Compte rendu - TP FTP

Hussein Elfakharany - Fabien Da Costa Barros

April 11, 2021

1 Introduction

Ce TP consiste à coder un protocole de transfert de fichiers (FTP) entre un serveur et des clients. Les clients envoient des requêtes au serveur pour demander des fichiers. Dans le cas où le client a entré la bonne commande et a demandé un fichier existant chez le serveur, respecant les permissions, le serveur envoie une copie du fichier demandé. Par ailleurs, le serveur s'occupe de gerer toutes sortes d'erreurs.

2 Paquetages

Il existe 4 paquetages (send, receive, utils, ftp) qui construisent le programme serveur ftpserver et le programme du client ftpclient.

Le paquetage ftp contient la fonction principale ftp appelée par le serveur. Cette fonction à son tour appelle les sous fonctions décrites ci dessous pour construire la squelette du serveur FTP.

Le paquetage send contient les fonctions qui s'occupent de l'envoie de soit les commandes par le client vers le serveur soit les fichiers du serveur vers le client. Le serveur renvoit un code de retour indiquant si le transfert (grace à la fonction envoieFichier) s'est passé sans erreurs ou pas.

Le paquetage receive contient les fonctions qui reçoivent soit les commandes de la part du client soit le fichier transferé de la part du serveur. La fonction lireMessage lit une chaine de caractères representant une commande à l'aide de la fonction rioreadline de la bibliothèque RIO. En revanche, la fonction lire recupère dans un buffer à l'aide du socket le contenu du fichier.

Le paquetage utils contient des fonctions qu'on a jugé utiles pour factoriser notre code et eviter les répétions de code. La fonction getCommand est une fonction appelée après la fonction lireMessage et qui recupère la commande sans le dernier caractère \n. La fonction afficher_code affiche un code en cas d'erreur chez le client. 0 pour fichier vide, 1 pour commande inconnue, 3 pour fichier manquant.

3 Principales fonctionnalités

Notre programme est une version simplifiée du vrai serveur FTP ne gérant pas toutes les commandes. On a commencé par réadapter le code du TP Serveur Echo pour en faire un serveur FTP simple qui accepte plusieurs des requetes de plusieurs clients en parallèle. Ensuite, on a ameiloré le programme pour qu'il reçoit une commande *get* suivie d'un nom de fichier et qui renvoit au client le fichier peut importe son extension qui les sauvegarde dans un répértoire local. On a implementé aussi la commande *bye* pour terminer la connexion entre le serveur et le client. Cette commande est entrée pour suspendre les plusieurs demandes de fichier par client.

A part la fonction get et bye, on a implementé d'autres fonctionnalités. La commande *cnt* suivie d'un nom de fichier continue le transfert d'un fichier en cas de panne coté client. Si le fichier demandé n'existe pas chez le client, la commande imite la fonction get et recupère tout le fichier de chez le serveur, sinon elle complète le transfert du fichier jusqu'à ce que la taille du fichier dans le repertoire du client soit égale à la taille du fichier chez le serveur.

Le programme aussi s'occupe bien de terminer le serveur et envoie bien les signaux SIGINT et SIGTERM aux fils pour les tuer. De même, le serveur reconnait la mort de ses fils avec un wait pour eviter leur transformation en zombie.

Il est aussi important de mentionner que le transfert de fichier se fait en découpage du fichier en plusieurs blocs. En effet, chaque bloc contient au maximum 8192 octets de données.

4 Protocole d'échange

Le protocole d'échange suit cette algorithme. On peut le voir dans la fonction ftp du paquetage éponyme ou dans ftp client. Du point de vue client cela se resume en ces différentes étapes.

- ◇ Envoie de la commande par le client
- ◇ Si la commande est valide
  - Reception de la taille du fichier
  - Si la taille vaut 0
    - ∗ Récupérer le code erreur et l'afficher
    - ∗ Recommencer du début
  - Sinon si la commande est cnt
    - ∗ Envoyer la taille déjà reçu
  - Récupérer la fin (cnt) ou tout (get) le fichier
  - Le stcoker dans un fichier
- ◇ Sinon redemander une commande à l'utilisateur

5 Tests

- Get des fichiers différents types (zip,jpg,pdf,mp4) avec des tailles variables voir dans le repertoire ServerFTP2/Test. On a utilisé la commande diff -s sur le fichier dans les deux répertoires pour les comparer: TEST VALIDE

- Test de la commande get et cnt get text.txt Ajouter du text sur la version serveur du fichier cnt text.txt diff -s sur le fichier dans les deux répertoires : TEST VALIDE

- Commande inconnue entrez "aezrz" Fichier manquant "get fichier n'existant pas sur le serveur": TEST VALIDE

- Test de la commande bye: TEST VALIDE

- Ctrl+C au moment d'entrer la commande avec le client ftpclient: Le serveur ferme la connexion et affiche un message d'erreur - se connecter avec ftpclientcrash1 qui simule un crash avant le transfert du fichier

- se connecter avec ftpclientcrash2 qui simule un crash après la reception d'un paquet du fichier, à utiliser avec un fichier de taille superieure à 8192 octets

6 Procédure de test classique

- Supprimer tous les fichiers restants des precedents tests
- Compiler *make*
- Copier les executables dans deux répertoires differents
- Lancer le serveur en bg depuis une autre fenetre dans ce second repertoire
- Lancer le client depuis le premier repertoire
- Lancer les commandes ou les executables souhaiter
- Verifier si le résultat obtenu est celui attendu
- cnt sur un fichier plein et sur un fichier dont on a pas reçu le début

7 Répartition du travail

Etant donné les circonstances, nous n'avons pas pu travailler ensemble physiquement fréquemment. Nous avons dû séparer de manière très précises les differentes parties du sujet et la collaboration n'a pu être qu'étroite. Cependant les échanges en lignes était assez fréquent et en cas de blocage nous avons pu discuter des implémentations en cas de blocage sur une fonction.

8 Conclusion

Pour la partie 3, nous n'avons pas eu le temps d'implémenter notre solution qui consistait a :

- ◇ Créer les esclaves (avec le même fonctionnement qu'à l'étape 2)
- ◇ Créer le maitre et lui donner en arguments le nom des 4 serveurs

- ◇ Se connecter et se déconnecter à tous les serveurs pour vérifier leurs existence et les stocker dans un tableau
- ◇ A chaque demande par un client vérifier par une connexion succinctes que le serveur esclaves que l'on veut attribuer est toujours disponibles
- ◇ Deconnecter le maitre et l'esclave
- ◇ Envoyer l'IP du serveur esclave attribué au client
- ◇ Connecter le client et l'esclave et reprendre le fonctionnement de la partie 2

Nous aurions voulu l'implémenter mais il nous a manqué un peu de temps sur la fin. Nous avons préférés faire la partie 2 du mieux que l'on pouvait plutôt que de faire rapidement la partie 3, comme conseillé.