

Cours de Système Semestre 2

Jean-François Pradat Peyre et Emmanuel Hyon

Jean-François.Pradat-Peyre@u-paris10.fr et Emmanuel.Hyon@u-paris10.fr

Université Paris Ouest Nanterre

Mars 2013

Les Signaux

Coexistence de plusieurs flots (processus)

- Il n'y a pas uniquement du séquentiel.
- Ces flots partagent des accès à une ressource commune.
 - ▶ Périphériques
 - ▶ Processeur(s)
 - ▶ données
- Cette concurrence peut être disjointe (pas d'inter-actions) ou non.

L'indication d'un changement d'état du processus au cours de l'exécution est appelée **événement**.

besoin de signaler à d'autres processus ces événements

Buts

Signal : moyen d'indiquer un événement ou de synchroniser des processus.

Buts

Signal : moyen d'indiquer un événement ou de synchroniser des processus.

Evénements

- externe au processus :
terminaison d'un fils, occurrence d'une interruption (par exemple), etc...
- interne au processus :
en cas d'erreurs : instruction illégale, violation de segment , etc.

Buts

Signal : moyen d'indiquer un événement ou de synchroniser des processus.

Evénements

- externe au processus :
terminaison d'un fils, occurrence d'une interruption (par exemple), etc...
- interne au processus :
en cas d'erreurs : instruction illégale, violation de segment , etc.

Synchronisation

- Fin du déroulement d'un processus
- Plus de lecteurs au bout du pipe

A tout signal est associé :

- Un envoi :
le signal est envoyé à un autre processus pour indiquer l'événement
- Un comportement à suivre dès réception
 - ▶ définit par défaut
 - ▶ définit par utilisateur (handler)
 - ▶ ignorance du signal
- Un armement
Il faut que le processus soit à l'écoute d'un tel signal

Exemple : Terminaison d'un processus

Un événement

Un processus se termine :

- Fin de ses instructions (termine sa demande)
- Suite à l'occurrence d'un événement externe

Signalement

- signalement de l'événement aux fils (SIGHUP)
- signalement de l'événement au père (SIGCHLD)

Conséquence une suite d'actions

La terminaison implique :

- la libération des ressources (mémoire, verrous..);
- la fermeture des fichiers;
- le rattachement des processus fils (orphelins) à l'init
- sauvegarde du code de retour et passage à l'état *zombie*

Envoi d'un signal

Raccourcis clavier

La commande `stty -a` donne la liste des raccourcis clavier associés au signaux (Attention c'est système dépendant).

```
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;  
eol = <undef>; eol2 = <undef>; start = ^Q; stop = ^S;
```

Ligne de commande

```
kill -Numero Pid
```

Dans un code C

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

- `pid > 0`, signal envoyé au processus de ce `pid`
- `pid = 0`, signal envoyé à tous les processus du groupe de l'appelant

Liste des principaux signaux

Commande `kill -l` pour la liste complète (ou fichier *signal.h*)

N°	Nom	Evenement associé	Effet
1)	<i>SIGHUP</i>	Fin de Session	Terminaison
2)	<i>SIGINT</i>	Interruption	Terminaison
3)	<i>SIGQUIT</i>	Caractère Quit	Core Dump
4)	<i>SIGILL</i>	Instruction illégale	Core Dump
8)	<i>SIGFPE</i>	Expression arithmétique faussée	Core Dump
9)	<i>SIGKILL</i>	Terminaison	Terminaison
10)	<i>SIGUSR1</i>	Signal utilisateur	Terminaison
12)	<i>SIGUSR2</i>	Signal utilisateur 2	Terminaison
13)	<i>SIGPIPE</i>	Signal Pipe Brisé	Terminaison
14)	<i>SIGALRM</i>	Alarme d'Horloge	Terminaison
17)	<i>SIGCHLD</i>	Fin Fils	Ignorer

Traitement d'un signal

Définition du traitement

Un **handler** est la fonction qui décrit la suite des instructions à effectuer lors de la réception d'un signal.

- A chaque type de signal est associé un handler par défaut défini dans `SIG_DFL`.
- Un même handler peut être **armé** par plusieurs signaux différents. Par exemple : les signaux 1, 2, 9, 13... pour terminer un processus.

Pas de traitements

- Signal ignoré : on associe au signal un handler vide : `SIG_IGN` ce handler vide est exécuté mais n'a pas d'effet.
- Signal bloqué : aucun handler n'est exécuté. Le signal attend la modification du masque il est dit *pendant*.

Démarche

- Dans le processus faisant interruption
Ecrire l'envoi du signal au moment de l'événement
- Dans le processus recevant le signal
 - ❶ Ecrire le handler qui va décrire le comportement du processus à la réception du signal.
 - ❷ Armement du signal dans la fonction (réarmement à chaque fois) sous UNIX)

Fonctions spécifiques

Fonction qui :

- Déclarée en dehors du `main`
- Pas de valeur de retour
- Invoquée quelque soit l'endroit du code
- Récupère le numéro du signal comme paramètre

Exemple :

```
void fonction_handler (int sig){  
    printf("le signal qui m'a lance est %d",sig);  
}
```

Appel système

Armer un signal c'est dire au système le comportement de celui-ci à la reception du signal. Comportement décrit par une fonction.

L'association fonction \leftrightarrow signal reçu est faite par l'appel système `signal()` (ANSI C) ou l'appel système `sigaction()` (POSIX).

`signal()` (ANSI C)

Son en tête est

```
signal(sig, handler);
```

Dans le cours du programme on aura par exemple :

```
signal (SIGHUP, fonction_handler);  
int pause(void);
```

Attention : Armement est considéré comme une seule instruction.

signal()

handler

Le paramètre **handler** peut prendre une des trois valeurs :

- **SIG_DFL** : un pointeur vers la fonction par défaut SIG_DFL() du système pour gérer le signal indiqué par le paramètre **sig**
- **SIG_IGN** : un pointeur vers la fonction SIG_IGN() qui permet d'ignorer le signal (sauf SIGKILL).
- L'adresse d'une fonction définie par l'utilisateur.

Signature par défaut d'un handler

```
void handler(int sig);
```

Exemple Final

But

A chaque fois qu'on tape **Ctrl C** au clavier *Aie* s'affiche à l'écran.

Le Pgm

```
#include <signal.h>
void fonction_handler(int sig){
    printf(" A cause de %d je dis Aie ! \n",sig);
    return;
}
int main(){
    signal(SIGINT,fonction_handler);
    while(1);
}
```


Envoi d'un signal d'un processus à un autre

Handler et fils

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

void handler(int sig){
    printf("Je veux
           rester dormir\n");
    exit(0);
}

void fonction_fils(){
    signal(SIGUSR,handler);
    while(1);
}
```

Pere

```
int main(){
    int k,pid,status;
    if (k=fork!=0){
        printf("Trompette pour
               les dormeurs\n");
        kill(k,SIGUSR);
        pid=wait(&status)
    }
    else{
        fonction_fils();
    }
    return;
```