02/05/2019

Projet FAS

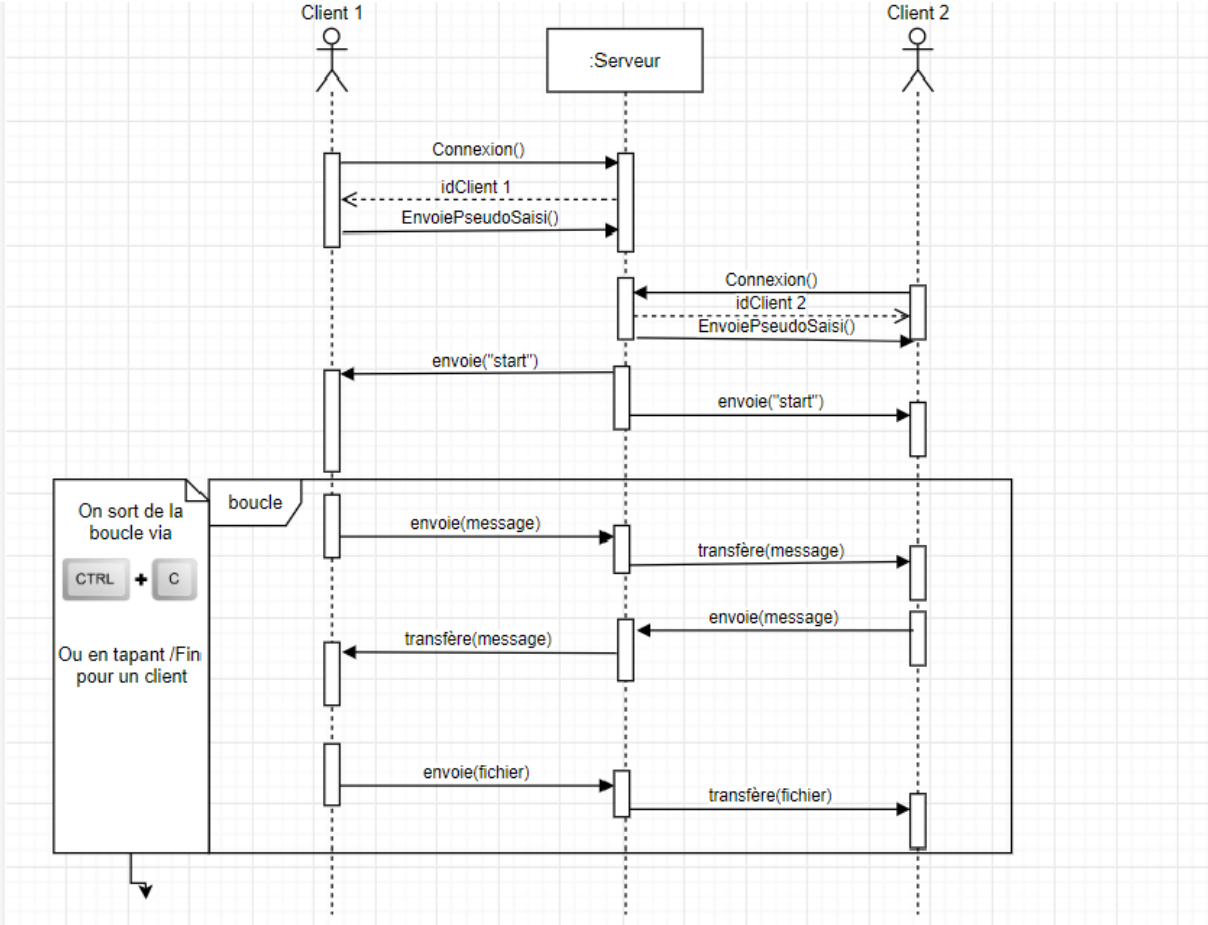
Itération 2

Lauren UNQUERA
Etienne SAIMOND

SOMMAIRE

I – Diagramme de séquence UML

Voici le diagramme de séquence décrivant le protocole de communication entre le(s) client(s) et le serveur.



II – Compilation et exécution du code

But : Créer une messagerie instantanée codée en C et basée sur le protocole Socket

Itération 3 : transfert de fichiers

À tout moment, un client peut décider d'envoyer un fichier, il saisit le mot « file ». Le client liste les fichiers d'un répertoire (dédié à l'application et dans lequel on doit ranger au préalable les fichiers pouvant être transférés) et demande à l'utilisateur de choisir, dans cette liste, un fichier à envoyer. Ceci est géré par un thread dédié. Comme ça, le client peut continuer à échanger des messages textuels. Un fichier reçu par le client est stocké dans un deuxième répertoire (dédié aux fichiers téléchargés). Un thread est dédié à la réception de fichiers.

Pour compiler les fichier :

A l'aide du bash, tapez la commande suivante :

make

(Nous avons réalisé un makefile afin de pouvoir compiler rapidement, efficacement et simplement).

Pour lancer le serveur :

A l'aide du bash, tapez la commande suivante :

bash launch_server.sh

Pour lancer le client :

A l'aide du bash, tapez la commande suivante :

bash launch_client.sh

Déroulement :

***Connexion**

1. Le Serveur attend une connexion.

2. Le Client X se connecte.

3. Le serveur envoie au Client X son numéro et attend de recevoir son pseudo

4. Le Client X saisi et envoie son pseudo au serveur.

5. Le Serveur envoie « start » au Client X

***Actions**

- Le Client X peut taper /help pour que le terminal affiche la liste des commandes possibles
- Le Client X peut taper /file pour transmettre un fichier au serveur, qui le transmettra à son tour à tous les autres clients
- Le Client X peut taper /fin pour mettre fin à la connexion
- Le Client X peut envoyer autant de message qu'il veut et tous les autres clients connectés réceptionneront son message.

Tout ceci est géré par des threads afin que les actions soient indépendantes les unes des autres

Arrêt du programme client :

1. Si un client envoie "fin" au serveur, il ferme ses port et le programme se termine.

2. Le serveur va lui fermer le socket du client concerné, et attendre la connexion d'un autre client.

3. Un nouveau client se connecte.

4. Le serveur envoie au nouveau client le numéro du client précedemment deconnecté.

5. Le serveur envoie au nouveau client "emi" (Mode reception).

6. Le serveur envoie aussi au nouveau client le dernier message envoyé au client precedemment deconnecté.

7. Le client va alors entrer dans son fonctionnement normal.

Fermeture du serveur :

1. Si le serveur est fermé via CTRL+C, il envoie aux deux client "exit" et ferme ses socket.

2. Les deux client à la reception du message "exit" vont fermer leurs socket

Sources :

- [Guide pour la programmation réseaux de Beej's](http://vidalc.chez.com/lf/socket.html)

- [Les sockets en C de developpez](https://broux.developpez.com/articles/c/sockets/#L3-2-1-c)

III – Difficultés du projet

1) Difficultés rencontrées

Une des difficultés qu'on a rencontrées a été sur l'ouverture d'un autre terminal pour le transfert de fichier. En effet, les utilisateurs de windows, en tout cas dans ce groupe, quand bien même étaient-ils dotés d'Ubuntu ne pouvaient pas exécuter cette fonction sur leur machine. La solution a été de privilégier le travail en salle de TP même si celles-ci étaient inaccessibles pendant les vacances.

Un autre problème que nous avons rencontré a été un envoi trop rapide lors de l'envoi de fichier, ce qui faisait sauter certaines lignes. Une solution apportée a été d'ajouter 50 millisecondes d'attente après l'envoi du paquet.

2) Evolution

On pourrait envisager une amélioration qui serait de mettre le système en Peer-to-peer, ce qui ferait du serveur un simple annuaire.

IV – Répartition du travail et organisation

Concernant la charge de travail, nous nous sommes réparti le travail plutôt en termes de fonctionnalité voire de temps. Cela a été plus pratique comme nous n'étions pas forcément disponibles aux mêmes moments, notamment lors des vacances.

Néanmoins, afin d'être plus performant et d'améliorer au mieux notre travail, nous mettons en commun nos différentes production et nous apportons des modifications, lorsqu'elles sont discutées et validées, sur le travail de l'autre. En effet, n'ayant pas les mêmes capacités ou la même vision des choses, nous jugeons bon de relire le travail de l'autre afin de perfectionner le code.

De plus, avec la mise en place d'une communication régulière et d'une bonne entente, nous pouvons nous entraider assez aisément, et cela a été très utile, que ce soit d'un côté ou de l'autre.