Concurrent Version System (CVS) Et Communication Interprocessus - TP2

Organisation des fichiers

Comme l'exigent les bonnes pratiques, ce travail dirrigé a été développé en utilisant le système de gestion de versions .git, afin de s'assurer d'une bonne cohérence et d'un développement plus sécurisé. Nous sommes partits de la version du TP1 disponible à l'adresse /root/Home/Ecemple_Meunier/Exemples/ exempleCVS-Concurrence-ExclusionMutuelle (sur le site FTP) (dmiftp.uqtr.ca).
L'organisation des fichiers de cette travail est la suivante :

gestionCVS_MAIN.C

Ce fichier est le point d'entrée de la version **serveur** du projet. Il contient la fonction main() ainsi que toutes les initialisations nécessaires, et l'appel à la fonction métier readTrans().

client.C

Ce fichier est le point d'entrée de la version **client** du projet. Il contient les initialisations nécéssaires ainsi que les fonctions permettant l'utilisation des ncurses de façon concurrentielle par la version cliente du projet.

gestionCVS.c

Ce fichier contient des utilitaires entrant dans la gestion concurrente de version, à savoir les fonctions permettant de charger, d'enregistrer des versions, et d'excécuter du code.

gestionListeChaineeCVS.c

Ce fichier contient les utilitaires et toutes les fonctions nécéssaires à la gestion concurrente de version, à savoir les fonctions permettant d'enregistrer une ligne de code (un commit), de faire des modifications, etc.

gestionListeChanneeCVS.h

Ce fichier d'entête contient les déclaration des fonctions du projet, ainsi que les structures de données utilisées dans le projet.

Makefile

Ce fichier permet de compiler facilement le code dans la console d'un environnement UNIX en tapant la commande **make.** Il faut cependant noter que ce fichier est configuré pour le compilateur gcc.

tmp/

Ce dossier est vide à la base, mais est sensé contenir les fichiers pipe qu'utiliseront les versions clientes et serveur du projet.

Fonctionnement du code

Compilation et exécution

Pour compiler le code, il suffit de se déplacer dans le dossier contenant les fichiers précités et de saisir la commande **make**. Deux exécutables seront générés :

1. **gestionCVS_MAIN :** Cet exécutable représente la version serveur du projet qu'il faudrait lancer en premier en tapant la commande **./gestionCVS_MAIN** dans un terminal. Nous reccommandons cependant de taper la commande **make clean** avant chaque lancement de cette version cliente afin de s'assurer que les fichiers non nécessaires (fichiers objets .o et anciens fichiers pipes) ont été supprimés et ne gèneront pas l'exécution du programme de serveur. Une fois lancé, un message d'acceuil s'affichera à l'écran, montrant que le serveur est bien lancé et attend des connexions entrantes.
2. **client** : Cet exécutable représente quant à lui la version cliente du projet. Pour le lancer, il suffit de taper la commande **./client** dans un terminal. Bien évidemment, cela va de soi que le serveur (expliqué précédemment soit d'abord lancé). Une fois lancé, une fenêtre intitulée sobrement "Transmission" vous invitera à entrer vos commandes. Les commandes doivent respecter le format décrit dans l'énoncé du présent tavrail dirrigé.

Remarque : Il se peut qu'au lancement du la version cliente, une seule fenêtre Transmission s'affiche. Pas de panique : En fait, les fenêtres transmission et réception sont threadées avec une exclusion mutuelle à l'affichage dans le terminal. Or, la fenêtre Transmission affiche un prompt avec la fonction **wgetstr** qui est une fonction bloquante. De ce fait, c'est après avoir appuyé la touche "entrée" que l'exclusion mutuelle donnera le "feu vert" à la fonction chargée d'afficher la fenêtre Réception.

Implémentation des threads

Les threads ont été implémentés en utilisant la fonction pthread_create() de la librairie **pthread.h.** Cette fonction est appelée à l'exécution de chaque transaction dans la fonction readTrans(), conformément aux suggestions de monsieur Meunier. A la fin de cette même

fonction, la fonction pthread_join() est appelée afin de s'assurer que la fonction attend la fin de l'exécution des thread avant de se terminer.

Implémentation de l'exclusion mutuelle

L'exclusion mutuelle a été effectuée en utilisant les sémaphores, conformément aux suggestions de monsieur Meunier. L'exclusion mutuelle concerne les fonctions de modifications de listes (ajout comme modification), et les fonctions faisant appel à l'affichage à la console. Les fonctions d'affichage côté serveur (Listage et exécutions) sont envoyée au client via son pipe pour être affichées.