Systèmes d'Exploitation
TP ❹ – Les Signaux

➲ *Exercice 1 :* `signal`

Ecrire un programme qui intercepte le CTRL-C et qui n'accepte de tuer le processus correspondant à ce programme qu'après vérification par un mot de passe.

```
#include <iostream.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>

char PASS[]="fleur";

void password(int s)
{
    // Ignore pendant le mot de passe les CtrlC :
    signal(s,SIG_IGN);
    cout<<"\nPassword: ";
    char entree[30];
    cin.getline(entree,'\n');
    if(strcmp(entree,PASS)==0)
    {
        // Retablit le controle C par defaut...
        signal(SIGINT,SIG_DFL);
        // ...Pour l'utiliser contre le present programme :
        kill(getpid(),SIGINT);
    }
    else
    {
        // Sinon retablit le controle par mot de passe :
        signal(SIGINT, password);
        cout<<"Mot de passe incorrect !";
    }
}

int main()
{
    // Traite le Control C :
    signal(SIGINT, password);
    // Boucle :
    while(1) cout<<"Salut ! ";
    return 0;
}
```

➲ *Exercice 2 :* `signal, alarm`

Ecrire un programme qui affiche d'une manière continuelle le message « bonjour », tout en affichant le message « au revoir » toutes les trois secondes.

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

void message(int s)
{
    printf("\n\nAU REVOIR !\n\n");
    // Retablit le signal alarme :
    alarm(3);
    signal(SIGALRM,message);
}

int main()
{
    // Declenche un signal SIGALRM au bout de 3 secondes :
    alarm(3);
    // Recupere les signaux d'alarme :
    signal(SIGALRM,message);
    // Boucle infinie :
    while(1) printf("bonjour,");
    return 0;
}
```

➲ *Exercice 3 :* signal, alarm, pause, kill

Ecrire un programme *timeout* qui reçoit comme paramètre un temps *t* et le *pid* d'un processus, et qui tue le processus au bout de *t* secondes.

```
#include <iostream.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

void alarme(int s)
    {cout << "Destruction." << endl;}

int main(int argc, char * argv[])
{
    // Recuperation des parametres :
    int t=atoi(argv[1]);
    int pid=atoi(argv[2]);
    if(pid<=0)
    {
        cout << "Usage : timeout <t> <pid>\n"
             << "        Detruit le processus <pid> au bout de <t> secondes." << endl;
        return 0;
    }
    cout << "Le processus " << pid << " va etre ejecte dans " << t << " sec. !" << endl;

    // Verification du temps :
    if (t<0) t = -t;
    if (t)
    {
        // Recupere les signaux d'alarme :
        alarm(t);
        signal(SIGALRM, alarme);
        // Attente du programme...
        cout << "Attente..."<<endl;
        pause();
    }
    // Detruit le processus pid :
    kill(pid,SIGKILL);
    return 0;
}
```

➲ *Exercice 4 :* Mini-Ordonnanceur

Ecrire deux programmes qui affichent d'une manière alternative le message « ceci est un bon exercice pour comprendre le mécanisme des signaux ». La synchronisation entre les processus correspondants aux deux programmes est assurée par un troisième processus : l'ordonnanceur.

Fichier : 'miniproc.cpp'

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

int n;

void interrupt(int s)
{
    cout << ++n << " >  Ceci est un bon exercice pour comprendre le"
         << "\n            mecanisme des signaux..." << endl;
    signal(s,interrupt);
}

int main()
{
    // Initialisation du compteur de ligne :
    n=0;

    // Pour des raisons pratiques...
    cout << "Mon PID est : " << getpid() << endl;
    signal(SIGUSR1,interrupt);

    // Boucle infinie :
    while(1) {pause();}
    return 0;
}
```

Fichier : 'miniordo.cpp'

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

// Numero de processus actuel, et pid correspondants :
int procactuel,pid1,pid2;

// Donne la main aux processsus l'un apres l'autre :
void swap(int s)
{
    if(procactuel==0)
    {
        cout << "Main au processus 2" << endl;
        kill(pid2,SIGUSR1);
        procactuel=1;
    }
    else
    {
        cout << "Main au processus 1" << endl;
        kill(pid1,SIGUSR1);
        procactuel=0;
    }

    // Reprise :
    alarm(1);
    signal(s,swap);
}

void detruit(int s)
{
    // L'ordonnanceur entraine dans sa chute les deux
    // processus fils :
    kill(pid1,SIGINT);
    kill(pid2,SIGINT);
    int mypid=getpid();
    kill(mypid,SIGINT);
}

int main(int argc, char * argv[])
{
    // Recuperation des parametres :
    pid1=atoi(argv[1]);
    pid2=atoi(argv[2]);

    // Verification des deux pid :
    if(pid1<=0 || pid2<=0)
    {
        cout << "Usage : miniordo <pid1> <pid2>\n"
             << "        Execute de maniere alternative deux processus <pid1> et <pid2>." << endl;
        return 0;
    }

    // Declaration des fonctions de traitement d'interruption :
    signal(SIGALRM,swap);
    signal(SIGINT,detruit);

    // Initialisation :
    cout << "Pressez CTRL-C pour quitter !" << endl;
    cout << "Mini-ordonnanceur pret." << endl;
    procactuel=0;
    kill(pid1,SIGUSR1);
    alarm(1);

    // Boucle infinie :
    while(1) {pause();}
    return 0;
}
```

Utilisation : Lancez trois terminaux, puis exécutez dans deux d'entre eux le programme *miniproc*. Puis démarrez dans le troisième terminal le programme *miniordo*, avec comme argument les *pid* des deux processus maintenant en attente. Pour quitter l'Ordonnanceur, il n'y a pas mieux qu'un bon CTRL-C...