Télécom Physique Strasbourg

Informatique & Réseaux

Semestre 6

Rapport de projet RIO

Communications concurrentes

Thomas Triouleyre

Sofian Mareghni

Introduction

L'objectif du projet est de développement un logiciel d'échange de messages entre plusieurs stations. Pour commencer, nous avons réalisé un tchat pair à pair en UDP afin de se familiariser avec la notion de socket. Ensuite, nous sommes passé en TCP pour réaliser un tchat multi-utilisateur. Dans la dernière version disponible, les clients connectés peuvent communiquer entre eux de pair à pair (en privé) ou en broadcast par le biais d'un serveur, s'attribuer des noms lors de la connexion et communiquer entre eux grâce à ces noms et quitter le tchat en tapant « **/quit** » dans le terminal. Le serveur peut aussi être fermé proprement en tapant « **/quit** ».

Ce projet ne fait pas appel à la notion de thread. Il fait les liens entre clients et serveurs à l'aide des fonctions vus en cours : **select(), connect(), accept(), recv()…**

A propos du projet :

Le projet est composé de :

- Un fichier « **server.c** » pour les opérations du serveur
- Un fichier « **client.c** » pour les opérations des clients
- Un fichier « **utiles.h** » qui contient des fonctions utiles pour server.c et des macros
- Un fichier « **Makefile** »

Commande pour utiliser le projet :

Make clean

Make

Lancement de plusieurs terminaux (ctrl + alt + t) :

- Un pour le serveur : ./server num_port > 5000
- Le autres terminaux sont pour les clients : ./client addr_IP num_port
    - Le premier message est le pseudonyme du client dans le tchat
    - Si un client veut communiquer en broadcast, il rentre son message normalement
    - Si un client veut communiquer en privé : pseudonyme_recepteur :: message
- /quit pour fermer un client
- /quit pour fermer le serveur. S'il existe encore des clients connectés, ils les ferment avant de finir le programme.

Note :

Les quatre points « :: » servent de détecteur lors de l'analyse du message par le serveur pour savoir si la communication est en broadcast ou en privé. Le pseudonyme et « :: » doivent être collé ! Dans le cas contraire le serveur recherchera l'utilisateur «pseudo_» qui n'existe pas.

PARTIE CLIENT

Le but de cette partie de créer des entités « client » qui désire communiquer entre eux. Pour cela, il est nécessaire qu'ils puissent recevoir des messages et aussi en transmettre. Les transmissions doivent se faire en sollicitant le code server.c, qui en fonction des données reçu, effectuera un broadcast ou un tchat pair à pair.

Détails sur le fonctionnement :

Nous avons fait le choix de ne pas diviser ce programme en plusieurs sous-fonctions car nous trouvons que cela ne facilite en rien la lisibilité vue sa taille. Nous aurions juste des fonctions de 3 lignes ce qui est inutile. En revanche, il se distingue plusieurs blocs dans ce programme :

- 1er bloc : sert à définir toutes les variables et structures dont nous aurons besoin par la suite comme le descripteur de socket ou encore les buffers.
- 2ème bloc : sert à initialiser la structure d'adresse du destinataire. Par exemple, il met en place la famille d'adresse, l'adresse IPv4 du destinataire, son port ou encore la demande de connexion au serveur. Il sera aussi demandé au client de rentrer son pseudonyme dans ce bloc.
- 3ème bloc : ce bloc est constitué de la boucle while. Il permet d'effectuer toutes les opérations que doit faire le client comme la réception et l'envoie de messages ou encore de quitter le tchat.

Description de la démarche **/quit** :

Pour quitter proprement, le client envoie un **/quit** au serveur. Celui-ci prend note de sa demande de déconnexion, notifie les autres clients et lui renvoie un **/quit** pour lui dire qu'il peut se déconnecter. À la suite de la réception de la réponse **/quit** du serveur, le client ferme son socket.

PARTIE SERVEUR

Le but de cette partie de créer un serveur qui sera le maître d'orchestre du tchat. Celui-ci doit pouvoir accepter des connexions et fermer des sockets, recevoir des messages des clients, les identifier, les analyser et agir en fonction (broadcast ou tchat pair à pair). Nous avons aussi fait en sorte d'afficher quelques messages pour l'utilisateur (client) dans le but qu'ils puissent voir si un des membres du tchat c'est connecté ou déconnecté.

Détails sur le fonctionnement :

De même pour cette partie, nous avons fait le choix de ne pas diviser ce programme en plusieurs sous-fonctions car nous trouvons que cela ne facilite en rien la lisibilité. En revanche, nous avons créé un fichier

utile.h où nous avons mis des fonctions auxiliaires utiles comme la séparation d'une chaîne de caractères en fonction d'un séparateur ou une recherche de maximum dans une liste. A l'instar du programme client, celui-ci est aussi fragmenté en en sous bloc dans la fonction main :

- 1er bloc : sert à définir toutes les variables et structures dont nous aurons besoin par la suite comme le descripteur de socket, les buffers, quatre tableaux différents pour contenir par exemple les noms des utilisateurs, les sockets, un détecteur de 1er message pour savoir si c'est le pseudonyme ou non…
- 2ème bloc : sert à initialiser la structure d'adresse du destinataire. Par exemple, il met en place la famille d'adresse, l'adresse IPv4 du destinataire ou encore son port.
- 3ème bloc : ce bloc est constitué de la boucle while. Il permet d'effectuer toutes les opérations que doit faire le serveur comme la mise en place des connexions, la réception et l'envoie de messages, mise en lien entre le nom de l'utilisateur et son numéro de socket ou encore de quitter le tchat.

Description de la démarche **/quit** :

Lorsque l'on souhaite déconnecter le serveur, nous pouvons taper **/quit** dans le terminal. Celui-ci va alors demander la fermeture de tous les clients et mettre fin à son programme.

Conclusion

Lors de ce projet, nous avions pour but de réussir le tchat pair à pair. Nous avons fait le choix de simplifier notre code en évitant d'utiliser des mallocs ou encore des structures. En effet, nous voulions dans un premier temps réussir à maitriser les nouveaux utiles que nous avons vu dans le cours sans chercher à optimiser la mémoire et le code en général grâce à des structures. D'autant plus que vu le nombre de clients connectés, cela n'est pas dramatique pour la memoire.

Pour aller plus loin, avec plus de temps nous aurions pu adapter ce code à la mise en place d'un tchat entre différentes machines. La mise en relation entre adresse IP et pseudonyme aurait été fait grâce à la structure struct hostent et les fonction *gethostbyname(), *gethostbyaddr() et sethostbyname.