

TD 3 : Gestion des processus

Création

Permet la création dynamique d'un nouveau processus qui s'exécute de façon concurrente avec le processus qui l'a créé.

Un appel par un processus, que nous appellerons processus père, à la fonction fork entraîne, si cela est possible, la création d'un nouveau processus fils. La syntaxe est la suivante :

```
#include <sys/times.h>

pid_t fork(void);
```

Le programme exécuté par les deux processus étant le même et les données étant identiques, il est nécessaire, pour que le comportement des deux processus ne soit pas identique dans la suite de leur exécution, de pouvoir distinguer dans le code le retour dans le processus père de celui dans le processus fils.

La valeur de retour de la fonction est :

0, dans le processus fils;

l'identité du processus fils créé, dans le processus père.

-1, si la primitive échoue (et qu'il n'y a pas donc de création d'un nouveau processus, comme dans le cas où l'utilisateur créé trop de processus ou si le nombre total de processus dans le système est trop élevé).

getpid	identité du processus en cours
getppid	identité du processus parent de celui en cours
getuid	propriétaire réel : en général celui du login
geteuid	propriétaire effectif : propriétaire du processus pour lequel le bit set-uid a été modifié pour permettre à un exécutable d'être exécuter par un autre utilisateur.
set-uid	est le bit à changer pour changer d'utilisateur en utilisant la fonction setuid
getgid	groupe réel
getegid	groupe propriétaire effectif
getpwn	répertoire de travail

Exemple:

```
#include <stdio.h>
#include <sys/times.h>
main(){ if (fork()==0) {printf("fin du processus fils de numéro %d \n ", getpid());
                    exit(2); }
    sleep(30);}
```

Les signaux

La signalisation constitue le mécanisme fondamental de communication non seulement entre processus mais également entre le noyau et l'ensemble des processus du système.

Un signal est défini par un code numérique (dans le fichier /usr/include/sys/signal.h) et se manifeste de manière asynchrone par notification d'un événement au processus concerné; cet événement est mémorisé dans la table des processus.

Contrairement aux autres mécanismes de communication interprocessus, les signaux ne transportent pas de données, mais indiquent des actions à effectuer dans certaines circonstances.

Types de signaux :

NOM	N°	FONCTION
SIGHUP	1	Signal hangup envoyé à tous les processus liés à un terminal, lors de la déconnexion de celui-ci (CTRL-D)
SIGINT	2	Interruption des processus d'un terminal (CTRL-C, BREAK, DEL)
SIGQUIT	3*	Interruption avec image mémoire : core (CTRL \)
SIGILL	4*	Instruction illégale
SIGTRAP	5*	Trappe envoyée à un processus en mode trace après exécution de chaque instruction
SIGIOT	6*	Instruction IOT : problème matériel
SIGEMT	7*	Instruction EMT : problème matériel
SIGFPE	8*	Exception de virgule flottante
SIGKILL	9	Permet de tuer un processus de façon brutale; ne peut être capté ni ignoré pour des priorités > PZERO=25
SIGBUS	10*	Erreur BUS
SIGSEGV	11*	Violation de segment
SIGSYS	12*	Mauvais argument dans un appel système
SIGPIPE	13	Ecriture dans un pipe sans lecteur
SIGALARM	14	Alarme horloge
SIGTERM	15	Fin normale d'un processus (kill)
SIGUSR1	16	Signal à la disposition de l'utilisateur (signal 1)
SIGUSR2	17	Signal à la disposition de l'utilisateur (signal 2)
SIGCLD	18	Mort d'un fils signalée au processus père
SIGPWR	19	Redémarrage après coupure de courant
SIGPOLL	22	Sélection d'événements dans les streams (événements multiples asynchrones)

* production de l'image mémoire sur disque (core dump)

Emission d'un signal

Cette primitive permet d'envoyer un signal à un processus ou à un groupe de processus.

```
#include <sys/signal.h>
int kill(pid, sig)
int pid, sig;
```

La commande *kill* permet d'envoyer un signal à un processus dont on connaît le numéro (il est facile de le déterminer grâce à la commande *ps*) :

```
kill -HUP 1664
```

Ici, on envoie le signal SIGHUP au processus numéro 1664 (notez qu'en utilisant la commande *kill*, on écrit le nom du signal sous forme abrégée, sans le SIG initial).

On aurait également pu utiliser le numéro du signal plutôt que son nom :

```
kill -1 1664
```

On envoie le signal numéro 1 (SIGHUP) au processus numéro 1664

Les entiers pid et sig désignent respectivement l'identificateur du processus destinataire ou groupe de processus, et le numéro du signal émis. Les processus émetteurs et destinataire doivent avoir le même identificateur (réel ou effectif) d'utilisateur.

- * pid > 0: le signal est destiné au processus dont le PID = pid.
- * pid = 0 : le signal est destiné à tous les processus de même groupe que le processus émetteur à l'exception des processus spéciaux : init, swapper, sched ...
- * pid = 1: et l'euid n'est pas celui de su le signal est envoyé à tous les processus (hormis les spéciaux) dont l'uid = euid émetteur.
- * pid = -1: et euid = root le signal est envoyé à tous les processus sauf les processus spéciaux.
- * pid < -1: le signal est envoyé à tous les processus dont l'identificateur de groupe de processus est égal à la valeur absolue de pid. Kill retourne un 0 s'il n'y a pas d'erreur et -1 dans le cas d'erreur.

Réception d'un signal par un processus

Le type de signal reçu par un processus indique la nature de l'événement qui a nécessité l'envoi de ce signal par un autre processus.

```
#include <sys/signal.h>
int (*signal(sig, func))()
int sig, (*func)();
```

La valeur de sig désigne le numéro du signal à recevoir. A la réception d'un signal par le processus, trois actions sont possibles selon la valeur de l'argument func (flags SIG-DFL et SIG-IGN ou une routine) :

Action par défaut : SIG-DFL; c'est la mort du processus

Signal ignoré : SIG-IGN; le signal reçu est ignoré par le processus, sauf le cas SIGKILL. Par exemple un processus peut ignorer le signal SIGCLD émis à la mort d'un fils:

```
signal(SIGCLD, SIG_IGN)
```

Exemple : Un processus n'attendant pas la mort du fils et ignorant le signal SIGCLD

```
#include <sys/signal.h>
main(){signal(SIGCLD, SIG_IGN); /* SIGCLD ignoré */
    if (fork()==0)
        { printf("processus fils PID= %d\n", getpid());
          exit(1);}
    printf("processus père PID=%d\n", getpid());
    sleep(30);
    printf("signal mort fils ignoré \n ");
    for(;;);
}
```