TD Programmation système "Signal"

1. Déroutement d'un signal (On ne peut pas dérouter SIGKILL)

Ecrire un programme qui crée un fils.
- Le père déroute les signaux SIGTERM et SIGKILL, puis s'endort
- Le fils lui envoie successivement ces deux signaux

Que constatez-vous?

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

void Reaction(int);
void erreur(const char *);

void main(void)
{
pid_t pid, ppid;
int retour;
struct sigaction action;
system("clear");
printf("\t COMMUNICATION ENTRE PROCESSUS PAR SIGNAUX\n");
pid = fork();
switch(pid)
        {
        case -1:
                erreur("Fork");
        case 0:
                sleep(3);
                ppid = getppid();
                printf("Le fils envoie SIGTERM %d au pere\n",SIGTERM);
                if ((retour= kill(ppid, SIGTERM)) == -1)
                        erreur("kill SIGTERM");
                sleep(5);
                printf("Le fils envoie SIGKILL %d au pere\n",SIGKILL);
                if ((retour= kill(ppid, SIGKILL)) == -1)
                        erreur("kill SIGKILL");
                sleep(3);
                printf("ppid du fils %d\n",getppid());
                printf("\nFils termine. Taper ENTER...\n");
                break;
        default:
                action.sa_handler = Reaction;
                sigaction(SIGTERM,&action,NULL);
                printf("Le pere a deroute SIGTERM.\n");
                sigaction(SIGKILL,&action,NULL);
                printf("Le pere a deroute SIGKILL.\n");
                sleep(100);
                break;
                }
        }
```

```
void Reaction(int sig)
        {
        printf("Signal recu: %d\n", sig);
        sleep(50);
        }
void erreur(const char * message)
{
        perror(message);
        exit(-1);
}
```

2. Synchronisation par SIGSTOP et SIGCONT

Ecrire un programme qui crée un fils. Le père doit effectuer deux tâches p1 et p2. Le fils doit effectuer deux tâche f1 et f2. On veut synchroniser la réalisation de ces tâches dans cet ordre f1, p1, f2, p2 en utilisant les signaux SIGSTOP et SIGCONT.

```
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
void erreur(const char *);
void main(int argc, char *argv[])
        {
        int pid, statut;
        char *p;

        pid = fork();
        switch(pid)
                {
                case -1:
                        erreur("Erreur fork:");
                case 0 :
                        printf("\nfils : f1\n");
                        sleep(4);
                        kill(getppid(),SIGCONT);
                        kill(getpid(),SIGSTOP);
                        printf("fils : f2\n");
                        sleep(4);
                        kill(getppid(),SIGCONT);
                        break;
                default :
                        kill(getpid(),SIGSTOP);
                        printf("pere : p1\n");
                        sleep(4);
                        kill(pid,SIGCONT);
                        kill(getpid(),SIGSTOP);
                        printf("pere : p2\n");
                        wait(&statut);
                }
        }

void erreur(const char *Message)
        {
        perror(Message);
        exit(-1);
        }
```

3. Le cas particulier de SIGALRM

Ecrire un programme qui attend que l'utilisateur saisisse une question. L'utilisateur a 5 secondes pour le faire. Si au bout de trois tentatives l'utilisateur ne l'a toujours pas fait alors le programme affiche " pas de question, pas de réponse !!", sinon il affiche la question posée

```
#include <signal.h>
#include <stdio.h>
#include <errno.h>
void erreur(const char *);
void f(int sig);
int saisie, cpt;

void main()
        {
        struct sigaction action;
        char question[20];
        action.sa_handler = f;
        if(sigaction(SIGALRM,&action,NULL)==-1)
                erreur("Erreur Sigaction:");
        saisie = 1;
        cpt = 1;
        while(saisie && cpt <= 3)
                {
                saisie = 0;
                alarm(5);
                printf("Quelle est la question\n");
                scanf("%s",question);
                if(saisie == 0)
                        alarm(0);
                }
        if(saisie == 0)
                printf("La question est %s\n",question);
        else
                printf("Pas de question pas de réponse\n");
        }

void f(int sig)
        {
        printf("reveillez-vous \n");
        saisie = 1;
        cpt++;
        }
void erreur(const char * message)
        {
        perror(message);
        exit(-1);
        }
```

4. Le traitement des zombies avec le signal SIGCHLD

Ecrire un programme qui crée un fils. Le père déroute le signal SIGCHLD pour supprimer le fils zombie, puis il fork. Le fils affiche son pid. Le programme ne se terminera pas, il faut le tuer à partir de la ligne de commande

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
void f(int);
void derouter(int);
void erreur(const char *);
int main()
{
        pid_t pid;
        derouter(SIGCHLD);
        pid = fork();
        switch(pid)
                {
                case -1:
                erreur("Erreur Fork:");

                case 0 :
                printf("Fils : %d\n",getpid());
                break;

                default:
                while(1)
                        {
                        printf("Je travaille\n");
                        sleep(3);
                        }
                }
       return 0;
}
void derouter(int sig)
{
        struct sigaction action;
        action.sa_handler = f;
        printf("\n signal %d recu et deroute par le pere\n",sig);
        if(sigaction(sig,&action,NULL)==-1)
                erreur("Erreur Sigaction:");
}
void f(int sig)
{
        int statut;
        printf("Le fils zombie est supprime %d\n", wait(&statut));
}
void erreur(const char * message)
{
  perror(message);
  exit(-1);
}
```