Les signaux

Un signal traduit l'apparition d'un événement asynchrone à l'exécution d'un processus.

Il peut être généré par :

- **Le noyau (par exemple, si un processus accède une zone mémoire non autorisée - signal SIGSEGV)**
- **Un appel système *kill* qui permet d'envoyer un signal vers un processus identifié par son pid.**
- **Le processus lui même (par exemple, un processus initialise une alarme (appel *alarm*) et lorsque que cette alarme expire, le noyau lui envoie le signal SIGALRM).**
- **la frappe d'une touche par l'utilisateur du terminal (par exemple [CTRL-C] génère le signal SIGINT dont l'action par défaut est de terminer le processus).**

Contrôle des signaux

A chaque signal est associé un pointeur vers une fonction qui est appelée lors de l'activation du signal.

Il est possible de modifier le pointeur de cette fonction sauf pour les signaux SIGKILL et SIGSTOP,

Trois comportements sont possibles dans la gestion des signaux :

- **c'est la fonction par défaut du Kernel qui traite le signal**
- **l'utilisateur fournit une fonction pour traiter le signal**
- **l'activation du signal est ignorée**

7-2) Type de signaux

Pour le programmeur un signal est représenté dans le système par un nom de la forme SIGXXX.

Signaux provoqués par une exception matérielle

Posix précise que le comportement d'un programme qui ignore les signaux d'erreur du type SIGFPE, SIGILL,SIGSEGV, SIGBUS est indéfini.

SIGFPE Floating-point exception : envoyé lors d'un problème avec une opération en virgule flottante mais il est aussi utilisé pour l'erreur de division entière par zéro. L'action par défaut est la mise àmort du processus en créant un fichier d'image mémoire *core*. Ce fichier est alors utilisé par l'outil de "debugger" *gdb.*

SIGILL Illegal instruction : envoyé au processus qui tente d'exécuter une instruction illégale. Ceci ne doit jamais se produire dans un programme normal mais il se peut que le fichier exécutable soit corrompu (erreur lors du chargement en mémoire). L'action par défaut est la mise àmort du processus en créant un fichier d'image mémoire *core*.

SIGSEV Segmentation violation : envoyé lors d'un adressage mémoire invalide (emploi d'un pointeur mal initialisé). L'action par défaut est la mise àmort du processus en créant un fichier d'image mémoire *core*.

SIGBUS Bus error : envoyé lors d'une erreur d'alignement des adresses sur le bus. L'action par défaut est la mise àmort du processus en créant un fichier d'image mémoire *core*.

SIGTRAP Trace trap : envoyé après chaque instruction, il est utilisé par les programmes de mise au point (debug).
L'action par défaut est la mise àmort du processus en créant un fichier d'image mémoire *core*.

Signaux de terminaison ou d'interruption de processus

SIGHUP Hangup : envoyé àtous les processus attachés àun terminal lorsque celui-ci est déconnecté du système. Il est courant d'envoyer le signal àcertains processus démons afin de leur demander de se réinitialiser. L'action par défaut est la miseàmort du processus.

SIGINT Interrupt : le plus souvent déclenché par [CTRL-C].
L'action par défaut est la mise àmort du processus.

SIGQUIT Quit : similaire àSIGINT mais pour [CTRL] [\].
L'action par défaut est la mise àmort du processus en créant un fichier d'image mémoire *core*.

SIGIOT I/O trap instruction : émis en cas de problème hardware (I/O). → *core*

SIGABRT Abort : Arrêt immédiat du processus (erreur matérielle). → *core*

SIGKILL Kill : utilisé pour arrêter l'exécution d'un processus.
L'action par défaut est non modifiable.

SIGTERM Software termination : C'est le signal qui par défaut est envoyé par la commande **kill**. L'action par défaut est la mise àmort du processus.

Signaux utilisateurs

SIGUSR1 User defined signal 1 : Définie par le programmeur. L'action par défaut est la mise àmort du processus.

SIGUSR2 User defined signal 2 : idem que SIGUSR1 .

Signal généré pour un tube

SIGPIPE Signal d'erreur d'écriture dans un tube sans processus lecteur.
L'action par défaut est la mise àmort du processus.

Signaux liés au contrôle d'activité

SIGCLD Signal envoyé au processus parent lorsque les processus fils terminent.
Aucune action par défaut.

SIGCONT L'action par défaut est de reprendre l'exécution du processus s'il est stoppé.
Aucune action si le processus n'est pas stoppé.

SIGSTOP Suspension du processus.
L'action par défaut est la suspension (!! non modifiable).

SIGTSTP Emission àpartir d'un terminal du signal de suspension [CTRL-Z]. Ce signal à le même effet que SIGSTOP mais celui-ci peut être capturé ou ignoré.
L'action par défaut est la suspension.

SIGTTIN Est émis par le terminal en direction d'un processus en arrière-plan qui essaie de lire sur le terminal. L'action par défaut est la suspension.

SIGTTOU Est émis par le terminal en direction d'un processus en arrière-plan qui essaie d'écrire sur le terminal. L'action par défaut est la suspension.

Signal lié àla gestion des alarmes

SIGALRM Alarm clock : généré lors de la mise en route du système d'alarme par l'appel système alarm().

kill

La fonction *kill()* permet d'envoyer un signal à un processus

```
#include<signal.h>

int kill (pid_t pid, int sig);

Retourne 0 si OK, -1 si erreur
```

Si l'argument pid

- **est positif, le signal est envoyé au processus identifié par pid.**
- **est nul, le signal est envoyé aux processus du groupe du processus émetteur.**
- **est égal à -1, le signal est envoyé à tous les processus, sauf le processus *init.***
- **est inférieur à -1, le signal est envoyé au groupe de processus identifié par la valeur absolue de pid.**

Si l'argument sig est zéro, une simulation d'émission de signal est réalisée c'est-à-dire qu'aucun signal n'est généré mais la fonction retourne les mêmes valeurs que dans le cas standard. Ce qui permet par exemple de tester la validité d'un PID.

Si une erreur s'est produite la variable errno contient l'une des valeurs suivantes:

- **EINVAL : Signal invalide**
- **ESRCH : Le processus ou groupe de processus n'existe pas.**
- **EPERM : L'euid du processus appelant est différente de celle du processus cible.**

signal

```
#include<signal.h>

void (*signal (int signum, void (*handler) (int))) (int) ;
```

Retourne le pointeur de fonction du signal si OK, la constante SIG_ERR si erreur

Le premier argument signum est le numéro du signal (qui doit être différent de SIGKILL ou SIGSTOP).

Le second argument handler définit la nouvelle disposition à associer au signal; ce sera l'adresse d'une fonction ou l'une des constantes symboliques SIG_DFL (traitement par défaut) ou SIG_IGN (le signal est ignoré).

La fonction de déroutement prend en paramètre un entier correspondant au signal qui l'a activée et ne renvoie aucun argument.

Cette fonction semble devenir obsolète et est remplacée par la fonction POSIX : sigaction( )

pause

```
#include<signal.h>

int pause( void );
```

Retourne -1 avec errno valant EINTR

L'appel système pause() endort indéfiniment un processus dans un mode interruptible. Le processus reprendra son cours à l'arrivée d'un signal si celui-ci n'est pas inhibé, ignoré ou si la fonction associée ne conduit pas le processus à terminer.

```
/*************************************************************
 *    Nom   : sig1.c
 *    But   : Se protéger des signaux
 *               SIGINT  :  [CTRL] [C]
 *               SIGTERM :  kill pid
 *            qui par défaut arrêtent le processus.
 *************************************************************/

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

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

      if (argc<2) {
            printf("Erreur : --> Appel : sig1 nbr_sec \n");
            exit(1);
      }

      sscanf(argv[1],"%d",&nbrsec);

      if (signal(SIGINT , SIG_IGN)== SIG_ERR) {      /* CTRL C     */
            perror("SIGINT");
            exit(1);
      }
      if (signal(SIGTERM, SIG_IGN) == SIG_ERR) {     /* KILL -TERM pid */
            perror("SIGTERM");
            exit(1);
      }

      sleep(nbrsec);
      exit(0);
}
```

Essai

```
jpol@Isil007:~> gcc –Wall –o sig1 sig1.c
jpol@Isil007:~> sig1 30 &
[1] 423
jpol@Isil007:~> kill -TERM 423
jpol@Isil007:~> ps
  PID TTY STAT TIME COMMAND
  131   1 S    0:00 -bash
  423   1 S    0:00 sig1 30
  424   1 R    0:00 ps
jpol@Isil007:~>
```

```
/*********************************************************
 *   Nom  :  sig2.c
 *   But  :  Dérouter les signaux SIGUSR1 et SIGUSR2
*********************************************************/

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

void handler( int) ;

int main()
{
      if ( signal(SIGUSR1, handler) == SIG_ERR){
            perror("SIGUSR1");
            exit(1);
      }
      if ( signal(SIGUSR2, handler) == SIG_ERR) {
            perror("SIGUSR2");
            exit(1);
      }
      while(1) pause();
}

void handler( int signo)
{
      if (signo == SIGUSR1)
            printf("Signal SIGUSR1 reçu \n");
      else
            printf("Signal SIGUSR2 reçu \n");

}
```

Essais

```
jpol@Isil007:~> sig2&
[1] 151
jpol@Isil007:~> kill -USR1 151
Signal SIGUSR1 reçu

jpol@Isil007:~> kill -USR2 151
Signal SIGUSR2 reçu

jpol@Isil007:~> kill -USR1 151
Signal SIGUSR1 reçu

jpol@Isil007:~> kill -TERM 151
jpol@Isil007:~> ps
  PID TTY STAT TIME COMMAND
  131   1 S    0:00 -bash
  152   1 R    0:00 ps
[1]+  Terminated              sig2
jpol@Isil007:~>
```

alarm

C'est un appel système qui permet de générer, à l'attention du processus appelant, un signal SIGALRM après un certain nombre de secondes (la valeur 0 provoque l'annulation de l'alarme).

#include<unistd.h>

unsigned alarm (unsigned sec);

Retourne 0 ou le nombre de secondes qui restait dans son compteur avant l'émission d'un nouvel appel. La nouvelle alarme annule l'ancienne.

```
/*********************************************************************
 *   Nom    : sleep1.c
 *   Buts   : Version simplifiée de la fonction sleep().
 *            Utilisation de l'appel système alarm().
 *********************************************************************/
#include<stdio.h>
#include<signal.h>
#include<unistd.h>

void sig_alarm( int signo)
{
      /* Ne rien faire */
      return;     /* Réveille le pause  */
}

int sleep1( unsigned nsecs)
{
      if (signal(SIGALRM, sig_alarm) == SIG_ERR)
            return (nsecs);
      alarm(nsecs);     /* Démarrer le timer */
      pause();          /* Endormir la fonction en attente d'un signal */

      /* Poursuite de la fonction après réception d'un signal */
      return( alarm(0) );     /* Arrête le timer et retourne le nombre de
                                 secondes qui reste dans le compteur    */

}

int main()
{
      printf("Attente de 5 secondes\n");
      sleep1(5);
      printf("Fin du traitement\n");
      exit(0);
}
```

```
/*********************************************************************
 *   Nom    : sig3.c
 *   But    : Effectuer un read avec timeout
 *********************************************************************/

#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#define MAXCAR 100

void sig_alarm( int signo)
{     /* Ne rien faire */
      return;     /* Interrompre le read  */
}

int main()
{
      int   n;
      char  ligne[MAXCAR];

      if (signal(SIGALRM, sig_alarm) == SIG_ERR) {
            perror("SIGALRM");
            exit(1);
      }
      siginterrupt( SIGALRM, 1);

      printf("Entrez une ligne ?\n");
      alarm(5);   /* Démarrer le timer */
      fgets(ligne,MAXCAR-1,stdin);
      if(alarm(0))
            printf("%s",ligne);
      else
            printf("Timeout\n");

      printf("Fin du traitement\n");
      exit(0);
}
```

Essais:

```
jpol@Isil007:~> gcc –Wall -o sig3 sig3.c
jpol@Isil007:~> sig3
Entrez une ligne ?
Timeout
Fin du traitement
jpol@Isil007:~> sig3
Entrez une ligne ?
isil
isil
Fin du traitement
jpol@Isil007:~>
```

Les appels système interruptibles

Si au cours d'un appel système (donc en mode noyau) un processus est amené à attendre des ressources temporairement indisponibles, il est endormi par le noyau.

Dans certains cas l'arrivée d'un signal va réveiller le processus, c'est-à-dire le rendre à nouveau éligible sans considération de la libération de la ressource et provoquer un retour prématuré de l'appel système. La valeur de retour de l'appel système est alors généralement -1, et la variable errno contient la valeur EINTR.

Afin de gérer le comportement d'un programme en cas d'interruption par un signal, les systèmes BSD fournissent la fonction *siginterrupt(3C)*

```
#include<signal.h>

int siginterrupt( int sig, int flag);

Retourne 0 si OK, -1 si erreur
```

Si la valeur du flag

- **est nulle, un appel système sera relancé s'il est interrompu par le signal sig.**
- **est égale à 1 et qu'aucune donnée n'a été transférée, alors un appel système interrompu par le signal sig renvoie -1 et la variable errno contient la valeur EINTR.**
- **est égale a 1 et que des données ont été transférées, alors un appel système interrompu par le signal sig renvoie la quantité de données transférées.**

Exemple:

```
/*************************************************************************
 *    Nom   : sigread.c
 *    But   : Utiliser la fonction siginterrupt pour contrôler une lecture
 *            bloquante sur le descripteur 0.
 *************************************************************************/
#include<unistd.h>
#include<stdio.h>
#include<signal.h>
#include<errno.h>

void sig_suspend( int numero)
{
      printf("Gestionnaire (CTRL-Z) \n");
}

int main()
{
      int car;

      if (signal(SIGTSTP, sig_suspend) == SIG_ERR) {
            perror("SIGTSTP");
            exit(1);
      }

      printf("Utilisation de siginterrupt avec flag=0 \n");
      siginterrupt(SIGTSTP, 0);
      printf("Entrez un caractère ? \n");
      if ( read(0, &car, sizeof(car)) <0)
            if (errno == EINTR ) printf("errno=EINTR\n");

      printf("Voici le caractère : %c\n", car);

      printf("Utilisation de siginterrupt avec flag=1 \n");
      siginterrupt(SIGTSTP, 1);
      printf("Entrez un caractère ? \n");
      if ( read(0, &car, sizeof(car)) <0) {
            if (errno == EINTR ) printf("errno=EINTR\n");
      }
      else
            printf("Voici le caractère : %c\n", car);

      exit(0);
}
```

Essais:

```
jpol@Isil007:~ > gcc -Wall -o sigread sigread.c
jpol@Isil007:~ > sigread
Utilisation de siginterrupt avec flag=0
Entrez un caractère ?
Gestionnaire (CTRL-Z)        ← CTRL Z
Gestionnaire (CTRL-Z)        ← CTRL Z
k                            ← k (enter)
Voici le caractère : k
Utilisation de siginterrupt avec flag=1
Entrez un caractère ?
Gestionnaire (CTRL-Z)        ← CTRL Z
errno=EINTR
jpol@Isil007:~ >
```

Les fonctions réentrantes

Si une fonction est appelée comme fonction de déroutement d'un signal elle peut provoquer des erreurs dans le déroulement du processus. En effet, certaines fonctions de bibliothèques manipulent des données statiques et ne sont pas réentrantes.

Par exemple, la fonction *malloc* gère les zones mémoire sous forme de listes chaînées: si un signal intervient durant l'appel à cette fonction et que la fonction de déroutement fait également appel à *malloc*, les listes risquent d'être dans un état incohérent et le fonctionnement du processus risque donc d'être erroné.

Il est donc préférable d'utiliser uniquement des fonctions réentrantes dans les fonctions de déroutement.

POSIX.1 spécifie les fonctions qui sont garanties être réentrantes :

_exit	**fork**	**pipe**	**stat**
abort	**fstat**	**read**	**sysconf**
access	**getegid**	**rename**	**tcdrain**
alarm	**geteuid**	**rmdir**	**tcflow**
cfgetispeed	**getgid**	**setgid**	**tcflush**
cfgetospeed	**getgroups**	**setpgid**	**tcgetattr**
cfsetispeed	**getpgrp**	**setsid**	**tcgetpgrp**
cfsetospeed	**getpid**	**setuid**	**tcsendbreak**
chdir	**getppid**	**sigaction**	**tcsetattr**
chmod	**getuid**	**sigaddset**	**tcsetpgrp**
chown	**kill**	**sigdelset**	**time**
close	**link**	**sigemptyset**	**times**
creat	**longjmp**	**sigfillset**	**umask**
dup	**lseek**	**sigismember**	**uname**
dup2	**mkdir**	**signal**	**unlink**
execle	**mkfifo**	**sigpending**	**utime**
execve	**open**	**sigprocmask**	**wait**
exit	**pathconf**	**sigsuspend**	**waitpid**
fcntl	**pause**	**sleep**	**write**

sigaction

```
#include <signal.h>

int sigaction ( int signum,
                const struct sigaction *nouveau,
                struct sigaction *ancien );
```

Retourne 0 si OK, -1 si erreur

Le premier paramètre représente le signal à dérouter. Les deuxième et troisième paramètres sont respectivement le nouveau et l'ancien comportement à adopter à l'arrivée du signal.

Si le pointeur *nouveau* est différent de NULL, alors le système modifie l'action du signal *signum*. Si le pointeur *ancien* est différent de NULL, alors le système retourne l'action précédente qui était prévue pour le signal *signum.* Si les pointeurs *nouveau* et *ancien* sont NULL, l'appel système teste uniquement la validité du signal.

Définition de la structure sigaction:

```
struct sigaction {
        void    (*sa_handler()) ();     /* Fonction de déroutement ,SIG_IGN ou SIG_DFL*/
        sigset_t  sa_mask;              /* Liste des signaux qui sont bloqués durant l'exécution
                                           de la fonction de déroutement                */
        unsigned long sa_flags;         /* Options définissant le comportement du processus à
                                           la réception du signal*/
};
```

Le signal ayant déclenché l'exécution du gestionnaire est automatiquement bloqué sauf si on demande explicitement le contraire (option SA_NODEFER).

Les options possibles pour sa_flags:

Options	*Signification*
SA_NOCLDSTOP	Si le signal est SIGCHLD alors ne pas recevoir de signal (SIGSTOP, SIGSTP, GIGTTIN, SIGTTOU)lorsque le processus fils se suspend.
SA_ONESHOT, SA_RESETHAND	Réinstaller la fonction de déroutement par défaut après la réception du signal.
SA_RESTART	Relancer automatiquement l'appel système interrompu par la réception du signal.
SA_NOMASK SA_NODEFER	Ne pas bloquer la réception du signal durant l'exécution de la fonction de déroutement.

Les erreurs possibles pour cet appel sont :

- EINVAL signum est un signal invalide
- EFAULT *nouveau* ou *ancien* contient une adresse invalide

Exemple:

```
/*************************************************************************
 *    Nom   : sigaction.c
 *    But   : Utiliser la fonction sigaction pour contrôler une lecture
 *            bloquante sur le descripteur 0.
 *            Une seconde occurence du signal SIGTSTP déclenchera le
 *            comportement par défaut prévu pour ce signal.
 *************************************************************************/

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

void sig_suspend( int numero)
{
      printf("Gestionnaire (CTRL-Z) \n");
}

int main()
{
      int car;
      struct sigaction action;

      action.sa_handler=sig_suspend;
      sigemptyset( &(action.sa_mask));
      action.sa_flags= SA_RESTART | SA_ONESHOT;

      if (sigaction(SIGTSTP, &action, NULL) != 0) {
            perror("SIGTSTP");
            exit(1);
      }

      printf("Entrez un caractère ? \n");
      if ( read(0, &car, sizeof(car)) <0) {
            if (errno == EINTR ) printf("errno=EINTR\n");
      }
      else
            printf("Voici le caractère : %c\n", car);


      exit(0);
}
```

Essais:

```
jpol@Isil007:~ > gcc -Wall -o sigaction sigaction.c
jpol@Isil007:~ > sigaction
Entrez un caractère ?
Gestionnaire (CTRL-Z)   ← CTRL Z
k                       ← k (enter)
Voici le caractère : k

jpol@Isil007:~ > sigaction
Entrez un caractère ?
Gestionnaire (CTRL-Z)   ← CTRL Z
                        ← CTRL Z (ONE_SHOT -> mise à mort du processus)
[1]+  Stopped                 sigaction
jpol@Isil007:~ >
```