جامعة ابن طفيل Université Ibn Tofail
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵅⵓ ⵟⵓⴼⴰⵢⵍ Ibn Tofail University

Système d'Exploitation
Signaux

Med. AMNAI
Filière SMI-S4
Département d'Informatique

2023-2024

Plan

Plan

Plan

Plan

Problématque ?

- La coopération entre les processus est nécessaire pour résoudre des problèmes ;
- Ces processus s'exécutent en parallèle sur un même ordinateur, ou bien sur des ordinateurs différents ;
- Communiquer pour synchroniser ou pour communiquer de l'information ;
- Besoin d'échanger et de partager les informations.

Besoin ==> Communication Inter Processus (CIP).
Solution ==> Les variables communes, les fichiers communs, les signaux, les messages et les tubes de communication.

Partage de données entre processus

Le partage des données entre les processus peut être distinguer selon deux formes :

- Des variables ou segments de données : **Espace mémoire** ;
- Des fichiers : **Espace disque** ;

Problématique ==> Accès concurrents.

Moyens de communication Interprocessus

Les moyens de communication interprocessus offerts par le système :

- **Signaux** ;
- Envois de messages par le biais de **tubes** (**pipe** ou **tuyau**) ;

Définition

- Les **interruptions logicielles** ou **signaux** sont utilisées par le système d'exploitation pour aviser les processus utilisateurs de l'occurrence d'un événement important ;
- Un signal est un moyen de communication **indiquant une action à entreprendre** à partir de conventions préétablies ;
- Chaque signal a une **signification** particulière qui détermine le **comportement** du processus (aucun échange d'information).
- Chaque signal est distingué par :
  - Un **numéro** entier
  - Un **nom symbolique** décrit dans **/usr/include/signal.h**

RQ : La commande **kill –l** permet d'afficher la liste des signaux.

Liste des signaux

On peut dégager un grand nombre de signaux communs à toutes les versions d'Unix :

Signal	Numéro	Description
SIGHUP	1	Rupture de ligne téléphonique.
SIGINT	2	Signal envoyé quand on tape Control-C
SIGFPE	8	Erreur de calcul (division par zéro)
SIGKILL	9	Tue le processus
SIGSEGV	11	Référence mémoire invalide (pointeur nul)
SIGPIPE	13	Tentative d'écriture dans un tuyau sans lecteur
SIGALRM	14	Alarme (chronomètre)
SIGTERM	15	Demande au processus de se terminer proprement.
SIGCHLD	20,17,18	Informe père qu'un des fils vient de se terminer
SIGUSR1	30,10,16	Signal utilisateur 1

Envoi d'un signal

Un signal peut être envoyé :

- Lors de la constations d'une **anomalie** matérielle.
- Suite à la frappe d'une **combinaison de touches** (par ex. **CTRL-C** pour envoyer le signal **SIGINT**) ;
- Par un autre processus utilisant la **commande kill** ou l'appel système **kill**.

Réception d'un signal

- Réaction du processus suite à la réception d'un signal :
  - **Ignorer** le signal ;
  - Le signal provoque l'**arrêt** du processus ;
  - **Appeler une procédure** utilisateur ;
- **RQ** : Lorsqu'un processus reçoit un signal pour lequel **il n'a pas indiqué de fonction de traitement**, par défaut le système d'exploitation réagit suivant les signaux :
  - soit, il **ignore** le signal ;
  - soit, il **termine** le processus.

Envoi Depuis un Terminal

Après avoir déterminer le **PID** d'un processus avec la commande **ps**, la commande KILL permet d'envoyer un signal en utilisant son **PID**.

```
KILL  - HUP  555
```

Ici, envoi du signal **SIGHUP** au processus numéro **555**.

RQ : Le numéro de signal peut changer, il vaux mieux d'utiliser le nome du signal au lieu de son numéro.

```
KILL  - 1  555
```

Appel Système kill()

int kill (pid_t pid, int signal)

- **kill()** envoi le signal numéro "**signal**" au processus "**pid**". En cas d'**erreur**, il retourne **-1** et **0** autrement.
- **RQ** : Un processus ne peut envoyer un signal à un autre que si le **propriétaire** (réel ou effectif) est le **même** pour les deux processus.

Exemple kill()

```
#include <sys/types.h>
#include <signal.h>

/* ... */

pid_t pid = 1664;

/* ... */

if ( kill(pid, SIGHUP) == -1 ) {

        /* erreur : le signal n'a pas pu etre envoye */

}
```

Appel système signal()

signal () permet d'envoyer un signal d'un processus à un autre.

- **signal (SIGINT, Faire)** : la fonction **Faire()** est exécutée à la réception du signal SIGINT. (avec void Faire(int sig))
- **Ex : signal (SIGINT, SIG_IGN)** : la réception de SIGINT devient inopérante ; le signal est **ignoré** ;

Exemple signal()

```c
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
void sighandler( int signum ) ;

int main( void ){

char buffer[ 256 ] ;
if ( signal(SIGTERM,&sighandler ) == SIG_ERR )
{
    printf( "Ne peut pas manipuler le signal\n" ) ;
    exit( 1 ) ;
}
while( 1 ){
    fgets( buffer , sizeof ( buffer ) , stdin );
    printf( " Input : %s " , buffer ) ;
    }
return 0 ;
}
void sighandler( int signum){
printf ( "Masquage du signal SIGTERM\n" ) ;
}
```

Exemple signal() (suite)

Exécution dans deux terminaux, afin de montrer le masquage de signaux et d'essayer de tuer le processus.

```
[amnai@localhost sig]$ ./sig1
salut
 Input : salut
 tout
 Input : tout
 le monde
 Input : le monde
 Masquage du signal SIGTERM
toto
 Input : toto
 titi
 Input : titi
```

Processus 1 qui exécute notre programme en **sig1.c** (terminal **1**)

```
 3185 pts/1    00:00:00 bash
 3200 pts/1    00:00:00 vim
 3213 pts/2    00:00:00 bash
 7492 pts/0    00:00:00 sig1
 7499 pts/2    00:00:00 ps
[amnai@localhost sig]$ kill 7492
```

Processus 2 qui essaye de tuer le processus 1 (terminal **2**)

Exemple signal() (suite)

```
[amnai@localhost sig]$ ./sig1
salut
 Input : salut
 tout
 Input : tout
 le monde
 Input : le monde
 Masquage du signal SIGTERM
toto
 Input : toto
 titi
 Input : titi
 Masquage du signal SIGTERM
koko
 Input : koko
 Processus arrêté
[amnai@localhost sig]$
```

```
 3185 pts/1    00:00:00 bash
 3200 pts/1    00:00:00 vim
 3213 pts/2    00:00:00 bash
 7492 pts/0    00:00:00 sig1
 7499 pts/2    00:00:00 ps
[amnai@localhost sig]$ kill 7492
[amnai@localhost sig]$
[amnai@localhost sig]$ kill -SIGTERM 7492
[amnai@localhost sig]$ kill -9 7492
```

Attente active

```
#include<stdio.h>//sig2.c
#include<signal.h>
int num_signal = 0 ;
void traiter_SIGINT()
{
    num_signal ++;
    printf("Signal CTRL  C capture !\n");
}
int main()
{
    if( signal ( SIGINT , & traiter_SIGINT ) == SIG_ERR ){
    printf ( " Erreur dans le gestionnaire\n" ) ;
    exit( 1 ) ;
}
while( num_signal < 5 ); // attente active
printf( "\n%d signaux SIGINT sont arrives\n" , num_signal );
return 0 ;
}
```

```
[amnai@localhost sig]$ ./sig2bis
^CSignal CTRL  C capture !
^CSignal CTRL  C capture !
^CSignal CTRL  C capture !
^CSignal CTRL  C capture !
^CSignal CTRL  C capture !

5 signaux SIGINT sont arrives
[amnai@localhost sig]$
```

Attente non active : pause()

pause (void) suspend (Attente non active) le processus courant jusqu'à réception d'un signal quelconque non ignoré.

```
#include<stdio.h>//sig2bis.c
#include<signal.h>
int num_signal = 0 ;
void traiter_SIGINT()
{
    num_signal ++;
    printf("Signal CTRL  C capture !\n");
}

int main()
{
    if( signal ( SIGINT , & traiter_SIGINT ) == SIG_ERR ){
    printf ( " Erreur dans le gestionnaire\n" ) ;
    exit( 1 ) ;
}
while( num_signal < 5 ); pause(); // attente  non active
printf( "\n%d signaux SIGINT sont arrives\n" , num_signal );
return 0 ;
```

```
[amnai@localhost sig]$ ./sig2bis
^CSignal CTRL  C capture !
^CSignal CTRL  C capture !
^CSignal CTRL  C capture !
^CSignal CTRL  C capture !
^CSignal CTRL  C capture !
^CSignal CTRL  C capture !

6 signaux SIGINT sont arrives
[amnai@localhost sig]$
```

Appel système alarm()

L'appel système **alarm** active une horloge qui déclenche un signal **SIGALRM** au bout du nombre de secondes demandé en paramètre.

```
void chrono(int sig){
        printf("Bien reçu %d \n", sig);
        printf("!!Dring dring.....Temps écoulé \n");
}

int main(){

printf ( " Test appel systeme alarme \n" ) ;
signal(SIGALRM, chrono);
printf( "\nReveil dans 10 secondes");

alarm(10);
sleep(13);
}
```

```
test appel systeme alarm
reveil dans 10 secondes
Bien reçu 14
Dring dring ... temps écoulé
```

Appel système sigaction()

int sigaction (int signo, const struct sigaction *act, struct sigaction *oldact) ;

L'appel système **sigaction** sert à modifier l'action effectuée par un processus à la réception d'un signal spécifique :

- **signo** : signal concerné (à l'execption de SIGKILL et SIGSTOP) ;
- si **act #NULL** : La nouvelle action pour le signal **signo** est définie par **act** ;
- si **oldact #NULL** : L'ancienne action est sauvegardée dans **oldact** ;
- V. Retour : **0** en cas de succès sinon **-1** .

```
struct sigaction {
    void (*sa_handler)();      // fonction de traitement (ou SIG DFL et SIG IGN)
    sigset_t        sa_mask;   // signaux additionnels à bloquer pendant le traitement
    int             sa_flags;  // options (SA_RESTART, SA_NOCLDWAIT,
                               // SA_NODEFER, SA_NORESETHAND, ... )
    }
```

Exemple sigaction()/1

```c
void TraiteSignal(int sig) {
    printf("Reception du signal numero %d.\n", sig);
}

int main(int argc, char *argv[]) {

struct sigaction act;

/* une affectation de pointeur de fonction */
/* c'est equivalent d'ecrire act.sa_handler = &TraiteSignal */
act.sa_handler = TraiteSignal;

/* le masque (ensemble) des signaux non pris en compte est mis */
/* a l'ensemble vide (aucun signal n'est ignore) */
sigemptyset(&act.sa_mask);

/* Les appels systemes interrompus par un signal */
/* seront repris au retour du gestionnaire de signal */
act.sa_flags = SA_RESTART;

/* enregistrement de la reaction au SIGUSR1 */
if ( sigaction(SIGUSR1,&act,NULL) == -1 ) {
/* perror permet d'afficher la chaine avec */
/* le message d'erreur de la derniere commande */
    perror("sigaction");
    exit(EXIT_FAILURE);
}
printf("Je suis le processus numero %d.\n" ,getpid());
```

Exemple sigaction()/2

```
for(;;) { /* boucle infinie equivalente a while (1) */
    sigset_t sigmask; /* variable locale a cette boucle */
    sigemptyset(&sigmask); /* mask = ensemble vide */

    /* on interromp le processus jusqu'a l'arrivee d'un signal */
    /* (mask s'il n'etait pas vide correspondrait aux signaux ignores) */
    sigsuspend(&sigmask);

    printf("Je viens de recevoir un signal et de le traiter\n");
}

exit(EXIT_SUCCESS);
}
```

```
[amnai@localhost signaux]$ ./sigaction
Je suis le processus numero 8218.
Reception du signal numero 10.
Je viens de recevoir un signal et de le traiter
```

```
amnai     8218  0.0  0.0   1908   240 pts/1    S+   11:50   0:00 ./sigaction
amnai     8219  2.0  0.0   4796  1036 pts/2    R+   11:51   0:00 ps -au
[amnai@localhost signaux]$
[amnai@localhost signaux]$ kill -SIGUSR1 8218
```