Module : Programmation Systèmes et Réseaux
Elément : Programmation Système
TP 4 : Gestion des signaux

La signalisation constitue le mécanisme fondamental de communication non seulement entre processus mais également entre le noyau et l'ensemble des processus du système.

Un signal est défini par un code numérique (dans le fichier **/usr/include/sys/signal.h**) et se manifeste de manière asynchrone par notification d'un événement au processus concerné; cet événement est mémorisé dans la table des processus. Contrairement aux autres mécanismes de communication interprocessus, les signaux ne transportent pas de données, mais indiquent des actions à effectuer dans certaines circonstances.

Emission d'un signal kill() : Cette primitive permet d'envoyer un signal à un processus ou à un groupe de processus.

```
#include <sys/signal.h>
int kill(pid, sig)
int pid, sig;
```

Les entiers **pid** et **sig** désignent respectivement l'identificateur du processus destinataire ou groupe de processus, et le numéro du signal émis. Les processus émetteurs et destinataire doivent avoir le même identificateur (réel ou effectif) d'utilisateur.

- pid > 0 : le signal est destiné au processus dont le PID = pid.
- pid = 0 : le signal est destiné à tous les processus de même groupe que le processus émetteur à l'exception des processus spéciaux : init, swapper, sched …

Réception d'un signal par un processus : Le type de signal reçu par un processus indique la nature de l'événement qui a nécessité l'envoi de ce signal par un autre processus.

```
#include <sys/signal.h>
int (*signal(sig, func))()
int sig, (*func)()
```

Question 1 : écrire un programme qui permet la création de deux processus père et fils. Le programme père ignore un signal de type SIGCLD.

Question 2 : écrire un programme qui permet que le processus ignore (si possible) le signal dont le raccourci est donné en ligne de commande.

Question 3
Ecrire un programme C qui compte les signaux qu'il reçoit et affiche ces compteurs.

Question 4 : Traitement des routines.
a) la routine sig-user sert à quoi ?
b) rectifier le code afin de n'envoyer le signal que à l'un des processus fils.
c) changer l'action à faire, par exemple interrompre l'exécution du processus fils.

```
#include <sys/signal.h>
#include <stdio.h>
int sig_user(), PID=0;
main(){signal(SIGCLD, SIG-IGN); /* SIGCLD ignoré */
        if ((fork())>0)
        {       printf("processus père PID= %d\n", getpid());
                sleep(30); /* laisse au fils le temps nécessaire pour traiter le signal */
                kill(PID, SIGUSR1);
                printf("envoie du signal au fils \n");}
        else{   if ((fork())<0) exit(1);
                printf("le fils arme le signal \n");
                signal(SIGUSR1, sig_user);/* ne pas mettre les parenthèses sig_usre()*/
                pause(); /* attente du signal */}
        exit(0);
}
/* routine de traitement du signal */
sig_user(){     printf("signal reçu par le fils \n");
                exit(0);
              }
```

Question 5 : Attente d'un signal

La primitive **pause** permet à un processus de se mettre en attente d'un signal qui ne doit pas être ignoré. On ne reviendra de l'appel pause que si le signal reçu entraîne la mort du processus appelant. Si le signal attendu doit être traité par une fonction et arrive pendant l'exécution de pause, on a un retour de -1 (erreur EINTR).

Exemple d'un signal attendu et ignoré

```
#include <sys/signal.h>
#include <stdio.h>
int sig, PID;
main(){int sig, PID;
        if ((PID=fork())<0)  exit(1);
        if (PID)
        {                                       /* processus père */
                sleep(10);
                kill (PID, SIGTERM);            /* signal au fils */}
        else{                                   /* processus fils */
                sig=signal(SIGTERM, SIG_IGN);
                printf("je suis toujours là \n");
                pause();                        /* attente */
             }
}
```

Exécuter le programme en commentant le kill pour voir si le signal est envoyé ou non, ensuite commenter le signal pour voir si le signal est ignoré si le processus reste bloqué sur le pause ou non.

Question 6 : Ecrire un programme C qui crée deux processus à l'aide de l'appel système `fork()`. Le père affichera les entiers pairs compris entre 1 et 100, le fils affichera les entiers impairs compris dans le même intervalle. Synchroniser les processus à l'aide des signaux pour que l'affichage soit 1 2 3 4 5 ... 98 99 100.

Question 7: Ecrire un programme qui intercepte deux signaux : le contrôle C de l'utilisateur (qui tente de l'arrêter) et le SIGUSR1 (venant d'un autre terminal) qui au contraire lui prolonge la vie. Il faut par exemple disposer d'un entier (déclaré en global, donc hors de toute fonction) valant 5 au début du programme. Chaque appel à CTR-C décrémente ce compteur, et la réception d'un signal SIGUSR1 lui ajoute 10. La fonction main quant à elle boucle tant que le nombre de vie ne vaut pas zéro.
Ecrire ce programme avec signal, puis avec sigaction.

Question 8: Ecrire un programme C qui utilise 3 processus H, M, S qui incrémentent les 3 «aiguilles » d'une horloge. S reçoit un signal SIGALRM chaque seconde et émet un signal à M. Quand M reçoit un signal, il incrémente son compteur. Quand son compteur passe de 59 à 0, M envoie un signal à H.

Question 9 : illustration du mécanisme de saut non local

Q 1 : Appréhender le comportement des deux programmes suivants:

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

static int i = 0;
static jmp_buf buf;

int
main()
{
    int j = 0;

    if (setjmp(buf))
        for (; j<5; j++)
            i++;
    else {
        for (; j<5; j++)
            i--;
        longjmp(buf,~0);
    }
   printf("%d\n", i );
}
```

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

int i = 0;
jmp_buf buf;

int
main()
{
    int j;

    if (setjmp(buf))
        for (j=0; j<5; j++)
            i++;
    else {
        for (j=0; j<5; j++)
            i--;
        longjmp(buf,~0);
    }
   printf("%d\n", i );
}
```

Q 2 : Exécuter les deux programmes. Commenter l'exécution.

Question 10: Armer un temporisation

Ecrire un programme qui permet de générer un signal SIGALARM en utilisant l'appel système alarm(). Utiliser un gestionnaire de signal qui affiche le numéro de signal pour lequel il a été appelé. Et qui met à 1 une variable globale volatile initialisée à 0. Appeler ce gestionnaire dans le contexte de l'appel système **sigaction** pour le signal SIGALARM. Armer une temporisation de 5s.

Vous devez avoir un affichage comme suit

Avant la boucle, le temps est Wed Apr 17 20:14:00 2013
Gestionnaire appelé pour le signal 14
Après la boucle, le temps est Wed Apr 17 20:14:05 2013
Le gestionnaire a pris le contrôle
La valeur du compteur est 290032

Utiliser la fonction suivante pour afficher le temps système.

void timestamp(char *str) { time_t t; time(&t); printf("%s the time is %s\n", str, ctime(&t)); }

Question 11 : Affichage horloge

Écrire un programme horloge qui affiche l'heure sous le format HH:MM:SS.CC (avec le centième de secondes). Le réaffichage se fera au départ toutes les INTERV secondes, avec INTERV == 1. Ce programme traite les signaux SIGINT (Ctrl c), SIGTSTP (Ctrl z) et SIGQUIT (Ctrl \ ) de la manière suivante:

- SIGINT double l'intervalle INTERV à chaque fois
- SIGTSTP divise par 2 l'intervalle INTERV à chaque fois
- SIGQUIT termine le programme.

Question 12 : Blocage de signaux

Ecrire un programme qui reçois un certain nombre de signaux passés en paramètres et effectue les opérations suivantes.

Q1 : Créer un masque contenant les signaux à bloquer
Q2 : Mettre en place le masque avec sauvegarde de l'ancien masque
Q3 : Endormir le processus pendant une certaine durée après laquelle vous récupérer les signaux pendants. Borner cette période par des messages affichés.
Q4 : Afficher les libellés et les numéros des signaux pendants

T à F :

1. Le travail demandé est individuel.

2. Implémenter les fonctions et les programmes C demandés.

3. Rapport explicatif .pdf contenant les prises d'écrans complètes (avec la barre des taches) non traitées (pas de paint, ni Photoshop ….).

4. Le code .c et .h

5. Envoyer les sources .rar à saadi_mo@yahoo.fr. (avant le 28 Avril 2019 minuit), faites comme objet d'émail : TP4_signaux_VotreNom tout autre objet sera ignoré.