Compte Rendu

Le but de ce TP est de réaliser un Serveur et un client pour le transfert de fichiers.

Pour se faire, nous avons décidé d'implémenter un serveur TCP qui est en lien avec un client. Cela permet d'avoir une communication entre le serveur et le client et donc d'avoir des réponses en cas de messages envoyés ou dans ce cas de commandes effectuées côté client.

Nous avons donc d'abord implémenté le code serveur et le code client afin qu'il puisse y avoir communications entre les deux puis nous l'avons modifié en implémentant les commandes demandées dans le TP.

Pour les dossiers et fichiers, il nous fallait donc une structure. Nous avons donc fait le choix d'utiliser la bibliothèque *<dirent.h>* qui implémente des structures de dossier. Cela nous a facilité l'implémentation et nous a permis de nous concentrer sur le codage des commandes.

Le programme fonctionne de la manière suivante :

Tout d'abord, dans le code *serv.c* ainsi que dans le code *client.c* nous créons le socket et nous le vérifions. Nous affichons dans le terminal si la création du socket a été effectuée avec succès ou si elle a ratée.

Après cela, nous faisons une assignation d'IP et de ports à la structure *sockaddr_in* préalablement crée afin d'indiquer que c'est une adresse IPv4, le port et les adresses IP disponibles. Nous le faisons dans les deux codes *serv.c* et *client.c*.

Ensuite du côté serveur donc dans le code *serv.c* on effectue le bind du socket afin de lui affecter une adresse IP et un numéro de port.

Une fois effectué, on va lancer une écoute du côté serveur puis une connexion du côté client et pour finir une acceptation du côté serveur. Et si tout se passe bien le serveur et le client sont à présent connecté et le client peut donc rentrer des commandes.

Une fois le client connecté sur le serveur, on lance donc dans les deux codes la fonction *fonction(int sockfd)* qui lance la communication.

Des deux côtés, dans la fonction *fonction()* on va tout d'abord initialiser des variables, tableaux de caractères comme le *buffer* et un fichier **f*.

Après cette étape, on gère dès le début l'authentification.

Du côté serveur, on crée une liste d'utilisateur qui est un tableau à deux dimensions que l'on initialise avec la fonction *fctlisteUtil()*. Cette fonction permet de remplir le tableau avec des prénoms choisis dans cette même fonction.

Ensuite, on a une séquence d'échanges entre le client et le serveur grâce aux fonctions *write()* et *read()* afin de demander au client d'entrer son Prénom et de vérifier dans le serveur si la personne est autorisée à entrer des commandes. A réception du Prénom par le serveur, on le vérifie avec la fonction *compare()* qui utilise la fonction *strcmp()* pour comparer le Prénom reçu avec tout ceux de la liste crée. Si le prénom correspond à l'un d'entre eux alors on envoie au client qu'il est autorisé. Sinon on envoie au client un *« exit »* qui ferme le client et on ferme aussi le serveur.

A chaque *write()* ou *read()* on utilise le buffer pour lire le message à envoyer ou reçu. On remet donc entre chaque utilisation de ces fonctions le buffer à 0 en utilisant la fonction *bzero()* qui remet tous les octets de la variable entrée en paramètre à 0.

Une fois le client autorisé, toutes les commandes se font côté client.

Dans les deux codes on a donc une boucle *while(1)* qui permet aux deux codes d'être exécuté jusqu'à ce que le client rentre la commande *exit*.

Côté serveur, on a tout de suite un *read()* pour attendre une commande du client pendant que dans le client on a un while afin de rentrer une commande et de l'envoyer au serveur.

Pour les commandes *cd*, *ls*, liste et *pwd*, la saisie de la commande côté client est envoyé au code serveur qui la traite.

Côté serveur, **si la commande est cd**, on utilise tout d'abord la fonction *getNameFile()* qui nous permet d'obtenir le nom du dossier dans lequel on veut aller.

Dans le cas où le *cd* est suivi de .. ou de . , on fait en sorte soit d'afficher au client « cd fait » si l'on reste dans le même répertoire ou que l'on est revenu d'un répertoire en arrière ou alors si l'utilisateur veut revenir un répertoire en arrière mais que l'on est à la racine, on lui affichera « cd impossible ».

Dans le cas où le *cd* est suivi par un nom de dossier, on fait un test d'appartenance avec la fonction *testApart()* pour vérifier que le dossier dans lequel on veut aller fait bien parti de celui dans lequel on est actuellement. Si c'est possible on envoie au client « cd fait », sinon, « dossier impossible ».

A chaque opération on a une chaine de caractères *path* que l'on modifie pour mettre le chemin du répertoire dans lequel on est.

Pour finir, on affiche côté serveur le répertoire dans lequel on est.

Si la commande est *ls*, on utilise la fonction *listdir()* qui nous affiche tout le contenu du dossier dans lequel on est.

On le met donc dans le buffer pour l'envoyer au client qui l'affichera sur le terminal.

Si la commande est *liste*, on met dans le buffer la liste des commandes que l'on envoie au client pour l'afficher sur le terminal.

Si la commande est *pwd*, on met notre chaine de caractères *path* dans le buffer afin de l'envoyer au client qui l'affichera dans le terminal.

Si la commande est *put*, tout d'abord côté client on va définir deux chaines de caractères, une dans laquelle on va mettre le nom du fichier dans lequel mettre des données que l'on va trouver en utilisant la focniton *getNameFile()* et une autre dans laquelle on va mettre le chemin de destination de ce fichier.

Ensuite on ouvre ou test si le fichier est ouvrable en lecture ou pas et on envoie au serveur la commande rentrée par le client. Côté serveur, les mêmes opérations vont être effectuées afin de trouver le nom de fichier et son chemin dans les dossiers.

Le serveur va à partir de la va lire tout ce que le client va lui envoyer à partir du fichier ouvert jusqu'à ce que le client lui envoie le mot « fin ». A chaque mot reçu, le serveur va le placer dans le buffer avant de le mettre dans le fichier en question sur le serveur. Une fois terminé, le serveur va fermer le fichier et va mettre dans le buffer « ecriture succesfull » et va l'envoyer au client afin qu'il l'affiche.

Si la commande est *get*, on va faire les mêmes opérations côté serveur et côté client que dans le put jusqu'à ce qu'on ait le nom du fichier et le chemin.

Ensuite, ce sera l'inverse, le serveur tant qu'il y aura des mots dans le fichier les mettra dans le buffer et les enverra au client qui les lira et les mettra dans le fichier. Ce, jusqu'à ce que le serveur envoi le mot « fin » et que le client arrête la lecture et ferme son fichier.

Si les commandes sont *lls*, *lcd* ou *lpwd*, c'est le client qui va faire les opérations en local. Le code et les fonctions sont exactement les mêmes que dans le serveur mais copiées dans le code client.