Buisson Valentin

Rapport Projet Network

1) Description

Le but de ce projet est de développer un serveur et un client en C. Le serveur doit pouvoir accepter la connexion de plusieurs lient en UDP ou en TCP. Une fois le client connecté, celui-ci doit s'identifier (mot de passe et nom d'utilisateur). Ces informations doivent être stocker dans une base de données SQLite3. Après identification, le client doit pouvoir accéder à la liste des fichiers sur le serveur, télécharger/uploader des fichiers, accéder à la liste de tout les clients connectés au serveur, envoyer des messages publics ou priver.

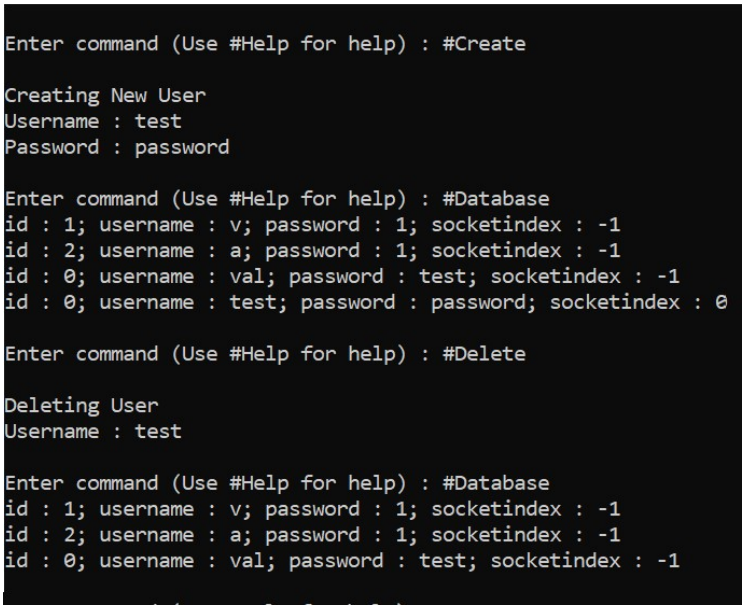
2) Fonctionnalités développées

Le Serveur :

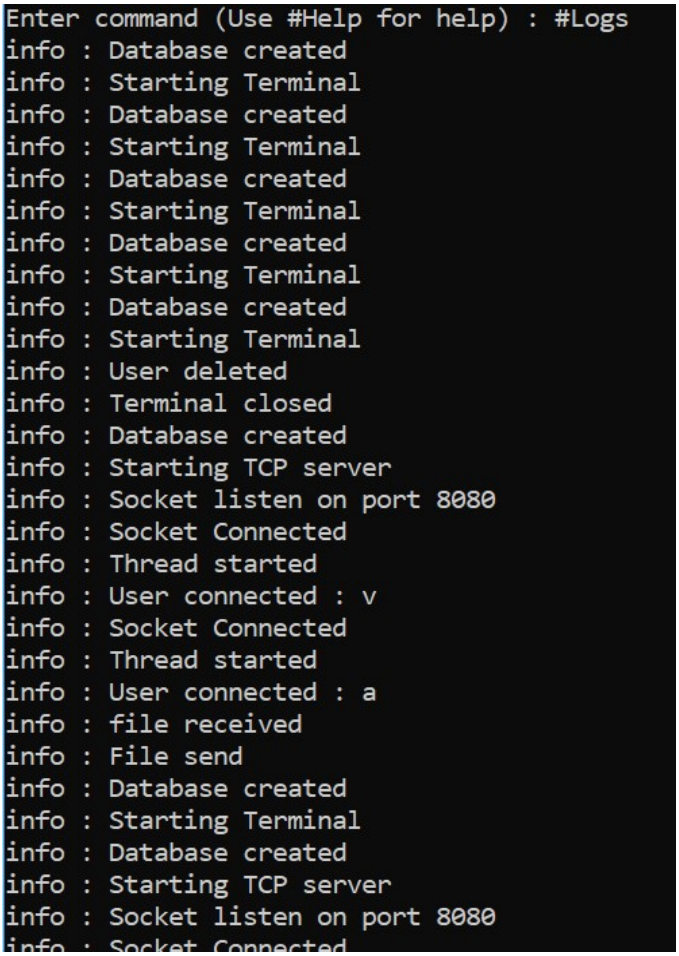
Après avoir lancé le serveur, l'utilisateur a 3 possibilités : démarrer le serveur en TCP, démarrer le serveur en UDP, ouvrir le terminal.

a) Le terminale permet à l'utilisateur d'accéder à la base de données. Une fois le terminal ouvert, l'utilisateur peut ajouter ou supprimer un utilisateur. Le terminal permet également d'afficher tous les utilisateurs dans la base de données. Il est également possible d'afficher tous les logs.

Exemple création et suppression d'un utilisateur :



Affichage des logs :



b) Serveur TCP

Le serveur TCP permet la connexion de plusieurs utilisateurs. Une fois qu'un utilisateur se connecte, une nouvelle thread est créée pour gérer les interaction de cet utilisateur.

Voici les différentes fonctionnalité sur le serveur.

-Identification :

Une fois connecté au serveur, le client doit se connecter pour avoir accès aux fonctionnalités, tant qu'il n'est pas connecté, il n'y a pas accès. Le serveur reçoit donc le nom d'utilisateur et le mot de passe du client, et vérifie dans la base de données s'il trouve une correspondance. Après avoir vérifié, le serveur renvoie -1 si aucune correspondance n'est trouvée ou 1 s'il en trouve une. Après connexion, le serveur stock les sockets dans un tableau, et l'index du tableau de la socket est stocké dans la base de données. Ainsi pour trouver la socket d'un utilisateur précis, il suffit de rechercher son nom dans la base de données.

-Liste des utilisateurs :

Pour obtenir la liste des utilisateurs connecté, il suffit de faire une requête à la base de données. En effet, tous les utilisateurs connectés on l'index du tableau des sockets supérieur ou égale à 0. A l'initialisation du serveur ou a la déconnexion d'un client, cette valeur est remise à -1. On récupère donc tous les nom d'utilisateurs dont les index sont supérieurs à 0.

-Envoie d'un message public :

Lorsqu'un client souhaite envoyer un message à tous les utilisateurs, il envoie le message au serveur, et en utilisant la même méthode qu'avant, le serveur cherche dans la base de données tous les index supérieurs à -1, puis récupère la socket dans le tableau et lui envoie un message.

-Envoie d'un message privé :

Le principe du message privé est à peu près similaire au message public, le client envoie le message à transmettre au serveur, puis il envoie le nom de l'utilisateur à qui envoyer le message. Le serveur va donc ensuite chercher dans la base de données, l'index du tableau correspondant au nom d'utilisateur reçu. Il récupère ensuite la socket dans le tableau et lui envoie le message.

-Liste des fichier sur le serveur :

Lorsqu'un fichier est enregistré sur le serveur, il est mis dans le répertoire courant. Pour obtenir la liste des fichier disponible il suffit de lister tout les fichier dans le répertoire courant et d'envoyer cette liste à l'utilisateur.

-Uploader un fichier :

Pour uploader un fichier, le client entre dans la console le nom du fichier qu'il souhaite transférer au serveur, une fois le nom reçu par le serveur, le fichier est ouvert en mode binaire et lu. Le client lit 1 caractère du fichier et si ce caractère n'est pas EOF, il est envoyé au serveur. Une fois que le caractère EOF est lu par le client, il envoie un message spéciale au serveur pour lui indiquer la fin de la transmission du serveur. Le serveur lui, dès qu'il reçoit le nom du fichier, le crée et l'ouvre en mode écriture binaire et chaque un charactère reçu, il l'écrit dans le fichier. Quand il reçoit le message de fin de transmission, il ferme le fichier.

-Télécharger un fichier

Pour télécharger un fichier cela fonctionne comme l'uploade mais dans l'autre sens. Le client choisi le nom du fichier qu'il veut télécharger, le serveur ouvre le fichier en mode binaire et envoie les caractères 1 par 1 au client jusqu'au message de fin. Le client lui écrit tout les caractère dans le fichier.

-Exit

Le client peut envoyer une commande exit qui fermera la connexion du client

c) Serveur UDP

Le serveur UDP n'offre pas autant de fonctionnalités que le serveur TCP. En effet je n'ai pas réussi à bien créer les thread me permettant de retrouver les différents utilisateur.

Le serveur UDP permet de lister les fichier et également de télécharger et uploader des fichiers.

Le client :

Quand le client est lancé, l'utilisateur a le choix entre le mode TCP ou UDP.

a) TCP

Quand l'utilisateur choisit le mode TCP, il doit tout d'abord s'identifier. Une fois qu'il est identifié, il peut commencer à utiliser les fonctionnalités. Une commande d'aide affiche les différentes commandes disponible.

```
Entrez votre commande(Use #Help for help) : #Help
#Exit : Close Connection
#ListU : List all users on server
#ListF : List all files on server
#TrfU : Upload a file to server
#TrfD : Download a file from server
#Private : Send a message to a user
#Public : Send a message to all users
```

Si l'utilisateur entre une commande qui n'existe pas, un message d'erreur est affiché.

Une fois que l'utilisateur s'est connecté au serveur, une thread est lancé. Cette thread s'occupe de recevoir les messages. Cela permet au client de recevoir un message à n'importe quel moment. La thread principale elle s'occupe de récupérer les input de l'utilisateur et d'envoyer des messages. Sans la thread pour écouter les messages, le client ne pourrait pas recevoir un message à n'importe quel moment puisse que les actions de récupérer un texte de l'utilisateur sont bloquantes.

b) UDP

Quand le client choisit le mode UDP, il a directement accès aux fonctionnalités, il n'a pas besoin de s'identifier, et il peut choisir parmi les différentes fonctionnalité : télécharger, uploader un fichier, avoir la liste des fichiers sur le serveurs.

Problèmes rencontré :

1) Saisi de texte : pour l'envoie d'un message, il faut que l'utilisateur saisisse un texte. La fonction gets() qui permet d'obtenir un string avec des espace ne fonctionne pas lorsque je l'utilise pour demander le message. Le programme s'exécute comme si la ligne n'existait pas. Pourtant, lorsque je test cette fonction au début du code, elle fonctionne.

2) Thread UDP : je n'ai pas réussi à implémenter le système de message en UDP car lorsque je lançais une thread, je ne pouvais pas envoyer de message avec la socket depuis cette thread. J'ai donc juste implémenté le système des fichiers.

Diagramme Serveur :

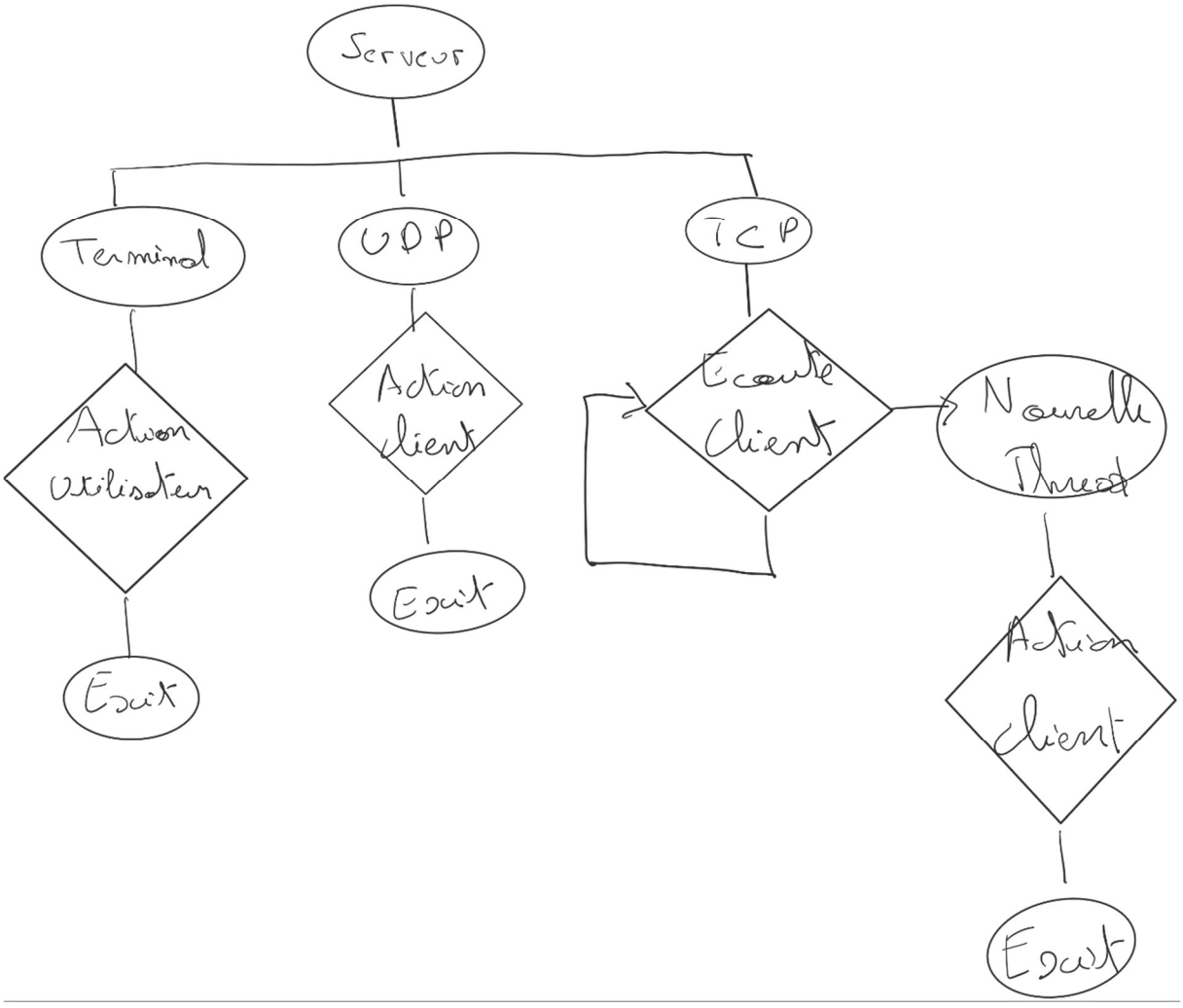


Diagramme client :