Les processus : exec, la communication par signaux

Luigi Santocanale

Laboratoire d'Informatique Fondamentale,
Centre de Mathématiques et Informatique,
39, rue Joliot-Curie - F-13453 Marseille

11 octobre 2005

Plan

exec : exécuter un fichier exécutable

But :

- lancer (exécuter) un fichier exécutable.

Recouvrement :

- remplacement en mémoire d'un processus par l'image d'un nouveaux exécutable,
- on garde l'identité d'un processus.

Attention :

- on ne revient pas d'un appel système exec.

exec : exécuter un fichier exécutable

But :

- lancer (exécuter) un fichier exécutable.

Recouvrement :

- remplacement en mémoire d'un processus par l'image d'un nouveaux exécutable,
- on garde l'identité d'un processus.

Attention :

- on ne revient pas d'un appel système exec.

exec : exécuter un fichier exécutable

But :

- lancer (exécuter) un fichier exécutable.

Recouvrement :

- remplacement en mémoire d'un processus par l'image d'un nouveaux exécutable,
- on garde l'identité d'un processus.

Attention :

- on ne revient pas d'un appel système exec.

Algorithme exec (Bach 1989)

```
algorithme exec
entrée: (1) nom d'un fichier
        (2) liste de paramètres
        (3) liste des variables d'environnement
sortie: néant
{
    accéder à l'i-noeud du fichier (algorithme iput);
    vérifier que le fichier est exécutable et que l'utilisateur a la
                                                permission de l'exécuter;
    lire l'en-tête du fichier, vérifier que c'est un module chargeable;
    copier les paramètres de l'exec de l'ancien espace
                                            d'adressage dans l'espace système;
    for (chaque région attachée au processus)
        détacher toutes les anciennes régions (algorithme detachreg);
    for (chaque région spécifiée dans le module chargeable)
    {
        s'allouer de nouvelles régions (algorithme allocreg);
        attacher les régions (algorithme attachreg);
        charger la région en mémoire si cela est approprié
                                            (algorithme loadreg);
    }
    copier les paramètres de l'exec dans la nouvelle région
                                                de pile de l'utilisateur;
    traitement spécial pour les programmes "setuid" ou mis au point;
    initialiser la zone de sauvegarde des registres  utilisateur de
                        telle façon à pouvoir retourner en mode utilisateur;
    libérer l'i-noeud du fichier (algorithme iput);
}
```

main

```
int main(int argc, char * argv[], char * arge[]);
```

argc : nombre d'arguments

argv[] : tableau de chaînes de caractères contenant la liste des paramètres – `argv[0]` la commande donné. Le tableau est terminé par `NULL`

arge[] : tableau de chaînes de caractères permettant l'acces à l'environnement

Remarques : *arge[]* style ancien C, on peut utiliser `getenv`.

main

```
int main(int argc, char * argv[], char * arge[]);
```

argc : nombre d'arguments

argv[] : tableau de chaînes de caractères contenant la liste des paramètres – `argv[0]` la commande donné. Le tableau est terminé par `NULL`

arge[] : tableau de chaînes de caractères permettant l'acces à l'environnement

Remarques : *arge[]* style ancien C, on peut utiliser `getenv`.

main

```
int main(int argc, char * argv[], char * arge[]);
```

argc : nombre d'arguments

argv[] : tableau de chaînes de caractères contenant la liste des paramètres – `argv[0]` la commande donné. Le tableau est terminé par `NULL`

arge[] : tableau de chaînes de caractères permettant l'acces à l'environnement

Remarques : *arge[]* style ancien C, on peut utiliser `getenv`.

main

```
int main(int argc, char * argv[], char * arge[]);
```

argc : nombre d'arguments

argv[] : tableau de chaînes de caractères contenant la liste des paramètres – `argv[0]` la commande donné. Le tableau est terminé par `NULL`

arge[] : tableau de chaînes de caractères permettant l'acces à l'environnement

Remarques : *arge[]* style ancien C, on peut utiliser `getenv`.

main

```
int main(int argc, char * argv[], char * arge[]);
```

argc : nombre d'arguments
argv[] : tableau de chaînes de caractères contenant la liste des paramètres – `argv[0]` la commande donné. Le tableau est terminé par `NULL`
arge[] : tableau de chaînes de caractères permettant l'acces à l'environnement

Remarques : *arge[]* style ancien C, on peut utiliser `getenv`.

getenv

```
#include <stdlib.h>
...
const char *name = "HOME";
char *value;


value = getenv(name);
```

Programme : exexec.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <wait.h>
 4  #include <unistd.h>
 5
 6  int main()
 7  {
 8    char *args[] =
 9      {
10        /* argv[0] : nom de la commande */
11        "monls",
12        /* argv[1] : premier paramètre */
13        "-l",
14        /* argv[2] : deuxième paramètre */
15        "monfichier",
16        NULL
17      };
18    int status;
```

Programme : exexec.c(II)

```
20    switch(fork())
21      {
22      case -1 : exit(EXIT_FAILURE);
23      case 0 :
24        if(execvp("ls",args) < 0)
25          exit(EXIT_FAILURE);
26        /* On ne revient pas ici */
27      default:
28        wait(&status);
29        if(WIFEXITED(status))
30          printf("Le processus nommée %s"
31                 " s'est terminé avec code %d.\n",
32                 args[0],WEXITSTATUS(status));
33      }
34
35    exit(EXIT_SUCCESS);
36  }
```

Session : exexec

```
[lsantoca@localhost exec]$ touch monfichier; a.out
-rw-r--r--  1 lsantoca lsantoca 0 oct 11 16:01 monfichier
Le processus nommée monls s'est terminé avec code 0.
[lsantoca@localhost exec]$ rm monfichier ; a.out
monls: monfichier: No such file or directory
Le processus nommée monls s'est terminé avec code 1.
[lsantoca@localhost exec]$
```

Structure de la famille exec

Quand on exécute un fichier exécutable on peut :

- le chercher
  - par rapport au répertoire de travail
  - `p` : dans les répertoires de la variable d'environnement `PATH`
- passer les arguments en tant que
  - tableau de paramètres
  - `l` : liste explicite de paramètres
- utiliser
  - l'enivrommenemnr courant,
  - `e` : un nouveau environnement.

Structure de la famille exec

Quand on exécute un fichier exécutable on peut :

- le chercher
  - par rapport au répertoire de travail
  - p : dans les répertoires de la variable d'environnement PATH
- passer les arguments en tant que
  - tableau de paramètres
  - l : liste explicite de paramètres
- utiliser
  - l'enivrommenemnr courant,
  - e : un nouveau environnement.

Structure de la famille exec

Quand on exécute un fichier exécutable on peut :

- le chercher
  - par rapport au répertoire de travail
  - p : dans les répertoires de la variable d'environnement PATH
- passer les arguments en tant que
  - tableau de paramètres
  - l : liste explicite de paramètres
- utiliser
  - l'enivrommenemnr courant,
  - e : un nouveau environnement.

Structure de la famille exec

Quand on exécute un fichier exécutable on peut :

- le chercher
  - par rapport au répertoire de travail
  - `p` : dans les répertoires de la variable d'environnement `PATH`
- passer les arguments en tant que
  - tableau de paramètres
  - `l` : liste explicite de paramètres
- utiliser
  - l'enivrommenemnr courant,
  - `e` : un nouveau environnement.

exec, avec tableaux

```
#include <unistd.h>
```

```
int
execv(const char * ref, const char * argv[]);
```

ref : le nom l'exécutable sur le disque

argv[] : le tableau des paramètres

```
int
execvp(const char * ref, const char * argv[]);
```

Remarques : On cherche l'exécutable dans le PATH

```
int execve(const char * ref, const char * argv[],
                             const char * arge[]);
```

arge[] : tableau contenant l'environnement

exec, avec tableaux

```
#include <unistd.h>
```

int
execv(const char * *ref*, const char * *argv[]*);

ref : le nom l'exécutable sur le disque

argv[] : le tableau des paramètres

int
execvp(const char * *ref*, const char * *argv[]*);

Remarques : On cherche l'exécutable dans le PATH

int execve(const char * *ref*, const char * *argv[]*,
const char * *arge[]*);

arge[] : tableau contenant l'environnement

exec, avec tableaux

```
#include <unistd.h>
```

int
execv(const char * *ref*, const char * *argv[]*);

ref : le nom l'exécutable sur le disque

argv[] : le tableau des paramètres

int
execvp(const char * *ref*, const char * *argv[]*);

Remarques : On cherche l'exécutable dans le PATH

int execve(const char * *ref*, const char * *argv[]*,
const char * *arge[]*);

arge[] : tableau contenant l'environnement

exec, avec tableaux

```
#include <unistd.h>
```

int
execv(const char * *ref*, const char * *argv[]*);

ref : le nom l'exécutable sur le disque

argv[] : le tableau des paramètres

int
execvp(const char * *ref*, const char * *argv[]*);

Remarques : On cherche l'exécutable dans le PATH

int execve(const char * *ref*, const char * *argv[]*,
const char * *arge[]*);

arge[] : tableau contenant l'environnement

exec, avec listes

```
#include <unistd.h>
```

```
int
execl(const char * ref,
      const char * arg0, const char * arg1, ..., NULL);
```

ref : le nom l'exécutable sur le disque
arg0 : le nom qu'on veut donner au processus, le argv[0]
arg1 : premier paramètre, le argv[1]
... : les argv[i], pour $i \geq 1$
NULL : terminateur

```
int
execlp(const char * ref, const char * arg, ..., NULL);
```

Remarques : On cherche l'exécutable dans le PATH

```
int execle(const char * ref,
      const char * arg, ..., NULL, const char * arge[]);
```

arge[] : tableau contenant l'environnement

exec, avec listes

```
#include <unistd.h>
```

int
execl(const char * *ref*,
const char * *arg0*, const char * *arg1*, ..., *NULL*);

ref : le nom l'exécutable sur le disque
arg0 : le nom qu'on veut donner au processus, le argv[0]
arg1 : premier paramètre, le argv[1]
... : les argv[i], pour $i \geq 1$
NULL : terminateur

int
execlp(const char * *ref*, const char * *arg*, ..., NULL);

Remarques : On cherche l'exécutable dans le PATH

int execle(const char * *ref*,
const char * *arg*, ..., NULL, const char * *arge[]*);

arge[] : tableau contenant l'environnement

exec, avec listes

```
#include <unistd.h>
```

```
int
execl(const char * ref,
      const char * arg0, const char * arg1, ..., NULL);
```

ref : le nom l'exécutable sur le disque
arg0 : le nom qu'on veut donner au processus, le argv[0]
arg1 : premier paramètre, le argv[1]
... : les argv[i], pour $i \geq 1$
NULL : terminateur

```
int
execlp(const char * ref, const char * arg, ... , NULL );
```

Remarques : On cherche l'exécutable dans le PATH

```
int execle(const char * ref,
      const char * arg, ... , NULL , const char * arge[]);
```

arge[] : tableau contenant l'environnement

exec, avec listes

```
#include <unistd.h>
```

int
execl(const char * *ref*,
const char * *arg0*, const char * *arg1*, ..., *NULL*);

ref : le nom l'exécutable sur le disque
arg0 : le nom qu'on veut donner au processus, le argv[0]
arg1 : premier paramètre, le argv[1]
... : les argv[i], pour $i \geq 1$
NULL : terminateur

int
execlp(const char * *ref*, const char * *arg*, ... , NULL);

Remarques : On cherche l'exécutable dans le PATH

int execle(const char * *ref*,
const char * *arg*, ... , NULL , const char * *arge[]*);

arge[] : tableau contenant l'environnement

system

```
#include <stdlib.h>
int system(const char * comm);
```

comm : un commande (contenant pararemètres et options) en forme en forme de chaîne de caractères.

Retourne : La valeur de retour du shell.

Sommaire : La commande *commande* est passée au shell.

Remarques : Fonction de la bibliothèque standard C.

system

```
#include <stdlib.h>
int system(const char * comm);
```

comm : un commande (contenant pararemètres et options) en forme en forme de chaîne de caractères.

Retourne : La valeur de retour du shell.

Sommaire : La commande *commande* est passée au shell.

Remarques : Fonction de la bibliothèque standard C.

system

```
#include <stdlib.h>
int system(const char * comm);
```

comm : un commande (contenant pararemètres et options) en forme en forme de chaîne de caractères.

Retourne : La valeur de retour du shell.

Sommaire : La commande *commande* est passée au shell.

Remarques : Fonction de la bibliothèque standard C.

system

```
#include <stdlib.h>
int system(const char * comm);
```

comm : un commande (contenant pararemètres et options) en forme en forme de chaîne de caractères.

Retourne : La valeur de retour du shell.

Sommaire : La commande *commande* est passée au shell.

Remarques : Fonction de la bibliothèque standard C.

system

```
#include <stdlib.h>
int system(const char * comm);
```

comm : un commande (contenant pararemètres et options) en forme en forme de chaîne de caractères.

Retourne : La valeur de retour du shell.

Sommaire : La commande *commande* est passée au shell.

Remarques : Fonction de la bibliothèque standard C.

Plan

La communication par signaux

Forme primitive de communication.

Un signal est

- envoyé par un processus,
- reçu par un autre processus,
- véhiculé par le noyau.

Le processus qui reçoit le signal

- a un comportement par défaut,

  en général, terminaison anormale du processus †.
- il peut modifier le comportement par défaut:

  exécuter une tache sans terminer.
- il peut ignorer le signal.

La communication par signaux

Forme primitive de communication.

Un signal est

- envoyé par un processus,
- reçu par un autre processus,
- véhiculé par le noyau.

Le processus qui reçoit le signal

- a un comportement par défaut,

  en général, terminaison anormale du processus †.
- il peut modifier le comportement par défaut:

  exécuter une tache sans terminer.
- il peut ignorer le signal.

La communication par signaux

Forme primitive de communication.

Un signal est

- envoyé par un processus,
- reçu par un autre processus,
- véhiculé par le noyau.

Le processus qui reçoit le signal

- a un comportement par défaut,

  en général, terminaison anormale du processus †.
- il peut modifier le comportement par défaut:

  exécuter une tache sans terminer.
- il peut ignorer le signal.

Origine des signaux

- Frappes de caractères:

touche	signal
CTRL-C	SIGINT
CTRL-\	SIGQUIT
CTRL-Z	SIGTSTP

Utilisateur à la console $\longrightarrow$ processus en avant-plan.

- Erreur du programme :

erreur	signal
Violation écriture mémoire,	SIGSEGV
Division par zero	SIGFPE

Noyau $\longrightarrow$ programme même

- Commande `kill`, appel système `kill` :
processus « quelconque »$\longrightarrow$ un processus « quelconque »

Origine des signaux

- Frappes de caractères:

touche	signal
CTRL-C	SIGINT
CTRL-\	SIGQUIT
CTRL-Z	SIGTSTP

Utilisateur à la console $\longrightarrow$ processus en avant-plan.

- Erreur du programme :

erreur	signal
Violation écriture mémoire,	SIGSEGV
Division par zero	SIGFPE

Noyau $\longrightarrow$ programme même

- Commande kill, appel système kill :
processus « quelconque »$\longrightarrow$ un processus « quelconque »

Origine des signaux

- Frappes de caractères:

touche	signal
CTRL-C	SIGINT
CTRL-\	SIGQUIT
CTRL-Z	SIGTSTP

  Utilisateur à la console $\longrightarrow$ processus en avant-plan.

- Erreur du programme :

erreur	signal
Violation écriture mémoire,	SIGSEGV
Division par zero	SIGFPE

  Noyau $\longrightarrow$ programme même

- Commande `kill`, appel système `kill` :
  processus « quelconque »$\longrightarrow$ un processus « quelconque »

Liste des signaux par la commande kill

```
[lsantoca@localhost lecture4]$ 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     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      32) SIGRTMIN    33) SIGRTMIN+1
34) SIGRTMIN+2  35) SIGRTMIN+3  36) SIGRTMIN+4  37) SIGRTMIN+5
38) SIGRTMIN+6  39) SIGRTMIN+7  40) SIGRTMIN+8  41) SIGRTMIN+9
42) SIGRTMIN+10 43) SIGRTMIN+11 44) SIGRTMIN+12 45) SIGRTMIN+13
46) SIGRTMIN+14 47) SIGRTMIN+15 48) SIGRTMAX-15 49) SIGRTMAX-14
50) SIGRTMAX-13 51) SIGRTMAX-12 52) SIGRTMAX-11 53) SIGRTMAX-10
54) SIGRTMAX-9  55) SIGRTMAX-8  56) SIGRTMAX-7  57) SIGRTMAX-6
58) SIGRTMAX-5  59) SIGRTMAX-4  60) SIGRTMAX-3  61) SIGRTMAX-2
62) SIGRTMAX-1  63) SIGRTMAX
```

Actions par défaut

action	explication
T	Terminaison anormale du processus.
A	Comme T + creation du fichier core (selon machine).
I	Ignorer ce signal.
S	Arrêter (Stop) le processus.
C	Continuer le processus si arrêté, ignorer sinon.

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	acces à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

kill, raise

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

int kill(pid_t *pid*, int *sig*);

pid : > 0 : pid du destinataire.
0 : tous les processus du groupe.
-1 : non defini.
< -1 : tous les processus du groupe abs (pid).

sig : le signal à envoyer.

Retourne : 0/-1

Sommaire : Envoie un signal à un processus.

int raise(int *sig*);

Sommaire : Envoie le signal à soi-même : `kill(getpid(),sig)`.

Remarques : bibliothèque standard C

kill, raise

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

int kill(pid_t *pid*, int *sig*);

pid : > 0 : pid du destinataire.
0 : tous les processus du groupe.
-1 : non defini.
< -1 : tous les processus du groupe abs (pid).

sig : le signal à envoyer.

Retourne : 0/-1

Sommaire : Envoie un signal à un processus.

int raise(int *sig*);

Sommaire : Envoie le signal à soi-même : `kill(getpid(),sig)`.

Remarques : bibliothèque standard C

kill, raise

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

int kill(pid_t *pid*, int *sig*);

pid : > 0 : pid du destinataire.
0 : tous les processus du groupe.
-1 : non defini.
< -1 : tous les processus du groupe abs (pid).

sig : le signal à envoyer.

Retourne : 0/-1

Sommaire : Envoie un signal à un processus.

int raise(int *sig*);
Sommaire : Envoie le signal à soi-même : `kill(getpid(),sig)`.

Remarques : bibliothèque standard C

Plan

signal

```
#include <signal.h>
#include <unistd.h>
void (*signal(int sig, void (*handler)(int)))(int) ;
```

sig : le signal à intercepter

handler : (l'adresse d'une) procédure qui gère le signal.
Le constantes SIG_DFL : comportement par défaut
SIG_IGN : ignorer le signal

Sommaire :

faire traiter le prochain signal *sig* par la procedure *handler*.

Retourne :

adresse de la procédure qui a géré ce signal jusqu'à maintenant.

signal

```
#include <signal.h>
#include <unistd.h>
void (*signal(int sig, void (*handler)(int)))(int) ;
```

sig : le signal à intercepter

handler : (l'adresse d'une) procédure qui gère le signal.
Le constantes SIG_DFL : comportement par défaut
SIG_IGN : ignorer le signal

Sommaire :

faire traiter le prochain signal *sig* par la procedure *handler*.

Retourne :

adresse de la procédure qui a géré ce signal jusqu'à maintenant.

signal

```
#include <signal.h>
#include <unistd.h>
void (*signal(int sig, void (*handler)(int)))(int) ;
```

sig : le signal à intercepter

handler : (l'adresse d'une) procédure qui gère le signal.
Le constantes SIG_DFL : comportement par défaut
SIG_IGN : ignorer le signal

Sommaire :

faire traiter le prochain signal *sig* par la procedure *handler*.

Retourne :

adresse de la procédure qui a géré ce signal jusqu'à maintenant.

signal

```
#include <signal.h>
#include <unistd.h>
void (*signal(int sig, void (*handler)(int)))(int) ;
```

sig : le signal à intercepter

handler : (l'adresse d'une) procédure qui gère le signal.
Le constantes SIG_DFL : comportement par défaut
SIG_IGN : ignorer le signal

Sommaire :

faire traiter le prochain signal *sig* par la procedure *handler*.

Retourne :

adresse de la procédure qui a géré ce signal jusqu'à maintenant.

Programme : exsignal1.c

```
1  #include <signal.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  void handler(int signum)
6  {
7    printf("Merci, et à la prochaine.\n");
8    exit(EXIT_SUCCESS);
9  }
10
11 int main(void){
12   signal(SIGINT,handler);
13   while(1);
14   exit(EXIT_SUCCESS);
15 }
```

Programme : exsignal2.c

```
 1  #include <signal.h>
 2  #include <limits.h>
 3  #include <unistd.h>
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6
 7  static void handler(int signum);
 8  static int sigalrm_recu=0;
 9
10  int main(void){
11    unsigned int i;
12    signal(SIGINT,handler);
13    signal(SIGALRM,handler);
14
15    while(1){
16      for(i=UINT_MAX;i;i--);
17      kill(getpid(),SIGALRM);
18    }
19    exit(EXIT_SUCCESS);
20  }
```

Programme : exsignal2.c(II)

```
22  void handler(int signum)
23  {
24    /* Ligne suivante mystérieuse !!! */
25    signal(SIGALRM,handler);
26
27    switch(signum)
28      {
29      case SIGINT:
30        printf("Merci, et A+.\n");
31        exit(EXIT_SUCCESS);
32      case SIGALRM:
33        printf("J'ai reçu %d signaux SIGALRM.\n",
34               sigalrm_recu);
35        return;
36      }
37  }
```

alarm, pause

```
#include <unistd.h>
```

unsigned alarm(unsigned *no*);

no : numero de secondes

Retourne :

nombre de secondes restant si un alarme étai déjà prévu.

Sommaire : se faire reveiller apres *no* secondes.

int pause(void);

Retourne : -1 si erreur

Sommaire :

le processus s'endorme en attente d'un signal quelconque.

Remarques : évite l'attente active.

alarm, pause

```
#include <unistd.h>
```

unsigned alarm(unsigned *no*);

no : numero de secondes

Retourne :

nombre de secondes restant si un alarme étai déjà prévu.

Sommaire : se faire reveiller apres *no* secondes.

int pause(void);

Retourne : -1 si erreur

Sommaire :

le processus s'endorme en attente d'un signal quelconque.

Remarques : évite l'attente active.

alarm, pause

```
#include <unistd.h>
```

unsigned alarm(unsigned *no*);

no : numero de secondes

Retourne :

nombre de secondes restant si un alarme étai déjà prévu.

Sommaire : se faire reveiller apres *no* secondes.

int pause(void);

Retourne : -1 si erreur

Sommaire :

le processus s'endorme en attente d'un signal quelconque.

Remarques : évite l'attente active.

Programme : exalarm.c

```
1  #include <signal.h>
2  #include <limits.h>
3  #include <unistd.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  static void handler(int signum);
8  static int sigalrm_recu=0;
9
10 int main(void){
11   unsigned int i;
12   signal(SIGINT,handler);
13   signal(SIGALRM,handler);
14
15   while(1){
16     alarm(5);
17     pause();
18   }
19   exit(EXIT_SUCCESS);
20 }
```

Programme : exalarm.c(II)

```
22  void handler(int signum)
23  {
24    /* Ligne suivante mystérieuse !!! */
25    signal(SIGALRM,handler);
26
27    switch(signum)
28      {
29      case SIGINT:
30        printf("Merci, et A+.\n");
31        exit(EXIT_SUCCESS);
32      case SIGALRM:
33        printf("J'ai reçu %d signaux SIGALRM.\n",
34               sigalrm_recu);
35        return;
36      }
```

Communication entre processus

- Un père, un fils,
- le père envoie un signal aux fils,
- le fils confirme la réception du message.

Programme : perefils.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <unistd.h>
 4  #include <signal.h>
 5
 6  #define afficher(message) \
 7  printf("[%4d] " message "\n",getpid())
 8
 9  void interruption(int);
10  void pere(pid_t);
11  void fils(void);
```

Programme : perefils.c(II)

```
13  int main(void)
14  {
15    pid_t pid;
16    signal(SIGUSR1,interruption);
17
18    switch(pid = fork())
19      {
20      case -1:
21        exit(EXIT_FAILURE);
22      case 0:
23        fils();
24      default:
25        pere(pid);
26      }
27    return EXIT_SUCCESS;
28  }
```

Programme : perefils.c(III)

```
30 void interruption(int signum)
31 {
32   return;
33 }
34
35 void pere(pid_t fils)
36 {
37   afficher("Je delivre ma recommandé à la poste");
38   kill(fils, SIGUSR1);
39   pause();
40   afficher("Avis de réception reçu");
41   exit(EXIT_SUCCESS);
42 }
43
44 void fils()
45 {
46   pause();
47   afficher("Message reçu");
48   afficher("Je délivre ma confirmation");
49   kill(getppid(), SIGUSR1);
50   exit(EXIT_SUCCESS);
51 }
```

À l'exécution

```
[lsantoca@localhost signaux]$ a.out
[16392] Je delivre ma recommandé à la poste
[16393] Message reçu
[16393] Je délivre ma confirmation
[16392] Avis de réception reçu
[lsantoca@localhost signaux]$
```

Que se passe-t-il si on enlève la ligne

```
signal(SIGUSR1,interruption);
```

?

Plan

Motivations

Problèmes avec les signaux System V :

- ne sont pas réliables, les signaux pouvant être perdus.
- leur sémantique est différente de la sémantique des signaux BSD.

POSIX reprends le mécanisme des signaux BSD.

Motivations

Problèmes avec les signaux System V :

- ne sont pas réliables, les signaux pouvant être perdus.
- leur sémantique est diffèrente de la sémantique des signaux BSD.

POSIX reprends le mécanisme des signaux BSD.

Motivations

Problèmes avec les signaux System V :

- ne sont pas réliables, les signaux pouvant être perdus.
- leur sémantique est diffèrente de la sémantique des signaux BSD.

POSIX reprends le mécanisme des signaux BSD.

Motivations

Problèmes avec les signaux System V :

- ne sont pas réliables, les signaux pouvant être perdus.
- leur sémantique est diffèrente de la sémantique des signaux BSD.

POSIX reprends le mécanisme des signaux BSD.

Motivations

Problèmes avec les signaux System V :

- ne sont pas réliables, les signaux pouvant être perdus.
- leur sémantique est diffèrente de la sémantique des signaux BSD.

POSIX reprends le mécanisme des signaux BSD.

Le gestionnaire des signaux

1	2	3		253	254	255	
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux pendant
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux bloqués
↓	↓	↓	...	↓	↓	↓	comportements

Un signal est :

- *pendant* : pas encore pris en compte par le processus.
- *délivré* : pris en compte par le processus.
- *bloqué* ou *masqué* : le processus veut retarder la délivrance d'un tel signal.

Comportement:

- la routine à utiliser à la prise en compte du signal.

Le gestionnaire des signaux

1	2	3		253	254	255	
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux pendant
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux bloqués
↓	↓	↓	...	↓	↓	↓	comportements

Un signal est :

- *pendant* : pas encore pris en compte par le processus.
- *délivré* : pris en compte par le processus.
- *bloqué* ou *masqué* : le processus veut retarder la délivrance d'un tel signal.

Comportement:

- la routine à utiliser à la prise en compte du signal.

Le gestionnaire des signaux

1	2	3		253	254	255	
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux pendant
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux bloqués
↓	↓	↓	...	↓	↓	↓	comportements

Un signal est :

- *pendant* : pas encore pris en compte par le processus.
- *délivré* : pris en compte par le processus.
- *bloqué* ou *masqué* : le processus veut retarder la délivrance d'un tel signal.

Comportement:

- la routine à utiliser à la prise en compte du signal.

Le gestionnaire des signaux

1	2	3		253	254	255	
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux pendant
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux bloqués
↓	↓	↓	...	↓	↓	↓	comportements

Un signal est :

- *pendant* : pas encore pris en compte par le processus.
- *délivré* : pris en compte par le processus.
- *bloqué* ou *masqué* : le processus veut retarder la délivrance d'un tel signal.

Comportement:

- la routine à utiliser à la prise en compte du signal.

Le gestionnaire des signaux

1	2	3		253	254	255	
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux pendant
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux bloqués
↓	↓	↓	...	↓	↓	↓	comportements

Un signal est :

- *pendant* : pas encore pris en compte par le processus.
- *délivré* : pris en compte par le processus.
- *bloqué* ou *masqué* : le processus veut retarder la délivrance d'un tel signal.

Comportement:

- la routine à utiliser à la prise en compte du signal.

Le gestionnaire des signaux

1	2	3		253	254	255	
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux pendant
0/1	0/1	0/1	...	0/1	0/1	0/1	signaux bloqués
↓	↓	↓	...	↓	↓	↓	comportements

Un signal est :

- *pendant* : pas encore pris en compte par le processus.
- *délivré* : pris en compte par le processus.
- *bloqué* ou *masqué* : le processus veut retarder la délivrance d'un tel signal.

Comportement:

- la routine à utiliser à la prise en compte du signal.

Manipulation des ensembles de signaux

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

struct sigset_t;
Sommaire : Un ensemble de signaux

int sigemptyset(sigset_t * *p_ens*);
Sommaire : p_ens = ∅.

int sigfillset(sigset_t * *p_ens*);
Sommaire : p_ens = tous les signaux.

int sigaddset(sigset_t * *p_ens*, int *sig*);
Sommaire : Ajoute sig à p_ens.

int sigdelset(sigset_t * *p_ens*, int *sig*);
Sommaire : Enlève sig de p_ens.

int sigismember(sigset_t * *p_ens*, int *sig*);
sig : cucu
Retourne : -1/0/1.
Sommaire : Teste sig ∈ p_ens

Manipulation des ensembles de signaux

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

struct sigset_t;
Sommaire : Un ensemble de signaux

int sigemptyset(sigset_t * *p_ens*);
Sommaire : p_ens = $\emptyset$.

int sigfillset(sigset_t * *p_ens*);
Sommaire : p_ens = tous les signaux.

int sigaddset(sigset_t * *p_ens*, int *sig*);
Sommaire : Ajoute sig à p_ens.

int sigdelset(sigset_t * *p_ens*, int *sig*);
Sommaire : Enlève sig de p_ens.

int sigismember(sigset_t * *p_ens*, int *sig*);
sig : cucu
Retourne : -1/0/1.
Sommaire : Teste $sig \in p\_ens$

Manipulation des ensembles de signaux

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

struct sigset_t;
Sommaire : Un ensemble de signaux

int sigemptyset(sigset_t * *p_ens*);
Sommaire : p_ens = $\emptyset$.

int sigfillset(sigset_t * *p_ens*);
Sommaire : p_ens = tous les signaux.

int sigaddset(sigset_t * *p_ens*, int *sig*);
Sommaire : Ajoute sig à p_ens.

int sigdelset(sigset_t * *p_ens*, int *sig*);
Sommaire : Enlève sig de p_ens.

int sigismember(sigset_t * *p_ens*, int *sig*);
sig : cucu
Retourne : -1/0/1.
Sommaire : Teste sig $\in$ p_ens

Manipulation des ensembles de signaux

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

struct sigset_t;
Sommaire : Un ensemble de signaux

int sigemptyset(sigset_t * *p_ens*);
Sommaire : p_ens = $\emptyset$.

int sigfillset(sigset_t * *p_ens*);
Sommaire : p_ens = tous les signaux.

int sigaddset(sigset_t * *p_ens*, int *sig*);
Sommaire : Ajoute sig à p_ens.

int sigdelset(sigset_t * *p_ens*, int *sig*);
Sommaire : Enlève sig de p_ens.

int sigismember(sigset_t * *p_ens*, int *sig*);
sig : cucu
Retourne : -1/0/1.
Sommaire : Teste $sig \in p\_ens$

Manipulation des ensembles de signaux

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

struct sigset_t;
Sommaire : Un ensemble de signaux

int sigemptyset(sigset_t * *p_ens*);
Sommaire : p_ens = $\emptyset$.

int sigfillset(sigset_t * *p_ens*);
Sommaire : p_ens = tous les signaux.

int sigaddset(sigset_t * *p_ens*, int *sig*);
Sommaire : Ajoute sig à p_ens.

int sigdelset(sigset_t * *p_ens*, int *sig*);
Sommaire : Enlève sig de p_ens.

int sigismember(sigset_t * *p_ens*, int *sig*);
sig : cucu
Retourne : -1/0/1.
Sommaire : Teste $sig \in$ p_ens

Manipulation des ensembles de signaux

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

struct sigset_t;
Sommaire : Un ensemble de signaux

int sigemptyset(sigset_t * *p_ens*);
Sommaire : p_ens = $\emptyset$.

int sigfillset(sigset_t * *p_ens*);
Sommaire : p_ens = tous les signaux.

int sigaddset(sigset_t * *p_ens*, int *sig*);
Sommaire : Ajoute sig à p_ens.

int sigdelset(sigset_t * *p_ens*, int *sig*);
Sommaire : Enlève sig de p_ens.

int sigismember(sigset_t * *p_ens*, int *sig*);
sig : cucu
Retourne : -1/0/1.
Sommaire : Teste sig $\in$ p_ens

Manipulation des ensembles de signaux

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

struct sigset_t;
Sommaire : Un ensemble de signaux

int sigemptyset(sigset_t * *p_ens*);
Sommaire : p_ens = $\emptyset$.

int sigfillset(sigset_t * *p_ens*);
Sommaire : p_ens = tous les signaux.

int sigaddset(sigset_t * *p_ens*, int *sig*);
Sommaire : Ajoute sig à p_ens.

int sigdelset(sigset_t * *p_ens*, int *sig*);
Sommaire : Enlève sig de p_ens.

int sigismember(sigset_t * *p_ens*, int *sig*);
sig : cucu
Retourne : -1/0/1.
Sommaire : Teste sig $\in$ p_ens.

La structure sigaction

```
struct sigaction {
  void (*sa_handler)(int);
          /* Pointeur à l'handler, ou SIG_DFL,SIG_IGN */

  sigset_t sa_mask;
          /* signaux supplementaires à bloquer */

  int sa_flags; /* options */
  ...
};
```

sa_flags : SA_RESETHAND, pour reinstaller SIG_DFL.

sigaction

```
int sigaction(int sig, const struct sigaction *
new, struct sigaction * old);
```

sig : le signal à intercepter
new : le nouvelles coordonnés de l'handler
old : ou sauver les anciennes coordonnes. NULL si on ne veut pas sauver ces coordonnés.

Retourne : 0/-1

sigaction

```
int sigaction(int sig, const struct sigaction *
new, struct sigaction * old);
```

sig : le signal à intercepter
new : le nouvelles coordonnés de l'handler
old : ou sauver les anciennes coordonnes. NULL si on ne veut pas sauver ces coordonnés.

Retourne : 0/-1

sigaction

```
int sigaction(int sig, const struct sigaction *
new, struct sigaction * old);
```

sig : le signal à intercepter
new : le nouvelles coordonnés de l'handler
old : ou sauver les anciennes coordonnes. NULL si on ne veut pas sauver ces coordonnés.

Retourne : 0/-1

sigaction

```
int sigaction(int sig, const struct sigaction *
new, struct sigaction * old);
```

sig : le signal à intercepter
new : le nouvelles coordonnés de l'handler
old : ou sauver les anciennes coordonnes. NULL si on ne veut pas sauver ces coordonnés.

Retourne : 0/-1

sigaction

```
int sigaction(int sig, const struct sigaction *
new, struct sigaction * old);
```

sig : le signal à intercepter
new : le nouvelles coordonnés de l'handler
old : ou sauver les anciennes coordonnes. NULL si on ne veut pas sauver ces coordonnés.

Retourne : 0/-1

Programme : exsigaction.c

```
1  #include <signal.h>
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  static void handler(int signum);
7  struct sigaction action;
8  static int no_bonjour=1;
9
10 int main(void){
11   action.sa_handler=handler;
12   action.sa_flags=0;
13   sigemptyset(&action.sa_mask);
14   sigaction(SIGINT,&action,NULL);
15   sigaction(SIGALRM,&action,NULL);
16
17   while(1){
18     alarm(5);
19     pause();
20   }
21   exit(EXIT_SUCCESS);
22 }
```

Programme : exsigaction.c(II)

```
24  void handler(int signum)
25  {
26    /* sigaction(SIGALRM,&action,NULL); */
27    switch(signum)
28      {
29      case SIGINT:
30        printf("Merci, et A+.\n");
31        exit(EXIT_SUCCESS);
32      case SIGALRM:
33        printf("Bonjour no. %d.\n",no_bonjour++);
34        return;
35      }
36  }
```

sigprocmask

```
#include <signal.h>
int
sigprocmask(int how, const sigset_p * new,
                                const sigset_p * old);
```

how : SIG_SET_MASK : *new
SIG_BLOCK : *old ∪ *new
SIG_UNBLOCK : *old \ *new

new : ensemble à copier dans le gestionnaire.
old : pointeur à un ensemble où sauver le contenu courant du gestionnaire.

Retourne : 0/code erreur.

Sommaire : examine et change les signaux bloqués.

sigprocmask

```
#include <signal.h>
int
sigprocmask(int how, const sigset_p * new,
                                const sigset_p * old);
```

how : SIG_SET_MASK : *new
SIG_BLOCK : *old ∪ *new
SIG_UNBLOCK : *old \ *new

new : ensemble à copier dans le gestionnaire.
old : pointeur à un ensemble où sauver le contenu courant du gestionnaire.

Retourne : 0/code erreur.

Sommaire : examine et change les signaux bloqués.

sigprocmask

```
#include <signal.h>
int
sigprocmask(int how, const sigset_p * new,
                                const sigset_p * old);
```

how : SIG_SET_MASK : *new
SIG_BLOCK : *old ∪ *new
SIG_UNBLOCK : *old \ *new

new : ensemble à copier dans le gestionnaire.

old : pointeur à un ensemble où sauver le contenu courant du gestionnaire.

Retourne : 0/code erreur.

Sommaire : examine et change les signaux bloqués.

sigprocmask

```
#include <signal.h>
int
sigprocmask(int how, const sigset_p * new,
                                const sigset_p * old);
```

how : SIG_SET_MASK : *new
SIG_BLOCK : *old ∪ *new
SIG_UNBLOCK : *old \ *new

new : ensemble à copier dans le gestionnaire.
old : pointeur à un ensemble où sauver le contenu courant du gestionnaire.

Retourne : 0/code erreur.

Sommaire : examine et change les signaux bloqués.

sigprocmask

```
#include <signal.h>
int
sigprocmask(int how, const sigset_p * new,
                                const sigset_p * old);
```

how : SIG_SET_MASK : *new
SIG_BLOCK : *old ∪ *new
SIG_UNBLOCK : *old \ *new

new : ensemble à copier dans le gestionnaire.
old : pointeur à un ensemble où sauver le contenu courant du gestionnaire.

Retourne : 0/code erreur.

Sommaire : examine et change les signaux bloqués.

sigprocmask

```
#include <signal.h>
int
sigprocmask(int how, const sigset_p * new,
                                const sigset_p * old);
```

how : SIG_SET_MASK : *new
SIG_BLOCK : *old ∪ *new
SIG_UNBLOCK : *old \ *new

new : ensemble à copier dans le gestionnaire.
old : pointeur à un ensemble où sauver le contenu courant du gestionnaire.

Retourne : 0/code erreur.

Sommaire : examine et change les signaux bloqués.

sigpending

```
#include <signal.h>
int sigpending(sigset_t * ens);
```

Retourne : 0/-1.

Sommaire : copie dans la structure ens l'ensemble des signaux bloqués et pendants.

sigpending

```
#include <signal.h>
int sigpending(sigset_t * ens);
```

Retourne : 0/-1.

Sommaire : copie dans la structure ens l'ensemble des signaux bloqués et pendants.

sigpending

```
#include <signal.h>
int sigpending(sigset_t * ens);
```

Retourne : 0/-1.

Sommaire : copie dans la structure ens l'ensemble des signaux bloqués et pendants.

Programme : exsigprocmask.c

```
 1  #include <stdio.h>
 2  #include <signal.h>
 3  #include <stdlib.h>
 4  #include <unistd.h>
 5
 6  sigset_t ens1,ens2;
 7  int sig;
 8
 9  int main()
10  {
11    /* Initialisation de la masque */
12    sigemptyset(&ens1);
13    sigaddset(&ens1,SIGINT);
14    sigaddset(&ens1,SIGQUIT);
15    sigprocmask(SIG_SETMASK,&ens1,NULL);
16
17    sleep(15);
```

Programme : exsigprocmask.c(II)

```
19    /*  Affichaghe des signaux bloqués */
20    sigpending(&ens2);
21    printf("Signaux pendants : ");
22    for(sig=1; sig < NSIG;sig++)
23      if(sigismember(&ens2,sig))
24        printf("%d ",sig);
25    putc('\n',stdout);
26
27    sleep(15);
28
29    /*  Debloquage des signaux */
30    sigemptyset(&ens1);
31    printf("Débloquage.\n");
32    sigprocmask(SIG_SETMASK,&ens1,NULL);
33
34    /* Fin normale */
35    printf("Fin normale !!!\n");
36    exit(0);
37  }
```