Systèmes d'Exploitation
TP ❺ – L'Ordonnanceur

Ecrire un programme qui permet de partager le temps d'exécution entre différents processus :

- Chaque processus doit pouvoir s'exécuter sans interruption pendant une seconde (*alarm* et *pause* dans le gestionnaire).
- Chaque processus nouvellement créé doit communiquer son PID au gestionnaire de tâches afin qu'il soit pris en compte.
- Les processus doivent communiquer leur PID via un ou plusieurs fichiers.
- Les processus doivent également connaître PID du gestionnaire.
- Le gestionnaire utilise les signaux SIGSTOP et SIGCONT pour arrêter et continuer l'exécution des processus.
- Le gestionnaire utiliser une liste circulaire doublement chaînée pour gérer les processus présents.
- Quand un processus se termine, il doit prévenir le gestionnaire. Ce dernier doit l'ôter de sa liste chaînée.
- Les processus utilisent les signaux SIGUSR1 et SIGUSR2 pour communiquer avec le gestionnaire.

Fichier : 'ordo.cpp' – Le gestionnaire de tâche

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

// Periode allouee a chaque processus successivement :
const int T_PERIODE=1;

// Maillon elementaire de liste :
class Proc
{
private:
        Proc * prec;
        Proc * suiv;
        long pid;

public:
        Proc(long p) {prec=NULL; suiv=NULL; pid=p;}
        ~Proc() {}

        Proc * getPrec() {return prec;}
        Proc * getSuiv() {return suiv;}
        long getPid() {return pid;}

        void setPrec(Proc * p) {prec=p;}
        void setSuiv(Proc * p) {suiv=p;}
};

// Liste doublement chainee circulaire classique (no comment) :
class Liste
{
private:
        Proc * debut;
public:
        Liste(){ debut = NULL; }
        ~Liste()
        {
                if(!debut) return;
                Proc * pprec,
                     * p=debut;
                while(p->getSuiv()!=debut)
                {
                        pprec=p;
                        p=p->getSuiv();
                        destroy(pprec);
                }
                destroy(p);
        }

        void insert( long pid )
        {
                Proc * nouveau=new Proc(pid);
```

```
                if(debut)
                {
                        debut->getPrec()->setSuiv(nouveau);
                        Proc * dernier = debut->getPrec();
                        debut->setPrec(nouveau);
                        nouveau->setSuiv(debut);
                        nouveau->setPrec(dernier);
                }
                else
                {
                        debut=nouveau;
                        nouveau->setSuiv(nouveau);
                        nouveau->setPrec(nouveau);
                }
        }

        void destroy(Proc * pkilled)
        {
                if(!pkilled) return;
                if(pkilled->getSuiv()==pkilled)
                {
                        delete pkilled;
                        debut=NULL;
                }
                else
                {
                        pkilled->getPrec()->setSuiv(pkilled->getSuiv());
                        pkilled->getSuiv()->setPrec(pkilled->getPrec());
                        if(debut==pkilled) debut=pkilled->getSuiv();
                        delete pkilled;
                }
        }

        void print(int n=10)
        {
                if(!debut) {cout << "Liste vide !";return;}
                Proc * p=debut;
                for(int i=0; i<n; i++)
                {
                        cout << p->getPid() << ";";
                        p=p->getSuiv();
                }
        }

        Proc * getDebut() {return debut;}
};

// Variables globales :
Liste listProcessus;
Proc * encours=NULL;

void newproc(int s)
{
        // Creation d'un nouveau processus :
        long pid;
        ifstream fproc("proc.pid");
        fproc >> pid;
        fproc.close();

        cout << "> Admission du processus " << pid << endl;
        listProcessus.insert(pid);

        // Si premier processus admis :
        if(!encours)
                encours=listProcessus.getDebut();
        else
                kill(pid,SIGSTOP);

        signal(s,newproc);
}

void killproc(int s)
{
        // Destruction du processus concerne :
        cout << "> Destruction du processus " << encours->getPid() << endl;
        Proc * vieux=encours;
        encours=encours->getSuiv();
        // Gestion du cas du dernier processus :
        if(encours==vieux) encours=NULL;
        // Destruction proprement dite :
        listProcessus.destroy(vieux);
        // Lancement du processus suivant :
        if(encours)
        {
                cout << "    Reprise du processus " << encours->getPid() << endl;
                kill(encours->getPid(),SIGCONT);
        }
        else cout << "    En attente..." << endl;

        signal(s,killproc);
}
```

```
void swap(int s)
{
        if(encours)
        {
                // Verifie qu'il y a plusieurs processus...
                if(encours->getSuiv()!=encours)
                {
                        cout << "    Interruption du processus " << encours->getPid() << endl;
                        kill(encours->getPid(),SIGSTOP);
                        encours=encours->getSuiv();
                        cout << "    Reprise du processus " << encours->getPid() << endl;
                        kill(encours->getPid(),SIGCONT);
                }
        }
        // Pour test :
        //cout << "Alarme! ";listProcessus.print(); cout << endl;

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

int main()
{
        ofstream fordo("ordo.pid");
        fordo << getpid();
        fordo.close();
        signal(SIGUSR1,newproc);
        signal(SIGUSR2,killproc);
        signal(SIGALRM,swap);

        // Boucle infinie :
        cout << "> Ordonnanceur pret." << endl;
        cout << "    En attente..." << endl;
        alarm(T_PERIODE);
        while(1) {pause();}

        return 0;
}
```

Fichier : 'proc.cpp' – Un processus de test (indiquer en argument le temps d'exécution voulu)

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

void erreur()
{
        cout << "Usage : proc <duree>\n"
             << "        Affiche des points pendant <duree> en secondes." << endl;
}

void debutProc()
{
        long pid;
        ifstream fordo("ordo.pid");
        ofstream fproc("proc.pid");
        fordo >> pid;
        fproc << getpid();
        fordo.close();
        fproc.close();
        kill(pid,SIGUSR1);
}

void finProc()
{
        ifstream fichier("ordo.pid");
        long pid;
        fichier >> pid;
        fichier.close();
        kill(pid,SIGUSR2);
}

int main(int argc, char * argv[])
{
        // Verification des arguments :
        if(argc<1) {erreur();return 0;}
        int temps=atoi(argv[1]);
        if(temps<1) {erreur();return 0;}

        // Initialisation :
        debutProc();

        cerr << "* DEMARRAGE - Duree du processus : "
             << temps << " secondes." << endl;
```

```
        for(int i=0; i<temps; i++)
                {sleep(1);cerr << ".";}

        cout << "\n* FIN" << endl;

        // Fin :
        finProc();
}
```

Fichier : 'x.cpp' – Ordonnancement de tout type de processus (indiquer en argument une ligne de commande classique, par exemple « *top* » ou « *netscape* ». Ceci permet à n'importe quel processus d'être ordonnancé par notre gestionnaire, pourvu qu'il soit exécuté avec un 'x' devant !)

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

void erreur()
{
        cout << "Usage : x <commande>\n"
             << "        Execute <commande> en gerant l'ordonnancement." << endl;
}

void initProc(const long & mypid)
{
        long pid;
        ifstream fordo("ordo.pid");
        ofstream fproc("proc.pid");
        fordo >> pid;
        fproc << mypid;
        fordo.close();
        fproc.close();
        kill(pid,SIGUSR1);
}

void retProc()
{
        ifstream fichier("ordo.pid");
        long pid;
        fichier >> pid;
        fichier.close();
        kill(pid,SIGUSR2);
}

void urgenceStop(int s)
{
        retProc();
        kill(getpid(),SIGINT);
}

int main(int argc, char * argv[])
{
        // Verification des arguments :
        if(argc<1) {erreur();return 0;}

        // Interception des CTRL-C
        signal(SIGINT,urgenceStop);

        // Lance le programme :
        int res;
        res=fork();
        if(res>0)
        {
                // Processus PERE : gere l'ordonnancement
                initProc(res);
                wait(&res);
                retProc();
        }
        else if(res==0)
        {
                // Processus FILS
                execvp(argv[1], argv+1);
                cout << "\nImpossible d'executer la commande " << argv[1] << endl;
        }
        else cout << "\nCreation d'un processus supplementaire impossible !";
}
```

Utilisation : L'Ordonnanceur doit être exécuté en premier dans un terminal à part. Utilisez d'autres terminaux pour lancer les divers processus, qui seront alors ordonnancés par le gestionnaire. Les processus et l'Ordonnanceur doivent être dans le même répertoire. Donc pour tester *x*, ajoutez éventuellement son répertoire dans la variable d'environnement *path*.