Cours "Système d'exploitation"

R3.05

BUT 2ème année

Département d'informatique

Développement système : les signaux

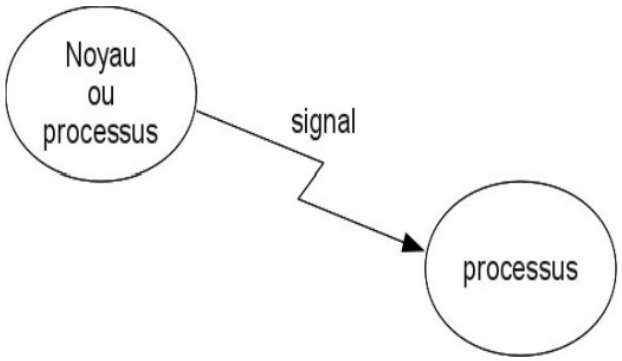
Table des matières

Partie 3 :

I/ Introduction :

Les signaux sous Linux sont des événements que le système génère et envoie à un processus. Cet envoi provoque une réaction qui peut être :

- ✓ **Interne** : détection d'une erreur. (Violation mémoire, instructions illicites, erreur mathématique comme une division par zéro etc.).
- ✓ **Externe** : provient d'un autre processus, frappe au clavier, provient d'un shell etc.



Un signal est également un mécanisme de synchronisation car il permet de réveiller, arrêter ou avertir un processus d'un événement.

Quand un signal est envoyé d'un processus vers un autre processus, il devient un moyen de transmission d'informations ou de modification de comportement du processus destinataire.

Lorsqu'un processus reçoit un signal il peut l'ignorer, déclencher l'action prévu par défaut ou déclencher un traitement spécial (handler).

II/Identification d'un signal :

Sur un système donné, on dispose de NSIG signaux numérotés de 1 à NSIG. Les différents signaux et les prototypes des fonctions qui les manipulent sont définis dans le fichier ***<signal.h>***.

Le noyau Linux admet 64 signaux différents qui possèdent un numéro et un nom différents.

- ➢ 0 : seul signal qui n'a pas de nom
- ➢ 1 à 31 : signaux classiques
- ➢ 32 à 63 : signaux « temps réels »

Il est possible d'obtenir la liste des signaux sous le Shell grâce à la commande ***kill – l***.

```
pi@raspberrypi:~ $ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
pi@raspberrypi:~ $
```

Portabilité :

Des efforts de normalisation ont été faits tant sur le plan de la dénomination des signaux que sur le comportement correspondant des systèmes d'exploitation UNIX. Il n'est cependant pas certain que la compatibilité soit assurée à 100%. La première précaution est d'utiliser impérativement les noms symboliques et non les valeurs. De plus il est nécessaire de bien connaitre le comportement des processus pour chaque signal, et donc de lire attentivement toutes les informations disponibles dans le "man" du système sur lequel on veut implémenter une application utilisant les signaux. Les traitements standards de chaque signal peuvent différer d'un système à un autre.

La portabilité est donc difficile à maintenir en ce qui concerne les signaux.

Il est possible d'obtenir un libellé de signal à l'aide des fonctions

char *strsignal (int sig)

et ***void psignal (int sig, const char *s).***

III/Comportement d'un signal :

Lorsqu'un signal parvient à un processus, il peut être traité, bloqué ou ignoré.

Un signal peut être bloqué de 3 façons :

- ✓ Soit par un masque positionné par différentes fonctions système ou fonctions standard.
- ✓ Soit parce que le processus est dans un état ***non interruptible*** (priorité < 25).
- ✓ Soit parce que le signal est en cours de traitement.

Un signal émis par un processus ne sera effectivement traité qu'au bout d'un délai imprévisible. En attendant le traitement il est appelé ***signal pendant.***

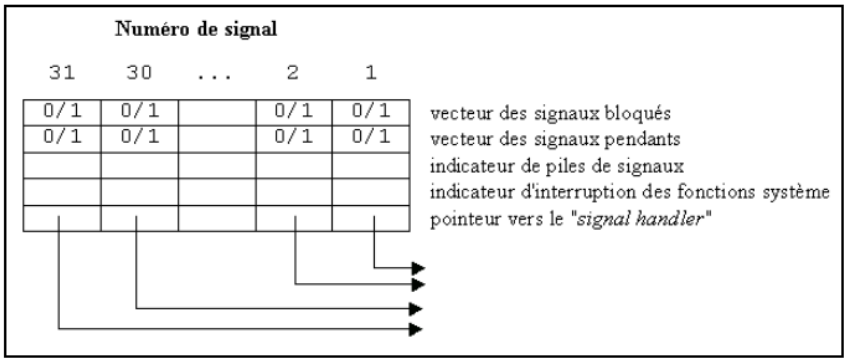
Un signal pendant n'est pas perdu, il pourra être traité soit quand la priorité remontera au-dessus de 25 ou quand il sera débloqué.

Un processus ne prend en compte un signal qu'il a reçu que lorsqu'il repasse en ***mode utilisateur***, ou s'il est dans l'état de sommeil. (État de sommeil par exemple provoqué par l'appel à l'une des fonctions ***sigpause()***, ***wait()*** ,***pause()*** ou ***sleep()***).

Un processus doit être en mesure d'associer au signal qu'il est susceptible de recevoir :

- Un traitement particulier (signal handler),
- Un bit indiquant si le signal est bloqué ou non,
- Un bit indiquant si le signal correspondant est arrivé mais pas encore traité (par exemple si un autre signal est en cours de traitement) ***signal pendant***
- Une indication indiquant si le signal doit être traité sur la pile utilisateur ou sur une pile spécifique à ce signal.
- Une indication spécifiant si le signal doit ou non provoquer l'interruption des fonctions systèmes, s'il arrive pendant l'exécution d'une telle fonction.

Toutes ces informations sont stockées dans le bloc de contrôle du processus (**PCB)**.



L'arrivée du même signal ne peut être mémorisée qu'une seule fois avant d'être traitée. Toutes les autres occurrences sont perdues.

Il n'existe pas de priorité entre les signaux, ils sont traités dans l'ordre croissant de leurs numéros. Le traitement d'un signal commence toujours par sauvegarde du contexte du processus concerné avant de donner la main au traitant du signal. Sin le traitant rend la main au processus, son contexte est restauré et il reprend au point où il a été interrompu.

Remarque :

Les signaux ***SIGKILL*** ***SIGSTOP*** ne peuvent être bloqués ou déroutés dans aucune version d'UNIX. Conformément à la norme Posix, le signal ***SIGCHLD*** ne devrait jamais être ignoré (***SIG_IGN***)

III-1/ Émettre un signal :

Un processus ne peut envoyer un signal à un autre processus que s'ils ont le même propriétaire.

III-1-1/ Sous le shell

Pour émettre un signal à un processus sous le shell, il faut utiliser la commande ***kill***.

Cette commande est capable d'émettre tous les signaux à un processus donné. Si on ne précise pas de signal, c'est le signal SIGTERM qui sera envoyé.

Exemple :

```
pi@raspberrypi:~/Documents/TubesAnonymes $ ./Td3Exo3
 pid du premier fils 3344
 pid du second fils 3345
Père en attente de la fin des fils...
```

Dans cet exemple un père crée deux fils et se met en attente de la fin de ses fils.

```
pi@raspberrypi:~ $ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       486  0.1  4.7 213040 44788 tty7     Ssl+ 19:25   0:15 /usr/lib/xorg/X
root       487  0.0  0.2   5620  2744 tty1     Ss   19:25   0:00 /bin/login -f
pi         657  0.0  0.3   8492  3676 tty1     S+   19:25   0:00 -bash
pi        2054  0.0  0.4   8668  3984 pts/0    Ss   20:19   0:00 bash
pi        3323  0.1  0.3   8516  3712 pts/1    Ss   21:37   0:00 bash
pi        3343  0.0  0.0   1720   324 pts/0    S+   21:38   0:00 ./Td3Exo3
pi        3344  0.0  0.0   1852    64 pts/0    S+   21:38   0:00 ./Td3Exo3
pi        3345  0.0  0.0   1720    68 pts/0    S+   21:38   0:00 ./Td3Exo3
pi        3346  0.0  0.2   9788  2504 pts/1    R+   21:38   0:00 ps au
```

On envoie le signal SIGKILL (-9) au fils de pid 3345, on remarque que le fils se termine.

```
pi@raspberrypi:~ $ kill -9 3345
pi@raspberrypi:~ $ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       486  0.2  4.7 213040 44788 tty7     Ssl+ 19:25   0:16 /usr/lib/xorg/X
root       487  0.0  0.2   5620  2744 tty1     Ss   19:25   0:00 /bin/login -f
pi         657  0.0  0.3   8492  3676 tty1     S+   19:25   0:00 -bash
pi        2054  0.0  0.4   8668  3984 pts/0    Ss   20:19   0:00 bash
pi        3323  0.1  0.3   8516  3712 pts/1    Ss   21:37   0:00 bash
pi        3343  0.0  0.0   1720   324 pts/0    S+   21:38   0:00 ./Td3Exo3
pi        3344  0.0  0.0   1852    64 pts/0    S+   21:38   0:00 ./Td3Exo3
pi        3359  0.0  0.2   9788  2528 pts/1    R+   21:39   0:00 ps au
```

On recommence mais cette fois mais sans préciser de signal au fils toujours actif de pid 3344 et on remarque qu'il se termine aussi et que le père se termine également.

```
pi@raspberrypi:~ $ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       486  0.2  4.7 213040 44788 tty7     Ssl+ 19:25   0:16 /usr/lib/xorg/X
root       487  0.0  0.2   5620  2744 tty1     Ss   19:25   0:00 /bin/login -f
pi         657  0.0  0.3   8492  3676 tty1     S+   19:25   0:00 -bash
pi        2054  0.0  0.4   8668  3984 pts/0    Ss+  20:19   0:00 bash
pi        3323  0.1  0.3   8516  3712 pts/1    Ss   21:37   0:00 bash
pi        3362  0.0  0.2   9788  2576 pts/1    R+   21:39   0:00 ps au
pi@raspberrypi:~ $
```

```
pi@raspberrypi:~/Documents/TubesAnonymes $ ./Td3Exo3
 pid du premier fils 3344
 pid du second fils 3345
Père en attente de la fin des fils...
terminaison du pere
```

Si on essaye d'envoyer un signal à un processus non-propriétaire, on remarque que cela n'a pas d'effet sur le processus et un message d'erreur nous prévient que l'envoi n'est pas permis.

```
pi@raspberrypi:~ $ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       486  0.2  4.7 213040 44788 tty7     Ssl+ 19:25   0:16 /usr/lib/xorg/X
root       487  0.0  0.2   5620  2744 tty1     Ss   19:25   0:00 /bin/login -f
pi         657  0.0  0.3   8492  3676 tty1     S+   19:25   0:00 -bash
pi        2054  0.0  0.4   8668  3984 pts/0    Ss+  20:19   0:00 bash
pi        3323  0.1  0.3   8516  3712 pts/1    Ss   21:37   0:00 bash
pi        3362  0.0  0.2   9788  2576 pts/1    R+   21:39   0:00 ps au
pi@raspberrypi:~ $ kill -9 487
bash: kill: (487) - Opération non permise
pi@raspberrypi:~ $
```

III-1-2/ En langage C

En langage C la fonction ***int kill(pid_t pid, int sig)*** permet à un processus donné d'envoyer un signal à un autre processus. Il faut lui préciser le signal que l'on souhaite envoyer et le PID du processus visé :

- Si PID est > 0, le signal est envoyé au processus ayant ce PID,
- Si PID = 0, le signal est envoyé à tous les processus du groupe auquel appartient le processus émetteur,
- Si PID = -1 (non POSIX), le signal est envoyé à tous les processus sauf celui qui a pour PID 1 (init). C'est ce qui est fait lors de l'arrêt du système (shutdown),
- Si PID < -1, le signal est envoyé à tous les processus de groupe |PID|.

Remarques :

- ✓ Si on envoie la valeur 0 pour ***sig***, ceci revient à tester l'existence du processus ***pid.***
- ✓ La fonction ***int raise (int sig)*** (non POSIX) permet d'envoyer un signal au processus courant (au processus lui-même).
- ✓ Les signaux sont évidemment sans effet sur les processus zombies puisqu'ils n'existent plus.

III-2/Interception et gestion des signaux

Nous avons vu qu'un signal pouvait être intercepté, transformé ou ignoré. Pour cela il faut utiliser une fonction écrite par le développeur, cette fonction sera appelée à la réception du signal spécifié. L'installation de cette fonction est appelée ***intercepter ou capturer un signal.***

La primitive ***signal()*** permet d'associer un traitement à la réception d'un signal.

Une autre primitive ***sigaction()*** permet de faire la même chose et présente l'avantage de définir précisément le comportement désiré. Elle permet également de ne pas poser de problèmes de compatibilité.

III-2-1/ *signal()* :

Définie par :

```
#include <signal.h>
 typedef void (*sighandler_t)(int);
 sighandler_t signal(int signum, sighandler_t handler);
```

Cette primitive permet d'intercepter et d'installer le gestionnaire ***handler*** pour le signal ***signum.*** La fonction ***handler*** reçoit un paramètre de type ***int*** pour identifier le signal reçu.
handler peut être SIG_IGN, SIG_DFL ou l'adresse d'une fonction définie par le programmeur.
Lors de l'arrivée d'un signal de numéro ***signum***, un des événements suivants se produit :

- ✓ Si le gestionnaire est SIG_IGN, le signal est ignoré.
- ✓ Si le gestionnaire est SIG_DFL, l'action associée au signal est entreprise.
- ✓ Si le gestionnaire est une fonction, alors ***handler*** est appelée avec le paramètre ***signum***.

III-2-2/*sigaction()* :

Définie par :

```
#include <signal.h>
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

Cette primitive permet d'intercepter le signal ***signum***. Le comportement de cette interception est décrit dans la structure ***sigaction *act***, le précédent comportement est mémorisé dans la structure ***sigaction *oldact***.

Le structure ***sigaction*** est définie comme suit :

```
struct sigaction
{
        void (* sa_handler) (int);
        void (* sa_sigaction) (int, siginfo t *, void *);
        sigse_ t sa_mask;
        int sa_flags;
        void (* sa_restorer) (void);
}
```

- ***sa_handler*** : est la fonction qui sera exécutée à la réception du signal ***signum*****,** ou la constante SIG_IGN pour ignorer le signal ou SIG_DFL pour exécuter l'action par défaut.
- ***sa_sigaction*** : structure utilisée uniquement lorsqu'on précise SA_SIGACTION dans le champ ***sa_flags*** (voir l'aide : man sigaction).
- ***sa_mask*** : est un masque de signaux à bloquer pendant l'exécution de la fonction de déroutement. Par défaut, le signal ayant appelé la fonction est lui aussi bloqué.
- ***sa_flags*** : spécifie un ensemble d'attributs qui modifie le comportement de la fonction de déroutement :
  - SA_NOCLDSTOP : si le siganl est SIG_CHILD, le processus ne recevra pas de notification de processus fils quand le fils reçoit un des signaux suivants : SIG_STOP, SIG_TSTP, SIG_TTIN, ou SIG_TTOU
  - SA_ONESHOT ou SA_RESETHAND : qui rétablit l'action à son comportement par défaut une fois que la fonction a été appelée.
  - SA_NOMASK ou SA_NODEFER : qui ne bloque pas le signal qui a appelé la fonction
  - SA_RESTART : qui réinitialise le gestionnaire de signal automatiquement.
  - SA_SIGINFO : qui reçoit deux arguments supplémentaires en complément du numéro du signal. Dans ce cas, il faut utiliser le champ *sa_sigaction* et non le champ *sa_handler*
- ***sa_restorer*** : obsolète et ne doit pas être utilisé.

La fonction *signal()* fait partie de l'interface C standard mais n'est pas POSIX. Il est donc préférable d'utiliser la fonction *sigaction()*.

Exemple d'utilisation de sigaction :

```
pi@raspberrypi:~/Documents/Signaux $ cat sigaction1.c
#define _GNU_SOURCE
#include <stdio.h>

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

void gestion_signal(int signum);

int main(void)
{
        struct sigaction action;
        action.sa_handler = gestion_signal;
        action.sa_flags = SA_RESTART;
        if (sigaction(SIGTERM,&action,NULL) != 0)
        {
```

```
                perror("sigaction");
                return EXIT_FAILURE;
        }
        while(1);
        return EXIT_SUCCESS;
}

void gestion_signal(int signum)
{
        char Texte[] = "\tFonction de déroutement\n";
        write(1,Texte,strlen(Texte));
        sprintf(Texte,"Numéro de signal reçu : %d\n",signum);
        write(1,Texte,strlen(Texte));
}
pi@raspberrypi:~/Documents/Signaux $
```

```
pi@raspberrypi:~/Documents/Signaux $ ./sigaction1 &
[1] 3618
pi@raspberrypi:~/Documents/Signaux $
```

```
pi@raspberrypi:~/Documents/Signaux $ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       486  0.2  4.9 215540 47216 tty7     Ssl+ 09:10   0:23 /usr/lib/xorg/X
root       488  0.0  0.2   5620  2796 tty1     Ss   09:10   0:00 /bin/login -f
pi         587  0.0  0.3   8492  3608 tty1     S+   09:10   0:00 -bash
pi        1292  0.0  0.4   8668  4112 pts/0    Ss   09:14   0:00 bash
pi        1306  0.0  0.3   8516  3628 pts/1    Ss+  09:15   0:00 bash
pi        3460  0.0  0.3   8492  3640 pts/2    Ss+  11:31   0:00 -bash
pi        3618 99.3  0.0   1720   340 pts/0    R    11:36   0:45 ./sigaction1
pi        3631  0.0  0.2   9788  2560 pts/0    R+   11:37   0:00 ps au
pi@raspberrypi:~/Documents/Signaux $
```

```
pi@raspberrypi:~/Documents/Signaux $ kill 3618
        Fonction de déroutement
Numéro de signal reçu : 15
pi@raspberrypi:~/Documents/Signaux $
```

```
pi@raspberrypi:~/Documents/Signaux $ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       486  0.2  4.9 215540 47216 tty7     Ssl+ 09:10   0:23 /usr/lib/xorg/X
root       488  0.0  0.2   5620  2796 tty1     Ss   09:10   0:00 /bin/login -f
pi         587  0.0  0.3   8492  3608 tty1     S+   09:10   0:00 -bash
pi        1292  0.0  0.4   8668  4112 pts/0    Ss   09:14   0:00 bash
pi        1306  0.0  0.3   8516  3628 pts/1    Ss+  09:15   0:00 bash
pi        3460  0.0  0.3   8492  3640 pts/2    Ss+  11:31   0:00 -bash
pi        3618 99.9  0.0   1720   340 pts/0    R    11:36   3:29 ./sigaction1
pi        3681  0.0  0.2   9788  2524 pts/0    R+   11:40   0:00 ps au
pi@raspberrypi:~/Documents/Signaux $ kill 3618
        Fonction de déroutement
Numéro de signal reçu : 15
pi@raspberrypi:~/Documents/Signaux $
```

```
pi@raspberrypi:~/Documents/Signaux $ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       486  0.2  4.9 215540 47216 tty7     Ssl+ 09:10   0:24 /usr/lib/xorg/X
root       488  0.0  0.2   5620  2796 tty1     Ss   09:10   0:00 /bin/login -f
pi         587  0.0  0.3   8492  3608 tty1     S+   09:10   0:00 -bash
pi        1292  0.0  0.4   8668  4112 pts/0    Ss   09:14   0:00 bash
pi        1306  0.0  0.3   8516  3628 pts/1    Ss+  09:15   0:00 bash
pi        3460  0.0  0.3   8492  3640 pts/2    Ss+  11:31   0:00 -bash
pi        3618  100  0.0   1720   340 pts/0    R    11:36   5:11 ./sigaction1
pi        3709  0.0  0.2   9788  2532 pts/0    R+   11:42   0:00 ps au
pi@raspberrypi:~/Documents/Signaux $ kill -SIGUSR1 3618
pi@raspberrypi:~/Documents/Signaux $ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       486  0.2  4.9 215540 47248 tty7     Ssl+ 09:10   0:24 /usr/lib/xorg/X
root       488  0.0  0.2   5620  2796 tty1     Ss   09:10   0:00 /bin/login -f
pi         587  0.0  0.3   8492  3608 tty1     S+   09:10   0:00 -bash
pi        1292  0.0  0.4   8668  4112 pts/0    Ss   09:14   0:00 bash
pi        1306  0.0  0.3   8516  3628 pts/1    Ss+  09:15   0:00 bash
pi        3460  0.0  0.3   8492  3640 pts/2    Ss+  11:31   0:00 -bash
pi        3736  0.0  0.2   9788  2464 pts/0    R+   11:43   0:00 ps au
[1]+  Signal #1 défini par l'usager      ./sigaction1
pi@raspberrypi:~/Documents/Signaux $
```

III-3/ Blocage de signaux

Un processus peut bloquer à volonté un ensemble de signaux, à l'exception du signal ***SIGKILL*** et du signal ***SIGSTOP***.

Cette opération est réalisée par l'appel système ***sigprocmask().***

```
#include <signal. H>
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

Cette fonction permet de bloquer ou de débloquer des signaux, mais aussi de fixer un nouveau masque ou de consulter l'ancien masque de blocage.

Elle permet donc l'installation d'un masque de blocage. Le nouveau masque est construit à partir de l'ensemble *set* et éventuellement du masque antérieur *oldset*. Le paramètre *how* permet de préciser la construction du nouvel ensemble :

- SIG_SETMASK : définit un nouveau masque,
- SIG_BLOCK : ajoute un ou des signaux,
- SIG_UNBLOCK : retire un ou des signaux.

L'utilité principal d'un blocage de signaux est la protection des sections critiques de code.la délivrance d'un signal s'effectue avant le retour de la fonction *sigprocmask.* Un

processus peut consulter la liste des signaux bloqués sans en demander la délivrance immédiate grâce à la fonction *sigpending*.

De même, la structure ***sigaction*** permet de bloquer certains signaux grâce au champ ***sa_mask***.

POSIX fournit des fonctions permettant la manipulation d'ensembles de signaux (type ***sigset_t***).

- ✓ int sigemptyset (sigset_t *set) : qui vide l'ensemble de signaux *set*,
- ✓ int sigfillset (sigset_t *set) : qui remplit totalement l'ensemble de signaux *set* en incluant tous les signaux,
- ✓ int sigaddset (sigset_t *set, int signum) : qui ajoute le signal *signum* à l'ensemble de signaux *set*,
- ✓ int sigdelset (sigset_t *set, int signum) : qui supprime le signal *signum* de l'ensemble de signaux *set*,
- ✓ int sigismember (const sigset_t *set, int signum) : qui teste si le signal *signum* est membre de l'ensemble de signaux *set*.

Exemple1 :

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#include <unistd.h>
#include <string.h>

void gestion_signal(int signum);

int main(void)
{
        struct sigaction action;
        sigset_t masque,pendant;

        action.sa_handler = gestion_signal;
        sigemptyset(&action.sa_mask);
        action.sa_flags = SA_RESTART;
        if (sigaction(SIGINT,&action,NULL) != 0)
        {
                perror("sigaction");
                return EXIT_FAILURE;
        }
        sigemptyset(&masque);
        sigaddset(&masque,SIGINT);
        sigprocmask(SIG_BLOCK,&masque,NULL);
        sleep(10);
```

```
        sigpending(&pendant);

        if (sigismember(&pendant,SIGINT))
                printf("Signal SIGINT pendant\n");
        sigprocmask(SIG_UNBLOCK,&masque,NULL);
                printf("Signal SIGINT débloqué\n");
        return EXIT_SUCCESS;
}

void gestion_signal(int signum)
{
        char Texte[50] = "\tFonction de déroutement\n";
        sprintf(Texte,"\tSignal reçu (%d) : %s\n",signum,strsignal(signum));
        write(1,Texte,strlen(Texte));
}
```

```
pi@raspberrypi:~/Documents/Signaux $ ./Blocage1 &
[1] 1603
pi@raspberrypi:~/Documents/Signaux $ kill -SIGINT 1603
pi@raspberrypi:~/Documents/Signaux $ Signal SIGINT pendant
        Signal reçu (2) : Interrupt
Signal SIGINT débloqué

[1]+  Fini                    ./Blocage1
pi@raspberrypi:~/Documents/Signaux $
```

Le signal SIGINT est dérouté vers la fonction *gestion_signal* qui affiche le numéro et le nom du signal reçu. Ce même signal est bloqué grâce aux fonctions de manipulations *sigaddset* et *sigprocmask*. Lorsque le processus est en sommeil, on envoie un signal SIGINT qui évidemment n'est pas pris en compte puisqu'il est bloqué. Avec les fonctions *sigpending* et *sigismember* on affiche un message si ce signal est pendant puis on démasque ce même signal. La fonction de déroutement est immédiatement appelée et le processus se termine.

III-4/ fonctions autorisées dans la fonction de gestion de signal

Si une fonction de déroutement est interrompue en cours d'exécution, lors de la reprise il est capital que le fonctionnement soit correct. Les fonctions utilisées dans une fonction de déroutement doivent donc être réentrantes. Les fonctions garanties réentrantes sont les suivantes :

access	alarm	cfgetispeed	cfgetospeed	cfsetispeed
cfsetospeed	chdir	chmod	chown	close
creat	dup2	dup	execle	execve
_exit	fcntl	fork	fstat	getegid
geteuid	getgid	getgroups	getpgrp	getpid
getppid	getuid	kill	link	lseek
mkdir	mkfifo	open	pathconf	pause
pipe	read	rename	rmdir	setgid
setpgid	setsid	setuid	sigaction	sigaddset

sigdelset	sigemtyset	sigfillset	sigismember	signal
Sigpending	sigprocmask	siguspend	sleep	stat
sysconf	tcdrain	tcflow	tcflush	tcgetattr
tcgetpgrp	tcsendbreak	tcsetattr	tcsetpgrp	time
times	umask	uname	unlink	utime
wait	waitpid	write		

*Si on veut afficher du texte à l'écran, il n'est donc pas recommandé d'utiliser **printf**. Il est préférable d'utiliser **write** avec 1 (stdout) comme "file descripteur".*

III-5/ Attente d'un signal

Un processus peut attendre l'arrivée d'un signal. Deux fonctions sont disponibles pour cela :

- ***int pause(void)***: qui attend n'importe quel signal. Cela permet au processus de ne pas consommer de temps CPU. Au réveil, le processus ne peut pas savoir quel signal il a reçu. (La fonction d'interception peut le savoir).
- ***Int sigsuspend(const sigset_t* mask)*** **:**cette fonction installe le masque de signaux désigné par l'ensemble *mask,* puis met le processus en sommeil jusqu'à l'arrivée du signal. Le précédent masque sera réinstallé au retour.

Exemple :

```
pi@raspberrypi:~/Documents/Signaux $ ./suspend &
[1] 3404
pi@raspberrypi:~/Documents/Signaux $ Tous les signaux sauf SIGTERM sont masqués
Envoi du signal SIGUSR1 du fils à       son père
Fin du fils
ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pi         629  0.0  0.3   8492  3736 tty1     S+   17:46   0:00 -bash
pi        1125  0.0  0.4   8980  4316 pts/0    Ss   17:53   0:00 bash
pi        3404  0.0  0.0   1852   316 pts/0    S    20:21   0:00 ./suspend
pi        3405  0.0  0.0      0     0 pts/0    Z    20:21   0:00 [suspend] <defu
pi        3406  0.0  0.2   9788  2496 pts/0    R+   20:21   0:00 ps u
pi@raspberrypi:~/Documents/Signaux $ kill -SIGILL 3404
pi@raspberrypi:~/Documents/Signaux $ kill 3404
        (pid    =       3404)   Signal  reçu    (15)    :Terminated
Déblocage de SIGUSR2
```

```
pi@raspberrypi:~/Documents/Signaux $ kill -SIGILL 3404
pi@raspberrypi:~/Documents/Signaux $ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pi         629  0.0  0.3   8492  3736 tty1     S+   17:46   0:00 -bash
pi        1125  0.0  0.4   8980  4316 pts/0    Ss   17:53   0:00 bash
pi        3404  0.0  0.0   1852   316 pts/0    S    20:21   0:00 ./suspend
pi        3405  0.0  0.0      0     0 pts/0    Z    20:21   0:00 [suspend] <defu
pi        3420  0.0  0.2   9788  2560 pts/0    R+   20:22   0:00 ps u
pi@raspberrypi:~/Documents/Signaux $ kill 3404
        (pid    =       3404)   Signal  reçu    (15)    :Terminated
Déblocage de SIGUSR1
        (pid    =       3404)   Signal  reçu    (10)    :User defined signal 1
Fin du père
pi@raspberrypi:~/Documents/Signaux $ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pi         629  0.0  0.3   8492  3736 tty1     S+   17:46   0:00 -bash
pi        1125  0.0  0.4   8980  4316 pts/0    Ss   17:53   0:00 bash
pi        3421  0.0  0.2   9788  2500 pts/0    R+   20:22   0:00 ps u
[1]+  Fini                    ./suspend
pi@raspberrypi:~/Documents/Signaux $
```

Explication :

Les signaux SIGTERM et SIGUSR1 sont déroutés vers la fonction *gestion_signal* qui affiche le numéro et le nom du signal reçu. Ensuite, le processus "fork" un fils qui lui envoie au bout d'une seconde un signal SIGUSR1. Pendant ce temps, le père a bloqué tous les signaux sauf SIGTERM et il se bloque en attente d'un signal non bloqué. Lorsqu'on lui envoie le signal SIGILL, le processus reste en attente. Lorsqu'on lui envoie le signal SIGTERM, il se réveille et libère le signal SIGUSR2 et se bloque à nouveau en attente d'un signal non bloqué. Le signal SIGILL est toujours sans influence alors que le signal SIGTERM le réveille. A son réveil, il libère SIGUSR1 et immédiatement la fonction de déroutement est appelée et réalise son affichage. On constate que pendant qu'il est en attente il ne consomme pas de temps CPU.

III-6/ Signal SIGALRM

Le signal SIGALRM est émis à un processus à la suite d'une demande formulée par le processus lui-même à l'aide de la fonction ***unsigned int alarm(unsigned int nb_sec);***

Cette fonction envoie le signal SIGALRM au bout de nb_sec secondes sauf si on fait appel à nouveau à cette fonction avant nb_sec secondes. On peut également utiliser la fonction ***int setitimer(int which, const struct itimerval* value, struct itimerval* ovalue)*** pour spécifier le temps.

Exemple :

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>

void gestion_signal(int signum);
int temps;

int main(void)
{
	struct sigaction action;
	action.sa_handler = gestion_signal;
	sigemptyset(&action.sa_mask);
	action.sa_flags = SA_RESTART;
	printf("\n je suis le processus : %d\n",getpid());
	if (sigaction(SIGALRM,&action,NULL) != 0)
	{
		perror("sigaction");
		return EXIT_FAILURE;
	}
	temps = 10;
	printf("Appuyez sur Entrée ou le programme s'arrête dans 10 secondes\n");
	alarm(1);
	getchar();
	alarm(0);
```

```
	printf("\n c'est fini \n");
	return EXIT_SUCCESS;
}
void gestion_signal(int signum)
{
	char Texte[50] = "\tFonction de déroutement\n";
	alarm(1);
	if (temps)
	{
		sprintf(Texte,"Il vous reste %d seconde",temps);
		if (temps != 1)
			strcat(Texte,"s");
		strcat(Texte,"\n");
		write(1,Texte,strlen(Texte));
		temps -= 1;
	}
	else
	{
		sprintf(Texte,"Trop tard, le programme se termine\n");
		write(1,Texte,strlen(Texte));
		exit(1);
	}
}
```

Exécution :

```
pi@raspberrypi:~/Documents/Signaux $ ./alarme
Appuyez sur Entrée ou le programme s'arrête dans 10 secondes
Il vous reste 9 secondes
Il vous reste 8 secondes
Il vous reste 7 secondes
Il vous reste 6 secondes
Il vous reste 5 secondes

pi@raspberrypi:~/Documents/Signaux $ ./alarme
Appuyez sur Entrée ou le programme s'arrête dans 10 secondes
Il vous reste 9 secondes
Il vous reste 8 secondes
Il vous reste 7 secondes
Il vous reste 6 secondes
Il vous reste 5 secondes
Il vous reste 4 secondes
Il vous reste 3 secondes
Il vous reste 2 secondes
Il vous reste 1 seconde
Trop tard, le programme se termine
pi@raspberrypi:~/Documents/Signaux $
```

III-7/ Signal SIGCHLD

Le signal SIGCHLD est automatiquement envoyé à un processus père lorsqu'un de ses processus fils se termine. Par défaut ce signal est ignoré. En interceptant ce signal, un processus peut prendre en compte de manière systématique la terminaison de ses processus fils et ainsi éliminer les processus zombie en utilisant les fonctions ***wait*** ou ***waitpid.***

III-8/ interception et recouvrement

Lors d'un recouvrement le code d'origine est écrasé, le comportement vis-à-vis des signaux n'est donc pas maintenu. Pour obtenir un comportement souhaité, il faut réinstaller les intercepteurs dans le nouveau code.