Université de Genève

-

Sciences Informatiques

Systèmes d'Exploitation - TP 04

Noah Munz - Gregory Sedykh

Novembre 2022

Contents

TP 04

2 Description

2.1 Contenu du projet

NB: Etant donné que nous travaillons sur WSL et que dessus, les locks posés par un processus ne sont repérable que par les enfants de ce processus n'avons pas pu tester notre programme à 100%. Il devrait, cependant, tout de même fonctionner correctement.

(1) Code

Le code de l'implémentation, qui se compile avec la *"target"* `all` du `Makefile`, se trouve dans:

- Le fichier `main.c` qui contient la fonction main qui permet de lancer le programme.
  Son fonctionnement sera expliqué en détail ci-après (l'executable qui lui fait directement appel est compilé, à la racine, sous le nom `./main`.)
- Le module `files.c`
  S'occupe des fichers, de la gestion des erreurs avec les fichiers (existence, type, taille etc.), des path des fichiers (absolute path, concatenation...).
- Le module `inp.c`
  Définit la struct `struct Inp` qui stock et gère l'input entré par l'utilisateur, pour ensuite l'utiliser dans `lock.c`.
- Le module `lock.c`
  Utilise l'entrée de l'utilisateur pour ensuite remplir un `struct flock` avec les données nécessaires, et effectuer l'appel correspondant à `fcntl` pour lock/unlock les fichers.
- Le module `optprsr.c`
  Parse les arguments (dans ce cas, le fichier) et l'input (ce que l'utilisateur veut faire).
- Le module `util.c`
  Contient quelques fonctions utiles pour la gestion des erreurs, pour obtenir des informations etc.

(2) Reste

Le reste du projet contient :

- Un dossier `res` qui contient un fichier test.txt, pour tester le fonctionnement du programme.
- Un dossier `out` qui contient le code compilé et les `.o` des différents modules. (Sauf pour `main.out` qui sera à la racine du projet)
- Et finalement, le Makefile.

2.2 Manuel

La commande `make init` permet de créer le dossier `out`, s'il n'existe pas, nécessaire pour stocker les `.o` et pour que les autres target du makefile s'exécutent correctement.
La commande `make all`, quant à elle, permet de compiler tous les fichiers/modules.

Comme dit précédemment, le makefile donne aussi d'autres possibilités de compilation (i.e. compiler seulement certaines modules situé dans `out/`)

On peut exécuter le programme d'une seule manière: `./main.out <file>`.
Où `<file>` est le fichier sur lequel opérer. Le programme affichera ensuite le PID du processus.
De même, le programme prendra ensuite une entrée utilisateur constitué de un de:

- `?`: Pour afficher un message d'aide sur comment utiliser le programme
- `Q`: quitter le programme.
- `cmd l_type length [whence]`: le format de l'entrée utilisateur pour utiliser le programme, i.e. comme affiché dans le help message, chacun prend un caractère ou un chiffre pour lock/unlock/obtenir un lock sur le fichier.

Pour plus de détails:

```
Format:  cmd    l_type    start    length    [whence]
   'cmd' -- 'g' (F_GETLK), 's' (F_SETLK), or 'w' (F_SETLKW)
   'l_type' -- 'r' (F_RDLCK), 'w' (F_WRLCK), or 'u' (F_UNLCK)
   'start' -- lock starting offset
   'length' -- number of bytes to lock
   'whence' -- 's' (SEEK_SET, default), 'c' (SEEK_CUR), or 'e' (SEEK_END)
```

5 Questions

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

  Si on essaie de déverrouiller un fichier qui n'est pas verrouillé, on obtiendra une erreur EINVAL (invalid operation) (Source: man 2 flock)

- *Que se passera-t-il si nous mettons un nouveau verrou sur une 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.*

  Lorsqu'on a 2 processus différents, Le type de verrou changera le résultat: si on a déjà un read lock sur un fichier, on peut quand même mettre des read locks dessus. Par contre, si un fichier a un write lock, on ne pourra mettre ni de read lock, ni de write lock supplémentaires.
  Pareil si on essai de mettre un write lock sur une portion de fichier protégé par un read lock. ont obtient l'erreur `Resource temporarily unavailable`.

  Par contre si on est dans le cas d'un seul processus, il n'y a pas ce problème. En effet, il ne peut s'auto-empêcher d'accéder / vérouiller le contenu qu'il a lui même protégé.