Chapitre

Signaux

7.1 Préliminaire : Pointeurs de fonctions

Un pointeur de fonctions en C est une variable qui permet de désigner une fonction C. Comme nimporte quelle variable, on peut mettre un pointeur de fonctions soit en variable dans une fonction, soit en paramètre dans une fonction.

On déclare un pointeur de fonction comme un prototype de fonction, mais on ajoute une étoile ($*$) devant le nom de la fonction. Dans l'exemple suivant, on déclare dans le main un pointeur sur des fonctions qui prennent en paramètre un `int`, et un pointeur sur des fonctions qui retournent un `int`.

```
#include <stdio.h>

int SaisisEntier(void)
{
   int n;
   printf("Veuillez entrer un entier : ");
   scanf("%d", &n);
   return n;
}

void AfficheEntier(int n)
{
   printf("L'entier n vaut %d\(\backslash\)n", n);
}

int main(void)
{
  void (*foncAff)(int); /* déclaration d'un pointeur foncAff */
  int (*foncSais)(void); \em{}/*déclaration d'un pointeur foncSais */
  inte entier;

  foncSais = SaisisEntier;  {/* affectation d'une fonction */
  foncAff = AfficheEntier; /* affectation d'une fonction */
```

```
  entier = foncSais(); /* on exécute la fonction */
  foncAff(entier);   /* on exécute la fonction */
  \retu 0;
}
```

Dans l'exemple suivant, la fonction est passée en paramètre à une autre fonction, puis exécutée.

```
#include <stdio.h>

int SaisisEntier(void)
{
   int n;
   printf("Veuillez entrer un entier : ");
   scanf("%d", &n);
   getchar();
   return n;
}

void AfficheDecimal(int n)
{
   printf("L'entier n vaut %d\n", n);
}

void AfficheHexa(int n)
}
   printf("L'entier n vaut %x\(\backslash\)n", n);
}

void ExecAffiche(void (*foncAff)(int), int n)
{
  foncAff(n);  /* exécution du paramètre */
}

int main(void)
{
  int (*foncSais)(\void); /*déclaration d'un pointeur foncSais */
  int entier;
  char rep;

  foncSais = SaisisEntier;  /* affectation d'une fonction */
  entier = foncSais(); /* on exécute la fonction */

  puts("Voulez-vous afficher l'entier n en décimal (d) ou en hexa (x)?");
  rep = getchar();
  /* passage de la fonction en paramètre : */
  if (rep == 'd')
```

```
    ExecAffiche(AfficheDecimal, entier);
  if (rep == 'x')
    ExecAffiche(AfficheHexa, entier);

  return 0;
}
```

7.2 Les principaux signaux

Un signal permet de prévenir un processus qu'un évennement particulier c'est produit dans le système pour dans un (éventuellement autre) processus. Certains signaux sont envoyés par le noyau (comme en cas d'erreur de violation mémoire ou division par 0), mais un programme utilisateur peut envoyer un signal avec la fonction ou la commande `kill`, ou encore par certaines combinaison de touches au clavier (comme `Ctrl-C`). Un utilisateur (à l'exception de `root`) ne peut envoyer un signal qu'à un processus dont il est propriétaire.

Les principaux signaux (décrits dans la norme `POSIX.1-1990`) (faire `man 7 signal` pour des compléments) sont expliqués sur le table 7.1.

7.3 Envoyer un signal

La méthode la plus générale pour envoyer un signal est d'utiliser soit la commande sheel `kill(1)`, soit la fonction C `kill(2)`.

7.3.1 La commande `kill`

La commande `kill` prend une option `-signal` et un `pid`.
Exemple.

```
$ kill -SIGINT 14764 {\em# interromp processus de pid 14764}
$ kill -SIGSTOP 22765 {\em# stoppe temporairement le process 22765}
$ kill -SIGCONT 22765 {\em# reprend l'exécution du prcessus 22765}
```

Compléments

- √ Le signal par défaut, utilisé en cas est `SIGTERM`, qui termine le processus.
- √ On peut utiliser un *PID* négatif pour indiquer un groupe de processus, tel qu'il est indiqué par le *PGID* en utilisant l'option `-j` de la commande `ps`. Cela permet d'envoyer un signal à tout un groupe de processus.

7.3.2 La fonction `kill`

La fonction C `kill` est similaire à la commande `kill` du *shell*. Elle a pour prototype :

```
int kill(pid_t pid, int signal);
```

Signal	Valeur	Action	Commentaire
SIGHUP	1	Term	Terminaison du leader de session (exemple : terminaison du terminal qui a lancé le programme ou logout)
SIGINT	2	Term	Interruption au clavier (par Ctrl-C par défaut)
SIGQUIT	3	Core	Quit par frappe au clavier (par Ctr − AltGr − \ par défaut)
SIGILL	4	Core	Détection d'une instruction illégale
SIGABRT	6	Core	Avortement du processus par la fonction abort(3) (généralement appelée par le programmeur en cas de détection d'une erreur)
SIGFPE	8	Core	Exception de calcul flottant (division par 0 racine carrées d'un nombre négatif, etc...)
SIGKILL	9	Term	Processus tué (kill)
SIGSEGV	11	Core	Violation mémoire. Le comportement par défaut termine le processus sur une erreur de segmentation
SIGPIPE	13	Term	Erreur de tube : tentative d'écrire dans un tube qui n'a pas de sortie
SIGALRM	14	Term	Signal de timer suite à un appel de alarm(2) qui permet d'envoyer un signal à une certaine date
SIGTERM	15	Term	Signal de terminaison
SIGUSR1	30,10,16	Term	Signal utilisateur 1 : permet au programmeur de définir son propre signal pour une utilisation libre
SIGUSR2	31,12,17	Term	Signal utilisateur 23 : permet au programmeur de définir son propre signal pour une utilisation libre
SIGCHLD	20,17,18	Ign	L'un des processus fils est stoppé (par SIGSTOP ou terminé)
SIGSTOP	17,19,23	Stop	Stoppe temporairement le processus. Le processus se fige jusqu'à recevoir un signal SIGCONT
SIGCONT	19,18,25	Cont	Reprend l'exécution d'un processus stoppé.
SIGTSTP	18,20,24	Stop	Processus stoppé à partir d'un terminal (tty) (par Ctrl-S par défaut)
SIGTTIN	21,21,26	Stop	saisie dans un terminal (tty) pour un processus en tâche de fond (lancé avec &)
SIGTTOU	22,22,27	Stop	Affichage dans un terminal (tty) pour un processus en tâche de fond (lancé avec &)

Table 7.1 : Liste des principaux signaux

Exemple. Le programme suivant tue le processus dont le *PID* est passé en argument seulement si l'utilisateur confirme.

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

int main(int argc, char **argv)
{
  pid_t pidToSend;
  char rep;
  if (argc != 2)
    {
      fprintf(stderr, "Usage %s pid\n", argv[0]);
      exit(1);
    }
  pidToSend = atoi(argv[1]);
  printf("Etes-vous sûr de vouloir tuer le processus %d? (o/n)",
         pidToSend);
  rep = getchar();
  if (rep == 'o')
     kill(pidToSend, SIGTERM);
  return 0;
}
```

7.3.3 Envoi d'un signal par combinaison de touches du clavier

Un certain nombre de signaux peuvent être envoyé à partir du terminal par une combinaison de touche. On peut voir ces combinaisons de touches par `stty -a` :

```
 stty -a
speed 38400 baud; rows 48; columns 83; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = M-^?; eol2 = M-^?;
swtch = M-^?; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V;
flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iuclc
ixany imaxbel iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt echoctlechoke
```

7.4 Capturer un signal

7.4.1 Créer un gestionnaire de signal (*signal handler*)

Un gestionnaire de signal (*signal handler*) permet de changer le comportement du processus lors de la réception du signal (par exemple, se terminer).

Voici un exemple de programme qui sauvegarde des données avant de se terminer lors d'une interruption par `Ctrl-C` dans le terminal. Le principe est de modifier le comportement lors de la réception du signal `SIGINT`.

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

int donnees[5];

void gestionnaire(int numero)
{
  FILE *fp;
  int i;
  if (numero == SIGINT)
    {
      printf("\nSignal d'interruption, sauvegarde...\n");
      fp = fopen("/tmp/sauve.txt", "w");
      for (i=0 ; i<5 ; i++)
        {
          fprintf(fp, "%d ", donnees[i]);
        }
      fclose(fp);
      printf("Sauvegarde terminée, terminaison du processus\n");
      exit(0);
    }
}

int main(void)
{
  int i;
  char continuer='o';
  struct sigaction action;
  action.sa_handler = gestionnaire; /* pointeur de fonction */
  sigemptyset(&action.sa_mask); /* ensemble de signaux vide */
  action.sa_flags = 0; /* options par défaut */
  if (sigaction(SIGINT, &action, NULL) != 0)
    {
      fprintf(stderr, "Erreur sigaction\(\backslash\)n");
      exit(1);
    }
```

```
  for (i=0 ; i<5 ; i++)
    {
      printf("donnees[%d] = ", i);
      scanf("%d", &donnees[i]); getchar();
    }

  while (continuer == 'o')
    {
      puts("zzz...");
      sleep(3);
      for (i=0 ; i<5 ; i++)
          printf("donnees[%d] = %d ", i, donnees[i]);
      printf("\(\backslash\)nVoules-vous continuer ? (o/n) ");
      continuer = getchar(); getchar();
    }
}
```

Voici le trace de ce programme, que l'on interrompt avec `Ctrl-C` :

```
 gcc sigint.c -o sigint
 ./sigint
donnees[0] = 5
donnees[1] = 8
donnees[2] = 2
donnees[3] = 9
donnees[4] = 7
zzz...
donnees[0] = 5 donnees[1] = 8 donnees[2] = 2 donnees[3] = 9 donnees[4] = 7
Voules-vous continuer ? (o/n)
Signal d'interruption, sauvegarde...
Sauvegarde terminée, terminaison du processus
 cat /tmp/sauve.txt
5 8 2 9 7
```

7.4.2 Gérer une exception de division par zéro

Voici un programme qui fait une division entre deux entiers y/x saisis au clavier. Si le dénominateur x vaut 0 un signal `SIGFP` est reçu et le gestionnaire de signal fait resaisir x. Une instruction `siglongjmp` permet de revenir à la ligne d'avant l'erreur de division par 0.

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <setjmp.h>

int x,y, z;
sigjmp_buf env;
```

```
void gestionnaire(int numero)
{
  FILE *fp;
  int i;
  if (numero == SIGFPE)
    {
      printf("Signal d'erreur de calcul flottant.\n");
      printf("Entrez x différent de zéro : ");
      scanf("%d", &x);
      siglongjmp(env, 1); /* retour au sigsetjmp */
    }
}

int main(void)
{
  int i;
  char continuer='o';
  struct sigaction action;
  action.sa_handler = gestionnaire; /* pointeur de fonction */
  sigemptyset(&action.sa_mask); /* ensemble de signaux vide */
  action.sa_flags = 0; /* options par défaut */
  if (sigaction(SIGFPE, &action, NULL) != 0)
    {
      fprintf(stderr, "Erreur sigaction\n");
      exit(1);
   }

    printf("Veuillez entrer x et y : ");
    scanf("%d %d", &x, &y); getchar();
    sigsetjmp(env, 1); /* on mémorise la ligne */
    z = y/x;  /* opération qui risque de provoquer une erreur */
    printf("%d/%d=%d\(\backslash\)n", y, x, z);
    return 0;
}
```

Voici la trace de ce programme lorsqu'on saisit un dénominateur x égal à 0 :

```
Veuillez entrer x et y : 0 8
Signal d'erreur de calcul flottant.
Entrez x différent de zéro : 0
Signal d'erreur de calcul flottant.
Entrez x différent de zéro : 2
8/2=4
```