

Programmation système



C. Gouinaud
2005-2006

Introduction

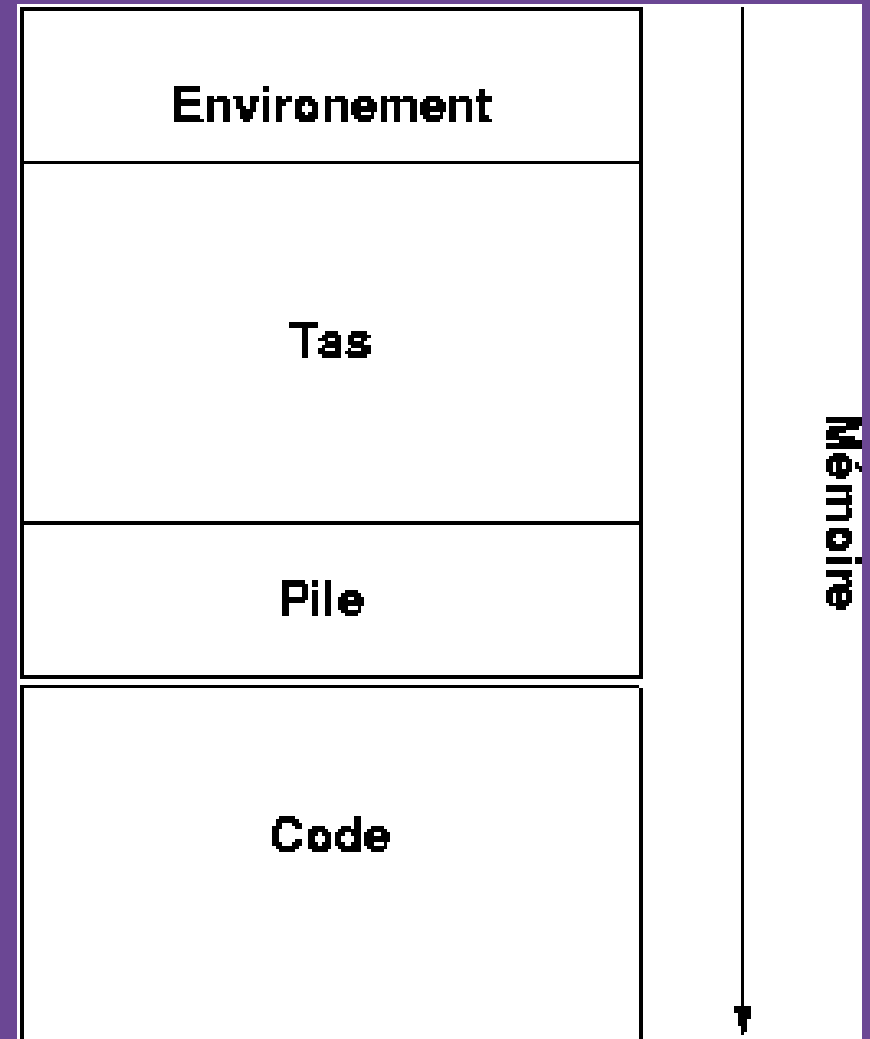
- Création et gestion de tâche
 - Communication interne
 - Fichiers création/verrouillage/manipulation
 - Outils système pour la programmation
 - Programmation bas niveau (minix)
 - Des TP qui ont quelques choses à voir avec le cours :)
-
-

Création et gestion des tâches

- Rappel de notions
 - Niveaux shell et système
 - Création de processus
 - Utilisation des processus
 - Création des threads
 - Tache sous Unix(posix)
 - Tache sous Windows 2000
-
-

Rappel de notions

- Un processus
 - Une mémoire
 - Un thread
- Un thread
 - Un contexte + une pile
 - Une fibre
- Une fibre
 - Scedule en mode user



Rappel (II) windows XP

- Job Collection de processus
- Processus Conteneur de ressources
- Thread tâche noyaux
- Fibre tâche gérée dans l'espace utilisateur



Niveaux shell et système (UNIX)

- Création via le shell
- Liste ps -aux ou ps -edf
- Arrêt, mise en sommeil
 - Kill -9, kill -STOP, kill -CONT, kill -HUP
- Visible dans /proc
- Manipulation de job :
 - CTRL+Z, fg, bg, jobs, kill

Examples

```
gouinaud@elo:~/text/cours/sys
[gouinaud@elo sys]$ ls /proc
1      2163  2331  2645  3290  3419  39      fb      mtrr
1070   2182  2332  2652  3313  3420  4      filesystems net
113    2201  2337  2656  3379  3421  40     fs      partitions
1405   2227  2397  2665  3382  3422  5      ide     pci
1452   2267  2426  2671  3384  3423  acpi    interrupts scsi
1453   2279  2442  2674  3387  3425  asound  iomem    self
1456   2290  2452  2675  3389  3448  buddyinfo ioports  slabinfo
1478   2309  2510  2689  3398  3473  bus     irq      stat
1698   2318  2519  27    3400  3475  cmdline kcore    swaps
1699   2322  2529  2702  3404  3522  cpuinfo kmsg     sys
1700   2323  2540  2715  3406  3653  crypto  loadavg  sysrq-trigger
1797   2324  2551  2716  3407  3655  devices locks    sysvipc
1902   2325  2598  2729  3409  3682  diskstats mdstat   tty
194    2326  2607  28    3411  3683  dma     meminfo  uptime
2      2327  2611  2880  3412  3686  dri     misc     version
2076   2328  2620  3      3415  37    driver   modules  vmnet
2159   2329  2628  3019  3417  38    execdomains mounts    vmstat
[gouinaud@elo sys]$ ls /proc/3655
attr  cmdline  environ  fd      mem      root  statm  task
auxv  cwd      exe      maps    mounts   stat  status wchan
[gouinaud@elo sys]$
[gouinaud@elo sys]$
[gouinaud@elo sys]$
```

Job en shell bash

```
gouinaud@elo:~/text/cours/sys
[gouinaud@elo sys]$ nedit

ls -jementape *

[3]+  Stopped                  nedit
[gouinaud@elo sys]$ bg
[3]+ nedit &
[gouinaud@elo sys]$ fg
nedit

ls -a

[3]+  Stopped                  nedit
[gouinaud@elo sys]$ bg
[3]+ nedit &
[gouinaud@elo sys]$ jobs
[1]  Running                  xv &
[2]-  Running                  nedit &
[3]+  Running                  nedit &
[gouinaud@elo sys]$ %3
nedit

[gouinaud@elo sys]$ kill %2
[gouinaud@elo sys]$
[2]+  Terminated              nedit
[gouinaud@elo sys]$ █
```


Être sympa !

- Règle de priorité : mis à jours chaque minute

Priorité = CPU_USAGE + nice + base

- Commande renice

usage: renice priority [[-p] pids] [[-g] pgrps] [[-u] users]

- Lancement avec nice

```
[gouinaud@elo sys]$ nice xv &
```

```
[gouinaud@elo sys]$ ps aux | grep xv
```

```
gouinaud 4317 0.0 0.4 4784 2384 pts/2 SN 18:46 0:00 xv
```

```
gouinaud 4325 0.0 0.1 5852 604 pts/2 S+ 18:48 0:00 grep xv
```

Création de processus

- Allocation de ressources mémoire
- Création de l'environnement
- Monté du code en mémoire
- Initialisation
- Lancement d'une première tâche
- Scheduling

Un système est caractérisé par cette gestion

Processus et ressources

- Structure noyaux
 - Décrit les ressources
- Structure apparente
 - Mémoire, fichiers
- Un thread
 - Un par défaut, lancera les autres

Répertoire de travail
Acl, Umask
File handle
Pages mémoire
Propriétaire, groupe

Scheduling
Pile
Registres
Signaux

Un thread

A black arrow originates from the text 'Un thread' and points diagonally upwards and to the left, terminating at the bottom-right corner of the light blue box that contains the list of resources.

Processus et partage

- Comment de fait l'héritage
Environnement/objet + base de registre
- Présence d'une hiérarchie
Arborescence/notion de job + handle
- Classe de processus ou pas
Pas/service/batch/spool

Influence le style de prog
Et les routines systèmes disponibles

Sous unix

- Création : `fork()`
Duplication de processus
 - Recouvrement : `exec()`
Remplacement de la partie code + raz
 - Manipulation d'environnement : `setenv()`
Manipulation de variable=valeur
 - Hiérarchie de processus `wait()`
Attente des fils !
-
-

Fork()

```
pid_t fork(void);
```

- Retourne un entier :
 - 1 si sa foire
 - 0 si c'est le fils
 - Fpid si c'est le père
- On connait les autre
getpid(), getppid()

- On wait :

```
pid_t wait(int *status);
```

```
pid_t waitpid(pid_t  
pid, int *status, int  
options);
```

```
options = (!) bloquant
```

Example

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{pid_t pere,fils; int vingt,x=1;
```

```
printf("C'est partis !\n");
```

```
fils = fork();
```

```
if (fils==-1) fprintf(stderr, "erreur je sort");
```

```
else
```

```
{
```

```
    if (fils==0) /* c'est le fils */
```

```
        { printf("%d: Je suis le fils de %d \n", getpid(), getppid());
```

```
          sleep(20); x=2;
```

```
          printf("%d: Je sort \n",getpid()); exit(20); }
```

```
    else
```

```
        { printf("%d: Je suis le père de %d \n", getpid(), fils);
```

```
          waitpid(fils, &vingt, 0);
```

```
          printf("%d: Le processus fils %d est sorti \n", getpid(), getppid());
```

```
        }
```

```
    }
```

```
} /* FIN du main */
```

```
[gouinaud@elo c]$ cc fork.c -o fork
```

```
[gouinaud@elo c]$ ./fork
```

```
C'est partis !
```

```
4830: Je suis le fils de 4829
```

```
4829: Je suis le père de 4830
```

```
4830: Je sort
```

```
4829: Le processus fils 3655 est sorti
```

Recouvrement

- Remplacement du code par un nouveau !
- Remise à zéro de la pile
- Remise à zéro du tas
- Copie des descripteurs noyaux
- Héritage de l'environnement

On n'en revient pas !



Exec et autres !

```
#include <unistd.h>
```

```
extern char **environ;
```

```
int execl(const char *path, const char *arg, ...);
```

```
int execlp(const char *file, const char *arg, ...);
```

```
int execl(const char *path,  
          const char *arg , ..., char * const  
          envp[]);
```

```
int execv(const char *path, char *const argv[]);
```

```
int execvp(const char *file, char *const argv[]);
```

Retourne le status quand sa foire !

Example

```
int main()
{
char *argu[3];
pid_t pere,fils; int vingt,x=1;

fils = fork();

if (fils==-1)
fprintf(stderr, "erreur je sort");
else
{
if (fils==0) /* c'est le fils */
{ printf("%d: Je suis le fils de %d \n", getpid(), getppid());
argu[0] = "Je met ce que je veux";
argu[1] = "-a";
argu[2] = NULL;
execv("/bin/uname", argu);
printf("invisible !");
}
else
{ printf("%d: Je suis le père de %d \n", getpid(), fils);
waitpid(fils, &vingt, 0);
printf("%d: Le processus fils %d est sorti \n", getpid(), getppid());
}
}
} /* FIN du main */
```

```
[gouinaud@elo c]$ cc exec.c -o exec
[gouinaud@elo c]$ uname -a
Linux elo 2.6.9-1.667 #1 Tue Nov 2 14:41:25
EST
[gouinaud@elo c]$ ./exec
4907: Je suis le fils de 4906
Linux elo 2.6.9-1.667 #1 Tue Nov 2 14:41:25
EST
4906: Je suis le père de 4907
```

```
4906: Le processus fils 3655 est sorti
```

Manipulation environnementale !

- Getenv

```
char *getenv(const char  
    *name);
```

- Setenv

```
int setenv(const char  
    *name, const char  
    *value, int overwrite);
```

- Unsetenv

```
void unsetenv(const char  
    *name);
```



Exemple

```
int main()
{
    pid_t pere, fils; int vingt, x=1;

    if (getenv("PERE" ))
        printf("%d: PERE=%s \n",
               getpid(), getenv("PERE" ));
    setenv("PERE", "NOEL");

    fils = fork();

    if (fils == -1) fprintf(stderr, "erreur je sort");
    else
    {
        if (fils == 0) /* c'est le fils */
        { if (getenv("PERE" ))
          printf("%d: PERE=%s \n", getpid(), getenv("PERE" ));
          printf("%d: Je sort \n", getpid()); exit(20); }
        else
        { printf("%d: Je suis le père de %d \n", getpid(), fils);
          waitpid(fils, &vingt, 0);
          printf("%d: Le processus fils %d est sorti \n", getpid(),
getppid());
        } }
    } /* FIN du main */
```

```
[gouinaud@elo c]$ export PERE=FOUETARD
[gouinaud@elo c]$ cc getenv.c -o getenv
[gouinaud@elo c]$ ./getenv
5015: PERE=FOUETARD
5016: PERE=NOEL
5016: Je sort
5015: Je suis le père de 5016
5015: Le processus fils 3655 est sorti
[gouinaud@elo c]$ echo $PERE
FOUETARD
```

Sous windows

Creation avec CreateProcess();

10 paramètres :

- Executable
- Ligne de commande
- Descripteur de sécurité
- First thread pointer
- Father handle bit
- Debug priority flag
- Char *env[];
- Windows handle
- 18 valeurs de retour

CreateProcess

```
(ByVal lpApplicationName As String,  
ByVal lpCommandLine As String,  
ByVal lpProcessAttributes As Long,  
ByVal lpThreadAttribute As Long,  
ByVal bInheritHandles As Long,  
ByVal dwCreationFlags As Long,  
ByVal lpEnvironment As Long,  
ByVal lpCurrentDirectory As Long,  
ByVal lpStatupInfo As Long,  
ByVal lpProcessInformation As Long)  
As Boolean
```

Exemple CreateProcess

Lancer des programmes (.exe)

Pour lancer Media Player :

```
Dim Retval as Boolean
```

```
Retval = CreateProcess("\Windows\Player.exe", "", 0, 0, 0, 0, 0, 0, 0, 0)
```

Pour lire le fichier "\Mes Documents\raste.mp3" :

```
Dim Retval as Boolean
```

```
Retval = CreateProcess("\Windows\Player.exe", "\Mes  
Documents\rasta.mp3",
```

```
0,0,0,0,0,0,0,0)
```

Les mécanisme du système
s'applique au programmeurs !



Exemple 2 en C

ShellExecuteEx(): lancement de notepad:

```
SHELLEXECUTEINFO ExecuteInfo;
memset(&ExecuteInfo, 0, sizeof(ExecuteInfo));

ExecuteInfo.cbSize= sizeof(ExecuteInfo);
ExecuteInfo.fMask = 0;
ExecuteInfo.hwnd  = 0;
ExecuteInfo.lpVerb="open";
ExecuteInfo.lpFile  ="c:\\windows\\notepad.exe";
ExecuteInfo.lpParameters="toto.dat";
ExecuteInfo.lpDirectory  = 0;
ExecuteInfo.nShow      = SW_SHOW;
ExecuteInfo.hInstApp    = 0;
if(ShellExecuteEx(&ExecuteInfo) == FALSE)
{
// erreur
}
```

Exemple 3 en C

CreateProcess(): lancement de notepad.

```
STARTUPINFO      siStartupInfo;
PROCESS_INFORMATION piProcessInfo;

memset(&siStartupInfo, 0, sizeof(siStartupInfo));
memset(&piProcessInfo, 0, sizeof(piProcessInfo));
siStartupInfo.cb = sizeof(siStartupInfo);
if(CreateProcess("c:\\windows\\notepad.exe",
                "toto.txt",0,0,FALSE,
                CREATE_DEFAULT_ERROR_MODE,0,0,
                &siStartupInfo,&piProcessInfo) == FALSE)
{
    // erreur
}
```

Création des threads

- Tous les threads d'un processus partagent :
 - Le tas
 - Le même processeur (hyperthreading !)
 - Les mêmes file handle
- Les thread d'un processus ont en particulier :
 - Une pile
 - Un contexte

outils spécifiques pour la gestion des partages

■ *Plusieurs styles d'implémentation*

- Posix - le plus standard
- Windows – adapter au système
- Solaris-suspension et continuation
- Java – dans la machine virtuelle
- Chacun à sa propre API



☰ *Code monothread versus multithread*

Pièges :

- Variables globales
- Débordement de piles
- Gestion des signaux
- Gestions des handles (clavier,souris)
- Gestion des réponses au événement (CTRL+C)



■ *Gestion concurrente du Tas*

Pb de conditions de concurrence
= Technique d'exclusions mutuelle

- Section critiques = région exclusive d'exécution
- Sémaphore vecteur d'entier vérifiable
- Mutex Verrou logiciel bloquant



☰ *Thread Posix*

- API pour créer et les contrôler
- Existe sur Unix, VMS, Windows, Linux
- Fonctions de la forme :

pthread_[objet]_operation_[np]

np = non portable

objet = conditionnel ou mutex

#include « pthread.h »

■ *Création*

```
int pthread_create(pthread_t  
    *tid, pthread_attr_t *attr,  
    void (*start_function)(void *),  
    void * arg);
```

- Paramètres :
 - tid identifiant de thread pid.x
 - attr attribut de création pthread_attr_default
 - start_function point d'entrée
 - Arg l'argument de la fonction (var ou struct)

■ *Identification et variable*

- L'identification se fait via :
`pthread_t pthread_sef(void);`
- La pile est propre a chaque thread mais dans un seul espace
ie : il est possible d'échanger des valeur via un tableau de pointeur globaux ...

`Js[tid] = &j;`

Bref, on fait ce que l'on veut !

■ *Arrêt des tâches*

- Exit() arrête tous les thread :(
- Fonction spécifique
 void pthread_exit(void *retval);
 Retval valeur de retour exploitable par les autres thread !
- Destruction des ressources si appel à :
 int pthread_detach(pthread_t tid);
 dans le thread initial !



☰ *Abandon d'une tâche*

Arrêt par un autre thread :

```
int pthread_cancel(pthread_t thread);
```

```
int pthread_setcancelstate(int state, int  
    *oldstate);
```

- CANCEL_ON ok on se calme
- CANCEL_OFF cause toujours

```
int pthread_setcanceltype(int type, int *oldt);
```

```
type = PTHREAD_CANCEL_ASYNCHRONOUS
```

```
void pthread_testcancel(void);
```

■ *Synchronisation de tâches*

Attente de la fin d'une autre tâche !

```
int pthread_join(pthread_t th, void  
    **thread_return)
```

Retourne :

- 0 okdac
- EINVAL : thread inconnus
- ESRCH : activité inexistante
- EDEADLK : situation de deadlock

Utilisation très courante dans les logiciel
interactif !

▢ *Les mutex*

Un mutex doit corespondre à une ressource !

- Création

```
int pthread_mutex_init(pthread_mutex_t  
*mutex, const pthread_mutex_attr_t  
*mutexattr);
```

- Destruction

```
pthread_mutex_destroy(p_mutex);
```

■ *Utilisation des mutex*

- appel bloquant a :

```
int pthread_mutex_lock(pthread_mutex_t  
*mutex);
```

- essai de verrouillage

```
int pthread_mutex_trylock(pthread_mutex_t  
*mutex);
```

- lâché du verrou :

```
int pthread_mutex_unlock(pthread_mutex_t  
*mutex);
```

☰ *Thread linux*

- Porté principalement par la fonction :
`int clone(int (*fn)(void *), void *child_stack,
int flags, void *arg);`
- Suivant les flags :
 - Même espace CLONE_VM
 - Même fichier CLONE_FILES
 - Même Pid CLONE_PID

Processus Léger = 'implémentation des thread !

☰ *Thread Windows*

Une fonction de création simple :

```
HANDLE CreateThread(  
    LPSECURITY_ATTRIBUTES attr, SIZE_T  
    taillepile, LPTHREAD_START_ROUTINE  
    fonction, LPVOID Parametre, DWORD  
    CreationFlags, LPDWORD Tid );
```

Pour créer un virus utiliser plutôt la fonction
CreateRemoteThread !



■ *Gestion courante :*

- Arrêt de la tâche :
`VOID ExitThread(DWORD ExitCode);`
 - Qui suis je :
`HANDLE GetCurrentThread(void);`
 - Ou vais je :
`BOOL TerminateThread(HANDLE hThread,
DWORD dwExitCode);`
 - Ect ... chouf msdn !
-
-

☐ *Fibre Windows*

```
LPVOID CreateFiber(  SIZE_T taillepile,  
                    LPFIBER_START_ROUTINE fonction, LPVOID  
                    Parametre);
```

Paramètres :

- Taillepile taille initial de la pile ,0= défaut.
- Fonction Pointer vers la fonction de démarrage.
- Parametre Pointer sur les données de la fonction.
- Retourne Null si erreur, id de fibre autrement.

☐ *Scheduling de fibre Windows*

- La fonction SwitchToFiber démarre une fibre. Elle ne marche que si on l'appelle dans une fibre.

VOID SwitchToFiber(LPVOID fibre_id);

- La fonction ConvertThreadToFiber permet la création de la première fibre. Elle doit être appelé avant la précédente !

LPVOID ConvertThreadToFiber(LPVOID par);

Leçon numéro 2 – communication entre processus

- Signaux
- Pipe
- Socket
- IPC



Communication par signaux !

Interruptions logiciels d'un processus vers un autre :

- Unix
 - Entre deux processus du même utilisateur
 - Primitive signal et kill
- Windows
 - Primitive signal
 - Moins de signaux

Programmation sous Windows :

- armer une interruption signal :

```
void *signal( int sig,  
             void (__cdecl *func ) (int [, int ] ) );
```

- utiliser un signal :

Ya pas c'est que des signaux système !

Liste des signaux Windows

<i>sig value</i>	Description
SIGABRT	Abnormal termination
SIGFPE	Floating-point error
SIGILL	Illegal instruction
SIGINT	CTRL+C signal
SIGSEGV	Illegal storage access
SIGTERM	Termination request

Signaux unix

- Armer un signal

```
#include <signal.h>
```

```
typedef void (*sighandler_t)(int);
```

```
sighandler_t signal(int signum, sighandler_t f);
```

- Utiliser un signal

```
int kill(pid_t pid, int sig);
```

- Suivant les implémentation il faut réarmer !



Signaux Unix !

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction
SIGTRAP	5*	trace trap
SIGABRT	6*	abort (generated by abort(3) routine)
SIGEMT	7*	emulator trap
SIGFPE	8*	arithmetic exception
SIGKILL	9	kill (cannot be blocked, or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe or other socket with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGURG	16@	urgent condition present on socket

SIGSTOP	17+	stop (cannot be caught, blocked, or ignored)
SIGTSTP	18+	stop signal generated from keyboard
SIGCONT	19@	continue after stop
SIGCHLD	20@	child status has changed
SIGTTIN	21+	background read attempted from control terminal
SIGTTOU	22+	background write attempted to control terminal
SIGIO	23@	I/O is possible on a descriptor (see fcntl(2V))
SIGXCPU	24	cpu time limit exceeded (see getrlimit(2))
SIGXFSZ	25	file size limit exceeded (see getrlimit(2))
SIGVTALRM	26	virtual time alarm (see getitimer(2))
SIGPROF	27	profiling timer alarm (see getitimer(2))
SIGWINCH	28@	window changed (see termio(4) and win(4S))
SIGLOST	29*	resource lost (see lockd(8C))
SIGUSR1	30	user-defined signal 1
SIGUSR2	31	user-defined signal 2

Signaux Posix

Fonctions posix :

sigaction, sigprocmask, sigpending, sigsuspend

SYNOPSIS

```
#include <signal.h>
```

```
int sigaction(int signum, const struct sigaction *act,  
              struct sigaction *oldact);
```

```
int sigprocmask(int how, const sigset_t *set,  
               sigset_t *oldset);
```

```
int sigpending(sigset_t *set);
```

```
int sigsuspend(const sigset_t *mask);
```

Attention aux signaux

- Pas de blocage d'interruptions
= Risque de comportements imprévisibles
- Blocage de fonctions systèmes
Notions de fonction instable
alarm(), bind(), mkfifo(), signal(), sleep(), read()
- Interblocage
A kill B kill C kill A
A kill B, B kill A trop tot

Bref faut réfléchir !

Utilisation des signaux en shell

- Kill ne fait pas que tuer !
Kill -STOP PID
Kill -CONT PID
Kill -HUP PID
- Trap permet de bloquer des signaux !
trap [-lp] [[arg] sigspec ...]
arg exécuté quand le shell reçoit sigspec
Récupère CTRL+C

Communication par pipeline !

- Tuyaux bidirectionnel
fifo entre deux processus
- Manipulable en shell
avec | !
- Nommable
il reçoivent une nom de fichier
- Manipulation analogue à des fichiers
read(),write()

Tube simple !

- Création

```
int pipe(int fidedes[2]);  
fidedes[0] ecrire, fidedes[1] lire  
avant fork
```

- Fermeture

```
size_t write(int fd, const void *buf, size_t count);
```

- Lecture, Ecriture

```
ssize_t read(int fd, void *buf, size_t count);  
size_t write(int fd, const void *buf, size_t count);
```



```

#include <stdio.h>
#define ENUF(msg, value) {perror(msg); exit(value); }

main(argc, argv)
int argc;
char *argv[];
{ int i, pid, fd[2]; char buffer[512];

  if ( pipe(fd) != 0) ENUF("pipe erreur", 1);
  i = 0;
  while ( (pid=fork()) == -1) /* on ressaye */
    if (++i > 2 ) ENUF(" fork 2x rate", 2) else sleep(5);

  if (pid == 0) {
    /* Fils */
    fprintf(stdout,
      "fils : fd[0] %d, fd[1] %d\n",fd[0],fd[1]);
    write(fd[1],"blabla!",8);
  }
  else {
    /* Père */
    fprintf(stdout,
      "pere fd[0] %d, fd[1] %d \n", fd[0], fd[1]);
    close(fd[1]); /* On ecrit pas*/
    sleep(2);
    read(fd[0],&buffer,8);
    fprintf(stdout,"b:%s\n",buffer);
  }
} /* main */

```

[gouinaud@elo c]\$ gcc pipe.c -o pipe
 [gouinaud@elo c]\$./pipe
 fils : fd[0] 4, fd[1] 5
 pere fd[0] 4, fd[1] 5
 b:blabla!

*Exemple
pipe !*

Tube nommés !

- Limitation du fait des descripteurs
 - Tube associés à un descripteurs de fichier, visible
 - Fonction hybrique, fichier/tube
 - Lecture bloquante par défaut
 - Manipulable en shell
 - Multi-utilisateur
 - Le créateur existe !
-
-

Tube nommés pratique !

- Création en C
`int mkfifo(const char *pathname, mode_t mode);`
mode façon 755
 - Création en Shell
`mkfifo [OPTION] NAME...`
option `-m 770` par exemple
 - Exploitation avec `open(« /tmp/tube », ..)`
 - Lecture, écriture, fermeture comme dab !
-
-

Tube nommé fonction utile !

- Link, unlink création d'un inode physique dans le sf
- Fstat(char *path) récupération d'information
- Dup, dup2 duplication de descripteur
- Fnctl contrôle de l'aspect bloquant :
 int fcntl(int fd, int cmd)
 x=fcntl(fd[0], F_GETFL)
 int fcntl(int fd, int cmd, long arg);
 fcntl(fd[0], F_SETFL, x| O_NONBLOCK);

Tube Windows !

- Fonctionne comme sous Unix
- Pipe nommés avec ACL
- Moins utilisé ...



Les IPC classique

Interprocess communication :

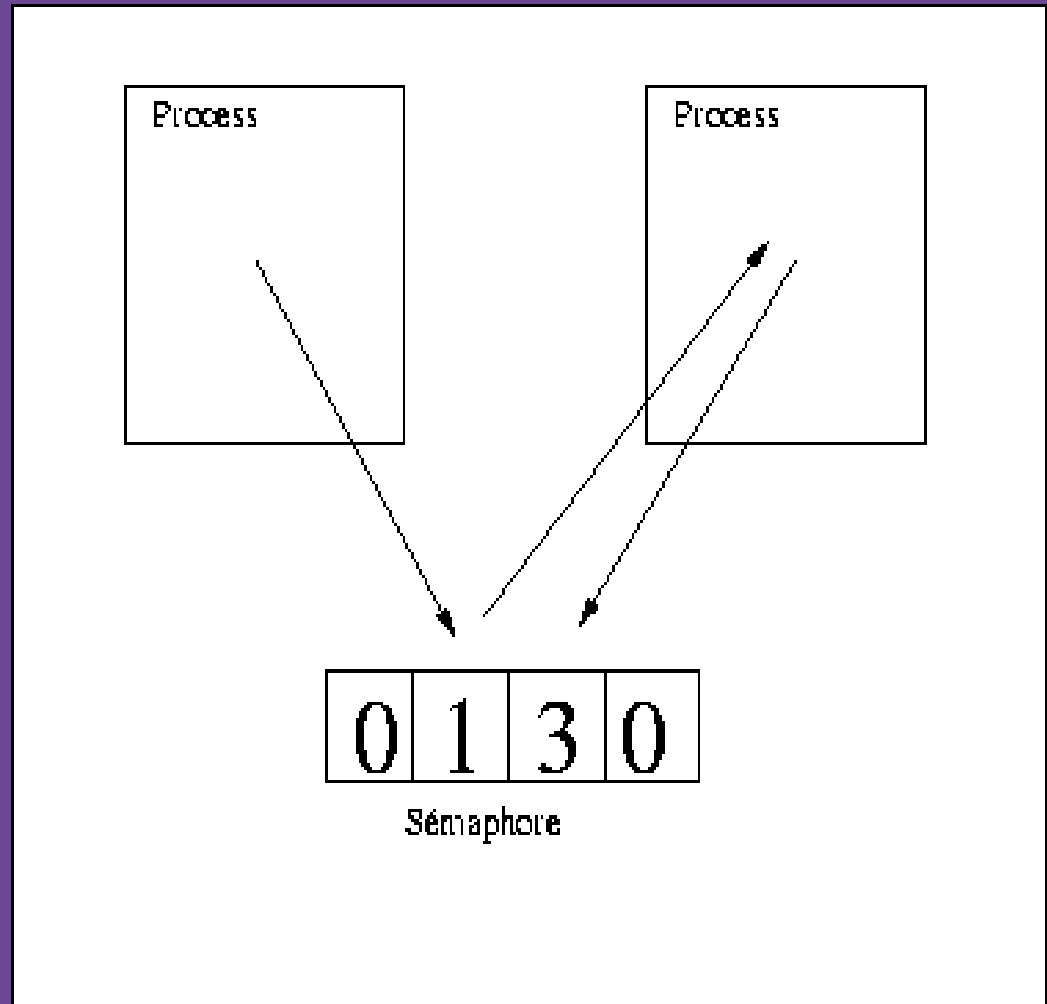
- Sémaphores
- Shared memory
- Files de messages

Ils implémentent les trois type de communications basiques : Fifo, LiFo, booléenne.



Les sémaphores

- Vecteurs d'entiers
- Accès synchrone
- But :
 - Permet d'attendre
 - Permet de signaler



Utilisation d'un sémaphore

- Primitive de création smget

```
int semget(key_t key, int nsems, int semflg);
```

- Modification d'un sémaphore

```
int semop(int semid, struct sembuf *sops,  
          unsigned nsops);
```

- Initialisation, configuration destruction

```
int semctl(int semid, int semnum, int cmd, ...);
```

Utilisation de semget : création

```
Main() {  
key_t key;  
int semid;  
  
// recuperont une clef  
  
key = ftok("/etc/hosts", getppid());  
  
// un sémaphore a trois entiers  
semid = semget(key, 3, 0660 | IPC_CREAT);  
}
```

Utilisation de semctl

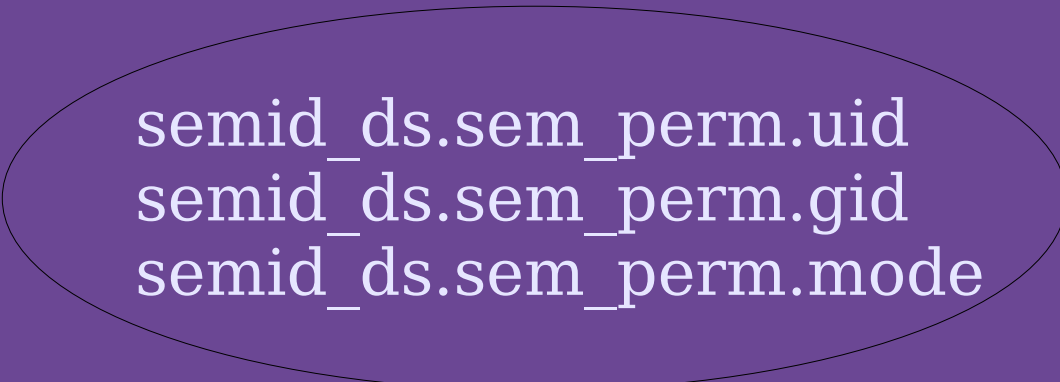
```
int semctl(int semid, int semnum, int cmd, ...);
```

- On set une valeur ou plusieurs
- On lit une valeur ou plusieurs
- Une union :

```
union semun {  
    int val; // valeur pour 1 struct  
    semid_ds *buf;  
    unsigned short *array;  
    struct seminfo *__buf; };
```

Commandes possibles

- IPC_STAT
- IPC_RMID
- GETALL, SETALL
- SETVAL, GETVAL
- IPC_SET



```
semid_ds.sem_perm.uid  
semid_ds.sem_perm.gid  
semid_ds.sem_perm.mode
```

Utilisation des sémaphores semop

```
int semop(int semid, struct sembuf *sops,  
unsigned nsops);
```

- sops est contenant les opérations à effectuer
- nsops le nombre de ces opérations
- structure sembuf = une opération

```
unsigned short sem_num,  
short sem_op  
short sem_flg
```

par exemple (1,2,IPC_WAIT) (0,-1,IPC_NOWAIT)

Astuce : si $\text{sem} + \text{sem_op} < 0$ on bloque !

Un exemple d'opération :

- Quand on manipule les sémaphores ils sont bloqués
- On incrémente dans un processus on décrémente dans l'autre
- On en utilise plusieurs

```
struct sembuf lock  
    = {0, -1, 0};  
struct sembuf rel  
    = {0, 1, 0};
```

```
semop(semid, &lock, 1);
```

```
semop(semid ,&rel, 1);
```


Les shared memory

- Bout de mémoire partagé
- Taille maximum 4Mo(souvent)
- Persistant
- Verrouillé en écriture pendant la lecture
- Identifié par une clef
- Lecture et écriture pas bidirectionnel



Shared memory pratique

- Créer un segment

```
int shmget(key_t key, int size, int shmflg);
```

- S'attacher à un segment

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- Se détacher d'un segment

```
int shmdt(const void *shmaddr);
```

- Supprimer un segment

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Utilisation des shared memory

- Création

```
int shmget(key_t key, size_t size, int shmflg);
```

- Attachement

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- Détachement

```
int shmdt(const void *shmaddr);
```

Opération sur les shm

Une fonction :

```
int shmctl(int shmid,  
           int cmd, struct  
           shm_id_ds *buf);
```

- Destruction

IPC_RMID

- rens/modif

IPC_STAT/IPC_SET

```
struct shm_id_ds {  
    struct ipc_perm shm_perm;  
    int shm_segsz;  
    time_t shm_atime;  
    time_t shm_dtime;  
    time_t shm_ctime;  
    unsigned short shm_cpid;  
    unsigned short shm_lpid;  
    short shm_nattch;  
    ...  
};
```

```

int main()
{
    pid_t pere,fils,ret;
    key_t key; int shmid,i,n; char *buffer;
    struct shmid_ds garbage;

    key=ftok("/etc/hosts",getpid());

    shmid=shmget(key, 1024*sizeof(char), 0660|
    IPC_CREAT);

    fils = fork();
    if (fils==-1) fprintf(stderr, "erreur je sort");
    else {
        if (fils==0) /* c'est le fils */
        { printf("%d: Je suis le fils de %d \n", getpid(),
        getppid());
        for (;;)
        {
            if (buffer=shmat(shmid,NULL,0))
            {
                n=time(NULL);
                sprintf(buffer,"fils PID:%d \t fils:%d \t time : %d\n",
                fils, getpid(),n);
                sleep(2);
            }else exit(1);
        }
        printf("%d: Je sort \n",getpid()); exit(20); }
    else

```

```

{ printf("%d: Je suis le père de %d \n", getpid(), fils);
    if (buffer=shmat(shmid,NULL,0))
        for (i=0;i<20;i++) {
            sleep(1);
            printf(buffer);
        }
    kill(fils,15);
    waitpid(fils, &ret, 0);
    printf("%d: Le processus fils %d est sorti \n",
    getpid(), getppid());
    for (i=0;i<20;i++)
        { sleep(1); printf("."); fflush(stdout); }
    shmdt(buffer);
    shmctl(shmid,IPC_RMID , &garbage);
}
} /* FIN du main */

```

```

[gouinaud@elo c]$ ./shmtest
C'est partis !
3808: Je suis le fils de 3807
3807: Je suis le père de 3808
fils PID:0      fils:3808      time : 1139173889
fils PID:0      fils:3808      time : 1139173891
fils PID:0      fils:3808      time : 1139173891
fils PID:0      fils:3808      time : 1139173893
fils PID:0      fils:3808      time : 1139173893

```

Un exemple

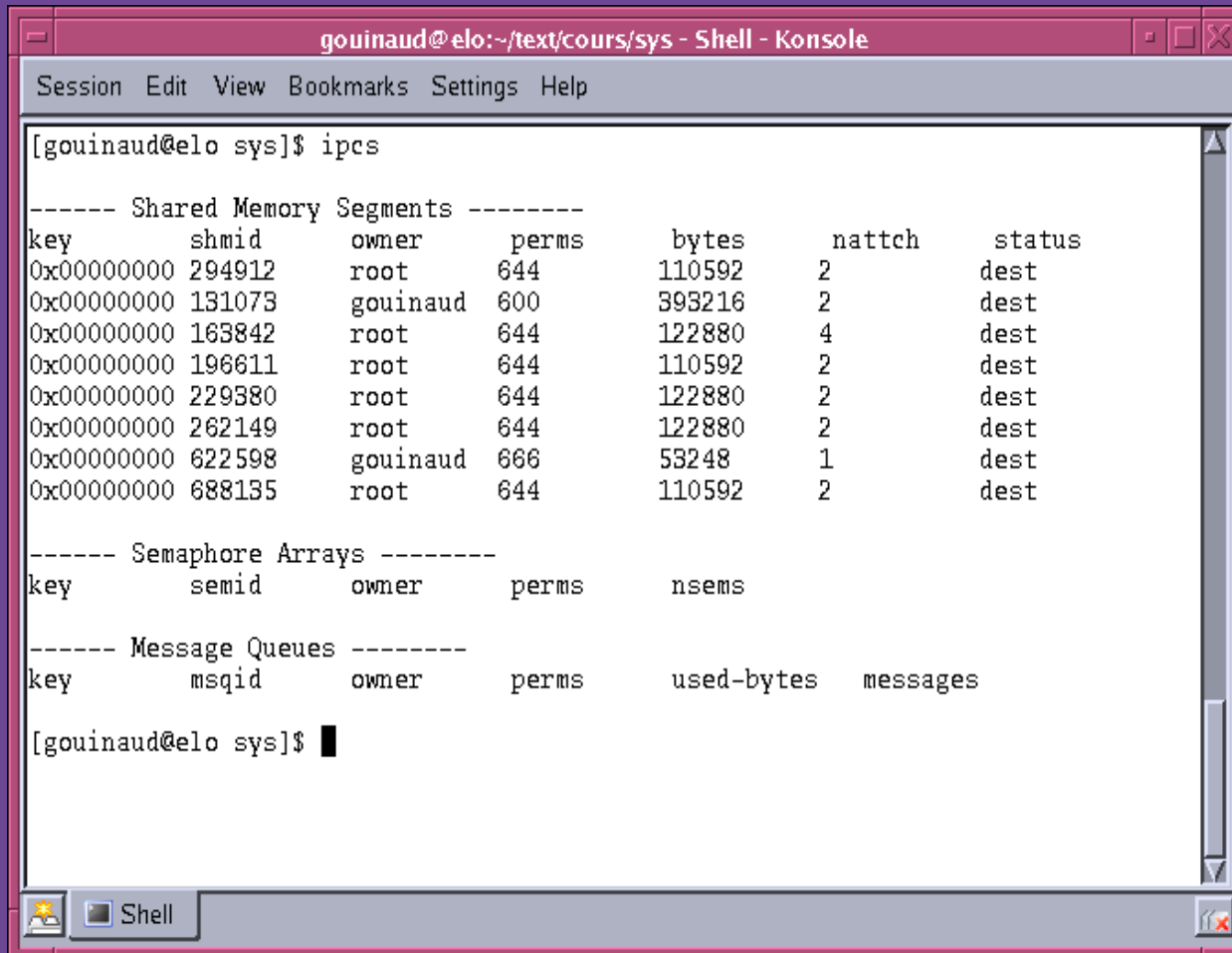
Droits des IPC

- Les ipc sont analogues au fichiers
- Le propriétaire à le droit de changer les droits
- Le root à le droit de changer le propriétaire

```
struct ipc_perm {  
    key_t  key;  
    ushort uid;  
    ushort gid;  
    ushort cuid;  
    ushort cgid;  
    ushort mode;  
    ushort seq;  
};
```

Ipcs et ipcrm

- Liste des ipc
 - ipcs
- Effacement des IPC
 - Programme
 - Ipcrm
 - s sémaphore
 - m shared memory
 - q file de message



```
[gouinaud@elo sys]$ ipcs

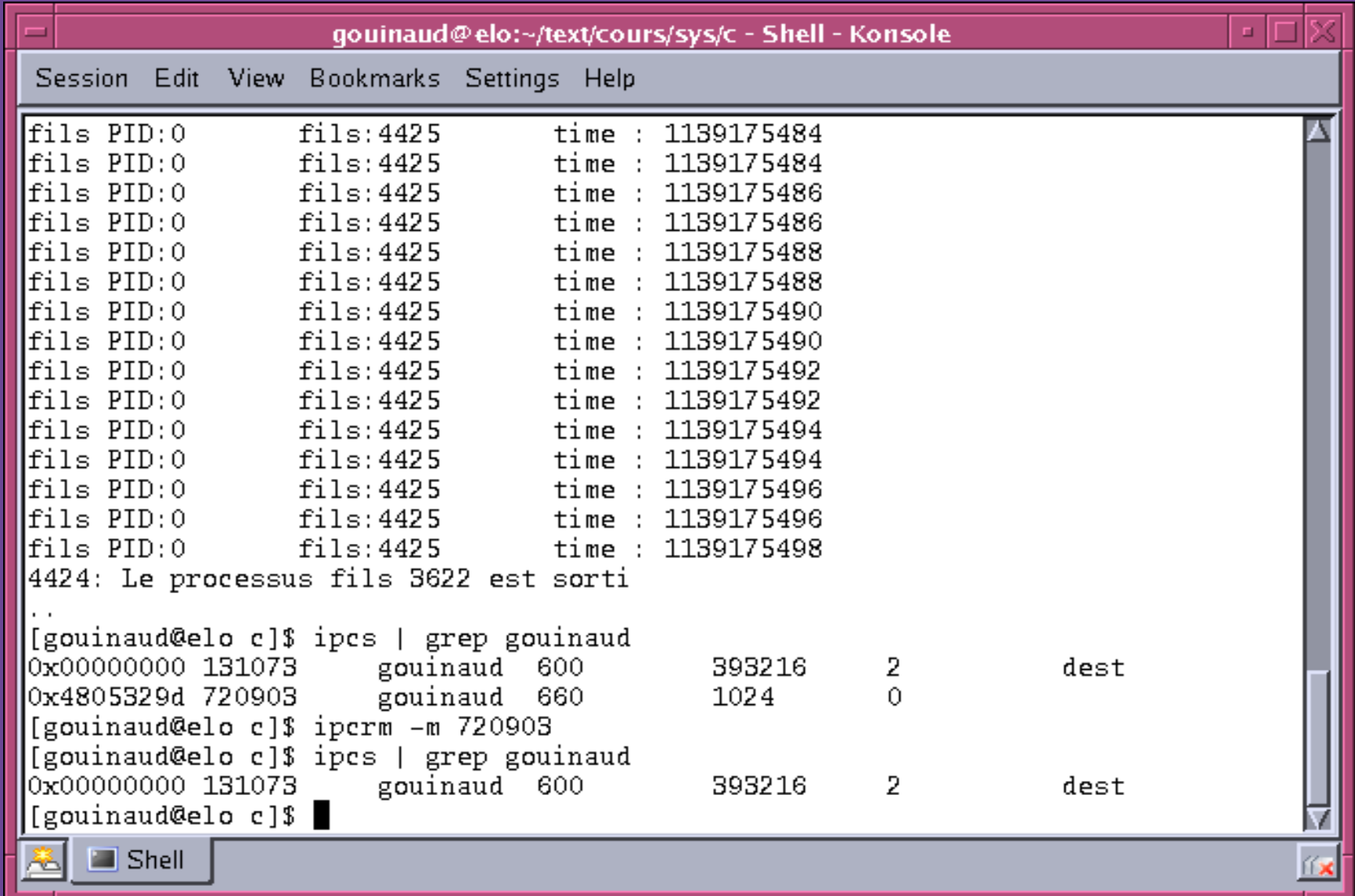
----- Shared Memory Segments -----
key          shmid    owner    perms    bytes    nattch   status
0x00000000   294912   root     644      110592    2        dest
0x00000000   131073   gouinaud 600      393216    2        dest
0x00000000   163842   root     644      122880    4        dest
0x00000000   196611   root     644      110592    2        dest
0x00000000   229380   root     644      122880    2        dest
0x00000000   262149   root     644      122880    2        dest
0x00000000   622598   gouinaud 666      53248     1        dest
0x00000000   688135   root     644      110592    2        dest

----- Semaphore Arrays -----
key          semid    owner    perms    nsems
0x00000000   294912   root     644      1
0x00000000   131073   gouinaud 600      1
0x00000000   163842   root     644      1
0x00000000   196611   root     644      1
0x00000000   229380   root     644      1
0x00000000   262149   root     644      1
0x00000000   622598   gouinaud 666      1
0x00000000   688135   root     644      1

----- Message Queues -----
key          msqid    owner    perms    used-bytes   messages
0x00000000   294912   root     644      0              0
0x00000000   131073   gouinaud 600      0              0
0x00000000   163842   root     644      0              0
0x00000000   196611   root     644      0              0
0x00000000   229380   root     644      0              0
0x00000000   262149   root     644      0              0
0x00000000   622598   gouinaud 666      0              0
0x00000000   688135   root     644      0              0

[gouinaud@elo sys]$
```

Exemple ipcs et ipcrm



The screenshot shows a terminal window titled "gouinaud@elo:~/text/cours/sys/c - Shell - Konsole". The window contains the following text:

```
Session Edit View Bookmarks Settings Help

fils PID:0      fils:4425      time : 1139175484
fils PID:0      fils:4425      time : 1139175484
fils PID:0      fils:4425      time : 1139175486
fils PID:0      fils:4425      time : 1139175486
fils PID:0      fils:4425      time : 1139175488
fils PID:0      fils:4425      time : 1139175488
fils PID:0      fils:4425      time : 1139175490
fils PID:0      fils:4425      time : 1139175490
fils PID:0      fils:4425      time : 1139175492
fils PID:0      fils:4425      time : 1139175492
fils PID:0      fils:4425      time : 1139175494
fils PID:0      fils:4425      time : 1139175494
fils PID:0      fils:4425      time : 1139175496
fils PID:0      fils:4425      time : 1139175496
fils PID:0      fils:4425      time : 1139175498
4424: Le processus fils 3622 est sorti
..
[gouinaud@elo c]$ ipcs | grep gouinaud
0x00000000 131073      gouinaud  600      393216      2      dest
0x4805329d 720903      gouinaud  660      1024      0
[gouinaud@elo c]$ ipcrm -m 720903
[gouinaud@elo c]$ ipcs | grep gouinaud
0x00000000 131073      gouinaud  600      393216      2      dest
[gouinaud@elo c]$
```

The terminal window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The status bar at the bottom shows a "Shell" icon and the text "Shell".

Files de messages

- Queue d'attente
- Pile lilo
- Message sérialisé et typé
- Multiple écouteur
- Peu utilisé car sémaphore + shared memory sont plus efficace



Emplois des files de messages

- Création

```
msgid = msgget ( key, flags )
```

- Ecriture

```
int msgsnd( msgid, msgbuf, size, flags )
```

- Lecture

```
int msgrcv( msgid, msgbuf, maxsize, type, flags )
```

- Destruction

```
int msgctl( msgid, command, buffer );
```

```
#define CLEF_REQUETES      123456
#define CLEF_REPONSES     123457
#define LG_MAX             512
struct msgform
{
    long mtype;
    char mtext[ LG_MAX ];
}
msg;
```

Exemple de message

```
void main(void)
{
    int res;
    int frequete, freponse;
    frequete = msgget(CLEF_REQUETES, 0700 | IPC_CREAT);
    if (frequete == -1) { perror("msgget"); exit(0); }
    freponse = msgget(CLEF_REPONSES, 0700 | IPC_CREAT);
    if (freponse == -1) { perror("msgget"); exit(0); }
    msg.mtype = getpid();
    strcpy(msg.mtext, "Hello");
    res = msgsnd(frequete, & msg, strlen(msg.mtext) + 1, 0);
    if (res == -1) { perror("msgsnd"); exit(0); }
    res = msgrcv(freponse, & msg, LG_MAX, getpid(), 0);
    if (res == -1) { perror("msgrcv"); exit(0); }
    printf("résultat : %s\n", msg.mtext);
    exit(0);
}
```

Conclusion IPC

- Trois mode de communication
 - Une façon de trouver les info `ftok()`
 - Fournis un mécanisme bas niveau
 - Nécessite un protocole
 - Attention à ne pas saturer
 - Revient à la mode
 - Versions Posix peu pratique ...
-
-

Leçon numéro 3 Fichiers ...

- Création et gestion des fichiers temporaire
- Memory-mapping et partage
- Fonction bas niveau sur les fichiers
 - Droit
 - Stat, unlink
- Gestion des verrous locaux
- Gestion des verrous à travers le réseaux



Fichiers temporaire !

- Sous UNIX

`FILE *tmpfile(void);`

créer et effacer en même temps dans /tmp

- Sous Windows yapas (je sais c'est naze)

`GetTempPath(512, sTmpPath)`

`GetTempFileName(sTmpPath, sPrefix, 0, sTmpName)`

et tu créer un fichier ordinaire, et tu l'efface !

`file.DeleteFile (filespec[, force]);`

Memory mapping

- Rappel : un pointeur = un fichier
- Partager entre les process
- Extra rapide
- Sous UNIX :

```
pa = mmap(addr, len, prot, flags, fildes, off);
```
- Sous Windows :

Voir la page suivante

Exemple Memory Mapping

```
if ( (fd = open("toto",
O_RDWR)) < 0)
    err_sys("open error");
if ( (area = mmap(0, SIZE,
    PROT_READ |
    PROT_WRITE,
    MAP_SHARED,
    fd, 0))
    == NULL)
    err_sys("mmap error");

area[2]=1; ...

close(fd); // can close
```

```
MapObject =
CreateFileMapping(
INVALID_HANDLE_VALUE, /
NULL, // no security
PAGE_READWRITE, //rw
0, // offset
SHMEMSIZE, // taille
"toto");

if (MapObject != NULL) {
    area = MapViewOfFile(
        MapObject,
        FILE_MAP_WRITE,
        0, 0, // offset ...
        0); // tout le fichier
}
```


Paramètre mmap Unix

mmap(addr, len, prot, flags, fildes, off)

- Paramètre prot

PROT_EXEC Pages may be executed.

PROT_READ Pages may be read.

PROT_WRITE Pages may be written.

PROT_NONE Pages may not be accessed.

- Paramètre flag

MAP_FIXED adresse spécifié multiple d'une page

MAP_SHARED passe les fork()

appel de msync(2) ou munmap(2) pour update

MAP_PRIVATE écriture fictive ...

Gestions bas niveau – catalogue unix

- Basés sur la structure dirent

```
struct dirent{  
    long d_ino;    //inode  
    off_t d_off;   //position dans le fichier dir  
    unsigned short d_reclen; //longueyr d_name  
    char d_name [NAME_MAX+1]; // file name  
}
```

- Le reste est dans les inodes

Catalogue unix 2

- Ouvrir un répertoire

```
DIR *opendir(const char *name);
```

- Obtenir une entrée

```
struct dirent *readdir(DIR *dir);
```

- Ou est on ?

```
char *getcwd(char *buf, size_t size);
```

```
struct dirent *readdir(DIR *dir);
```

- Comment on y va

```
int chdir(const char *path);
```

J'affirme que :

Celui qui fait :

```
system(« /bin/ls > /tmp/ls.txt »)
```

est une grosse :



Pourquoi ?

Fichiers Unix près du système

- Effacer un fichier
`int unlink(const char *pathname);`
- Ajouter une ref
`int link(char *oldpath, const char *newpath);`
- Lire les attributs d'un fichier
`int stat(char *file_name, struct stat *buf);`
`int fstat(int filedes, struct stat *buf);`
- Sous linux commandes shell !

Structure stat

```
struct stat {  
    dev_t      st_dev;      /* device */  
    ino_t      st_ino;      /* inode */  
    mode_t     st_mode;     /* protection */  
    nlink_t    st_nlink;    /* number of hard links */  
    uid_t      st_uid;      /* user ID */  
    gid_t      st_gid;      /* group ID */  
    dev_t      st_rdev;     /* si device le type */  
    off_t      st_size;     /* taille total en octets */  
    blksize_t  st_blksize;  /* blocksize for filesystem I/O */  
    blkcnt_t   st_blocks;   /* number de blocks alloués */  
    time_t     st_atime;    /* time dernier accès*/  
    time_t     st_mtime;    /* time dernier changement */  
    time_t     st_ctime;    /* date de modif de l'inode */  
};
```

Bas niveaux – Droit UNIX linux

- Propriétaire et groupe

```
int chown(const char *path, uid_t owner, gid_t group);
```

```
int fchown(int fd, uid_t owner, gid_t group);
```

```
int lchown(const char *path, uid_t owner, gid_t group);
```

- Droit

```
int chmod(const char *path, mode_t mode);
```

```
int fchmod(int fildes, mode_t mode);
```

- Le retour du concombre umasqué

```
mode_t umask(mode_t mask);
```

API WIN 32 fichiers

Win32	Unix	
CreateFile	Open	Créer ou ouvrir un fichier
DeleteFile	Unlink	Détruire un fichier
CloseHandle	Close	Fermé un fichier
ReadFile	Read	Lire des données
WriteFile	Write	Ecrire des données
SetFilePointeur	Lseek	Positionnement
GetFileAttributes	Stat	Lecture des propriétés
LockFile	Fnctl	Verrouillage
Unlockfile	Fnctl	Déverrouillage

Api win32 directory

Api win32	Unix	
CreateDirectory	Mkdir	Créer un répertoire
RemoveDirectory	Rmdir	Effacer un répertoire
FindFirstFile	Opendir	Ouverture répertoire
FindNextFile	Readdir	Lire une entrée
MoveFile	Rename	Renommé un fichier
SetCurrentDirec...	Chdir	Changement du WD

```
BOOL CopyFile(  
    LPCTSTR lpExistingFileName,  
    LPCTSTR lpNewFileName,  
    BOOL bFailIfExists  
);
```

Verrouillage de fichier

Problème non trivial :

- Préemptif ?
- Fichiers en réseau ?
- Tout ou partie
- Risque de dealock
- Verrouillage conjoint

Nécessaire mais rarement fait !



Verrouillage Unix simple

```
void verouille_base(dir)
char *dir;
{
char buffer[512];

strcpy(buffer,dir);
strcat(buffer,"base.lock");

verrou = open(buffer,O_RDWR);

if (verrou==-1)
    progerr("Impossible ");

flock(verrou,LOCK_EX);

} /* fin verrouille base */
```

```
void deverouille_base()
{

flock(verrou,LOCK_UN);

close(verrou);

} /* fin verrouille base */
```

Verrouillage en réseaux

Problème des états du serveur :

- Protocole sans états

Le serveur ne connaît pas l'état du fichier sur le client

- Protocole avec états

Le fichiers est ouvert sur le client et sur le serveur



Différence UNIX-Windows

- Unix NFS protocole sans états

basé sur le daemon rpc.statd appelé en RPC
verrouillage externe au protocole

- Windows CIFS (ex SMB) protocole à états

Système de redirection complet

Pas d'état sur le client



Utilisation de fcntl

Fonction à trois formes :

```
int fcntl(int fd, int cmd);  
int fcntl(int fd, int cmd, long arg);  
int fcntl(int fd, int cmd, struct flock *lock);
```

Permet de manipuler la structure « pointer »
par fd après ouverture d'un fichier

Marche aussi sur d'autre type d'objet
(socket, pipe)

Verouillage avec *fnctl*

```
int fnctl(int fd, int cmd, struct flock *lock);
```

- F_SETLK

Créer ou défait un verrou
pas d'attente

- F_SETLKW

Demande avant de faire
attente bloquante

- F_GETLK

Demande seulement

```
struct flock { ...  
    short l_type;  
    /* Type : F_RDLCK,  
    F_WRLCK, F_UNLCK */  
    short l_whence;  
    /* start: SEEK_SET,  
    SEEK_CUR, SEEK_END */  
    off_t l_start; /* Debut */  
    off_t l_len;   /* taille */  
    pid_t l_pid;   /* PID  
    du process qui lock  
    (en cas de F_GETLK only) */  
    ...};
```

▢ *Modif ou lecture des flags*

Utilisation de la deuxième forme de fcntl :

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

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

- F_GETFL retourne les flags
permet de savoir après exec ou entre thread
- F_SETFL change les flags
rendre ouvrable, écrire à la fin :
fcntl(fd, F_SETFL, O_APPEND)

Faire man open une fois dans votre vie !

Verouillage POSIX

- Fonction lockf :
`int lockf(int fd, int cmd, off_t len);`
Cmd = F_LOCK, F_TLOCK, F_ULOCK, F_TEST
par de la position courante
 - Portable sous windows
 - Passe entre les réseaux
 - Détecte les interblocages
-
-

Interblockage simple :

A run

B run

A run

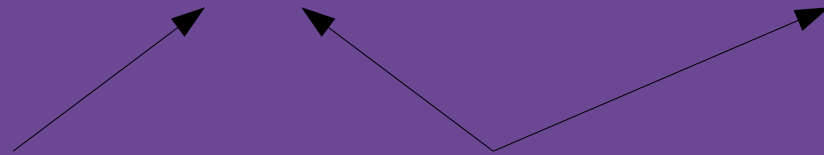
B iow

A iow

B iow

Section 1

Section 2



Process A lock
section 1

Process B lock
section 2
Process B lock
Section 1

Process A lock
section 2



Fonctions pratiques

- Madvise

```
int madvise(void *start, size_t length, int  
advice);
```

- MADV_NORMAL
- MADV_RANDOM
- MADV_SEQUENTIAL
- MADV_WILLNEED on va les demander
- MADV_DONTNEED

- Truncate

```
int truncate(const char *path, off_t length);  
int ftruncate(int fd, off_t length);
```

Leçon numéro 4 :

commandes système pratique



Strace – truss Affiche les appels système

```
open("/etc/group", O_RDONLY)          = 4
fcntl64(4, F_GETFD)                  = 0
fcntl64(4, F_SETFD, FD_CLOEXEC)      = 0
fstat64(4, {st_mode=S_IFREG|0644, st_size=701, ...}) = 0
read(4, "root:x:0:root\nbin:x:1:root,bin,d"... , 4096) = 701
close(4)                             = 0
open("/mnt", O_RDONLY|O_NONBLOCK|
      O_LARGEFILE|O_DIRECTORY) = 4
fstat64(4, {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
fcntl64(4, F_SETFD, FD_CLOEXEC)      = 0
getdents64(4, /* 5 entries */, 4096)  = 120
getdents64(4, /* 0 entries */, 4096)  = 0
close(4)                             = 0
fstat64(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(136,
2), ...}) = 0
write(1, "dk sk\ttest\n", 12dk sk    test)      = 12
```

Lsof

Affiche les descripteurs ouverts

```
[gouinaud@elo sys]$ /usr/sbin/lsof progsys.odp
```

COMMAND	PID	USER	FD	TYPE	DEVICE
---------	-----	------	----	------	--------

SIZE	NODE	NAME
------	------	------

soffice.b	3557	gouinaud	28uW	REG	3,7 204823
519280		progsys.odp			

```
[gouinaud@elo sys]$ /usr/sbin/lsof /tmp/ksocket-gouinaud/kdeinit__0
```

COMMAND	PID	USER	FD	TYPE	DEVICE
---------	-----	------	----	------	--------

SIZE	NODE	NAME
------	------	------

kdeinit	3439	gouinaud	7u	unix	0x19acddc0
---------	------	----------	----	------	------------

12378		/tmp/ksocket-gouinaud/kdeinit__0			
-------	--	----------------------------------	--	--	--

kdeinit	3466	gouinaud	5u	unix	0x13677700
---------	------	----------	----	------	------------

12678		/tmp/ksocket-gouinaud/kdeinit__0			
-------	--	----------------------------------	--	--	--

fsuser

Recherche de processus

Deux modes d'utilisation :

```
[gouinaud@elo sys]$ /sbin/fuser progsys.sxi
```

```
progsys.sxi:      3557
```

```
[gouinaud@elo sys]$ /sbin/fuser -k progsys.sxi
```

```
progsys.sxi:      3557
```

```
[gouinaud@elo sys]$ /usr/local/bin/soffice: line 224:
```

```
3557 Killed      "$sd_prog/$sd_binary" "$@"
```

```
[1] Done          soffice progsys.odp
```

```
[gouinaud@elo sys]$
```

```
[gouinaud@elo sys]$ /sbin/fuser -k progsys.sxi
```