

TD-TP de Système n° 7 : 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 définie 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 déjà un verrou bloquant, le processus est endormi jusqu'à ce qu'il n'y est plus de verrou incompatible ou que l'appel soit interrompu.
- { `F_GETLK` : Test d'existence d'un verrou incompatible avec le verrou donné. 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 décrit les possibilités de cohabitation des différents verrous :

- { `F_RDLCK` (verrou partagé) : plusieurs verrous de ce type peuvent cohabiter, c'est-à-dire avoir des portées 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.

Considérons 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);
    }
}

```

Quel est l'effet 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 créer un agenda pour un groupe de personnes. Pour cela, on dispose :

- { d'un type struct date qui décrit une date,
- { d'une fonction struct date *string2date(const char *) qui convertit une chaîne de caractères au format "jj.mm.aaaa" en un pointeur sur struct date (pointeur NULL si la chaîne n'est pas au bon format ou ne correspond pas à une date existante),
- { d'une fonction char *date2string(struct date) qui réalise l'opération inverse.

Les fonctions qui restent à implémenter sont :

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

Pour simplifier, on supposera qu'on ne peut noter qu'un événement par jour, et que l'événement prévu à la date jj.mm.aaaa est décrit dans le fichier \$AGENDA/jj.mm.aaaa, ou \$AGENDA est une variable d'environnement. Pour que l'utilisation de cet agenda se fasse sans heurt, il faut évidemment que personne ne puisse écrire dans un fichier que quelqu'un consulte et que personne ne puisse consulter un fichier sur lequel on est en train d'écrire...

1. Proposez un programme avec lequel l'utilisateur attend la ressource demandée jusqu'à ce qu'elle soit libérée.
2. Proposez une variante où la main est rendue à l'utilisateur avec un message d'erreur quand la ressource demandée est occupée.