GESTION DES SIGNAUX

PLAN

1. Introduction
2. Interruptions
3. Signaux
4. Exemples

INTRODUCTION

- Événements qui arrivent au moment de l'exécution d'un processus.

- Événements d'origine
  - matérielle : Touche clavier, fin papier, trame, …
  - logicielle : erreur d'adressage, dépassement de capacité de la pile (Stack overflow), …

INTRODUCTION

- Prise en compte matérielle de ces événements :
  - Variation d'état d'une des lignes du bus : LIGNE d'INTERRUPTION.
  - Processeur détecte cette variation
  - Le S.E la prend en charge et la fait passer au processus concerné

- Sous Unix, l'outil utilisé pour répercuter les interruptions aux processus s'appelle: signal

INTERRUPTION : ÉMISSION

- **Qui peut émettre une interruption ?**
  - Un périphérique peut émettre une interruption (fin d'une E/S)
  - Horloge (exemple : échéance d'une alarme)
  - etc.
- **Comment émettre une interruption ?**
  - Changement d'état sur une des lignes du processeur.
  - Ce dernier avant, d'exécuter n'importe quelle instruction, vérifie si elle a changé d'état.

INTERRUPTION : FONCTIONNEMENT

1. L'instruction en cours se termine.

2. L'adresse de l'instruction suivante ainsi que le contenu des registres sont sauvegardés dans la pile d'interruption.

3. Les interruptions de niveau inférieur ou égal sont masqués.

4. Le processeur consulte une table contenant l'adresse de la routine à exécuter (en fonction du type d'interruption reçue)

INTERRUPTION : TRAITEMENT

- Programme en cours d'exécution …

  ➔ Interruption

➔SAUVEGARDE DU CONTEXTE

➔EXÉCUTION DE LA ROUTINE D'INTERRUPTION

➔RETOUR ET RESTAURATION DU CONTEXTE

- Reprise du processus interrompu ou autre processus

SIGNAUX: PRÉSENTATION

- Un signal informe sur l'occurrence d'un événement qui peut être :
  - **interne** (provoqué par le processus lui-même)
    - Involontaire (division par zéro, débordement pile, …)
    - Volontaire (gestion d'une alarme, gestion d'une E/S, …)
  - **externe** (provoqué par un autre processus : terminaison d'un processus fils, … ou par le S.E: fin d'une E/S, …)
- Si l'événement est d'origine matérielle, les signaux sont le moyen utilisés par le S.E pour informer les processus concernés de l'apparition de l'interruption

SIGNAUX: ÉMISSION

- **Qui peut émettre un signal ?**
  - Le **noyau** (à la suite d'une Divz, KILL, terminaison d'un processus (émission de SIGCHLD), …)
  - Un **utilisateur** : utilisant le clavier du terminal ou l'utilisation de la commande kill.
  - Un **processus** (appel à kill() ou alarm())

SIGNAUX: ÉMISSION

Il existe un nombre déterminé de signaux (32 sous Linux 2.0, 64 depuis Linux 2.2).

Chaque signal dispose d'un nom défini sous forme de constante symbolique commençant par SIG et d'un numéro associé.

Il n'y a pas de nom pour le signal numéro 0, car cette valeur a un rôle particulier.

Toutes les définitions concernant les signaux se trouent dans le fichier d'en-tête <signal.h>

SIGNAUX: ÉMISSION

- Pour chaque processus, le SE définit un tableau de signaux.

- Pour le processus concerné, le SE positionne un bit dans le tableau spécifique au signal envoyé à ce processus.

SIGHUP	SIGINT	...
0	1	

SIGNAUX: ÉMISSION

- L'appel système **kill (pid_t pid, int sig)**
  permet d'envoyer le signal **sig** au processus identifié par **pid**.

- L'appel système **alarm (unsigned int sec)**
  permet d'envoyer le signal **SIGALRM** au processus courant dans **sec** secondes.

➔ $ man kill et man alarm

SIGNAUX: RÉCEPTION

- La gestion d'un signal dépend de l'état dans lequel se trouve le processus cible du signal.
  - Processus **actif** (en exécution)
  - Processus **bloqué** (en attente)

SIGNAUX: RÉCEPTION

- **Processus actif (en exécution)**

  – Si le processus exécute un programme utilisateur

  ➔ traitement immédiat si signal reçu.

  – Si le processus exécute un SVC

  ➔ le traitement du signal sera différé jusqu'à ce que le SVC soit terminé et le processus revienne en mode utilisateur.

SIGNAUX: RÉCEPTION

- **Processus bloqué (dans un SVC)**

  – Dans la plupart des cas, le signal est traité à la sortie du SVC.

  – Dans les autres cas, le signal est traité dès sa réception et le processus sort de la fonction bloquante et passe à l'état prêt (exp : arrivée de SIGCHLD pendant que le processus est bloqué sur une lecture réseau ou clavier).

SIGNAUX: TRAITEMENT

- **Après avoir détecté son occurrence, comment un processus peut-il traiter un signal ?**

  – Pour chaque signal, le processus peut :

    - Ignorer ce signal (si le noyau le permet)
    - Se contenter du traitement par défaut
    - Exécuter un traitement personnalisé (si le noyau le permet)

SIGNAUX: TRAITEMENT

- Le tableau de signaux comporte une entrée par signal. Chaque entrée est initialisée à SIG_DFL (Traitement par défaut) à la création du processus.

- Le processus peut modifier une entrée de ce tableau s'il veut changer le comportement par défaut à la réception du signal correspondant.

Sig_Num	FLAGS (IGN, BLK)	Handler
1		
2		

SIGNAUX: TRAITEMENT

- **Comment modifier le traitement par défaut ?**

  – On utilise l'appel système **signal (Num_Sig, Trt)**

  Où **Num_Sig** est le numéro du signal reçu

  et **Trt** est le nouveau traitement à exécuter à la réception de Num_Sig.

  – Il existe trois options pour le traitement :
    - **Signal (Num_Sig, SIG_IGN)** : Ignorer le signal
    - **Signal (Num_Sig, SIG_DFL)** : Traitement par défaut
    - **Signal (Num_Sig, Trt)** : Installer un nouveau traitant.

SIGNAUX: FONCTIONS UTILISÉES

int sigaction (int signum, const struct sigaction *new, struct sigaction *old)

➔ Modifier l'action effectuée par le processus à la réception d'un signal.

signum : numéro du signal (sauf SIGKILL et SIGSTOP)

new : est la nouvelle action effectuée par le processus.

old : s'il n'est pas nul, l'ancienne action est stockée dans **old**

```
struct sigaction {
    void        (*sa_handler)(int);
    sig_set     sa_mask;
    int         sa_flags ;
};
```

SIGNAUX: FONCTIONS UTILISÉES

```
struct sigaction {
    void        (*sa_handler)(int);
    sig_set     sa_mask;
    int         sa_flags ;
};
```

sa_handler: L'action affectée au signal ***signum***. Elle peut être: ***SIG_DFL***, ***SIG_IGN*** ou un pointeur sur une fonction de gestion de signaux.

sa_mask: Masque de signaux à bloquer pendant l'exécution du gestionnaire + le signal ayant appelé le gestionnaire.

sa_flags: Ensemble d'attributs modifiant le comportement du gestionnaire de signaux.

SIGNAUX: FONCTIONS UTILISÉES

Un processus pouvait bloquer à volonté un ensemble de signaux, sauf SIGKILL et SIGSTOP. Cette opération se fait principalement grâce à l'appel-système sigprocmask( ).

int sigprocmask (int action, const sigset_t *set, sigset_t *oldset)

➔ Utilisée pour changer la liste des signaux actuellement bloqués. Son comportement dépend de la valeur de l'argument **action** qui peut valoir :

SIG_BLOCK L'ensemble des signaux bloqués est l'union de l'ensemble actuel et de l'argument ***set*.** Il s'agit d'une addition au masque en cours.

SIG_UNBLOCK Les signaux dans l'ensemble ***set*** sont supprimés de la liste des signaux bloqués.

SIG_SETMASK L'ensemble des signaux bloqués est égal à l'argument ***set***.

SIGNAUX: FONCTIONS UTILISÉES

int sigpending (sigset_t *set)

➔Examen des signaux en attente (pendants : qui se sont déclenchés en étant bloqués). Cette routine remplit l'ensemble transmis en argument **set** avec les signaux en attente.

int sigsuspend (const sigset_t *set)

➔Cette primitive permet de manière atomique de modifier le masque des signaux **set** et de bloquer en attente. Une fois qu'un signal non bloqué arrive. sigsuspend( ) restitue le masque original avant de se terminer.

SIGNAUX: FONCTIONS UTILISÉES

int pause (void)

➔Force le processus appelant à s'endormir jusqu'à ce qu'un signal soit reçu qui le termine ou lui fasse invoquer un gestionnaire de signaux.

unsigned int sleep (unsigned int n_sec)

➔Fait endormir le processus appelant jusqu'à ce que ***n_sec*** secondes se soient écoulées ou jusqu'à ce qu'un signal soit reçu.

unsigned int alarm (unsigned int n_sec)

➔Fait programmer une temporisation pour que le signal SIGALRM soit envoyé au processus appelant dans ***n_sec*** secondes. Si le signal n'est pas masqué ou intercepté, sa réception terminera le processus.

SIGNAUX: FONCTIONS UTILISÉES

int sigemptyset (sigset_t *set)

➔ Fait vider l'ensemble des signaux fourni par ***set***. Tous les signaux sont exclus de cet ensemble.

int sigfillset (sigset_t *set)

➔ Fait remplir l'ensemble de signaux ***set***. Tous les signaux sont inclus dans cet ensemble.

int sigaddset (sigset_t *set, int signum)

➔ Fait ajouter le signal ***signum*** à l'ensemble de signaux pointé par ***set***.

int sigdelset (sigset_t *set, int signum)

➔ Fait retirer le signal ***signum*** de l'ensemble de signaux pointé par ***set***.

int sigismember (const sigset_t *set, int signum)

➔ Teste si le signal ***signum*** appartient à l'ensemble ***set***.

SIGNAUX: EXEMPLES

```c
/* Exemple simple d'utilisation de la primitive alarm et du signal SIGALRM */

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

int n;

void handler(int signum)
{
  printf("Expiration des %d secondes ...\n",n);
  fflush(stdout);
  exit(0);
}

int main(int argc, char **argv)
{
  signal(SIGALRM,handler);
  n=atoi(*(argv+1));
  alarm(n);
  while(1);
}
```

SIGNAUX: EXEMPLES

```c
/* Deuxième exemple d'utilisation de la primitive alarm */
#include "include.h"

int handle;

void handler(int signum)
{
    printf("Pas de saisie dans les 5 secondes... Opération annulée !\n");
    close(handle);
}

int main(int argc, char **argv)
{
  char chaine[20];
  signal(SIGALRM,handler);
  alarm(5);
  printf("Donner une chaine ... \n");
  scanf("%s",chaine);
  alarm(0); /* Si chaîne saisie, annuler la temporisation */
  handle=open("Junk",O_WRONLY|O_CREAT);
  write(handle,chaine,strlen(chaine));
  close(handle);
}
```

SIGNAUX: EXEMPLES

```c
#include "include.h"

/* Exemple montrant l'utilisation des primitives sigaction() et pause() */

void traitant(int sig_num)
{
  printf("Signal SIGINT reçu !\n");
  exit(0);
}
int main(void)
{
  struct sigaction action;
  action.sa_handler=traitant;

  sigaction(SIGINT,&action, NULL);

  pause();

  exit(0);
}
```

SIGNAUX: EXEMPLES

```c
#include "include.h"
void bonjour(int signum)
{
  printf("Bonjour à tous !\n");
}


void bonsoir(int signum)
{
  printf("Bonsoir à tous !\n");
}

int main(void)
{

  signal(SIGUSR1,bonjour);
  signal(SIGUSR2,bonsoir);
  for(;;);
}
```

SIGNAUX: EXEMPLES

```c
#include "include.h"
void handler(int signum)
{
  pid_t pid_fils;
  pid_fils=wait(NULL);
  printf("Fin d'exécution du fils %d\n",pid_fils);
}
int main(void)
{
  struct sigaction action;
  action.sa_handler=handler;
  sigaction(SIGCHLD,&action,NULL);
  if(fork()==0)
  {
  printf("Je suis le processus fils de pid %d,je me suspend ...\n",getpid());
  sleep(2);
  exit(0);
  }
  printf("Pid du processus courant : %d\n",getpid());
  printf("Attente de la terminaison du processus fils...\n");
  pause();
}
```