CHRISTOFOROU ANTHONY

RAPPORT

Variables

Notre but pour ce TP etait de creer un programme qui pose des verrous de maniere interactive pour l'utilisateur. On commence par initialiser toutes les varibales que l'on va utilisater tout le long du programme. On a tout d'abord input qui va etre la varibale qui va recevoir toute l'entree tapee par l'utilisateur sans formatage. On utilisera plus tard la fonction `sscanf` pour parser l'entree. On initialise input_w a 's' pour qu'on est une valeur par defaut plus tard. Finalement, EXIT_PROG et HELP sont ce qui va nous permettre de quitter le programme ou demander de l'aide.

Input

La premiere chose a faire va etre de creer un descriptif du fichier que l'on veut utiliser. Pour cela on utilise l'appel systeme de la ligne 23 `open(char* pathname, int flags, mode_t mode);`, qui va permettre de creer un descriptif de fichier qui est normalement le plus bas si il n'ent existe aucun donc 0. Ici on utilise le flag `O_RDWR` pour pouvoir l'utiliser en lectuer et ecriture. On test ensuite ce que retourne la fonction pour savoir si le fichier s'est bien ouvert, si oui on continue, si non en renvoie sur la sur la sortie standard d'erreur et on laisse errno nous dire ce qu'il s'est passee. On a un petit GUI (ligne 30/31) qui est cree assez rapidement, on print le `PID` (Processus ID) en utilisant `getpid()` et en affichant le tout. On utilise `fflush` a la ligne 32, tout simplement pour "nettoyer" la sortie. On a que fflush va forcer l'affichage mais avant ca il mettra nos caracteres en memoire. On arrive a la ligne 35 ou l'on va utiliser `fgets(char* str, int n, FILE* stream)` pour recuperer l'input d'un user. Avant d'en venir au vraies input, on peut utiliser `?` ou `exit`, qui respectivement affiche l'aide, et quitte le programme. On parse notre input avec `sscanf(const char* str, cons char* format, ...)` ou l'on va mettre les caracteres dans des variables uniques pour chaque.

Locking

On a trois switch/case. On va comparer chaque valeur pour pouvoir la mettre dans la structure `flock` qui va nous permettre de poser le lock sur le fichier. La structure `struct flock fl` contient `l_type` qui est lw type de lock que l'on voudra poser, `l_start` ou va t-il commencer, `l_len` la longueur du lock et enfin `l_whence` le placement du curseur dans le fichier.

Un fois que tout ca est fait on va utiliser la fonction `fcntl(int fd, int cmd, struct flock *fl)` a la ligne 98 qui va nous permettre de poser le lock.

Error Control

A partir de la ligne 100 on va controler les erreurs et les sortie en cas de reussite. On commence par quand cmd est `F_GETLK` si tout s'est bien passer et donc que la fonction a retourner 0 on va tester si `l_type` est egal a `F_UNLK`, si s'est le cas alors on peut acceder au lock ou il n'en a aucun. Dans le cas ou il n'est pas egal a `F_UNLK` on affiche un erreur d'acces et on regarde par quel lock il est bloquer, ou et par quel processus il est utiliser. Tout ca est enregistrer dans la structure dans ce cas. Dans le cas ou la fonction a retourner autre chose que 0 on va comparer errno avec une erreur existante, ici j'ai choisi `EINVAL` qui allait le mieux. On fait la meme

chose pour si cmd est egal a autre chose que `F_GETLK` et dans ce cas la on va juste retourner si le fichier a ete lock ou unlock. Pour errno jai choisi de tester sur `EACESS` et `EAGAIN`.

Results

En testant le programme on peut voir que on peut poser les lock que l'on veut sur un fichier mais du moment ou on va vouloir y acceder ou en poser en plus par un autre processus on va commencer a avoir des erreur. Si on test avec un lock exclusif on aura un erreur d'acces si un autre precessus essaie de recuprer ou poser un lock en plus. Dans le cas d'un lock partager on peut toujours en poser/recuperer des lock du meme type mais au moment ou l'on met un lock exclusif plus aucun processus ne peut acceder au fichier.