TD-TP de Système n° 8 : Verrous de fichiers

Exercice 1 : Manipulation des verrous

Voici la description de la fonction `fcntl` : `int fcntl(in fd, int op, struct flock *verrou);` ou la structure `struct flock` est de nie comme suit :

```
struct flock
{
    short int l_type; /* F_RDLCK, F_WRLCK ou F_UNLCK */
    short int l_whence; /* SEEK_SET, SEEK_CUR ou SEEK_END */
    off_t l_start;    /* position relative a l_whence */
    off_t l_len;      /* longueur de l'intervalle */
    pid_t l_pid;      /* pid du processus auquel appartient le verrou */
};
```

et op est parmi :

- `F_SETLK` : Demande de pose non bloquante de verrou.
- `F_SETLKW` : Demande de pose bloquante de verrou. Si il existe deja un verrou bloquant, le processus est endormi jusqu'a ce qu'il n'y ait plus de verrou incompatible ou que l'appel soit interrompu.
- `F_GETLK` : Test d'existence d'un verrou incompatible avec le verrou donne. S'il n'existe pas, le champ `l_type` de `verrou` vaut `F_UNLCK` sinon, `verrou` est rempli avec les informations sur ce verrou incompatible.

Le type du verrou decrit les possibilites de cohabitation des di erents verrous :

- `F_RDLCK` (verrou partage) : plusieurs verrous de ce type peuvent cohabiter, c'est-a-dire avoir des portees non disjointes, par exemple les verrous [80,150] et [100,123] ;
- `F_WRLCK` (verrou exclusif) : pas de cohabitation possible avec un autre verrou quel que soit son type.

Considerons maintenant le programme verrou.c suivant :

```
#include <stdlib.h>
#include <sys/file.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>
void erreur(char *c){
    perror(c); exit(EXIT_FAILURE);
}
int main(int argc,char **argv){
    int fic; short mode; struct flock lck;
    assert(argc == 4);
    switch (argv[1][1]) {
        case 's' : mode = F_RDLCK; break;
        case 'x' : mode = F_WRLCK; break;
        default : erreur ("mode");
    }
    if ((fic = open("test", O_RDWR)) == -1) erreur("ouverture");
    lck.l_type = mode;
```

```
    lck.l_whence = SEEK_SET;
    lck.l_start = atoi(argv[2]);
    lck.l_len = atoi(argv[3]);
    if (fcntl(fic, F_GETLK, &lck)) erreur("test verrou");
    if (lck.l_type == F_UNLCK) {
        lck.l_type = mode;
        if (fcntl(fic, F_SETLK, &lck)) erreur("pose verrou");
        printf("Verrou mis\n"); while(1);
    }
    else {
        switch (lck.l_type) {
            case F_RDLCK : printf("verrou partage pose par %d\n", lck.l_pid); break;
            case F_WRLCK : printf("verrou exclusif pose par %d\n", lck.l_pid); break;
        }
        printf("intervalle bloque : [%d,%d[\n", lck.l_start, lck.l_start+lck.l_len);
    }
    return 0;
}
```

Quel est l'e et des suites de commandes suivantes :

```
> verrou -s 10 10 &          > verrou -x 10 5 &
> verrou -s 5 10 &           > verrou -s 5 0 &
> verrou -x 10 5 &           > verrou -x 20 5 &
> verrou -x 20 0 &           > verrou -x 0 10 &
```

Que se passerait-il dans chacun de ces cas sans le test `if(lck.l_type == F_UNLCK)` ?

Exercice 2 :

On veut creer un agenda pour un groupe de personnes. Pour cela, on dispose :

- d'un type `struct date` qui decrit une date,
- d'une fonction `struct date *string2date(const char *)` qui convertit une cha^ne de caracteres au format `"jj.mm.aaaa"` en un pointeur sur `struct date` (pointeur `NULL` si la cha^ne n'est pas au bon format ou ne correspond pas a une date existante),
- d'une fonction `char *date2string(struct date)` qui realise l'operation inverse.

Les fonctions qui restent a implementer sont :

- `int consulter(const struct date *)`,
- `int noter(const struct date *)`,
- `int annuler(const struct date *)`,
- `int modifier(const struct date *)`.

Pour simpli er, on supposera qu'on ne peut noter qu'un evenement par jour, et que l'evenement prevu a la date `jj.mm.aaaa` est decrit dans le chier `$AGENDA/jj.mm.aaaa`, ou `$AGENDA` est une variable d'environnement. Pour que l'utilisation de cet agenda se fasse sans heurt, il faut evidemment que personne ne puisse ecrire dans un chier que quelqu'un consulte et que personne ne puisse consulter un chier sur lequel on est en train d'ecrire...

1. Proposez un programme avec lequel l'utilisateur attend la ressource demandee jusqu'a ce qu'elle soit liberee.
2. Proposez une variante ou la main est rendue a l'utilisateur avec un message d'erreur quand la ressource demandee est occupee.