Manipulation des signaux – première partie

Compte-rendu :

LE CR du TP doit comporter un seul fichier pdf contenant :
- une copie des codes sources,
- les captures d'écran des tests exécutés,
- votre analyse et vos commentaires associés à chaque exercice.

1 Présentation des signaux

Les signaux ont des origines diverses, ils peuvent être :

1. transmis par le noyau : `division par zéro`, `overflow`, `instruction interdite, ...`
2. envoyés depuis le clavier par l'utilisateur (touches : `<CTRL>Z`, `<CTRL>C`, `...`)
3. émis par la commande `kill` depuis le shell ou depuis un programme C par appel à la primitive `kill`

Emission :

- `kill (num_du_processus, num_du_signal)` en C
- `kill -num_du_signal num_du_processus` en Shell

Remarque : l'émetteur ne peut pas savoir si le destinataire a reçu ou non le signal, et le destinataire ne peut pas savoir si l'événement correspondant au signal reçu a réellement eu lieu ou non.

Réception :

Comportements possibles du destinataire du signal :

Ignorer le signal	`signal(num_du_signal, SIG_IGN)`
Repositionner le traitement par défaut	`signal(num_du_signal, SIG_DFL)`
Définir un traitement spécifique	`signal(num_du_signal, fonction)`

L'ensemble des signaux est décrit dans <signal.h> et s'obtient par la commande kill -l

2 Ignorer les signaux

EXERCICE 1

Ecrire un programme qui ignore TOUS les signaux.

Le schéma de programmation est donné ci-dessous. Le rôle du while() est de boucler pour attendre la réception d'un signal.

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

int main(void){
  int Nb_Sig;
  for(Nb_Sig = 1; Nb_Sig < NSIG ; Nb_Sig ++){
    ...
    ...
  }
  while(1){
    sleep(5);
  } /* Attendre des signaux */
  return 0;
}
```

1. Tester la valeur de retour de la fonction signal pour relever les signaux qu'on ne peut ignorer.
2. Utiliser la fonction char* strsignal(int sig) de la bibliothèque signal.h pour afficher les constantes symboliques associées aux signaux qu'on ne peut ignorer.
3. Faire <CTRL> C dans la fenêtre où le programme s'exécute. Envoyer également des signaux vers ce programme par le biais de kill depuis une autre fenêtre. Constater que SIGKILL (signal numéro 9) termine ce programme.

3 Utilisation du signal SIGINT

EXERCICE 2

Ecrire un programme qui génère deux processus, père et fils. Le fils se met dans une boucle d'attente. Après 5 secondes, le père lui envoie le signal SIGINT pour l'arrêter. A la réception, le fils attend 2 secondes et s'arrête. Le père attend la fin du fils avant de s'arrêter lui aussi.

4 Utilisation des signaux `SIGUSR1` et `SIGUSR2`

EXERCICE 3

Ecrire un programme qui :

1. Affiche son numéro (`pid`) via l'appel à `getpid()`
2. Traite tous les signaux, sauf `SIGUSR1` et `SIGUSR2`, par une fonction `fonc` qui se contente d'afficher le numéro du signal reçu.
3. Traite le signal `SIGUSR1` par une fonction `fonc1` et le signal `SIGUSR2` par `fonc2` :

(a) `fonc1` affiche le numéro du signal reçu et la liste des utilisateurs de la machine (appel à la commande `who` par `system("who")`)

(b) `fonc2` affiche le numéro du signal reçu et l'espace disque utilisé sur la machine (appel à la commande `df .` par `system("df .")`)

Lancer le programme et lui envoyer des signaux, dont `SIGUSR1` et `SIGUSR2`, depuis une autre fenêtre, à l'aide de la commande `kill`.

Schéma du programme :

```
#include <signal.h>
#include <unistd.h>
int main (void){
  /* ...
    Mettre ici le traitement pour tous les signaux
    sauf SIGUSR1 et SIGUSR2. */
  ... */
  /* Mettre ici le traitement pour SIGUSR1 et SIGUSR2
  ... */
  while (1){
    sleep(5);
  } /* Attendre les signaux */
  return 0;
}
/*************** La fonction fonc *************/
void fonc (int NumSignal){
//...
}
/*************** La fonction fonc1 *************/
void fonc1 (int NumSignal){
//...
}
/*************** La fonction fonc2 *************/
void fonc2 (int NumSignal){
//...
}
```