VANSON Nathan

TP4 Systèmes Informatiques Rapport :

Introduction / Objectifs :

Le but de ce TP est de pouvoir créer un verrou d'enregistrement interactif. En effet, sous Linux, il existe deux fonctions de verouillage que sont :

- flock : permet d'ajouter des verrous de manière "facultative" (par défaut), et il peut également, s'il le souhaite, ajouter un verrou obligatoire. Il peut donc verouiller une fichier ou une partie d'un fichier. (`man flock` pour le manuel de "flock", section 1 à 2)
- fcntl : généralement utiliser pour verouiller une partie d'un fichier (c'est-à-dire le "verrouillage d'enregistrement"); utilisé pour synchroniser deux processus visitant la même partie d'un fichier. (`man fcntl` pour le manuel de "fcntl", section 1 à 2)

Ainsi, nous devons verrouiller / déverrouiller une partie d'un fichier de manière interactive. On a du également faire en sorte que les utilisateurs puissent verrouiller et déverrouiller une partie spécifiquede fichier, obtenir les informations de verrouillage, ainsi que le choix du type de verrouillage et définir le décalage pour ce dernier.

Programme :

Parties TP :

Initialisation du Programme :

Avant de pouvoir réellement s'attaquer au programme, on commence par définir la structure ainsi que les variables nécessaire à son fonctionnement :

- *Ligne 18*, notre programme doit être capable de vérifier correctement le nombre d'argument, d'ou le :

```
if (argc != 2)
```

et il nous sort un message d'erreur dans le cas contraire.

- On pense également à initialiser la variable (pour la suite du programme):

```
char input_whence = 's';
```

- *Ligne 32-36* : Ouverture du fichier et traitement du cas ou ce dernier ne peut pas s'ouvrir (il nous renvoie alors un message d'erreur) par l'intermédiaire de :

```
int fd = open(argv[1], 0_RDWR);
```

- C'est donc à partir de la *Ligne 38* que l'on commence à boucler sur le programme. On commence premièrement par lire l'entrée de l'utilisateur (récupérer l'input) et ensuite la traiter avec `fgets()` :

```
fgets(input, sizeof(input), stdin);
```

On compare en suite avec les différents possibilités de l'input (voir dans la partie Utilisation du Programme).

- On sépare ensuite l'input, pour les utiliser dans la structure définie au début :

```
scanf(input, "%c %c %d %d %c", &input_cmd, &input_type, &input_start,
&input_lenght, &input_whence);
```

Conversion des commandes en flag, Locking :

C'est dans cette partie qu'on va pouvoir effectuer le verrouillage et/ou le déverrouillage d'un verrou.

- On a 3 `switch case`, nous permettant ainsi de comparer chacune des valeurs dans la structure `flock`. Il y a :

-> `cmd` avec : `F_GETLK` (obtenir des informations sur un verrou, puis retourne); `F_SETLK` (poser ou enlever un verrou sur une partie d'un fichier) ;et `F_SETLKW` (poser ou enlever un verrou, puis attend en cas de conflit qui se produit).

-> `type` avec : `F_WRLCK` (Un seul et unique verrou peut être posé sur un fichier); `F_RDLCK` (On peut poser plusieurs verrous vu qu'il s'agit d'un verrou partagé); et `F_UNLCK` (Permet de débloquer un verrou existant)

-> `whence`

- les inputs `start` et `length` n'ont pas besoin d'être convertis car ils se trouvent déjà dans le bon format.

- Afin de poser ou enlever un verrou, il est nécessaire d'utiliser l'appel système à la *Ligne 118* qui est celui-ci :

```
int status = fcntl(fd, cmd, &fl);
```

Gestion des Erreurs :

C'est donc à la *ligne 118* du programme que commence la partie de la gestion des erreurs. L'objectif dans cette section est de faire en sorte que dès qu'une erreur se produit, alors se dernière va nous spécifier son type et faire en sorte de savoir quelles erreurs ont été produit par l'intermédiaire du module `errno.h`. Ainsi, on va contrôler les erreurs et les sorties en cas de réussite.

- On effectue les traitements d'erreurs -> `cmd` pour : `F_GETLK`, puis pour `F_SETLK` et `F_SETLKW`.md
- Les types d'erreurs pour ce programe sont :

-> `EINVAL`: Nous dit que l'argument pris est invalide

-> `EACCES` ou `EAGAIN`: Nous dit qu'il est impossible de pouvoir accéder au verrou (voir la partie Questions).

Parties Questions :

- Que se passera-t-il si nous déverrouillons un fichier (ou une partie du fichier) qui n'est pas verrouillé ?

-> Si on tente de déverrouiller une fichier (ou une partie du fichier choisi) qui n'a pas été verrouillé, alors il ne va rien se produire, car il n'y a aucune interaction entre lui et le verrou vu que, par définition, un fichier est considéré comme un fichier déjà non verrouillé.

- Que se passera-t-il si nous mettons un nouveau verrou sur un section déjà verrouillée ? Le type de verrou changera-t-il le résultat ? Expliquer dans la situation avec le même processus et avec 2 processus différents.

-> Premièrement, dans une situation ou on a le même processus, alors lorsque l'on met un nouveau verrou sur une section déjà verrouillé, alors le verrou placé sur cette section sera mis à jour (update).

-> Deuxièment, dans une situation ou on a deux processus différents, alors on doit traiter 3 cas :

- 1er cas -> Le *verrou paratagé* `F_RDLCK` : Ce verrou va permettre à plusieurs processus de placer un verrou sur la même section (sur la même partie du fichier selectionné).
- 2ème cas -> Le *verrou exclusif* `F_WRLCK` : Alors le nouveau processus aura un message d'erreur, et ne pourra donc pas pouvoir placer un verrou (un verrou exclusif `F_WRLCK` sur une section ne peut être placé que si aucun autre verrou (partagé ou exclusif) ne se trouve sur la section correspondate).
- 3ème cas -> Le *verrou de type* `F_SETLKW`: En consultant le manuel, on voit que par rapport à notre programme, ce dernier va tenter de placer un verrou, puis attendra si un conflit ce produit. De plus, si un signal est capté pendant l'attente, alors l'appel est interrompu et (après que le gestionnaire de signal soit revenu) revient immédiatement (avec la valeur de retour -1 et errno défini à EINTR).

Uitilisation du Programme :

En utilisant un fichier texte ou autre, on utilise un terminal pour se familiariser dans les débuts avec le programme :

```
./verrou [file name]
```

Ainsi l'utilisateur aura le choix entre :

- `?` : Demander de l'aide sur le fonctionnement du programme
- `exit` : Quitter le programme
- effectuer ce qu'il a à faire pour utiliser le programme

On peut également s'amuser à tester avec 2 processus en simulant les 2 questions au dessus.

FIN DU RAPPORT.