UNIVERSITÉ DE ROUEN	ANNÉE 2005-2006
FACULTÉ DES SCIENCES	SYSTÈME D'EXPLOITATION
LICENCE INFORMATIQUE 3ème ANNÉE	FICHE DE TP N° 3

Ordonnancement de processus

Exercice 1 Écrire un programme qui prend en paramètre le pid d'un processus et qui retourne sa priorité. Indication : utiliser l'appel système *sched_getparam()*.

Exercice 2 Sous Linux les processus sont répartis en trois classes d'ordonnancement : SCHED_FIFO, SCHED_RR, et SCHED_OTHER. Les deux premières classes sont utilisées par les processus «temps réel» et la troisième est réservée aux processus ordinaires.

Les processus de la classe SCHED_OTHER ont une priorité statique de 0, ceux de la classe SCHED_FIFO ou SCHED_RR peuvent avoir une priorité statique allant de 1 à 99. Seuls les processus disposant des privilèges du super utilisateur peuvent obtenir une priorité statique supérieure à 0 afin d'être mis dans la classe SCHED_FIFO ou SCHED_RR.

L'intervalle de priorités associé à une classe d'ordonnancement varie d'un système à l'autre.

1) En utilisant les appels sytème *sched_get_priority_min()* et *sched_get_priority_max()*, écrire un programme qui affiche l'intervalle de priorités asscocié à chacune des classes d'ordonnancement de votre système Linux.

2) L'appel système *sched_getscheduler(pid)* prend en paramètre un numéro de processus et retourne en cas de succès l'une des trois valeurs SCHED_FIFO, SCHED_RR ou SCHED_OTHER selon la classe d'ordonnancement du processus de pid *pid*.

Écrire un programme qui prend en argument un numéro de processus et affiche la classe d'ordonnancement qui lui est associée.

Exercice 3 Le pseudo-répertoire */proc* contient des fichiers virtuels qui contiennent des informations fournies par le noyau. Parmi ces sous-répertoires on trouve des répertoires dont le nom est un nombre. Chacun de ces nombres correspond au pid d'un processus qui tourne sur le système. Par exemple

```
$ ls -l /proc/8551
total 0
dr-xr-xr-x  2 ziadi ziadi 0 oct 10 17:58 attr
-r--------  1 ziadi ziadi 0 oct 10 17:58 auxv
-r--r--r--  1 ziadi ziadi 0 oct 10 17:58 cmdline
lrwxrwxrwx  1 ziadi ziadi 0 oct 10 17:58 cwd -> /home/ziadi
-r--------  1 ziadi ziadi 0 oct 10 17:58 environ
lrwxrwxrwx  1 ziadi ziadi 0 oct 10 17:58 exe -> /bin/bash
dr-x------  2 ziadi ziadi 0 oct 10 17:58 fd
-r--------  1 ziadi ziadi 0 oct 10 17:58 maps
-rw-------  1 ziadi ziadi 0 oct 10 17:58 mem
-r--r--r--  1 ziadi ziadi 0 oct 10 17:58 mounts
lrwxrwxrwx  1 ziadi ziadi 0 oct 10 17:58 root -> /
-r--r--r--  1 ziadi ziadi 0 oct 10 17:58 stat
-r--r--r--  1 ziadi ziadi 0 oct 10 17:58 statm
-r--r--r--  1 ziadi ziadi 0 oct 10 17:58 status
dr-xr-xr-x  3 ziadi ziadi 0 oct 10 17:58 task
-r--r--r--  1 ziadi ziadi 0 oct 10 17:58 wchan
```

– *cmdline* : contient le nom de la commande exécutée,
– *cwd* : contient le répertoire de travail du processus,
– *environ* : contient les variables d'environnement,
– *status* : contient l'état du processus,
– *fd* : répertoire contenant la liste des descripteurs des fichiers ouverts par le processus,
– ....

1) Écrire une fonction *caracteristique(pid_t pid)* qui prend en argument un numéro de processus et qui affiche ses caractéristiques (pid, commande exécutée, état, ...).

Voici quelques fonctions qui peuvent servir : *fopen(), fgets(), fputs(), sprintf(), strncmp()*. À vos man.

La fonction *pid_t *liste_proc()* retourne un tableau d'entiers. Le premier élément du tableau contient le nombre de processus qui tournent sur le système. Les éléments suivants contiennent les pids des processus.

```
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>


pid_t *liste_proc(void){

struct proc {
        pid_t pid;
        struct proc *suivant;
};
typedef struct proc Proc;

struct dirent *entree;
DIR *rep;
Proc *lst_proc=NULL, *p;
int nbproc=0, *tab;

if ((rep = opendir("/proc"))==NULL){
        perror("opendir");
        exit(1);
}

while ((entree = readdir(rep))) {
        if (isdigit(entree->d_name[0])){
                if ((p=(Proc *)malloc(sizeof(Proc)))==NULL){
                        perror("malloc");
                        exit(2);
                }
                nbproc++;
                p->pid = atoi(entree->d_name);
                p->suivant = lst_proc;
                lst_proc = p;
        }
}
```

```
closedir(rep);

if ((tab = (int *) malloc((nbproc+1)*sizeof(pid_t)))==NULL){
        perror("malloc");
        exit(3);
}

p = lst_proc;
tab[0] = nbproc;
while(nbproc--){
        tab[nbproc+1] = p->pid;
        p = p-> suivant;
}

free(lst_proc);
return(tab);
};
```

2) En utilisant les deux fonctions précédentes, écrire un programme qui affiche les caractéristiques de tous les processus qui tournent sur le système.

3) On désire écrire un programme qui crée 4 processus (1 processus père et 3 processus fils). Le père crée les trois processus, affiche son état et celui de ses fils et s'endort pendant 1mn. Au réveil le père attend la terminaison de ses fils et se place en attente de lecture d'un caractère. Chaque fils affiche son pid, son état, l'état de son père et s'endort pendant 30s pour enfin se terminer.