved

2.5.2- Utilisation

- Création/ouverture

  `int shmget (key-t clé, int taille, int flag);`
  - **(cf. 2.4.3 et 2.4.4)**
  - **retourne l'identificateur du segment**
  - **1er appel** ⇨ **création et ouverture du segment**
  - **appels suivants** ⇨ **ouverture du segment après vérification des droits**
- Accès
  - **nécessite un attachement**

    `char *shmat (int id, char *adr, int flag);`
  - **avec adr :**
    - 0 ⇨ le système trouve une adresse
    - > 0 ⇨ attaché à cette adresse ou à une adresse arrondie si SHM_RND
  - **et flag :**
    - 0 ⇨ accès en lecture/écriture
    - SHM_RDONLY
  - **retourne l'adresse d'attachement, ou -1 si erreur**

copyright © 1999 vincent roca; all rights reserved

Utilisation... suite

- Détachement

  `int shmdt (char *adr);`
  - **retourne 0 si OK, -1 si erreur**
- Libération
  - **fait par `shmctl()` avec `IPC_RMID`**
- Opération de contrôle

  int shmctl (int id, int cmd, struct shmid_ds *buf);

  avec :
  - IPC_STAT ⇨ obtenir les infos système
  - IPC_SET ⇨ modifier les permissions
  - IPC_RMID ⇨ libération
- Quels problèmes pose l'utilisation de la mémoire partagée ?

copyright © 1999 vincent roca; all rights reserved

2.6- Sémaphores

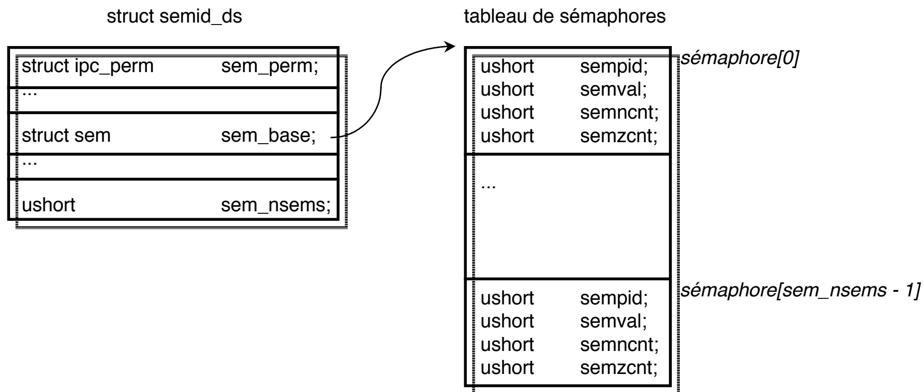
2.6.1- Principe

- synchronisation inter-processus
- gestion de l'accès à des ressources partagées
- exemple :
  - **section critique**
  - **synchronisation d'accès à la mémoire partagée**
  - **synchronisation d'un processus père avec N fils**
- version IPC très générale :
  - **tableau de sémaphores**
  - **à chaque sémaphore est associé un compteur >= 0**

copyright © 1999 vincent roca; all rights reserved

Principe... suite

- Structures de données système



copyright © 1999 vincent roca; all rights reserved

2.6.2- Utilisation des sémaphores

- Création/ouverture

```
int semget (key-t clé, int nb, int flag);
```

  - **retourne l'identificateur du tableau**
  - **1er appel ⇨ création et ouverture du tableau de `nb` sémaphores**
  - **appels suivants ⇨ ouverture du tableau après vérification des droits**

- Opérations de contrôle

```
int semctl (int id, int indice, int cmd, union semun arg);
```

avec :

```
union semun {
    int               val;     /* pour SETVAL */
    struct semid_ds   *buf;    /* pour IPC_STAT et IPC_SET */
    ushort            *array;  /* pour GETALL et SETALL */
}
```

  - **opérations soit sur un sémaphore, soit sur l'ensemble**

copyright © 1999 vincent roca; all rights reserved

Utilisation des sémaphores... suite

- opérations de contrôle... suite

  - **nombreuses possibilités :**

    initialisation et lecture des compteurs
    - SETALL
    - SETVAL
    - GETALL
    - GETVAL
    - GETNCNT
    - GETZCNT

    obtention d'informations
    - IPC_STAT
    - IPC_SET

    libération
    - IPC_RMID

copyright © 1999 vincent roca; all rights reserved

Utilisation des sémaphores... suite

- Opérations
  - **effectuer une liste d'opérations de façon atomique :**

```
int semop (int id, struct sembuf *ops, uint nops);
```

avec :

ops ⇨ liste d'opérations
nops ⇨ nombre d'opérations

```
struct sembuf {
    short    sem_num;      /* indice dans tableau */
    short    sem_op;       /* opération */
    short    sem_flg;      /* flag de l'opération : rien */
                           /* ou IPC_NOWAIT ou IPC_UNDO  */
};
```

  - **opérations :**
    - sem_op > 0 ⇨ semval += sem_op
    - sem_op == 0 ⇨ attente jusqu'à ce que : semval == 0
    - sem_op < 0 ⇨ attente jusqu'à ce que : semval >= | sem_op |
      puis : semval -= | sem_op |

copyright © 1999 vincent roca; all rights reserved

2.6.3- Exemple

- **Fonction de vérrouillage à base de sémaphores**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define PERMS  0660          /* permissions */

static struct sembuf op_lock = {
        0, 0, 0,             /* on attend que semval passe à 0 */
        0, 1, SEM_UNDO       /* et on incrémente */
};

static struct sembuf op_unlock = {
        0, -1, (IPC_NOWAIT I SEM_UNDO)          /* décremente semval de 1 à 0 */
};

int semid = -1;
```


copyright © 1999 vincent roca; all rights reserved

Exemple… suite

```
void lock (void)
{
      if (semid < 0) {
                  key_t       key;
                  if ((key = ftok("/tmp/my_lock_file", 1)) < 0)        exit -1;
                  if ((semid = semget(semid, op_lock, 2) < 0)        exit -1;
      }
      if (semop(semid, op_lock, 2) < 0)                              exit -1;
}

void unlock (void)
{
      if (semid < 0 II semop(semid, op_unlock, 1) < 0)               exit -1;
}
```

- Remarques :
  - Comment détruire le sémaphore ?
  - Lors de la création du sémaphore, le compteur vaut 0 !
  - On aurait pu inverser la technique, mais...

copyright © 1999 vincent roca; all rights reserved

2.7- Files de messages

- **2.7.1- Principe**
  - messages de taille arbitraire
  - appartiennent à une file donnée (idf de file)
  - files stockées dans le noyau
  - accessibles à tous les processus ayant des droits compatibles
  - structure générique associée à chaque message :

```
struct msgbuf {
  long       mtype;            /* type du message */
  char       mtext[1];         /* partie données du message */
};
```

  - ➪ en pratique définir une structure recouvrant msgbuf

copyright © 1999 vincent roca; all rights reserved

2.7.2- Utilisation des files de messages

- Création/ouverture

```
int msgget (key-t clé, int flag);
```

  - **retourne l'identificateur de la file de messages**
  - **1er appel** ➪ **création et ouverture de la file**
  - **appels suivants** ➪ **ouverture de la file après vérification des droits**

- Opérations de contrôle

```
int msgctl (int id, int cmd, struct msqid_ds *buf);
```

avec :
  - IPC_STAT ➪ obtenir les infos système
  - IPC_SET ➪ modifier les permissions
  - IPC_RMID ➪ libération

copyright © 1999 vincent roca; all rights reserved

Utilisation des files de messages... suite

- Envoi de données

```
int msgsnd (int id, struct msgbuf *msgp, int lg, int flag);
```

avec

- msgp — pointeur sur structure utilisateur

```
struct mes_messages {
  long      mtype;   /* interprété seulement par appli */
  char      mtext[100];      /* ma zone de données */
};
```

- lg — taille de la partie mtext de cette structure (ici 100)
- flag — action à effectuer si les ressources systèmes ne permettent pas la prise en compte d'un nouveau message
  - 0 ➩ appel bloquant
  - IPC_NOWAIT ➩ non bloquant, retourne erreur si file saturée

copyright © 1999 vincent roca; all rights reserved

Utilisation des files de messages... suite

- Réception de données

```
int msgrcv (int id, struct msgbuf *msgp, int lg,
            long mtype, int flag);
```

avec

- msgp — pointeur sur structure utilisateur
- lg — taille de la partie mtext de cette structure
- mtype — choix sur le message à lire
  - mtype == 0 ➩ lit 1er message
  - mtype > 0 ➩ lit 1er message dont le type est celui donné
  - mtype < 0 ➩ lit 1er message dont le type est < | mtype |
- flag — comportement
  - IPC_NOWAIT ➩ non bloquant
  - IPC_EXCEPT ➩ si mtype>0, lecture du 1er msg de type != mtype
  - IPC_NOERROR ➩ tronquer des messages trop longs

copyright © 1999 vincent roca; all rights reserved

Chapitre 3 : les communications inter-machines

- **3.1- Les Sockets**
  - Introduction
  - Survol des Sockets
  - Communications locales avec le domaine Unix
  - Adressage
  - Description des fonctions usuelles
  - Description des fonctions de la bibliothèque réseau
  - Description des fonctions avancées
  - Gestion des options
  - Multiplexage des E/S
- **3.2- L'API TLI/XTI**
  - Introduction

copyright © 1999 vincent roca; all rights reserved

3.1- Les Sockets

- **3.1.1- Introduction**
  - introduit en 1982 dans BSD 4.1,
  - mature en 1986 avec BSD 4.3
  - mise en œuvre plus complexe que les IPC car :
    - désignation plus complexe :
      {famille, protocole, adr locale, idf local, adr distante, idf distant}
    - non symétrique (client/serveur)
    - hétérogénéité des protocoles (connectés ou datagrammes)
    - supporte plusieurs familles de protocoles
      - TCP/UDP ➩ AF_INET ou AF_INET6
      - Unix connecté ou datagrammes ➩ AF_UNIX
      - TP0 ou TP5 de l'OSI ➩ AF_OSI
    - API (Application Programming Interface) identique dans tous les cas

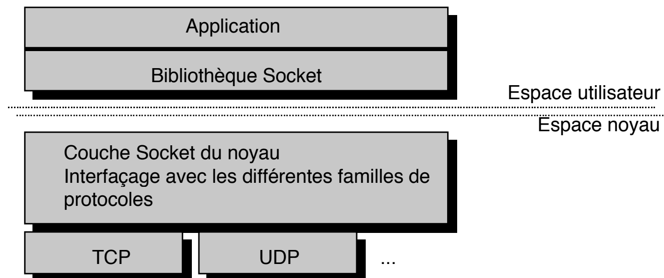
copyright © 1999 vincent roca; all rights reserved

Introduction… suite

Architecture générale

distinguer :

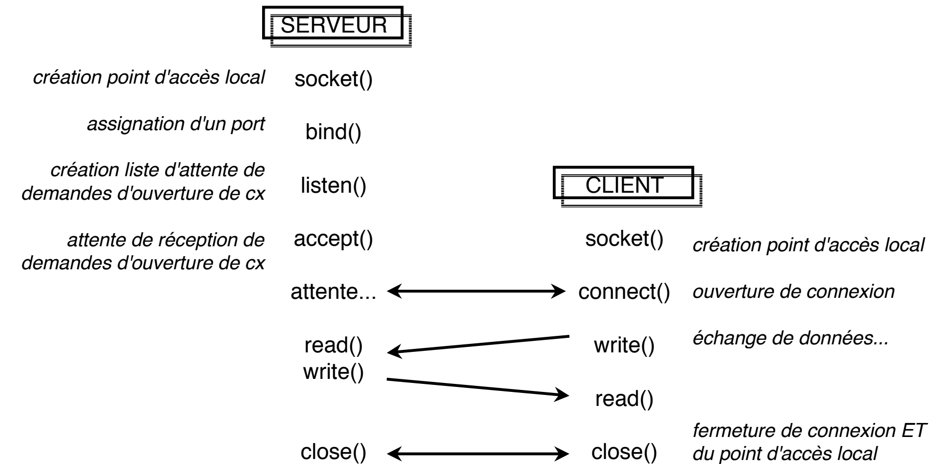
- la bibliothèque Socket (linkée avec l'application)
- la couche Socket (dans le noyau) implémentant les appels système de la bibliothèque



copyright © 1999 vincent roca; all rights reserved

3.1.2- Survol des Sockets

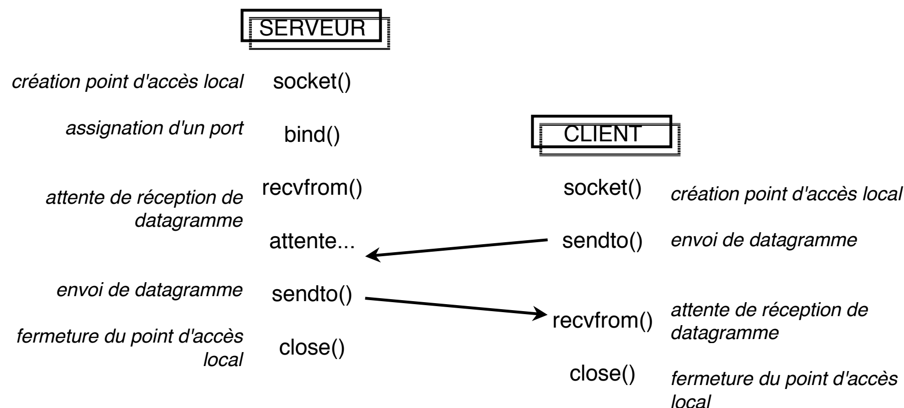
Mode connecté



copyright © 1999 vincent roca; all rights reserved

Survol des Sockets... suite

Mode datagrammes



copyright © 1999 vincent roca; all rights reserved

Survol des Sockets... suite

Liste des fonctions Socket

- gestion locale
  - **socket/close**
  - **bind**
- gestion de la connexion
  - **listen/accept**
  - **connect**
- transferts en mode connecté
  - **read/write**
  - **recv/send**
  - **readv/writev**
- transferts en mode datagrammes
  - **recvfrom/sendto**
  - **recvmsg/sendmsg**
- attentes multiples
  - **select**
- options
  - **getsockopt/setsockopt**
  - **fcntl**
  - **ioctl**
- informations
  - **getsockname**
  - **getpeername**
- bibliothèque réseau
  - **gethostbyname**
  - **gethostbyaddr**
  - **getservbyname**
  - **getservbyport**
  - **inet_addr**
  - **inet_ntoa**

copyright © 1999 vincent roca; all rights reserved

3.1.3 Communications locales avec le domaine Unix

- **Principes**
  - IPC avec une API Socket
  - communication locale à une machine
  - les identificateurs sont des noms de fichiers
    - **sont de type S_IFSOCK**
    - **inutile de les créer**
    - **peuvent ne pas être visibles dans l'arborescence**
- **Remarque:**
  - on peut aussi communiquer localement avec AF_INET et une adresse de rebouclage (ou “loopback”) : 127.0.0.1… différences ?

copyright © 1999 vincent roca; all rights reserved

3.1.4- Adressage

- **Structure générique**

```
#include <sys/socket.h>
struct sockaddr {
    ushort   sa_family;        /* famille d'adresse AF_... */
    char     sa_data[14];      /* données */
}
```

- **Famille AF_INET**

```
#include <netinet/in.h>
struct sockaddr_in {
    ushort   sin_family;       /* AF_INET */
    ushort   sin_port;         /* numéro de port TCP/UDP */
    struct in_addr sin_addr;   /* adresse sur 4 octets */
    char     sin_zero[8];      /* bourrage */
}
struct in_addr {
    ulong    s_addr;           /* les 4 octets */
}
```


copyright © 1999 vincent roca; all rights reserved

Adressage... suite

- **Famille AF_UNIX**

```
#include <sys/un.h>
struct sockaddr_un {
ushort      sun_family;      /* AF_UNIX */
char        sun_path[108];   /* chemin + nom fichier */
}
```

  - structure plus grande que la structure générique !
  - conséquence : dans les appels système, on passe :
    - **pointeur sur struct sockaddr**
    - **taille effective de la structure**

copyright © 1999 vincent roca; all rights reserved

3.1.5- Description des fonctions usuelles

- **socket ()**
  - **ouverture d'un point d'accès local du type voulu**

    `int socket (int af, int type, int protocole);`
  - **avec :**
    - af — AF_UNIX, AF_INET, etc. (PF_UNIX, PF_INET sont équivalents)
    - type — SOCK_STREAM, SOCK_DGRAM, ou SOCK_RAW
    - protocole — souvent à 0 car {af; type} suffit

type	*AF_UNIX*	*AF_INET*
SOCK_STREAM	oui, protocole 0	oui, protocole 0 ou IPPROTO_TCP
SOCK_DGRAM	oui, protocole 0	oui, protocole 0 ou IPPROTO_UDP
SOCK_RAW	non	oui, deux possibilités : IPPROTO_ICMP ⇨ accès ICMP IPPROTO_RAW ⇨ accès IP

  - **retourne un descripteur de socket (similaire fd)**
  - **aucun transfert réseau pour l'instant !**

copyright © 1999 vincent roca; all rights reserved

Fonctions usuelles... suite

- **bind ()**
  - **assignation d'une adresse/numéro de port locaux à une socket**
    ```
    int bind (int sockfd, struct sockaddr *mon_adr, int lg_adr);
    ```
  - **quand l'utiliser ?**
    - *serveur* ➩ enregistre son adresse publique (en fait numéro de port)
    - *client* ➩ n'en a pas besoin en général
  - **il n'est pas indispensable de spécifier son adresse ➩ adresse générique**
    ```
    mon_adr.sin_addr.s_addr = htonl(INADDR_ANY);
    ```
  - **le numéro de port peut être :**
    - numéro réservé connu de tous et conservé dans /etc/services
    ```
    struct servent       *sp;
    if (!(sp = getservbyname("login", "tcp")))
        /* erreur... */
    mon_adr.sin_port = htons(sp->s_port);
    ```
    - numéro privé (>=1024)
      se mettre d'accord avec le client si c'est un serveur

copyright © 1999 vincent roca; all rights reserved

Fonctions usuelles... suite

- **connect ()**
  - **utilisé par un client pour se connecter au serveur**
    ```
    int connect (int sockfd, struct sockaddr *adr_serveur,
                 int lg_adr);
    ```
  - **en mode connecté :**
    - déclenche échange de messages sur le réseau "3-way handshake"
    - finit de remplir les infos de désignation
      {famille, protocole, adr locale, idf local, *adr distante, idf distant*}
  - **en mode datagrammes :**
    - semble paradoxal !
    - permet :
      - de stocker l'adresse du serveur ➩ on ne la spécifie plus
      - de spécifier que l'on n'acceptera de datagrammes que de cet hôte
      - de savoir si le destinataire existe

copyright © 1999 vincent roca; all rights reserved

Fonctions usuelles... suite

- **listen ()**
  - **le serveur, en mode connecté, indique qu'il est prêt à recevoir des demandes d'ouvertures de connexion**
    ```
    int listen (int sockfd, int taille_file);
    ```
  - **pourquoi une file ?**
- **accept ()**
  - **le serveur accepte la connexion**
    ```
    int accept (int sockfd, struct sockaddr *adr, int *lg_adr);
    ```
  - **avec :**
    - adr pointeur sur le buffer alloué par l'application
    - lg_adr pointeur sur la taille du buffer
  - **au retour :**
    - le buffer contient l'adresse du client
    - lg_adr contient la taille effective de l'adresse
    - le code de retour est le nouveau sockfd si >0, un code d'erreur si <0
    - l'ancien sockfd permet de rester en attente de nouvelles connexions

copyright © 1999 vincent roca; all rights reserved

Fonctions usuelles... suite

- **accept () ... suite**
  - **appel bloquant**
  - **Attention:**
    - En cas de réception de signal (exemple : SIGCHLD), l'appel système retourne un code d'erreur (EINTR: Error INTeRrupted).
    - Tester impérativement et relancer `accept()` le cas échéant.
  - **remarques : deux niveaux d'acceptation d'ouverture de connexion**
    - niveau TCP acceptation immédiate ("3-way handshake")
      la nouvelle connexion ("control block TCP") est conservée sur une liste à part
    - niveau application acceptation de l'application crée un nouveau point d'accès et y transfert la connexion ("control block TCP")

copyright © 1999 vincent roca; all rights reserved

Fonctions usuelles... suite

- **exemple : serveur concurrent**

```
int sockfd, nouv_sockfd;

if ((sockfd = socket (...)) < 0)       ... erreur
if ( bind (sockfd, ...) < 0)           ... erreur
if ( listen (sockfd, SOMAXCONN) < 0)   ... erreur
while (1) {
    if ((nouv_sockfd = accept (sockfd, ...)) < 0)
            if (errno == EINTR) continue; else ... erreur
    if ((pid = fork ()) == 0) {
            /* fils */
            close (sockfd);           /* inutile ici */
            <traitements avec nouv_sockfd>
            exit (0);
    } else if (pid == 0) {
            /* père */
            close (nouv_sockfd);      /* inutile ici */
    } else   ... erreur
}
```


copyright © 1999 vincent roca; all rights reserved

Fonctions usuelles... suite

- **send/recv et sendto/recvfrom**
  - **transmission et réception de données**

```
int send (int sockfd, char *buf, int noctets, int flags);
int sendto (int sockfd, char *buf, int noctets, int flags,
            struct sockaddr *to, int lg_adr);

int recv (int sockfd, char *buf, int noctets, int flags);
int recvfrom (int sockfd, char *buf, int noctets, int flags,
            struct sockaddr *from, int *lg_adr);
```

  - **avec :**
    - flags combinaison de

MSG_OOB	(Out Of Band) lect/transmission de données urgentes
MSG_PEEK	lecture non destructives de données reçues
MSG_DONTROUTE	évite passage par le code de routage

copyright © 1999 vincent roca; all rights reserved

Fonctions usuelles... suite

- **close ()**
  - **fermeture de la socket (point d'accès local)**

```
int close (int sockfd);
```

  - **en mode connecté, TCP essaie de transmettre les données restantes avant d'initier la fermeture de connexion**
  - **par défaut non bloquant**
  - **rôle de l'option SO_LINGER (cf. setsockopt) :**
    - si temps == 0, alors les données non transmises sont détruites
    - si temps > 0, alors, on transmet ce qui reste et le close () retourne ensuite

copyright © 1999 vincent roca; all rights reserved

3.1.6- Description des fonctions de la bibliothèque réseau

- **Conversion de format des mots machine**
  - **Format sur le réseau est par *convention* « big endian »**
  - **Mais d 'autres existent suivant les processeurs (INTEL utilise « little endian »)**

```
u_long htonl (u_long entier_long);     /*h: host, n: network*/
u_short htons (u_short entier_court);  /*l: long, s: short  */

u_long ntohl (u_long entier_long);
u_short ntohs (u_short entier_court);
```

- **Conversion d 'adresses**
  - **Passage entre chaîne ASCII de la notation décimale pointée, et entier long**

```
u_long inet_addr (char *chaîne);
char * inet_ntoa (struct in_addr inaddr); /* format réseau */
int inet_aton (char *chaîne, struct in_addr *inaddr);
```


copyright © 1999 vincent roca; all rights reserved

Fonctions de la bibliothèque réseau… suite

- **Recherche via le DNS**
  - **rechercher l 'adresse IP d 'une machine dont on connaît le nom**

```
struct hostent * gethostbyname (char* name);
```

  - **rechercher le nom d 'une machine dont on connaît l 'adresse IP**

```
struct hostent * gethostbyaddr (char* addr, int len, int
  type);
```

- **Recherche d 'un service**
  - **rechercher le numéro de port d 'un service dont on connaît le nom**

```
struct servent * getservbyname (char* serv_name, char* proto);
struct servent * getservbyport (int port, char* proto);
```

  - **Utilise le fichier « /etc/services »**

copyright © 1999 vincent roca; all rights reserved

Fonctions de la bibliothèque réseau… suite

- Exemple :

```
struct sockaddr_in    sin;
struct hostent        *h;
struct servent        *sp;

memset((char*)&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
if ((h = gethostbyname(«sirac.lip6.fr»)) == NULL) {
    perror(«gethostbyname»); exit(-1);
}
memcpy ((char*)&sin.sin_addr.s_addr, h->h_addr_list[0],
        h->h_length);
if (!(sp = getservbyname(«login», «tcp»))) {
    perror(«getservbyname»); exit(-1);
}
sin.sin_port = htons(sp->s_port);
```


copyright © 1999 vincent roca; all rights reserved

3.1.7- Description des fonctions avancées

- **shutdown ()**
  - **fermeture de l'un ou l'autre des sens de transfert**

```
int shutdown (int sockfd, int mode);
```

  - **avec :**
    - mode == 0 ➩ fermeture en lecture
    - mode == 1 ➩ fermeture en écriture
    - mode == 2 ➩ fermeture en lecture/écriture

- **readv/writev**
  - **transmission et réception de données avec possibilité de "scattering" / "gathering"**

```
int readv (int sockfd, struct iovec iov[], int iovcount);
int writev (int sockfd, struct iovec iov[], int iovcount);
```

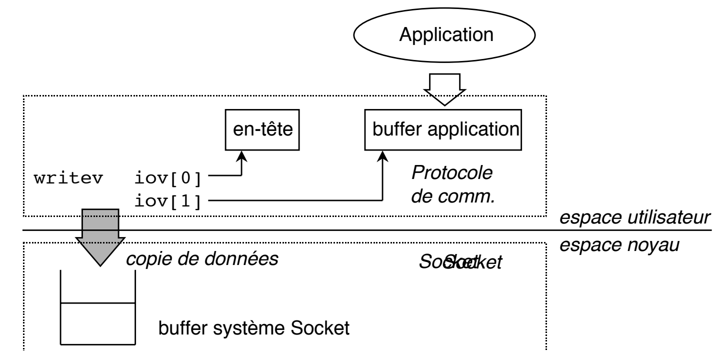
  - **avec :**

```
#include <sys/uio.h>
struct iovec {
    caddr_t  iov_base;        /* départ adresse buffer */
    int      iov_len;         /* taille du buffer */
}
```


copyright © 1999 vincent roca; all rights reserved

Fonctions avancées... suite

- **readv/writev... suite**
  - **intérêt : optimisation de performances**
  - **exemple : protocole implémenté dans l'espace utilisateur ➩ 1 seule copie !**



copyright © 1999 vincent roca; all rights reserved

Fonctions avancées... suite

- **recvmsg/sendmsg**
  - **forme la plus générale**
  - **permet scattering/gathering**
  - **permet le passage de descripteurs de fichiers entre deux processus avec AF_UNIX**
- **getsockname/getpeername**
  - **recherche de l'adresse/identificateur local (resp. distant)**

```
int getsockname (int sockfd, struct sockaddr *adr_locale,
                 int *lg_adr);
int getpeername (int sockfd, struct sockaddr *adr_dist,
                 int *lg_adr);
```

  - **getsockname : stocke dans adr_locale l'adr/idf local et initialise lg_adr en conséquence**
  - **getpeername : idem avec adr/idf distant**
  - **permet à un client de connaître le n° port local ou distant**

copyright © 1999 vincent roca; all rights reserved

3.1.8- Gestion des options

- **Trois possibilités...**
  - **getsockopt() / setsockopt()** — **spécifique Sockets**
  - **fcntl()** — **}**
  - **ioctl()** — **} plus généraux**
- **1- fonctions getsockopt() / setsockopt()**

```
int getsockopt (int sockfd, int niveau, int nom,
                char *valeur, int *lg);
int setsockopt (int sockfd, int niveau, int nom,
                char *valeur, int lg);
```

  - **avec :**
    - niveau — protocole concerné
    - nom — nom de l'option
    - valeur — buffer contenant la valeur lue (resp. à positionner) pour l'option
    - lg — pointeur avec getsockopt () car modifié par effet de bord, entier avec setsockopt () car seulement lu

copyright © 1999 vincent roca; all rights reserved

Gestion des options... suite

- **quelques options :**

niveau	*nom*	*Get/Set*	*description*	*type*
IPPROTO_TCP	TCP_MAXSEG	G	Max Segment Size	int
	TCP_NODELAY	G/S	force envoi immédiat	int (flag)
SOL_SOCKET	SO_LINGER	G/S	comportement durant close	struct linger
	SO_RCVBUF	G/S	taille socket de réception	int
	SO_SNDBUF	G/S	taille socket d'émission	int

- **exemple :**

```
int taille_sndbuf;

taille_sndbuf = 16384;
if (setsockopt (sockfd, SOL_SOCKET, SO_SNDBUF,
    (char*)&taille_sndbuf, sizeof (taille_sndbuf)) < 0)
          /* erreur */
```


copyright © 1999 vincent roca; all rights reserved

Gestion des options... suite

- **2- Fonction ioctl ()**

```
int ioctl (int fd, ulong requête, char *arg);
```

  - **utilisation très large :**
    - fichiers (cf. chap 1)
    - socket
    - routage
    - interfaces réseaux
  - **convention de nommage des noms de requêtes pour routage/interfaces**
    - SIOC ➩ Socket IOCtl
    - S ou G ➩ Set ou Get
    - IF ➩ InterFace
  - **exemples pour le contrôle d'interfaces**
    - SIOCGIFADDR ➩ stocke l'adresse de l'interface dans une structure ifreq passée en paramètre
    - SIOCGIFBRDADDR ➩ idem pour l'adresse de broadcast de l'interface

copyright © 1999 vincent roca; all rights reserved

Gestion des options... suite

- **fonction ioctl () ... suite**
  - **exemples pour une socket**
    - FIOASYNC ➪ positionne mode de réception asynchrone (signal SIGIO)
    - FIONREAD ➪ retourne le nombre de données reçues
    - FIONBIO ➪ positionne ou retire le mode d'E/S non bloquant
      si l'E/S ne peut se faire immédiatement, retourne -1 et initialise `errno` à `EWOULDBLOCK`
- **3- fonction fcntl ()**

```
int fcntl (int fd, int cmd, long arg);
```

  - **un peu redondant avec ioctl ()**
  - **exemples**
    - FNDELAY ➪ idem FIONBIO de ioctl (mais sans possibilité d'en sortir)
    - FASYNC ➪ idem FIOASYNC de ioctl (mais sans possibilité d'en sortir)

copyright © 1999 vincent roca; all rights reserved

3.1.9- Multiplexage d'E/S

- **plusieurs techniques**
  - **polling après passage en mode non bloquant**
    ➪ gaspillage ressources
  - **création d'un fils par socket**
    ➪ lourd
  - **utilisation d'E/S synchrones (signal SIGIO)**
    ➪ traitement des signaux couteux
  - **utilisation de select ()**
    ➪ conçu pour !!!
- **fonction select ()**

```
int select (int maxfd, fd_set *readfds, fd_set *writefds,
            fd_set *exceptfds, struct timeval *timeout);
```

  - **avec :**
    - maxfd valeur maxi des descripteurs devant être examinés + 1
      (le système considère tous les fd de valeur 0, 1, ... , maxfd - 1)

copyright © 1999 vincent roca; all rights reserved

Multiplexage d'E/S... suite

- readfds tableau de Sockets où l'on attend des lectures
- writefds tableau de Sockets où l'on attend de pouvoir transmettre
- exceptfds talbeau de Sockets où l'on attend l'arrivée d'exceptions
- (données urgentes, OOB)
- timeout temps pendant lequel on est prêt à attendre
  – si 0 ➪ retourne immédiatement (polling)
  – si > 0 ➪ attendre au plus le temps indiqué
  – si pointeur NULL ➪ attendre infiniment

- **initialisation des tableaux**
  - FD_ZERO met tout à zéro
  - FD_SET le fd doit être considéré
  - FD_CLR le fd ne doit plus être considéré
  - FD_ISSET teste si un fd est prêt suite au retour de la fonction select
  - NB : en cas de boucle, réinitialiser les différents tableaux avant nouveau select
- **code de retour**
  – si > 0 le nombre de fd prêts, il faut ensuite trouver lesquels !!!
  – si == 0 arrivée à échéance du timer, aucun fd n'est prêt
  – si < 0 erreur

copyright © 1999 vincent roca; all rights reserved

Multiplexage d'E/S... suite

- **exemple**

```
int                 fd;
fd_set              set;
struct timeval      timeout;

FD_ZERO(&set);
FD_SET(4, &set);
FD_SET(5,&set);
timeout.tv_sec = 5;
timeout.tv_usec = 0;
if ((val = select (6, &set, NULL, NULL, &timeout)) > 0) {
    for (fd = 0; fd < 6 ; fd++)
            if (FD_ISSET(fd))
                    <traitements associés>
} else if (val == 0) {
    printf("rien reçu\n");
} else {
    ... erreur
}
```


copyright © 1999 vincent roca; all rights reserved

3.2 L'API TLI/XTI

Introduction

- **deux noms :**
  - TLI (Transport Layer Interface) ➩ version originelle
  - XTI (X/Open Transport Interface) ➩ version X/Open

- **Présent essentiellement sur les OS dérivés de System V R4 (exemple : Solaris)**

- **API concurrente aux Sockets**
- **Associée historiquement à l'environnement d'implémentation de piles de protocoles STREAMS**
- **Fonctionne aussi avec les piles TCP/IP d'origine BSD**
- **Indépendant de la famille de protocoles**

- **Plus lourd que les Sockets à l'usage (surtout pour gérer des options) !**

copyright © 1999 vincent roca; all rights reserved