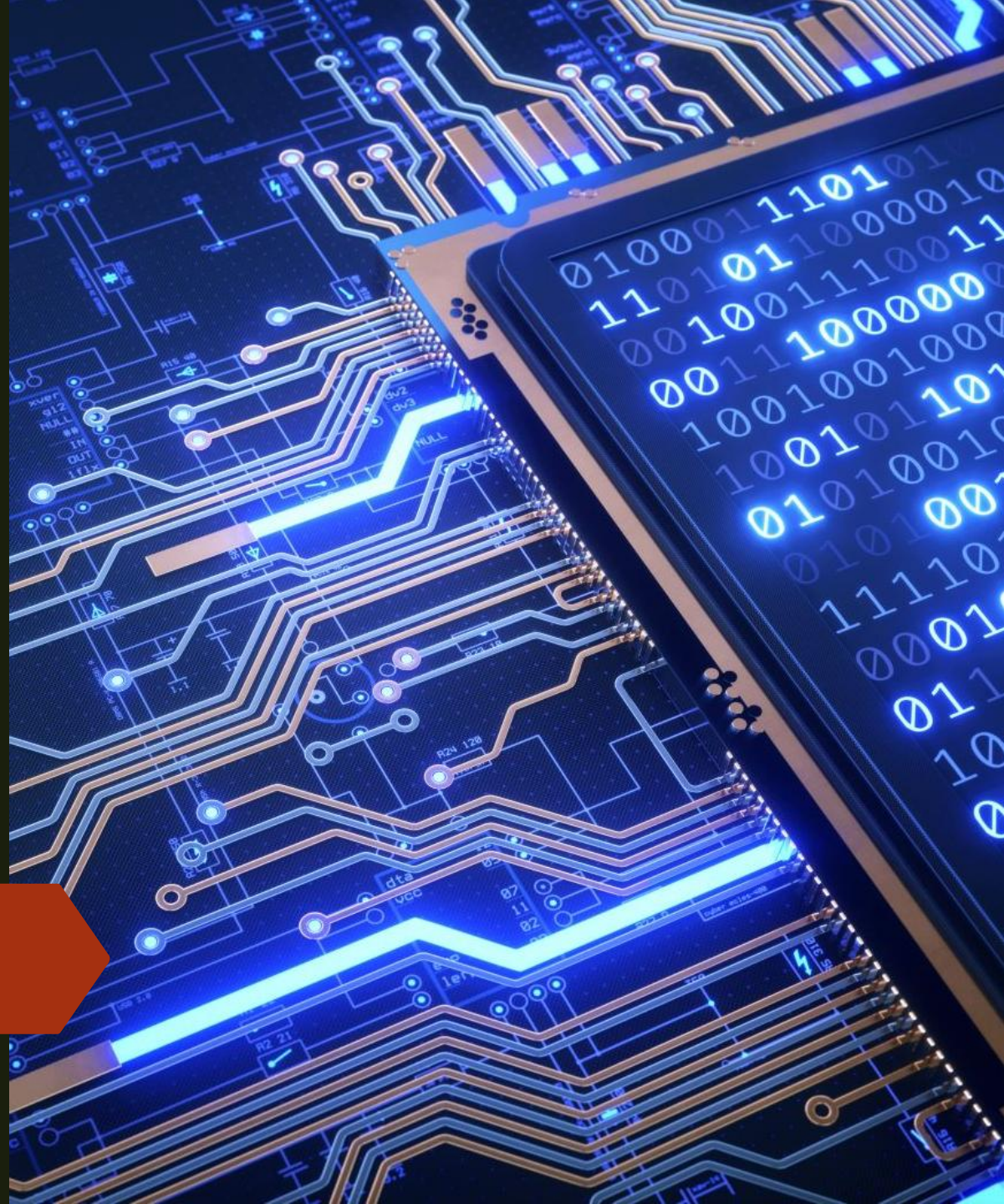


Développement Système

Les signaux



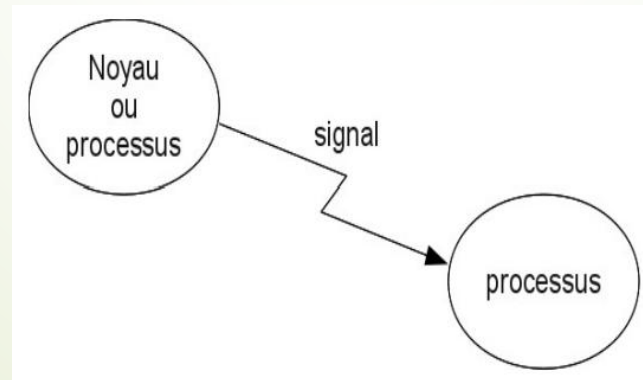
Développement système : les signaux

Introduction et définitions:

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

Sous Linux c'est un événement 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, d'une frappe au clavier, d'un Shell etc.,



Développement système : les signaux

➤ Un signal :

- ✓ est un événement asynchrone,
- ✓ peut-être déclenché par :

➤ le système :

- ✓ violation d'espace mémoire : **SIGSEGV**
- ✓ écriture dans un tube sans lecteur : **SIGPIPE**
- ✓ ...

➤ externe

- ✓ tuer un processus : `kill -9 pid` ou `kill SIGKILL pid`

➤ Les signaux peuvent être :

- ignorés,
- interceptés,
- bloqués.

Pour la plupart . . .

Développement système : les signaux

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

Les noms des signaux sont définis dans le fichier d'en-tête [*signal.h*](#). Il est possible d'obtenir la liste des signaux sous le Shell grâce à la commande *kill -l*.

- 0 : seul signal qui n'a pas de nom
- 1 à 31 : signaux classiques
- 32 à 64 : signaux « temps réels »

Développement système : les signaux

Ni ignoré ni intercepté

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	

Développement système : les signaux

Portabilité :

Le comportement des signaux et des systèmes d'exploitation UNIX n'est pas tout à fait normalisé. La compatibilité n'est pas assurée malgré les efforts de normalisation.

Il faut donc utiliser les noms symboliques et non les valeurs numériques.

Les traitements standards de chaque signal peuvent différer d'un système UNIX à un autre.

*Pour bien connaître le comportement des processus pour chaque signal il est conseillé de lire les informations dans le "**man**" du système.*

- ➡ 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).*

Développement système : les signaux

Comportement d'un signal:

Lorsqu'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.

il peut être imaginé comme une sorte d'impulsion qui oblige le processus cible à prendre immédiatement une mesure spécifique

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

Développement système : les signaux

Comportement d'un signal: (suite)

Quand un processus reçoit un signal, il peut le traiter, le bloquer ou l'ignorer.

- 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.

Développement système : les signaux

Comportement d'un signal: (suite)

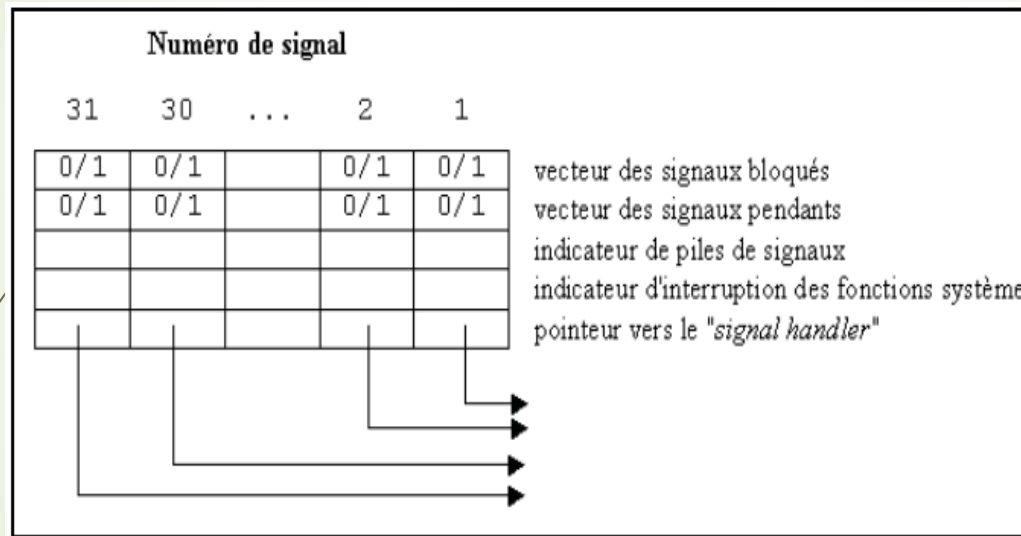
Un processus doit être en mesure d'associer au signal qu'il est susceptible de recevoir :

- 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 précisant 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.

Développement système : les signaux

Comportement d'un signal: (suite)

Les informations précédentes 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

Développement système : les signaux

Emission d'un signal :

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

- ***Sous le shell :***

kill – SIGxx PID

- SIGXX : le signal à émettre (SIGUSR1, SIGPIPE, SIGILL, etc)
- PID : le processus qui recevra le signal

La commande kill est capable d'émettre tous les signaux vers un processus donné(man 1 kill).

Si on ne précise pas de signal, c'est le signal SIGTERM qui sera envoyé.

- ***En langage C :***

int kill(pid_t pid, int sig);

- pid : le processus qui recevra le signal
- sig : le signal à émettre

Si sig vaut 0 : on teste l'existence du processus

Développement système : les signaux

Émission d'un signal :

Exemple :

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

int interrompre(int pid)
{
    if (kill(pid, SIGINT) == 0) // ou : kill(pid, 2);
        return 0;
    else perror("kill");
    return 1;
}
```

SIGINT : signal d'interruption émis par l'utilisateur (ctrl C)

Développement système : les signaux

Interception d'un signal

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

Intercepte le signal *signum* et exécute la fonction *handler* du type : *void nom_fonction(int);*

(non POSIX)

➡ **int sigaction(int *signum*, const struct sigaction **act*, struct sigaction **oldact*);**

Intercepte le signal *signum* et installe le comportement décrit par la structure *act*.

L'ancien comportement est décrit par la structure *oldact*.

Développement système : les signaux

Information d'un signal

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

int main (int argc, char* argv[])
{
    if (argc>1)
    {
        int sig = atoi(argv[1]);
        printf("information sur le signal n° %d \n:
%s\n",sig,signal (sig) );
        sleep(10);
    }
    else printf("\n usage : <%s> n° signal\n ");
}
```

On teste (EX1)...

Développement système : les signaux

```
1  #include <stdio.h>
2  #include <signal.h>
3  #include <unistd.h>
4  void monTraitement(int numSignal);
5
6  int main(void)
7  {
8      if(signal(SIGUSR1,monTraitement)==SIG_ERR) printf("\nsignal non intercepte\n");
9      printf("Merci de m'envoyer 3 signaux par la commande:\n");
10     printf("kill - SIGUSR1 %d\n",getpid());
11     pause();
12     printf("plus que 2 signaux SIGUSR1");
13     pause();
14     printf("plus que 1 signal SIGUSR1");
15     pause();
16     printf("C'est termine !!\n");
17     return 0;
18 }
19
20 void monTraitement(int numSignal)
21 {
22     signal(numSignal,monTraitement);
23     printf("\n j'ai reçu le signal %s\n",sys_siglist[numSignal]);
24 }
```

On teste (EX2)...

Développement système : les signaux

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

int compteur = 1; /* variable globale */

void gestionnaire(int numSig)
{
    signal(numSig, gestionnaire); /* réinstaller le gestionnaire de signal */
    printf("Compteur=%d\n", compteur++);
}

int main()
{
    printf("Je suis le PID=%d\n", getpid());
    signal(SIGUSR1, gestionnaire); /* installe un gestionnaire de signal pour SIGUSR1 */
    while(1); /* peut faire autre chose ... */

    return 0;
}
```

On teste (Ex3)

Développement système : les signaux

Description du comportement

```
struct sigaction
```

```
{
```

```
void (* sa_handler) (int);
```

```
void (* sa_sigaction) (int, siginfo_t *, void *);
```

```
sigset_t sa_mask;
```

```
int sa_flags;
```

```
void (* sa_restorer) (void);
```

```
}
```

Fonction exécutée

Non utilisé si
sa_handler utilisé

Signaux masqués pendant
l'exécution de sa_handler

Attribut de
comportement

SA_ONESHOT

SA_RESTART

Obsolète

Développement système : les signaux

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

int compteur = 1; /* variable globale */

void gestionnaire(int numSig)
{
    printf("Compteur=%d\n", compteur++);
}

int main()
{
    struct sigaction action;

    printf("Je suis le PID=%d\n", getpid());

    action.sa_handler = gestionnaire;
    sigemptyset(&action.sa_mask);
    action.sa_flags = 0;
    sigaction(SIGUSR1, &action, NULL);

    while(1); /* peut faire autre chose ... */

    return 0;
}
```

On teste (Ex4)

Développement système : les signaux

Voir les signaux interceptés et ignorés : **ps s**

Ignorés Interceptés

```
pi@raspberrypi:~ $ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	587	00000000	00000000	00380004	4b817efb	S+	tty1	0:00	-bash
1000	1292	00000000	00010000	00380004	4b817efb	Ss	pts/0	0:00	bash
1000	1304	00000000	00000000	00000000	00000200	R+	pts/0	0:51	./compt
1000	1306	00000000	00010000	00380004	4b817efb	Ss	pts/1	0:00	bash
1000	1327	00000000	00000000	00000000	<f3d1fe79	R+	pts/1	0:00	ps s

```
pi@raspberrypi:~ $
```

En binaire :

0000 0010 0000 0000

Bit10 : SIGUSR1

Développement système : les signaux

Blocage des 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);
```

Elle permet 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.

Développement système : les signaux

Blocage des signaux

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

int sigemptyset (sigset_t *set);

ne positionne aucun signal dans l'ensemble de signaux *set*

int sigaddset (sigset_t *set, int signum);

ajoute le signal *signum* à l'ensemble *set*

int sigfillset (sigset_t *set);

positionne tous les signaux de l'ensemble *set*

int sigdelset (sigset_t *set, int signum);

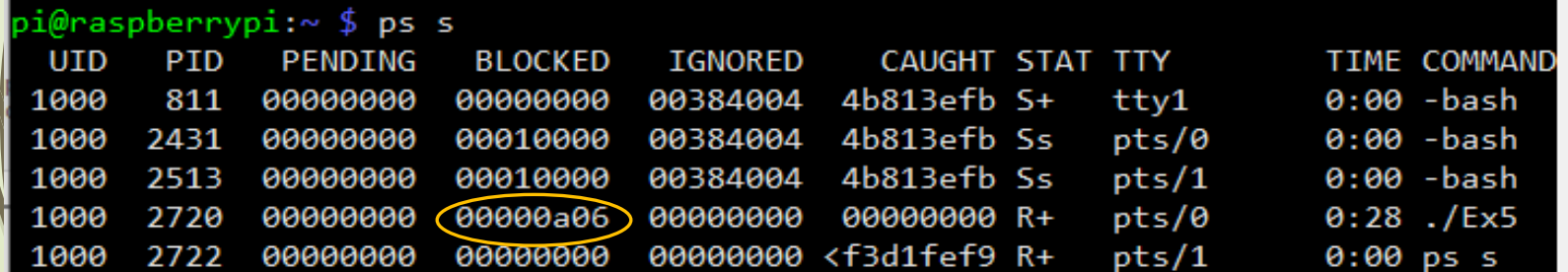
supprime le signal *signum* de l'ensemble *set*

Développement système : les signaux

Exemple: (Ex5)

```
int main(void)
{
    sigset_t ens1;
    sigemptyset(&ens1);
    sigaddset(&ens1, SIGINT);
    sigaddset(&ens1, SIGQUIT);
    sigaddset(&ens1, SIGUSR1);
    sigaddset(&ens1, SIGUSR2);
    sigprocmask(SIG_SETMASK, &ens1, NULL);
    while(1);
    return EXIT_SUCCESS;
}
```

1010 0000 0110



```
pi@raspberrypi:~ $ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	811	00000000	00000000	00384004	4b813efb	S+	tty1	0:00	-bash
1000	2431	00000000	00010000	00384004	4b813efb	Ss	pts/0	0:00	-bash
1000	2513	00000000	00010000	00384004	4b813efb	Ss	pts/1	0:00	-bash
1000	2720	00000000	00000a06	00000000	00000000	R+	pts/0	0:28	./Ex5
1000	2722	00000000	00000000	00000000	<f3d1fef9	R+	pts/1	0:00	ps s

Développement système : les signaux

Exemple : (Ex6)

```
int main(void)
{
    sigset_t ens1;
    sigfillset(&ens1);
    sigdelset(&ens1,SIGUSR1);
    sigdelset(&ens1,SIGUSR2);
    sigprocmask(SIG_SETMASK,&ens1,NULL);
    while(1);
    return EXIT_SUCCESS;
}
```

1111 1110 0111 1111 1111 1011 1111 0100 1111 1111

9,10,12,19,32 et 33

pi@raspberrypi:~ \$ ps s

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	811	00000000	00000000	00384004	4b813efb	S+	tty1	0:00	-bash
1000	2431	00000000	00010000	00384004	4b813efb	Ss	pts/0	0:00	-bash
1000	2513	00000000	00010000	00384004	4b813efb	Ss	pts/1	0:00	-bash
1000	2872	00000000	<7ffbf4ff	00000000	00000000	R+	pts/0	1:00	./Ex6
1000	2874	00000000	00000000	00000000	<f3d1fef9	R+	pts/1	0:00	ps s

Développement système : les signaux

Exemple 7:

```
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 débloquent\n");
    return EXIT_SUCCESS;
}
```


Développement système : les signaux

```
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));
}
```

Développement système : les signaux

```
pi@raspberrypi:~ $ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	811	00000000	00000000	00384004	4b813efb	S+	tty1	0:00	-bash
1000	2431	00000000	00010000	00384004	4b813efb	Ss	pts/0	0:00	-bash
1000	2513	00000000	00010000	00384004	4b813efb	Ss	pts/1	0:00	-bash
1000	3098	00000000	00000002	00000000	00000002	S+	pts/0	0:00	./Ex7
1000	3099	00000000	00000000	00000000	<f3d1fef9	R+	pts/1	0:00	ps s

```
pi@raspberrypi:~ $ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	811	00000000	00000000	00384004	4b813efb	S+	tty1	0:00	-bash
1000	2431	00000000	00010000	00384004	4b813efb	Ss	pts/0	0:00	-bash
1000	2513	00000000	00010000	00384004	4b813efb	Ss	pts/1	0:00	-bash
1000	3098	00000002	00000002	00000000	00000002	S+	pts/0	0:00	./Ex7
1000	3100	00000000	00000000	00000000	<f3d1fef9	R+	pts/1	0:00	ps s

```
pi@raspberrypi:~/Documents/ExCoursSignaux $ ./Ex7
```

je suis le processus : 3098
Signal reçu (2) : Interrupt
Signal SIGINT débloqué

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

Développement système : les signaux

Les signaux pending

- Un processus peut bloquer des signaux.
- Si un autre processus émet un signal bloqué à ce processus, ce signal devient pending.
- Lorsque le processus débloque ce signal, le système lui envoie le signal qui était en attente.
- Il est possible de connaître les signaux pendants :

```
int sigpending(sigset_t *set);
```

remplit set avec les signaux en attente.

Développement système : les signaux

Exemple 8:

```
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
    sigset_t ens1, ens2;
    int sig;
    sigemptyset(&ens1);
    sigaddset(&ens1, SIGINT);
    sigaddset(&ens1, SIGQUIT);
    sigaddset(&ens1, SIGUSR1);
    sigaddset(&ens1, SIGUSR2);
    sigprocmask(SIG_SETMASK, &ens1, NULL);
    printf("\t\nMasque installe (pause 30 secondes)\n");
    sleep(30);
    printf("\t\nLecture des signaux pendants\n");
    sigpending(&ens2);
    for (sig = 1; sig<31; sig++)
    {
        if (sigismember(&ens2, sig))
            printf("%d\t", sig);
    }
    printf("\t\nDeblocage des signaux\n");
    sigemptyset(&ens1);
    sigprocmask(SIG_SETMASK, &ens1, NULL);
    printf("\t\nFin du processus\n");    //sans doute pas exécuté
    return EXIT_SUCCESS;
}
```

Développement système : les signaux

Exécution :

```
pi@raspberrypi:~/Documents/ExCoursSignaux $  
je suis le processus : 3171  
  
Masque installe (pause 30 secondes)  
kill -10 3171  
pi@raspberrypi:~/Documents/ExCoursSignaux $ ps s  
  UID    PID  PENDING   BLOCKED   IGNORED   CAUGHT  STAT  TTY      TIME COMMAND  
  1000    811    00000000   00000000   00384004  4b813efb S+    tty1     0:00 -bash  
  1000   2431    00000000   00010000   00384004  4b813efb Ss    pts/0    0:00 -bash  
  1000   2513    00000000   00000000   00384004  4b813efb Ss+   pts/1    0:00 -bash  
  1000   3171    00000200   00000a06   00000000  00000000 S     pts/0    0:00 ./Ex8  
  1000   3172    00000000   00000000   00000000  <f3d1fef9 R+    pts/0    0:00 ps s  
pi@raspberrypi:~/Documents/ExCoursSignaux $ kill -2 3171  
pi@raspberrypi:~/Documents/ExCoursSignaux $ ps s  
  UID    PID  PENDING   BLOCKED   IGNORED   CAUGHT  STAT  TTY      TIME COMMAND  
  1000    811    00000000   00000000   00384004  4b813efb S+    tty1     0:00 -bash  
  1000   2431    00000000   00010000   00384004  4b813efb Ss    pts/0    0:00 -bash  
  1000   2513    00000000   00000000   00384004  4b813efb Ss+   pts/1    0:00 -bash  
  1000   3171    00000202   00000a06   00000000  00000000 S     pts/0    0:00 ./Ex8  
  1000   3173    00000000   00000000   00000000  <f3d1fef9 R+    pts/0    0:00 ps s  
pi@raspberrypi:~/Documents/ExCoursSignaux $  
Lecture des signaux pendants  
2      10  
Deblocage des signaux  
pi@raspberrypi:~/Documents/ExCoursSignaux $
```


Développement système : les signaux

Exécution :

```
pi@raspberrypi:~/Documents/Signaux $ kill -SIGINT 5754
pi@raspberrypi:~/Documents/Signaux $ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	629	00000000	00000000	00380004	4b817efb	S+	tty1	0:00	-bash
1000	1125	00000000	00010000	00380004	4b817efb	Ss	pts/0	0:00	bash
1000	4887	00000000	00000000	00380004	4b817efb	Ss+	pts/1	0:00	bash
1000	5754	00000a02	00000a06	00000000	00000000	S	pts/0	0:00	./Exemple3
1000	5771	00000000	00000000	00000000	<f3d1fef9	R+	pts/0	0:00	ps s

```
pi@raspberrypi:~/Documents/Signaux $
Lecture des signaux pendants
2      10      12
Déblocage des signaux

[1]+  Interromptre      ./Exemple3
pi@raspberrypi:~/Documents/Signaux $
```

Développement système : les signaux

Attendre un signal

C'est une attente non active.

```
int pause(void);
```

attend n'importe quel signal.

```
int sigsuspend(const sigset_t *mask);
```

installe le masque *mask* et attend un signal non masqué par *mask*.

Développement système : les signaux

Exemple 9:

```
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void gestion_signal(int signum);
int main(void)
{
    struct sigaction action;
    sigset_t masque;
    sigset_t masque_attente;
    action.sa_handler = gestion_signal;
    action.sa_flags = SA_RESTART;
    printf("\n je suis le processus : %d\n",getpid());
    sigaction(SIGUSR1,&action,NULL);
    sigaction(SIGUSR2,&action,NULL);
    sigaction(SIGTERM,&action,NULL);
    printf("Le signal SIGTERM est masque\n");
    sigemptyset(&masque);
    sigaddset(&masque,SIGTERM);
    sigprocmask(SIG_SETMASK,&masque,NULL);
    sigfillset(&masque_attente);
    sigdelset(&masque_attente, SIGUSR1);
    sigsuspend(&masque_attente);
    printf("Reception du signal SIGUSR1 Fin \n");
    return EXIT_SUCCESS;
}

void gestion_signal(int signum)
{
    char Texte[50] = "\tFonction de deroutement\n";
    write(1,Texte,strlen(Texte));
    sprintf(Texte,"\t(pid = %d) Signal recu (%d) : %s\n",getpid(),signum,signal(signum));
}
```

Développement système : les signaux

Exécution :

```
pi@raspberrypi:~/Documents/ExCoursSignaux $ ./Ex9 &
[1] 3270
pi@raspberrypi:~/Documents/ExCoursSignaux $
  je suis le processus : 3270
  Le signal SIGTERM est masque
ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	811	00000000	00000000	00384004	4b813efb	S+	tty1	0:00	-bash
1000	2431	00000000	00010000	00384004	4b813efb	Ss	pts/0	0:00	-bash
1000	2513	00000000	00000000	00384004	4b813efb	Ss+	pts/1	0:00	-bash
1000	3270	00000000	<7ffbfcff	00000000	00004a00	S	pts/0	0:00	./Ex9
1000	3271	00000000	00000000	00000000	<f3d1fef9	R+	pts/0	0:00	ps s

```
pi@raspberrypi:~/Documents/ExCoursSignaux $ kill -2 3270
pi@raspberrypi:~/Documents/ExCoursSignaux $ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	811	00000000	00000000	00384004	4b813efb	S+	tty1	0:00	-bash
1000	2431	00000000	00010000	00384004	4b813efb	Ss	pts/0	0:00	-bash
1000	2513	00000000	00000000	00384004	4b813efb	Ss+	pts/1	0:00	-bash
1000	3270	00000002	<7ffbfcff	00000000	00004a00	S	pts/0	0:00	./Ex9
1000	3272	00000000	00000000	00000000	<f3d1fef9	R+	pts/0	0:00	ps s

Développement système : les signaux

Exécution :

```
pi@raspberrypi:~/Documents/ExCoursSignaux $ kill -12 3270
pi@raspberrypi:~/Documents/ExCoursSignaux $ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	811	00000000	00000000	00384004	4b813efb	S+	tty1	0:00	-bash
1000	2431	00000000	00010000	00384004	4b813efb	Ss	pts/0	0:00	-bash
1000	2513	00000000	00000000	00384004	4b813efb	Ss+	pts/1	0:00	-bash
1000	3270	00000802	<7ffbfccff	00000000	00004a00	S	pts/0	0:00	./Ex9
1000	3273	00000000	00000000	00000000	<f3d1fef9	R+	pts/0	0:00	ps s

```
pi@raspberrypi:~/Documents/ExCoursSignaux $ kill -10 3270
(pid = 3270) Signal reçu (10) : User defined signal 1
pi@raspberrypi:~/Documents/ExCoursSignaux $ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	811	00000000	00000000	00384004	4b813efb	S+	tty1	0:00	-bash
1000	2431	00000000	00010000	00384004	4b813efb	Ss	pts/0	0:00	-bash
1000	2513	00000000	00000000	00384004	4b813efb	Ss+	pts/1	0:00	-bash
1000	3274	00000000	00000000	00000000	<f3d1fef9	R+	pts/0	0:00	ps s

```
[1]+  Interrompte ./Ex9
pi@raspberrypi:~/Documents/ExCoursSignaux $
```


Développement système : les signaux

Signaux particuliers:

➡ SIGALRM

Ce signal est émis à un processus à la suite d'une demande formulée par le processus lui-même à l'aide de la fonction alarm :

`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.

Développement système : les signaux

➤ SIGCHLD

Que se passe-t-il lors de la création d'un fils ?

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*.

Que se passe-t-il lors d'un 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.

Développement système : les signaux

Exemple 11

```
#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;
    action.sa_handler = gestion_signal;
    sigemptyset(&masque);
    action.sa_flags = SA_RESTART;
    printf("\n je suis le processus : %d\n",getpid());
    sigaction(SIGUSR1,&action,NULL);
    sigaction(SIGTERM,&action,NULL);
    if (fork() == 0)
    {
        sleep(5);
        printf("Envoi du signal SIGUSR1 du fils à son père\n");
        kill(getppid(),SIGUSR1);
        printf("Fin du fils\n");
        return EXIT_SUCCESS;
    }
}
```

Développement système : les signaux

Exemple 11 : (suite)

else

{

printf("Tous les signaux sont masqués sauf SIGTERM \n");

sigfillset(&masque);

sigdelset(&masque,SIGTERM);

sigprocmask(SIG_SETMASK,&masque,NULL);

sigsuspend(&masque);

printf("\nDéblocage de SIGUSR1\n");

sigdelset(&masque,SIGUSR1);

sigprocmask(SIG_SETMASK,&masque,NULL);

printf("Fin du père\n");

return EXIT_SUCCESS;

}

}

Développement système : les signaux

Exemple 11: (suite)

```
void gestion_signal(int signum)
{
    char Texte[50] = "\tFonction de déroutement\n";
    write(1,Texte,strlen(Texte));
    sprintf(Texte,"\t(pid = %d) Signal reçu (%d) :%s\n",getpid(),signum,signal(signum));

    write(1,Texte,strlen(Texte));
}
```

Développement système : les signaux

Exemple 11 : Exécution

```
pi@raspberrypi:~/Documents/ExCoursSignaux $ ./Ex11
```

```
je suis le processus : 2764  
Tous les signaux sont masqués on débloquent SIGTERM  
Envoi du signal SIGUSR1 du fils à son père  
Fin du fils
```

```
pi@raspberrypi:~ $ ps s
```

UID	PID	PENDING	BLOCKED	IGNORED	CAUGHT	STAT	TTY	TIME	COMMAND
1000	823	00000000	00000000	00384004	4b813efb	S+	tty1	0:00	-bash
1000	2270	00000000	00010000	00384004	4b813efb	Ss	pts/0	0:00	-bash
1000	2297	00000000	00010000	00384004	4b813efb	Ss	pts/1	0:00	-bash
1000	2764	00010200	07ffbbeff	00000000	00004200	S+	pts/0	0:00	./Ex11
1000	2765	00000000	00000000	00000000	00004200	Z+	pts/0	0:00	[Ex11]
1000	2777	00000000	00000000	00000000	<f3d1fef9	R+	pts/1	0:00	ps s

```
pi@raspberrypi:~ $ kill -15 2764
```

```
Fonction de déroutement
```

```
(pid = 2764) Signal reçu (15) :Terminated
```

```
Déblocage de SIGUSR1
```

```
Fonction de déroutement
```

```
(pid = 2764) Signal reçu (10) :User defined signal 1
```

```
Fin du père
```