UE Programmation Unix

TP3 Gestion des Signaux

partie 2

Emission d'un signal

kill

La commande `kill` permet d'envoyer au processus (ou groupe de processus) d'identification donnée le signal désigné.

```
[ev00000@saphyr ~]$ ps
  PID TTY          TIME CMD
 1059 pts/3    00:00:00 bash
 1173 pts/3    00:00:02 a.out
 1174 pts/3    00:00:00 ps
[ev00000@saphyr ~]$ kill -9 1173
```

Les signaux sont identifiés par des nombres entiers (numéros absolus dans le système tels que fournis par la commande `ps`) ou par des noms symboliques tels qu'ils apparaissent dans le fichier `signal.h` (privés du préfixe `SIG`). Les noms des signaux reconnus sont affichés par la commande `kill -l`.

```
KILL(1)                    User Commands                    KILL(1)

NAME
       kill - send a signal to a process

SYNOPSIS
       kill [options] <pid> [...]

DESCRIPTION
       The  default  signal  for kill is TERM.  Use -l or -L to list available
       signals.  Particularly useful signals include HUP, INT, KILL, STOP,
       CONT,  and  0.  Alternate  signals may be specified in three ways: -9,
       -SIGKILL or -KILL.  Negative PID values may be  used  to  choose  whole
       process  groups; see the PGID column in ps command output.  A PID of -1
       is special; it indicates all processes except the kill  process  itself
       and init.
```

```
[ij04ll5@saphyr:~]:ven. sept. 04$ 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
```

La commande kill

La commande `kill` envoie un signal aux processus actifs. Le numéro du signal et le PID du processus sont indiqués sur la ligne de commande. Si aucun numéro de signal n'est précisé, c'est SIGTERM (15) qui est envoyé. Ce signal tue les processus, sauf ceux qui l'interceptent. Pour ces processus, il faut envoyer explicitement le signal SIGKILL (9), qui ne peut être intercepté. L'administrateur *root* peut arrêter tous les processus avec la commande kill. Les autres utilisateurs ne peuvent gérer que les processus dont ils sont propriétaires.

Les différentes syntaxes de cette commande sont :

```
$ kill -num_signal pid1 pid2 etc.
$ kill -s num_signal pid1 pid2 etc.
$ kill -l
```

où num_signal est le numéro ou le nom de signal à envoyer, et `pid1`, `pid2`, etc., sont les numéros de processus vers lesqueks envoyer le signal. Les PID des processus sont obtenus avec la commande ps. La commande kill avec l'option `-l` affiche la liste et le nom des signaux qui peuvent être administrés.

Comment arrêter un programme ?

```
$ kill -9 pid_du_programme
```

Name	Description	ISO C	SUS	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Default action
SIGABRT	abnormal termination (abort)	•	•	•	•	•	•	terminate+core
SIGALRM	timer expired (alarm)		•	•	•	•	•	terminate
SIGBUS	hardware fault		•	•	•	•	•	terminate+core
SIGCANCEL	threads library internal use						•	ignore
SIGCHLD	change in status of child		•	•	•	•	•	ignore
SIGCONT	continue stopped process		•	•	•	•	•	continue/ignore
SIGEMT	hardware fault			•	•	•	•	terminate+core
SIGFPE	arithmetic exception	•	•	•	•	•	•	terminate+core
SIGFREEZE	checkpoint freeze						•	ignore
SIGHUP	hangup		•	•	•	•	•	terminate
SIGILL	illegal instruction	•	•	•	•	•	•	terminate+core
SIGINFO	status request from keyboard			•		•		ignore
SIGINT	terminal interrupt character	•	•	•	•	•	•	terminate
SIGIO	asynchronous I/O			•	•	•	•	terminate/ignore
SIGIOT	hardware fault			•	•	•	•	terminate+core
SIGJVM1	Java virtual machine internal use						•	ignore
SIGJVM2	Java virtual machine internal use						•	ignore
SIGKILL	termination		•	•	•	•	•	terminate
SIGLOST	resource lost						•	terminate
SIGLWP	threads library internal use			•			•	terminate/ignore
SIGPIPE	write to pipe with no readers		•	•	•	•	•	terminate
SIGPOLL	pollable event (poll)				•		•	terminate
SIGPROF	profiling time alarm (setitimer)			•	•	•	•	terminate
SIGPWR	power fail/restart				•		•	terminate/ignore
SIGQUIT	terminal quit character		•	•	•	•	•	terminate+core
SIGSEGV	invalid memory reference	•	•	•	•	•	•	terminate+core
SIGSTKFLT	coprocessor stack fault				•			terminate
SIGSTOP	stop		•	•	•	•	•	stop process
SIGSYS	invalid system call		XSI	•	•	•	•	terminate+core
SIGTERM	termination	•	•	•	•	•	•	terminate
SIGTHAW	checkpoint thaw						•	ignore
SIGTHR	threads library internal use			•				terminate
SIGTRAP	hardware fault		XSI	•	•	•	•	terminate+core
SIGTSTP	terminal stop character		•	•	•	•	•	stop process
SIGTTIN	background read from control tty		•	•	•	•	•	stop process
SIGTTOU	background write to control tty		•	•	•	•	•	stop process
SIGURG	urgent condition (sockets)		•	•	•	•	•	ignore
SIGUSR1	user-defined signal		•	•	•	•	•	terminate
SIGUSR2	user-defined signal		•	•	•	•	•	terminate
SIGVTALRM	virtual time alarm (setitimer)		XSI	•	•	•	•	terminate
SIGWAITING	threads library internal use						•	ignore
SIGWINCH	terminal window size change			•	•	•	•	ignore
SIGXCPU	CPU limit exceeded (setrlimit)		XSI	•	•	•	•	terminate or terminate+core
SIGXFSZ	file size limit exceeded (setrlimit)		XSI	•	•	•	•	terminate or terminate+core
SIGXRES	resource control exceeded						•	ignore

Figure 10.1 UNIX System signals

La fonction main()

Arguments de la ligne de commandes

- Langage C offre des mécanismes qui permettent d'intégrer parfaitement un programme C dans l'environnement hôte
  - environnement orienté ligne de commande (Unix, Linux)
- Programme C peut recevoir de la part de l'interpréteur de commandes qui a lancé son exécution, une liste d'arguments

⇔ ligne de commande qui a servi à lancer l'exécution du programme

- Liste composée
  - du nom du fichier binaire contenant le code exécutable du programme
  - des paramètres de la commande

Entête à inclure

```
#include <stdlib.h>  // <cstdlib> en C++
```

Fonction *atoi*

```
int atoi( const char * theString );
```

Cette fonction permet de transformer une chaîne de caractères, représentant une valeur entière, en une valeur numérique de type int. Le terme d'atoi est un acronyme signifiant : ASCII to integer.

⚠ **ATTENTION :** la fonction atoi retourne la valeur 0 si la chaîne de caractères ne contient pas une représentation de valeur numérique. Du coup, il n'est pas possible de distinguer la chaîne "0" d'une chaîne ne contenant pas un nombre entier. Si vous avez cette difficulté, veuillez préférer l'utilisation de la fonction strtol qui permet bien de distinguer les deux cas.

La fonction main ()

Un processus débute par l'exécution de la fonction `main()` du programme correspondant

Definition

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

- `argc`: nombre d'arguments de la ligne de commande y compris le nom du programme
- `argv[]`: tableau de pointeurs vers les arguments (paramètres) passés au programme lors de son lancement

A NOTER

- `argv[0]` pointe vers le nom du programme
- `argv[argc]` vaut `NULL`

- argc (argument count)
  - nombre de mots qui compose la ligne de commande (y compris le nom de la commande qui a servi à lancer l'exécution du programme)
- argv (argument vector)
  - tableau de chaînes de caractères contenant chacune un mot de la ligne de commande
  - argv[0] est le nom du programme exécutable

Exemple de cours

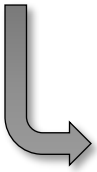
```
#include <stdio.h>

int main (int argc, char *argv[]){
int i;
for (i=0;i<argc;i++){
    printf ("argv[%d]: %s\n",i, argv[i]);
}
return (0);
}// main

>./affich_param Bonjour à tous
argv[0]: ./affich_param
argv[1]: Bonjour
argv[2]: à
argv[3]: tous
```



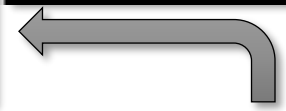
```
[ij04115@saphyr:~/unix_tp1]:ven. sept. 11$ nano affich_param.c
```



saphyr.ens.math-info.univ-paris5.fr - PuTTY

```
 GNU nano 2.5.3            Fichier : affich param.c
#include <stdio.h>

int main(int argc, char *argv[]){
        int i;
        for (i = 0 ; i < argc ; i++){
                printf("argv[%d]: %s\n", i, argv[i]);
        }
        return (0);
}//main
```



Spécifier le nom de l'exécutable avec l'option `-o`

```
ProgC > gcc -o toto premierProg.c
ProgC > ls
toto    premierProg.c
ProgC > ./toto
```

```
[ij04115@saphyr:~/unix_tp1]:sam. sept. 12$ gcc -o affich_param affich_param.c
[ij04115@saphyr:~/unix_tp1]:sam. sept. 12$ ls
affich_param  affich_param.c
[ij04115@saphyr:~/unix_tp1]:sam. sept. 12$
```

```
[ij04115@saphyr:~/unix_tp1]:sam. sept. 12$ ./affich_param Bonjour à tous
argv[0]: ./affich_param
argv[1]: Bonjour
argv[2]: à
argv[3]: tous
[ij04115@saphyr:~/unix_tp1]:sam. sept. 12$
```

Le fichier standard des erreurs stderr

Structure *FILE ** et variables *stdin*, *stdout* et *stderr*

Entête à inclure

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

Structure *FILE ** et variables *stdin*, *stdout* et *stderr*

```
FILE * stdin;
FILE * stdout;
FILE * stderr;
```

La structure FILE permet de stocker les informations relatives à la gestion d'un flux de données. Néanmoins, il est très rare que vous ayez besoin d'accéder directement à ses attributs.

Effectivement, il existe un grand nombre de fonctions qui acceptent un paramètre basé sur cette structure pour déterminer ou contrôler divers aspects.

- **stdin** (Standard input) : ce flot correspond au flux standard d'entrée de l'application. Par défaut, ce flux est associé au clavier : vous pouvez donc acquérir facilement des données en provenance du clavier. Quelques fonctions utilisent implicitement ce flux (**scanf**, par exemple).
- **stdout** (Standard output): c'est le flux standard de sortie de votre application.
  Par défaut, ce flux est associé à la console d'où l'application à été lancée. Quelques fonctions utilisent implicitement ce flux (**printf**, par exemple).
- **stderr** (Standard error) : **ce dernier flux est associé à la sortie standard d'erreur de votre application.** Tout comme stdout, ce flux est normalement redirigé sur la console de l'application.

fprintf it is the same as printf,
except now you are also specifying the place to print to :

```
printf("%s", "Hello world\n"); // "Hello world" on stdout (using printf)
fprintf(stdout, "%s", "Hello world\n"); // "Hello world" on stdout (using fprintf)
fprintf(stderr, "%s", "Stack overflow!\n");//Error message on stderr (using fprintf)
```

Messages d'erreurs affiches avec perror

C Library - <stdio.h>

C library function - perror()

Description

The C library function **void perror(const char *str)** prints a descriptive error message to stderr. First the string **str** is printed, followed by a colon then a space.

Declaration

Following is the declaration for perror() function.

```
void perror(const char *str)
```

Parameters

- **str** – This is the C string containing a custom message to be printed before the error message itself.

Return Value

This function does not return any value.

```c
#include <stdio.h>

int main () {
   FILE *fp;

   /* first rename if there is any file */
   rename("file.txt", "newfile.txt");

   /* now let's try to open same file */
   fp = fopen("file.txt", "r");
   if( fp == NULL ) {
      perror("Error: ");
      return(-1);
   }
   fclose(fp);

   return(0);
}
```

Let us compile and run the above program that will produce the following result because we are trying to open a file which does not exist –

```
Error: : No such file or directory
```

Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction

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

- Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affiches avec `perror`.
- Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`.

4 Le gestionnaire de signal POSIX sigaction( )

La primitive

```
int sigaction(int signo, const struct sigaction *act, struct sigaction *oldact);
```

permet de modifier l'action effectuée par un processus lors de la réception d'un signal spécifique

Les fonctions POSIX

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

Permet de déterminer ou de modifier l'action associée à un signal particulier

- signo : signal pour lequel le gestionnaire est installé
- si act ≠ NULL, il s'agit d'une modification de l'action associée au signal signo
- si oldact ≠ NULL, le système retourne l'ancienne action pour le signal signo

```
struct sigaction {
    void (*sa_handler)();     // adresse du handler, ou SIG_IGN, ou SIG_DFL
    sigset_t sa_mask;         // signaux additionnels à bloquer
    int sa_flags;             // options (SA_RESTART, SA_NOCLDWAIT,
                              //          SA_NODEFER, SA_NORESETHAND, ... )
}
```

Retourne : 0 en cas de succès et -1 sinon

The `sigaction` function allows us to examine or modify (or both) the action associated with a particular signal. This function supersedes the `signal` function from earlier releases of the UNIX System.

```
#include <signal.h>

int sigaction(int signo, const struct sigaction *restrict act,
              struct sigaction *restrict oact);

                                        Returns: 0 if OK, –1 on error
```

→ **The argument `signo` is the signal number whose action we are examining or modifying.**

→ **If the `act` pointer is non-null, we are modifying the action.**

→ **If the `oact` pointer is non-null, the system returns the previous action for the signal through the `oact` pointer.**

This function uses the following structure:

```
struct sigaction {
  void     (*sa_handler)(int);  /* addr of signal handler, */
                                /* or SIG_IGN, or SIG_DFL */
  sigset_t sa_mask;             /* additional signals to block */
  int      sa_flags;            /* signal options, Figure 10.16 */

  /* alternate handler */
  void     (*sa_sigaction)(int, siginfo_t *, void *);
};
```

When changing the action for a signal,

→ **if the `sa_handler` field contains the address of a signal-catching function (as opposed to either of the constants `SIG_IGN` or `SIG_DFL`),**

→ **then the `sa_mask` field specifies a set of signals that are added to the signal mask of the process before the signal-catching function is called.**

  → **If and when the signal-catching function returns, the signal mask of the process is reset to its previous value.**

Exemple (naïf) de mise en place d'un gestionnaire de signaux

Source : http://bruno-garcia.net/www/Unix/Docs/Signaux.html

Le code suivant met en place un gestionnaire de signaux très simple : Celui ci se contente d'émettre un message indiquant le numéro du signal reçu.

Les signaux `SIGINT`, `SIGQUIT` et `SIGTERM` étant interceptés, il sera possible de tuer ce processus avec un signal `SIGHUP`.

```
[ij04115@saphyr:~]:ven. sept. 04$ 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
```

Name	Description	Default action
SIGHUP	hangup	terminate
SIGINT	terminal interrupt character	terminate
SIGQUIT	terminal quit character	terminate+core
SIGTERM	termination	terminate

```c
#include <stdio.h> // printf(),fflush()
#include <signal.h> // sigaction(), sigemptyset()

/*Gestionnaire naif se contentant d'indiquer qu'un signal
 *a été recu accompagné de son numéro
 */
void handler(int theSignal) {
     printf("Je receptionne le signal %d\n", theSignal);
     fflush(stdout); // int fflush(FILE *stream)
}

int main(void) {

    // Structure pour la mise en place des gestionnaires
    struct sigaction prepaSignal;

    /* Remplissage de la structure */

    // Adresse du gestionnaire
    prepaSignal.sa_handler = &handler;

    // Mise a zero du champ sa_flags théoriquement ignoré
    prepaSignal.sa_flags = 0;


    /* int sigemptyset(sigset_t *set);
     * initialise a VIDE l'ensemble de signaux pointé par set
     * retourne 0 en cas de succès ou -1 en cas d'erreur
     */

     // On ne bloque pas de signaux spécifiques
     sigemptyset( &prepaSignal.sa_mask );

     /* int sigaction(int sigo, const struct sigaction *act,
        struct sigaction *oldact); */

     // Mise en place du gestionnaire bidon pour 3 signaux
     sigaction(SIGINT, &prepaSignal, 0);
     sigaction(SIGQUIT, &prepaSignal, 0);
     sigaction(SIGTERM, &prepaSignal, 0);

     while(1){ }
     return 0;
}
```

saphyr.ens.math-info.univ-paris5.fr - PuTTY

```
[ij04115@saphyr:~/unix_tp3]:sam. nov. 21$ nano question4_test.c
[ij04115@saphyr:~/unix_tp3]:sam. nov. 21$ ./question4_test
Je receptionne le signal 2
Je receptionne le signal 3
Je receptionne le signal 15
Fin de la connexion (raccroché)
[ij04115@saphyr:~/unix_tp3]:sam. nov. 21$
```

```
[ij04115@saphyr:~]:sam. nov. 21$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ij04115  13958  0.0  0.1  21328 10520 pts/0    Ss   02:19   0:00 -bash2
ij04115  14022  0.0  0.1  21160 10332 pts/2    Ss   02:20   0:00 -bash2
ij04115  14040  101  0.0   2260   676 pts/0    R+   02:23   1:06 ./question4_test
ij04115  14042  0.0  0.1  21320  8528 pts/2    R+   02:24   0:00 ps u
[ij04115@saphyr:~]:sam. nov. 21$ kill -INT 14040
[ij04115@saphyr:~]:sam. nov. 21$ kill -QUIT 14040
[ij04115@saphyr:~]:sam. nov. 21$ kill -TERM 14040
[ij04115@saphyr:~]:sam. nov. 21$ kill -HUP 14040
[ij04115@saphyr:~]:sam. nov. 21$
```

a Ecrire un programme `ring.c` qui se lance de la façon suivante

```
./ring n m
```

→ L'exécution de ce programme créera n processus de P1 à Pn (n>1) dans cet ordre.

→ Chaque processus Pi envoie le signal SIGUSR1 au processus P(i-1) puis attend à son tour l'arrivé de SIGUSR1.

→ Chaque Pi répète cette séquence m (m>1) fois puis se termine.

→ Le précèdent de P1 est Pn.

→

→ P1 est le résultat de ./ring n m.

?

```
#include <stdio.h> // printf(), fprintf(), perror(), fflush()
#include <stdlib.h> // exit(), malloc(), atoi()
#include <signal.h> // sigaction(), sigemptyset(), kill()
#include <unistd.h> // getpid(), fork(), pause()

int num_processus;
void handler(int theSignal);

int main(int argc, char *argv[]){
    int i, j,k, kill_result;

    if(argc < 3){
       fprintf(stderr,"Usage : %s n m \n", argv[0]);
       exit(1);
    }

    struct sigaction prepaSignal;
    prepaSignal.sa_handler = &handler;
    prepaSignal.sa_flags = 0;
    sigemptyset( &prepaSignal.sa_mask );
    int sigaction_result = sigaction(SIGUSR1, &prepaSignal, 0);
    if(sigaction_result < 0){
       perror("Function sigaction() : ");
       exit(1);
    }

    int *fork_result_array = (int*)malloc( (atoi(argv[1])+1)* sizeof(int) );
    if(fork_result_array == NULL){
       fprintf(stderr,"Erreur function malloc()");
       exit(1);
    }

    fork_result_array[1] = getpid();
    for(i = 2 ; i <= atoi(argv[1]) ; i++) {
       fork_result_array[i] = fork();
       if(fork_result_array[i] < 0) {
          perror("Function fork() : ");
          exit(1);
       }
       else
       {
          if(fork_result_array[i] == 0) // CODE DU FILS Pi
          {
             num_processus = i;
             for(j = 1 ; j<= atoi(argv[2]) ; j++) {
                  printf("Wait for signal – P%d \n", num_processus);
                  pause(); // Suspend le processus jusqu'à un signal
                  kill_result = kill(fork_result_array[i-1], SIGUSR1);
                  if(kill_result < 0) {
                        perror("Function kill() : ");
                        exit(1);
                  }
                  printf("Sent from P%d to P%d\n", num_processus, (i-1));
            } // for(j)
         } // FIN CODE DU FILS Pi
      } // else
   } // for(i)
```

?

```
   // CODE DU PERE p1
   num_processus = 1;
   for(k = 1 ; k <= atoi(argv[2]) ; k++) {
      kill_result = kill(fork_result_array[atoi(argv[1])], SIGUSR1);
      if(kill_result < 0) {
         perror("Function kill() : ");
         exit(1);
      }
      printf("Sent from P%d to P%d\n", num_processus, atoi(argv[1]));

      printf("Wait for signal - P%d\n", num_processus);
      pause();
   } // for
} // main()


void handler(int theSignal) {
     printf("P%d-receptionne signal %d\n",num_processus, theSignal);
     fflush(stdout);
} // handler()
```

```
[ij04115@saphyr:~/unix_tp3]:sam. nov. 21$ nano ring.c
[ij04115@saphyr:~/unix_tp3]:sam. nov. 21$ gcc ring.c -o ring
[ij04115@saphyr:~/unix_tp3]:sam. nov. 21$ ./ring
Usage : ./ring n m
[ij04115@saphyr:~/unix_tp3]:sam. nov. 21$ ./ring 2 2
Sent from P1 to P2
Wait for signal - P1
P0-receptionne signal 10
Wait for signal - P2
```

saphyr.ens.math-info.univ-paris5.fr - PuTTY

```
ij04115@saphyr.ens.math-info.univ-paris5.fr's password:
Last login: Sat Nov 21 15:53:27 2020 from 88.121.6.235
[ij04115@saphyr:~]:sam. nov. 21$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ij04115  15499  0.0  0.1  21424 10648 pts/0    Ss   15:53   0:00 -bash2
ij04115  15557  0.0  0.0   2392   640 pts/0    S+   16:03   0:00 ./ring 2 2
ij04115  15558  0.0  0.0   2392    96 pts/0    S+   16:03   0:00 ./ring 2 2
ij04115  15608  0.1  0.1  21256 10432 pts/1    Ss   16:03   0:00 -bash2
ij04115  15615  0.0  0.1  21320  8332 pts/1    R+   16:04   0:00 ps u
[ij04115@saphyr:~]:sam. nov. 21$
```

Deadlocks (interblocages)

because the SIGUSR1 signal can be delivered to a process before `pause()`.

So the process receives a signal before `pause()` and therefore two processes are blocked on `pause()`

Solution: block the SIGUSR1 signal when a process does not wait for this signal (unblock while waiting)

Masque des signaux d'un processus

Définition

Ensemble courant des signaux bloqués par le processus

Consultation, modification du masque des signaux d'un processus

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

- Si oldset $\neq$ NULL, le masque courant du processus est retourné à travers oldset.
- Si set $\neq$ NULL alors how indique comment le masque courant est modifié :
  - SIG_BLOCK $\rightarrow$ *nouveau_masque* = {*masque_courant* $\cup$ set}
  - SIG_UNBLOCK $\rightarrow$ *nouveau_masque* = {*masque_courant* $-$ set}
  - SIG_SETMASK $\rightarrow$ *nouveau_masque* = {set}
- Si set = NULL alors le masque courant n'est pas modifié et how est ignoré

Retourne 0 en cas de succès et -1 sinon

Attente d'un signal avec remplacement temporaire du masque des signaux

```
#include <signal.h>
int sigsuspend(const sigset_t *sigmask);
```

- Remplace temporairement le masque courant des signaux bloqués par `sigmask`
- Met en sommeil (suspend) le processus jusqu'à l'arrivée soit :
  - d'un signal non masqué, non ignoré et non capté qui met fin au processus
  - d'un signal non masqué, non ignoré et capté.
    Si le gestionnaire se termine
    alors `sigsuspend` :
    – **se termine,**
    – **retourne au processus appelant**
    – le masque des signaux est restauré à sa valeur avant l'appel à `sigsuspend`
    sinon `sigsuspend` ne retourne pas et le processus se termine

Retourne **toujours** -1 et `errno` = `EINTR` (appel système interrompu)

IMPORTANT

- Remplacement du masque et mise en sommeil du processus sont réalisés de manière **atomique**

```c
#include <stdio.h> // printf(), fprintf(), perror(), fflush()
#include <stdlib.h> // exit(), malloc(), atoi()
#include <signal.h> // sigaction, sigemptyset, kill, sigsuspend,sigaddset,sigprocmask
#include <unistd.h> // getpid(), fork(), pause()

int num_processus;
void handler(int theSignal);

int main(int argc, char *argv[]){
    int i, j, kill_result;

    if(argc < 3){
       fprintf(stderr,"Usage : %s n m \n", argv[0]);
       exit(1);
    }

    struct sigaction prepaSignal;
    prepaSignal.sa_handler = &handler;
    prepaSignal.sa_flags = 0;
    sigemptyset( &prepaSignal.sa_mask );
    int sigaction_result = sigaction(SIGUSR1, &prepaSignal, 0);
    if(sigaction_result < 0){
       perror("Function sigaction() : ");
       exit(1);
    }

    /* Add of SIGUSR1 to the mask of blocked signals */
    sigset_t set;
    sigemptyset(&set); // sigemptyset(sigset_t *set);
    sigaddset(&set, SIGUSR1); //sigaddset(sigset_t *set, int signo);
    sigprocmask(SIG_BLOCK, &set, NULL); // SIG_BLOCK → nouveau_masque

    int *fork_result_array = (int*)malloc( (atoi(argv[1])+1)* sizeof(int) );
    if(fork_result_array == NULL){
       fprintf(stderr,"Erreur function malloc()");
       exit(1);
    }

    fork_result_array[1] = getpid();
    for(i = 2 ; i <= atoi(argv[1]) ; i++) {
       fork_result_array[i] = fork();
       if(fork_result_array[i] < 0) {
          perror("Function fork() : ");
          exit(1);
       }
       else
       {
          if(fork_result_array[i] == 0) // CODE DU FILS Pi
          {
             num_processus = i;
             for(j = 1 ; j<= atoi(argv[2]) ; j++) {
                    printf("Wait for signal - P%d \n", num_processus);
                    sigset_t set_vide;
                    sigemptyset(&set_vide); // set_vide is empty
                    sigsuspend(&set_vide);//no signal blocked while process suspended
           // Now the current mask is restored and therefore SIGUSR1 is blocked again
                    kill_result = kill(fork_result_array[i-1], SIGUSR1);
                    if(kill_result < 0) {
                           perror("Function kill() : ");
                           exit(1);
                    }
                    printf("Sent from P%d to P%d\n", num_processus, (i-1));
             } // for(j)
          } // FIN CODE DU FILS Pi
       } // else
    } // for(i)
```

?

```
   // CODE DU PERE p1
   num_processus = 1;
   for(i = 1 ; i <= atoi(argv[2]) ; i++) {
      kill_result = kill(fork_result_array[atoi(argv[1])], SIGUSR1);
      if(kill_result < 0) {
         perror("Function kill() : ");
         exit(1);
      }
      printf("Sent from P%d to P%d\n", num_processus, atoi(argv[1]));

      printf("Wait for signal - P%d\n", num_processus);
      sigset_t set_vide;
      sigemptyset(&set_vide);
      sigsuspend(&set_vide);
   } // for
} // main()


void handler(int theSignal) {
     printf("P%d-receptionne signal %d\n",num_processus, theSignal);
     fflush(stdout);
} // handler()
```

Le but de cette exercice est de construire un anneau de processus et chaque processus *wait for* la réception d'un signal du processus qui le précède dans l'anneau et ensuite une fois qu'il reçoi ce signal, il l'envoie au suivant. Donc on fait tourner comme ça le signal un certain nombre de fois.

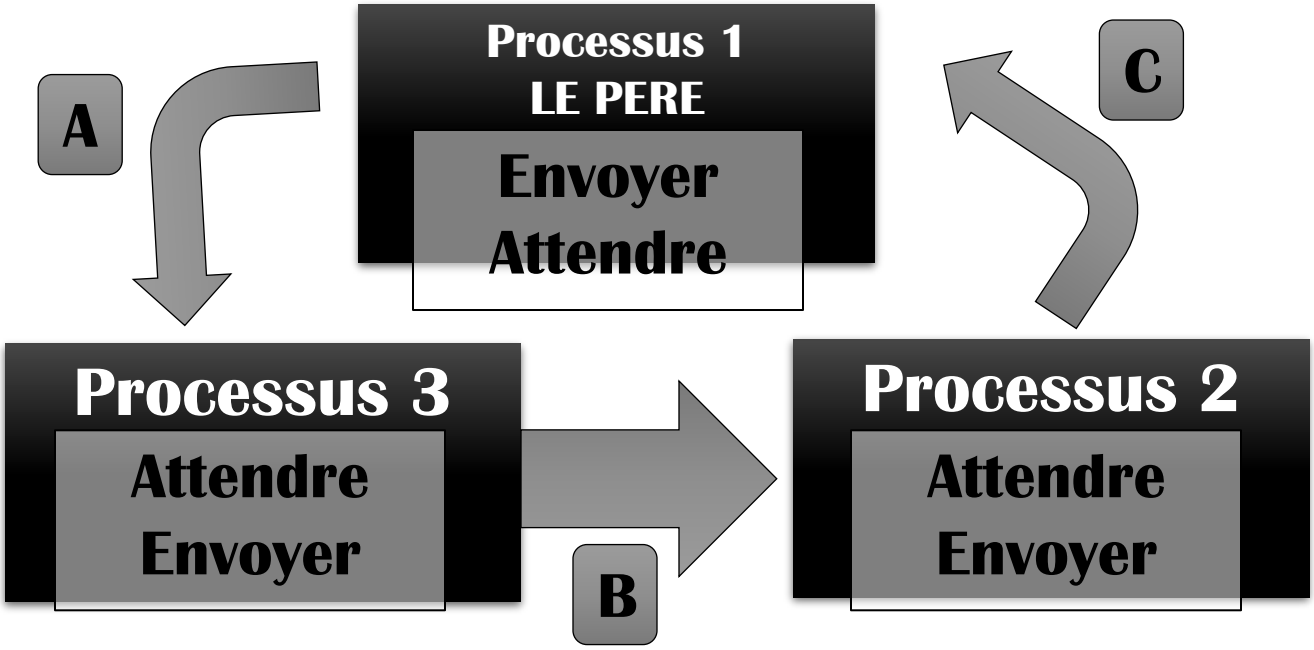
./ring n m

n – nombre de processus dans l'anneau

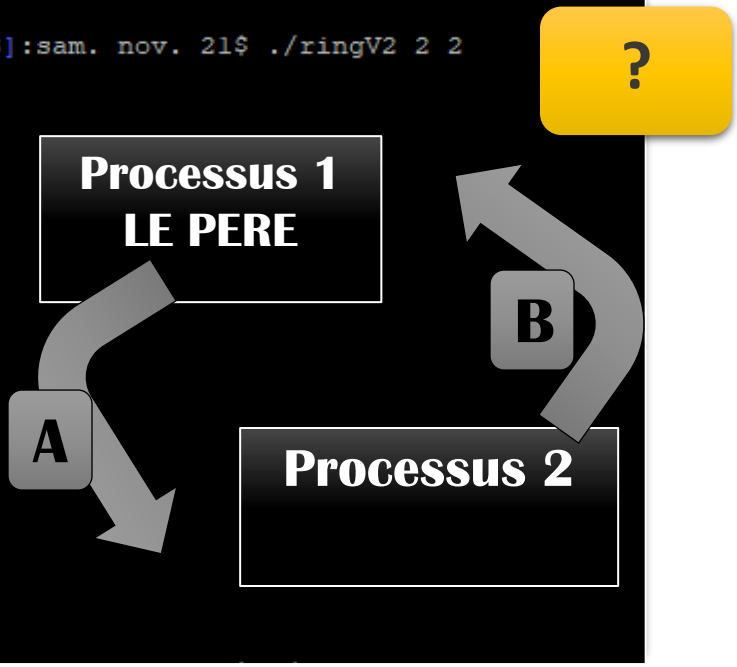
m – le nombre de fois que le signal circule dans l'anneau.

Exemple : on veut créer un anneau de 3

Un père crée 2 processus et ensuite envoie un signal…




```
[ij04l15@saphyr:~/unix_tp3]:sam. nov. 21$ nano ringV2.c
[ij04l15@saphyr:~/unix_tp3]:sam. nov. 21$ gcc ringV2.c -o ringV2
[ij04l15@saphyr:~/unix_tp3]:sam. nov. 21$ ./ringV2
Usage : ./ringV2 n m
[ij04l15@saphyr:~/unix_tp3]:sam. nov. 21$ ./ringV2 2 2
Sent from P1 to P2
Wait for signal - P1
Wait for signal - P2
P2-receptionne signal 10
Sent from P2 to P1
P1-receptionne signal 10
Sent from P1 to P2
Wait for signal - P1
Wait for signal - P2
P2-receptionne signal 10
Sent from P2 to P1
Sent from P1 to P2
Wait for signal - P1
P1-receptionne signal 10
P1-receptionne signal 10
Sent from P1 to P2
Wait for signal - P1
P1-receptionne signal 10
```



b **Est-il possible de faire circuler SIGUSR1 en sens inverse, c-à-d de Pi vers P(i+1) ?**

Justifiez votre réponse

Non.

Pour qu'un processus puisse envoyer un signal a un autre processus, il faut qu'il connaisse sont pid. Et vu que dans cette exercice on crée les processus par ordre de P1 a Pn donc forcement pour un Pi donné, tous ce qui peut hériter comme information c'est les PID des processus qui on été crée avant lui, et étant donner que Pi+1 sera créé après Pi, Pi ne peut pas connaitre le pid de Pi+1. Donc, ce n'est PAS possible.

c **Modifiez le programme écrit en a)**

afin que chaque processus affiche :

1. **Le masque courant au tout début de son exécution**
2. **Le masque des signaux pendants au tout début de son exécution**
3. **Le masque courant juste avant l'envoi de SIGUSR1**
4. **Le masque des signaux pendants et le masque courant lorsqu'il attend SIGUSR1**
5. **Le masque courant lorsqu'il a reçu SIGUSR1**

Masque des signaux d'un processus

Définition

Ensemble courant des signaux bloqués par le processus

Signaux pendants

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

Permet de connaître les signaux pendants du processus appelant

- set : ensemble des signaux pendants (c.-à-d. bloqués et non délivrés) du processus

Retourne 0 en cas de succès et -1 sinon

`sigprocmask()` Function

→ **The signal mask of a process is the set of signals currently blocked from delivery to that process.**

→ **A process can examine its signal mask, change its signal mask, or perform both operations in one step by calling the following function.**

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

- Si oldset $\neq$ NULL, le masque courant du processus est retourné à travers oldset.
- Si set $\neq$ NULL alors how indique comment le masque courant est modifié :
  - SIG_BLOCK → *nouveau_masque* = {*masque_courant* ∪ set}
  - SIG_UNBLOCK → *nouveau_masque* = {*masque_courant* − set}
  - SIG_SETMASK → *nouveau_masque* = {set}
- Si set = NULL alors le masque courant n'est pas modifié et how est ignoré

Retourne 0 en cas de succès et -1 sinon

Exemple – cours page 29

Affichage des masques de signaux d'un processus

```
#include <stdio.h> // perror(), printf()
#include <stdlib.h> // exit()
#include <signal.h> // sigprocmask(), sigismember(), sigset_t

//...

void affich_mask(char *message, sigset_t *mask) {
     sigset_t set_signaux;
     int i;

     if(mask == NULL) { // Affichage du masque courant
        if( sigprocmask(0, NULL, &set_signaux) < 0 ){
           perror("Function sigprocmask() : ");
           exit(1);
    } else {
           // Affichage du masque passé en paramètre
           set_signaux = *mask;
    }

     printf("%s{", message);
     for(i = 1 ; i < NSIG ; i++){
         if( sigismember(&set_signaux,i) )
            printf("%d ", i);
     }
     printf("}\n");
} // affich_mask
```

```
int sigismember(const sigset_t *set, int signo);
```

Teste l'appartenance du signal signo à l'ensemble de signaux pointé par set
Retourne 1 si signo appartient à l'ensemble, 0 si signo n'appartient pas à l'ensemble
ou -1 en cas d'erreur

- signo : numéro du signal
  - < 0 : valeur incorrecte
  - $>$ NSIG : valeur incorrecte
  - $\in$ [1..NSIG] : signal de numéro signo

```c
#include <stdio.h> // printf(), fprintf(), perror(), fflush()
#include <stdlib.h> // exit(), malloc(), atoi()
#include <signal.h> // sigaction, sigemptyset, kill, sigsuspend,sigaddset,sigprocmask
                    // sigismember, sigpending
#include <unistd.h> // getpid(), fork(), pause()

int num_processus;
void affich_mask(char *message, sigset_t *mask);
void handler(int theSignal);

int main(int argc, char *argv[]){
    int i, j, kill_result;

    if(argc < 3){
       fprintf(stderr,"Usage : %s n m \n", argv[0]);
       exit(1);
    }

    struct sigaction prepaSignal;
    prepaSignal.sa_handler = &handler;
    prepaSignal.sa_flags = 0;
    sigemptyset( &prepaSignal.sa_mask );
    int sigaction_result = sigaction(SIGUSR1, &prepaSignal, 0);
    if(sigaction_result < 0){
       perror("Function sigaction() : ");
       exit(1);
    }

    /* Add of SIGUSR1 to the mask of blocked signals */
    sigset_t set;
    sigemptyset(&set); // sigemptyset(sigset_t *set);
    sigaddset(&set, SIGUSR1); //sigaddset(sigset_t *set, int signo);
    sigprocmask(SIG_BLOCK, &set, NULL); // SIG_BLOCK → nouveau_masque

    int *fork_result_array = (int*)malloc( (atoi(argv[1])+1)* sizeof(int) );
    if(fork_result_array == NULL){
       fprintf(stderr,"Erreur function malloc()");
       exit(1);
    }

    fork_result_array[1] = getpid();
    for(i = 2 ; i <= atoi(argv[1]) ; i++) {
       fork_result_array[i] = fork();
       if(fork_result_array[i] < 0) {
          perror("Function fork() : ");
          exit(1);
       }
       else
       {
          if(fork_result_array[i] == 0) // CODE DU FILS Pi
          {
             num_processus = i;
             printf("P%d : ", num_processus);
             affich_mask("masque courant au tout début de lexecution", NULL);
             sigset_t set_pending;
             sigpending(&set_pending);
             printf("P%d : ", num_processus);
             affich_mask("signaux pendants au début de lexecution", &set_pending);
             for(j = 1 ; j<= atoi(argv[2]) ; j++) {
                    printf("Wait for signal – P%d \n", num_processus);
                    sigset_t set_vide;
                    sigemptyset(&set_vide); // set_vide is empty
                    sigsuspend(&set_vide);//no signal blocked while process suspended
           // Now the current mask is restored and therefore SIGUSR1 is blocked again
                     printf("P%d : ", num_processus);
                     affich_mask("masque courant juste avant l'envoi de SIGUSR1", NULL);
                     kill_result = kill(fork_result_array[i-1], SIGUSR1);
                     if(kill_result < 0) {
                            perror("Function kill() : ");
                            exit(1);
                     }
                     printf("Sent from P%d to P%d\n", num_processus, (i-1));
             } // for(j)
          } // FIN CODE DU FILS Pi
       } // else
    } // for(i)
```

?

```
   // CODE DU PERE p1
   num_processus = 1;
    printf("P%d : ", num_processus);
    affich_mask("masque courant au tout début de lexecution", NULL);
    sigset_t set_pending;
    sigpending(&set_pending);
    printf("P%d : ", num_processus);
    affich_mask("signaux pendants au début de lexecution", &set_pending);
   for(i = 1 ; i <= atoi(argv[2]) ; i++) {
      printf("P%d : ", num_processus);
      affich_mask("masque courant juste avant l'envoi de SIGUSR1", NULL);
      kill_result = kill(fork_result_array[atoi(argv[1])], SIGUSR1);
      if(kill_result < 0) {
         perror("Function kill() : ");
         exit(1);
      }
      printf("Sent from P%d to P%d\n", num_processus, atoi(argv[1]));

      printf("Wait for signal - P%d\n", num_processus);
      sigset_t set_vide;
      sigemptyset(&set_vide);
      sigsuspend(&set_vide);
   } // for
} // main()


void handler(int theSignal) {
     printf("P%d-receptionne signal %d\n",num_processus, theSignal);

      printf("P%d : ", num_processus);
      affich_mask("masque courant - reception signal", NULL);
     fflush(stdout);
} // handler()


void affich_mask(char *message, sigset_t *mask) {
     sigset_t set_signaux;
     int i;

     if(mask == NULL) { // Affichage du masque courant
        if( sigprocmask(0, NULL, &set_signaux) < 0 ){
           perror("Function sigprocmask() : ");
           exit(1);
        }
    } else {
           // Affichage du masque passé en paramètre
           set_signaux = *mask;
    }

     printf("%s{", message);
     for(i = 1 ; i < NSIG ; i++){
        if( sigismember(&set_signaux,i) )
           printf("%d ", i);
     }
     printf("}\n");
}
```

```
[ij04115@saphyr:~/unix_tp3]:mer. nov. 25$ ./ringV3
Usage : ./ringV3 n m
[ij04115@saphyr:~/unix_tp3]:mer. nov. 25$ ./ringV3 2 2
P1 : masque courant au tout début de lexecution{10 }
P1 : signaux pendants au début de lexecution{}
P1 : masque courant juste avant l'envoi de SIGUSR1{10 }
Sent from P1 to P2
Wait for signal - P1
P2 : masque courant au tout début de lexecution{10 }
P2 : signaux pendants au début de lexecution{10 }
Wait for signal - P2
P2-receptionne signal 10
P2 : masque courant - reception signal{10 }
P2 : masque courant juste avant l'envoi de SIGUSR1{10 }
Sent from P2 to P1
P1-receptionne signal 10
P1 : masque courant - reception signal{10 }
Wait for signal - P2
P1 : masque courant juste avant l'envoi de SIGUSR1{10 }
Sent from P1 to P2
P2-receptionne signal 10
P2 : masque courant - reception signal{10 }
P2 : masque courant juste avant l'envoi de SIGUSR1{10 }
Wait for signal - P1
Sent from P2 to P1
P1-receptionne signal 10
P1 : masque courant au tout début de lexecution{10 }
P1 : signaux pendants au début de lexecution{}
P1 : masque courant - reception signal{10 }
P1 : masque courant juste avant l'envoi de SIGUSR1{10 }
Sent from P1 to P2
Wait for signal - P1
P1-receptionne signal 10
P1 : masque courant - reception signal{10 }
P1 : masque courant juste avant l'envoi de SIGUSR1{10 }
Sent from P1 to P2
Wait for signal - P1
P1-receptionne signal 10
P1 : masque courant - reception signal{10 }
[ij04115@saphyr:~/unix_tp3]:mer. nov. 25$
```

?

d **Si vous avez écrit le programme en a) avec `pause`, écrivez le avec `sigsuspend`.**

Si vous l'avez écrit avec `sigsuspend`, écrivez le avec `pause`:

```
#include <signal.h>
int sigsuspend(const sigset_t *sigmask);
```

- Remplace temporairement le masque courant des signaux bloqués par `sigmask`
- Met en sommeil (suspend) le processus jusqu'à l'arrivée soit :

```
#include <unistd.h>
int pause(void);
```

Suspend le processus appelant jusqu'à l'arrivée d'un signal quelconque
Retourne -1 et errno = EINTR si un signal a été capté et que le gestionnaire du signal s'est terminé

5 La primitive alarm( )

L'appel à la primitive

```
unsigned int alarm(unsigned int sec);
```

correspond à une requête au système d'envoyer au processus appelant le signal `SIGALARM` dans `sec` secondes.

```
[ij04ll5@saphyr:~]:ven. sept. 04$ 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
```

Name	Description	Default action
SIGALRM	timer expired (`alarm`)	terminate

The `alarm` function allows us to set a timer that will expire at a specified time in the future. When the timer expires, the `SIGALRM` signal is generated. If we ignore or don't catch this signal, its default action is to terminate the process.

```
#include <unistd.h>

unsigned int alarm(unsigned int seconds);

                        Returns: 0 or number of seconds until previously set alarm
```

The `seconds` value is the number of clock seconds in the future when the signal should be generated.

There is only one of these alarm clocks per process.

If, when we call `alarm` ,a previously registered alarm clock for the process has not yet expired, the number of seconds left for that alarm clock is returned as the value of this function.

That previously registered alarm clock is replaced by the new value.

If a previously registered alarm clock for the process has not yet expired and if the seconds value is 0, the previous alarm clock is canceled. The number of seconds left for that previous alarm clock is still returned as the value of the function.

Although the default action for `SIGALRM` is to terminate the process, most processes that use an alarm clock catch this signal.

If the process then wants to terminate, it can perform whatever cleanup is required before terminating.

If we intend to catch `SIGALRM`, we need to be careful to install its signal handler before calling `alarm`. If we call `alarm` first and are sent `SIGALRM` before we can install the signal handler, our process will terminate.

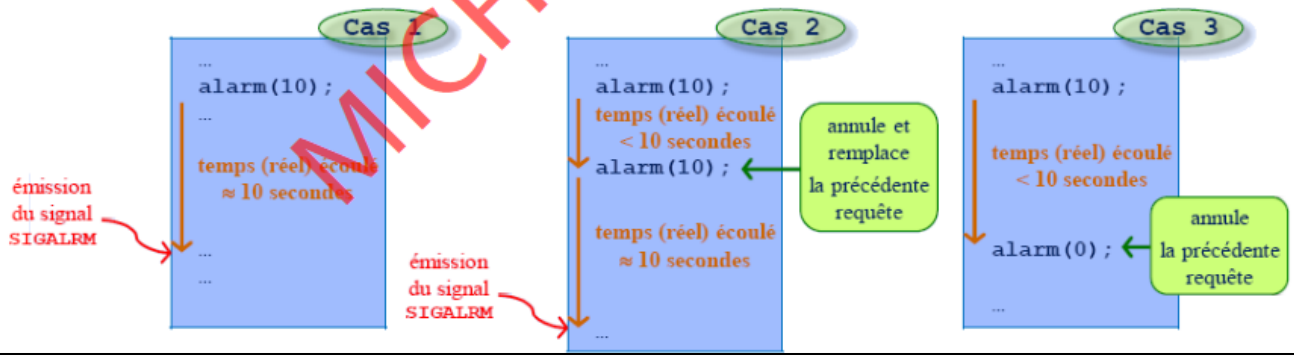
La primitive alarm

```
#include <unistd.h>
unsigned int alarm(unsigned int secondes);
```

Demande au système d'envoyer au processus appelant le signal SIGALRM dans, au plus tôt, secondes secondes.
– Utile pour implémenter des temporisateurs

- Si secondes > 0 : annule et remplace une éventuelle requête déjà en cours
- Si secondes $= 0$: annule la précédente requête (sans la remplacer)

Retourne le nombre de secondes restantes avant que la précédente requête génère le signal SIGALRM ou 0 si aucune requête n'est en cours (jamais d'échec)



Exemple – cours page 21

```c
#include <stdio.h> // printf(), fflush(), scanf()
#include <unistd.h> // alarm()

#define DELAI 5
int main(int argc, char **argv) {
    int valeur;
    alarm(DELAI);

    /* L'utilisateur à DELAI sec, pour entrer sa valeur
     * avant que SIGALRM ne cause la fin du processus
     */

    printf("Vous avez %d s pour saisir une valeur : ", DELAI);
    fflush(stdout);
    scanf("%d", &valeur);

    alarm(0); // Annulation de la requête alarm(DELAI)
    printf("Vous avez saisi : %d \n", valeur);

    return 0;
} // main
```

```
saphyr.ens.math-info.univ-paris5.fr - PuTTY
login as: ij04115
ij04115@saphyr.ens.math-info.univ-paris5.fr's password:
Last login: Sun Nov 22 23:53:27 2020 from 88.121.6.235
[ij04115@saphyr:~]:mar. nov. 24$ cd unix_tp3
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$ nano test_alarm.c
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$ gcc test_alarm.c -o test_alarm
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$ ./test_alarm
Vous avez 5 s pour saisir une valeur : Minuterie d'alerte
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$ ./test_alarm
Vous avez 5 s pour saisir une valeur : Minuterie d'alerte
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$ ./test_alarm
Vous avez 5 s pour saisir une valeur : 1
Vous avez saisi : 1
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$
```

a

Ecrire un programme qui lit cinq données depuis son entrée standard et qu'il sauvegarde dans les cinq entrées d'un vecteur.

Pour chaque donnée, il doit faire une requête à l'utilisateur en affichant un message lui demandant de l'introduire.

Si la donnée n'est pas introduite par l'utilisateur au bout d'un temps d'attente (2 secondes, par exemple)

le programme réitère sa requête jusqu'à ce qu'elle le soit.

A chaque nouvelle requête, pour une même donnée, le temps d'attente est augmenté d'une seconde.

A la fin du programme, pour chaque donnée, afficher le nombre de tentatives de requêtes qu'il a fallu ainsi que le temps écoulé avant son introduction.

```c
#include <stdio.h>  // perror(), printf(), fflush(), scanf()
#include <stdlib.h> // exit()
#include <signal.h> // sigaction(), sigemptyset()
#include <unistd.h> // alarm()

#define DELAI 2
#define NB_DATA 5 // "Ecrire  un  programme  qui  lit  cinq  données"

int delai, i;
int tentatives[NB_DATA] = {1, 1, 1, 1 , 1};
int temps_passe[NB_DATA] = {0, 0, 0 , 0 , 0};
int data[NB_DATA];

void handler(int theSignal);

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

    // Structure pour la mise en place des gestionnaires
    struct sigaction prepaSignal;

    /* Remplissage de la structure */
    prepaSignal.sa_handler = &handler; // Adresse du gestionnaire


    /* SA_RESTART  : System calls interrupted by this signal are
    automatically restarted. */
    prepaSignal.sa_flags = SA_RESTART;

   /* int sigemptyset(sigset_t *set);
    * initialise a VIDE l'ensemble de signaux pointé par set
    * retourne 0 en cas de succès ou -1 en cas d'erreur.
    */

   // On ne bloque pas de signaux spécifiques
   sigemptyset( &prepaSignal.sa_mask );

   /* int sigaction(int sigo, const struct sigaction *act,
  struct sigaction *oldact); */
  int sigaction_result = sigaction(SIGALRM, &prepaSignal, 0);
  if(sigaction_result < 0) {
     perror("Function sigaction() : ");
     exit(1);
  }

  for( i = 0 ; i < NB_DATA ; i++)
  {
      delai = DELAI;
      printf("Vous avez %d s pour saisir la valeur # %d \n", delai, (i + 1));
      fflush(stdout);


      alarm(delai);
      scanf("%d", &data[i]);

     temps_passe[i] = temps_passe[i] + ( ( delai ) - ( alarm(0) ) );
  }

  for( i = 0 ; i < NB_DATA ; i++){
      printf("DATA # %d \n", i+1);
      printf("data : %d \n", data[i]);
      printf("tentatives : %d \n", tentatives[i]);
      printf("temps_passe : %d \n", temps_passe[i]);
 }
}
```

```
void handler(int theSignal) {
	temps_passe[i] += delai;
	tentatives[i]++;
	delai++;

	printf("Vous avez %d s pour saisir la valeur # %d \n", delai, (i + 1));
	fflush(stdout);


	alarm(delai);
}
```

```
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$ nano question5a.c
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$ gcc question5a.c -o question5a
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$ ./question5a
Vous avez 2 s pour saisir la valeur # 1
1
Vous avez 2 s pour saisir la valeur # 2
Vous avez 3 s pour saisir la valeur # 2
2
Vous avez 2 s pour saisir la valeur # 3
Vous avez 3 s pour saisir la valeur # 3
Vous avez 4 s pour saisir la valeur # 3
3
Vous avez 2 s pour saisir la valeur # 4
Vous avez 3 s pour saisir la valeur # 4
Vous avez 4 s pour saisir la valeur # 4
Vous avez 5 s pour saisir la valeur # 4
4
Vous avez 2 s pour saisir la valeur # 5
Vous avez 3 s pour saisir la valeur # 5
Vous avez 4 s pour saisir la valeur # 5
Vous avez 5 s pour saisir la valeur # 5
Vous avez 6 s pour saisir la valeur # 5
5
DATA # 1
data : 1
tentatives : 1
temps_passe : 1
DATA # 2
data : 2
tentatives : 2
temps_passe : 4
DATA # 3
data : 3
tentatives : 3
temps_passe : 7
DATA # 4
data : 4
tentatives : 4
temps_passe : 11
DATA # 5
data : 5
tentatives : 5
temps_passe : 16
[ij04115@saphyr:~/unix_tp3]:mar. nov. 24$
```

b **Définissez un point de reprise (`sigsetjmp`)**

juste avant la requête demandant à l'utilisateur d'introduire une donnée.

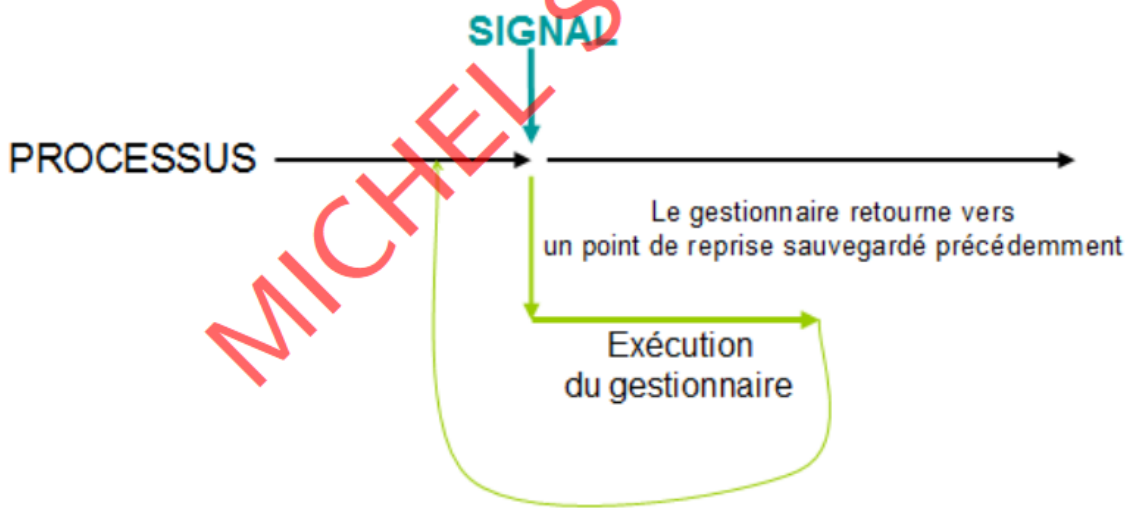
Reprenez l'exécution du programme à ce point de reprise (`siglongjmp`)

chaque fois que l'utilisateur n'a pas introduit la donnée au bout du temps d'attente

Contrôle du point de reprise

La gestion d'erreurs de bas niveau peut être plus efficace en retournant dans la boucle principale du programme plutôt qu'après l'endroit du programme où l'erreur s'est produite.
Pour cela, il faut pouvoir :

- mémoriser le point de reprise
- retourner vers le point de reprise mémorisé



```
#include <setjmp.h>
                int sigsetjmp(sigjmp_buf env, int savemask);
```

Mémorise l'environnement et le masque courant

- `env` : variable où sera mémorisé le contexte d'exécution du point de reprise
- `savemask` : si $\neq 0$, le masque courant des signaux est aussi sauvegardé dans `env`

Retourne 0 pour un appel direct, et une valeur non nulle si elle retourne depuis un appel à `siglongjmp()`.

```
#include <setjmp.h>
                void siglongjmp(sigjmp_buf env, int val);
```

Simule un retour de l'appel à la fonction `sigsetjmp()` avec un retour de valeur égal à `val` (si `val` $= 0$, alors retour de la valeur 1),
Restaure l'environnement sauvegardé par l'appel à `sigsetjmp()`.
Si le masque des signaux a été sauvegardé par `sigsetjmp`, il est aussi restauré.

- `env` : variable où a été mémorisé le contexte d'exécution du point de reprise
- `val` : valeur qui sera retournée par le retour simulé de `sigsetjmp`

Cette fonction ne retourne jamais

```c
#include <stdio.h>  // perror(), printf(), fflush(), scanf()
#include <stdlib.h> // exit()
#include <signal.h> // sigaction(), sigemptyset()
#include <unistd.h> // alarm()
#include <setjmp.h> // sigsetjmp(), siglongjmp()
#define DELAI 2
#define NB_DATA 5 // "Ecrire  un  programme  qui  lit  cinq  données"

int delai, i;
int tentatives[NB_DATA] = {1, 1, 1, 1 , 1};
int temps_passe[NB_DATA] = {0, 0, 0 , 0 , 0};
int data[NB_DATA];

sigjmp_buf env;

void handler(int theSignal);

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

    // Structure pour la mise en place des gestionnaires
    struct sigaction prepaSignal;

    /* Remplissage de la structure */
    prepaSignal.sa_handler = &handler; // Adresse du gestionnaire

    prepaSignal.sa_flags = 0;

   /* int sigemptyset(sigset_t *set);
    * initialise a VIDE l'ensemble de signaux pointé par set
    * retourne 0 en cas de succès ou -1 en cas d'erreur.
    */

   // On ne bloque pas de signaux spécifiques
   sigemptyset( &prepaSignal.sa_mask );

   /* int sigaction(int sigo, const struct sigaction *act,
  struct sigaction *oldact); */
  int sigaction_result = sigaction(SIGALRM, &prepaSignal, 0);
  if(sigaction_result < 0) {
     perror("Function sigaction() : ");
     exit(1);
  }

  for( i = 0 ; i < NB_DATA ; i++)
  {
      delai = DELAI;
      printf("Vous avez %d s pour saisir la valeur # %d \n", delai, (i + 1));
      fflush(stdout);


      alarm(delai);
      sigsetjmp(env, 1);
      scanf("%d", &data[i]);

     temps_passe[i] = temps_passe[i] + ( ( delai ) - ( alarm(0) ) );
  }

  for( i = 0 ; i < NB_DATA ; i++){
      printf("DATA # %d \n", i+1);
      printf("data : %d \n", data[i]);
      printf("tentatives : %d \n", tentatives[i]);
      printf("temps_passe : %d \n", temps_passe[i]);
 }
}
```

3

```
void handler(int theSignal) {
	temps_passe[i] += delai;
	tentatives[i]++;
	delai++;

	printf("Vous avez %d s pour saisir la valeur # %d \n", delai, (i + 1));
	fflush(stdout);


	alarm(delai);
	siglongjmp(env, 1);
}
```

```
[ij04ll5@saphyr:~/unix_tp3]:mar. nov. 24$ gcc question5b.c -o question5b
[ij04ll5@saphyr:~/unix_tp3]:mar. nov. 24$ ./question5b
Vous avez 2 s pour saisir la valeur # 1
Vous avez 3 s pour saisir la valeur # 1
3
Vous avez 2 s pour saisir la valeur # 2
Vous avez 3 s pour saisir la valeur # 2
Vous avez 4 s pour saisir la valeur # 2
Vous avez 5 s pour saisir la valeur # 2
8
Vous avez 2 s pour saisir la valeur # 3
Vous avez 3 s pour saisir la valeur # 3
1
Vous avez 2 s pour saisir la valeur # 4
0
Vous avez 2 s pour saisir la valeur # 5
Vous avez 3 s pour saisir la valeur # 5
Vous avez 4 s pour saisir la valeur # 5
Vous avez 5 s pour saisir la valeur # 5
Vous avez 6 s pour saisir la valeur # 5
5
DATA # 1
data : 3
tentatives : 2
temps_passe : 4
DATA # 2
data : 8
tentatives : 4
temps_passe : 10
DATA # 3
data : 1
tentatives : 2
temps_passe : 3
DATA # 4
data : 0
tentatives : 1
temps_passe : 1
DATA # 5
data : 5
tentatives : 5
temps_passe : 16
[ij04ll5@saphyr:~/unix_tp3]:mar. nov. 24$
```

-