TP N°7
Traitement des signaux

Sylvain Chevallier

Vous testerez vos programmes avec des jeux d'essai, c'est-à-dire différentes valeurs pour vérifier son bon fonctionnement. Vous indiquerez en commentaires dans le code source ou dans un fichier séparé, le jeu d'essai que vous avez testé et les résultats obtenus.

Préambule

Notions : Traitement des signaux `kill()`, `signal()` et synchronisation entre processus

Rappel

```c
#include <unistd.h>
int pause(void);
unsigned int alarm(unsigned int seconds);
#include <signal.h>
sighandler_t signal(int signum, sighandler_t handler);
int kill(pid_t pid, int sig);
```

La fonction `pause()` met le processus en sommeil, le processus est réveillé lorsqu'il reçoit un signal. Attention, par défaut la plupart des signaux provoque la terminaison du processus.

La fonction `alarm(n)` prépare l'envoi d'un signal `SIGALRM` après `n` secondes. Si un nouvel appel à `alarm()` est exécuté avant que `n` secondes soient écoulé, l'envoi du précédent `SIGALRM` est annulé. Ainsi vec `alarm(0)`, l'alarme précédente est annulée et aucune nouvelle alarme n'est programmée.

Il est possible d'associer une fonction de type `void handler (int s)` à l'envoi d'un signal avec la fonction `signal()`. Si le processus reçoit le signal `signum`, c'est la fonction `handler` qui sera exécutée.

Pour envoyer un signal depuis un programme en C, il est possible d'utiliser la fonction `kill()`, en spécifiant le `pid` du processus ciblé et le numéro du signal sig choisi.

1 Gestion des signaux

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

int main() {
  printf( "processus %d attend\n",getpid());
  pause ();
  printf("fin processus\n");;
  exit (EXIT_SUCCESS);
}
```

Le processus correspondant a pour numéro `pid`. Que se passe-t-il si on tape la commande `kill -USR1 pid` dans un autre terminal ? Si le programme est modifié comme suit, peut-il afficher le message de fin ?

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

void gestionnaire(int s) {
  printf(" Signal recu %d\n",s) ;
}
```

```c
int main() {
  signal (SIGUSR1,gestionnaire);
  printf( "processus %d attend\n",getpid());
  pause ();
  printf("fin processus\n");;
  exit (EXIT_SUCCESS);
}
```

2 Gestion du signal d'interruption

1. Que se passe-t-il si on tape plusieurs Ctl-C pendant l'exécution du programme ci-dessous ?
2. Comment alors arrêter le programme ?

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

void recup(int s) {
  printf ("signal recu %d\n", s);
}

int main() {
  signal (SIGINT, recup);
  while (1)
   pause() ;
  exit (EXIT_SUCCESS);
}
```

3 Communication par signaux

Commentez le programme suivant. Quels sont les affichages produits ?

```c
int recu = 0;

void inter(int s) {
  //signal(s, inter);
  recu++;
  printf ("signal recu\n");;
}

void sortie(int s) {
  printf ("sortie du pere par Ctl-C, il a recu %d signaux\n", recu);
  exit(EXIT_SUCCESS);
}

int main() {
  pid_t pid;
  pid = fork();
  if (pid == -1){
   perror ("erreur fork");
   exit(EXIT_FAILURE);
  }
  if (pid == 0){
   for(int i=0; i<10; i++){
    kill(getppid(),SIGUSR1);
    sleep(1);
   }
```

```
    printf ("fin du fils\n");
27    exit (EXIT_SUCCESS);
   }
29   else {
    signal(SIGUSR1, inter);
31    signal(SIGINT, sortie);
    while(1)
33     pause ();
    printf ("fin du pere\n");;
35    exit(EXIT_SUCCESS);
   }
37 }
```

4 Alarme

Qu'affiche le programme ci-dessous ? Comment le modifier pour changer l'affichage ?

```
1 void reveil(int s) {
   printf("l'alarme sonne !\n");
3 }

5 int main()
{
7   signal(SIGALRM, reveil);
   while (1)
9     alarm(1);
   exit(EXIT_SUCCESS);
11 }
```

5 Récupération de signaux

Reprenez le code de l'exercice 2. Modifiez le pour qu'il écrive "bonjour" lorsqu'il reçoit le signal `SIGUSR1`. Puis modifiez votre programme pour qu'il écrive également "bonsoir" lorsqu'il reçoit le signal `SIGUSR2`.

6 Échange de signaux entre plusieurs processus

En vous inspirant de l'exercice précédent et de l'exercice 4 du TD, écrivez un programme qui crée deux fils. On considère que chacun des fils a une tâche à faire et le père doit attendre que ses fils aient envoyé un signal pour les terminer et se terminer à son tour. Le premier fils attend un peu avec `sleep()` et envoie vers son père `SIGUSR1`. Le deuxième attend également avant d'envoyer vers son père un signal `SIGUSR2`. Le père attend que ses fils aient envoyé leur signaux, puis il leur envoie un `SIGTERM` et se termine à son tour.

7 Échange de signaux

Écrivez un programme qui crée deux fils, affichant alternativement des messages à l'écran. Le fils 1 affiche "Joueur 1" à l'écran et le fils 2 affiche "Joueur 2" à l'écran. La communication entre les deux fils se fait en utilisant les signaux `SIGUSR1` et `SIGUSR2`. Par exemple, le fils 1 déclenche son affichage écran lorsqu'il reçoit le signal `SIGUSR1` et le fils 2 lorsqu'il reçoit le signal `SIGUSR2`. Vous veillerez à proposer une solution qui permette de terminer correctement les processus.

Pensez à utiliser le père pour arrêter les deux processus fils. Une possibilité est d'utiliser un `handler` pour le signal `SIGINT` (généré par un `Ctrl-C`) chez le père.

8 Une première approche de l'alarme

Avec `alarm()`, un processus peut s'envoyer un signal après un certain temps. Écrivez un programme qui fait un calcul très long, comme une boucle infini qui incrémente un compteur. Vous utilisez une alarme afin de sauvegarder périodiquement (toutes les 3 secondes par exemple) le résultat intermédiaire. Vous utiliserez un `handler` associé au signal `SIGALRM` pour effectuer cette sauvegarde et réarmer l'alarme.

9 Autre usage de l'alarme

En utilisant `alarm(n)`, une temporisation de n seconde est armée, à l'issue de laquelle le processus reçoit un `SIGALRM`. Lorsqu'on utilise `alarm(0)`, la temporisation en cours est annulée et le processus ne recevra pas de `SIGALRM`. Un signal reçu pendant un appel système bloquant fait échouer l'appel système, qui renvoie alors un code d'erreur de valeur `EINTR` (lisible avec la fonction `perror()`).

Écrivez un programme qui demande à l'utilisateur de saisir un nombre et qui affiche son carré. Le programme enclenche une temporisation de 5 secondes avant de demander à l'utilisateur de rentrer une valeur au clavier (avec `scanf()`, appel système bloquant). Si aucune entrée n'a été lue au bout de 5 secondes, le programme continue en utilisant une valeur arbitraire (par exemple `6`).