20/04/2019

Projet FAS

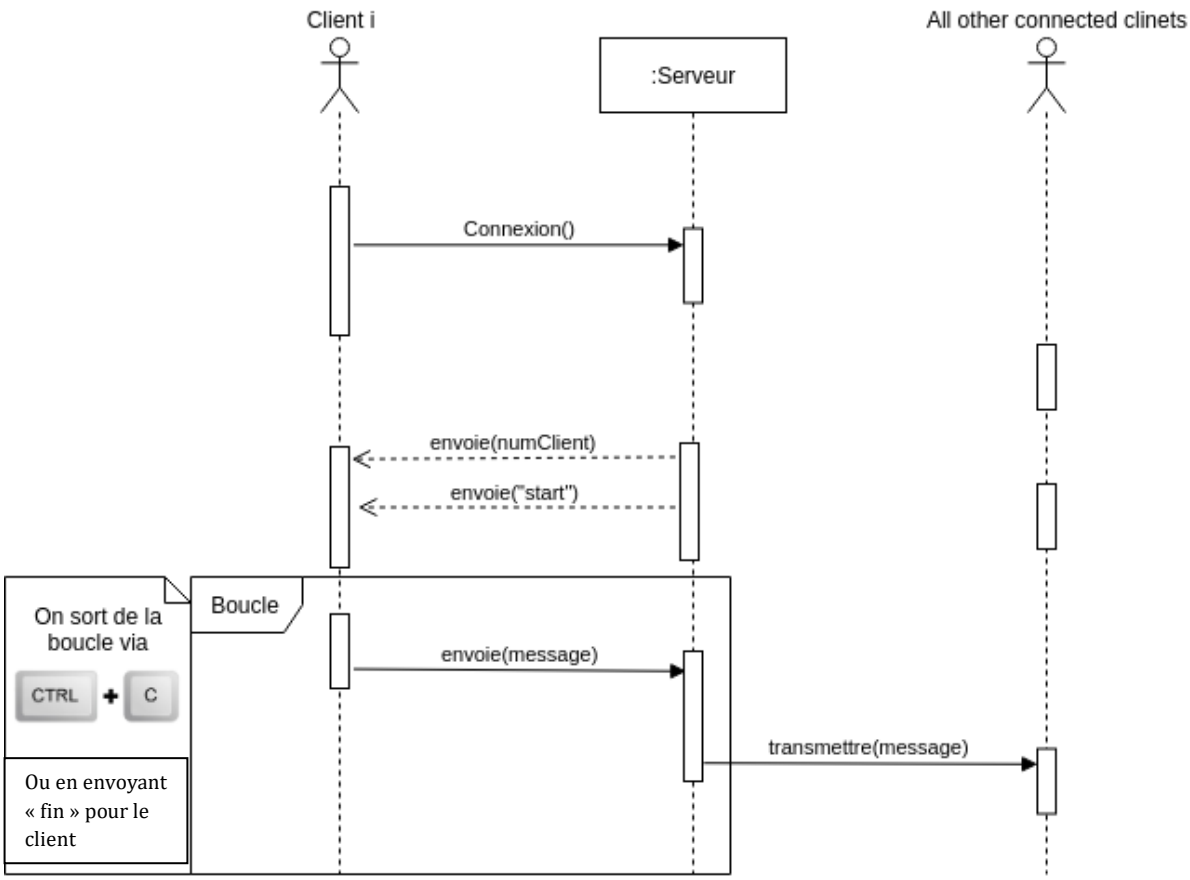
Itération 1

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 2 : 1 serveur et 1 client multi-threadés

Utiliser le multi-threading pour gérer l'envoi de messages dans n'importe quel ordre (on ne sera plus contraint par le fait que c'est le premier client qui se connecte qui envoie un message en premier et attend que l'autre lui écrive un message pour pouvoir envoyer un autre message)

programme client : 1 thread pour la saisie (avec fgets) et l'envoi du message au serveur et 1 autre thread pour la réception des messages du serveur et leur affichage (avec puts)

programme serveur : 1 thread pour relayer des messages du client 1 vers le client 2 et 1 autre pour relayer les messages du client 2 vers le client 1

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 :

1. Le Serveur attend une connexion.
2. Le Client 1 se connecte.
3. Le serveur envoie au Client 1 son numéro.
4. Le Client 1 attend de recevoir son role du serveur.
5. Le Serveur attend une autre connexion.
6. Le Client 2 se connecte.
7. Le serveur envoie au Client 2 son numéro.
8. Le Client 2 attend de recevoir son role du serveur.
9. Le Serveur envoie au Client 1 "emi" (Mode émission).
10. Le Serveur envoie au Client 2 "rec" (Mode réception).

*** Debut de boucle**

11. Le client 1 envoie son message au serveur et se met en mode reception.
12. Le Serveur le recoit et le retransmet au Client 2.
13. Le client 2 recoit le message du client 1.
14. Le client 2 envoie son message au serveur et se met en mode reception.
15. Le Serveur le recoit et le retransmet au Client 1.
16. Le client 1 recoit le message du client 2.

*** Fin de Boucle**

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 le passage de paramètre dans l'appel aux threads. Afin de surmonter cette difficulté, nous avons crée une structure nommée « arg_struct » qui est destinée à contenir ces paramètres.

L'autre difficulté qu'on pourrait mentionner est la réalisation de la 2ème version de cette itération avec la gestion de n clients. Afin de résoudre les problèmes liés à cela, nous avons utilisé des tableaux. Nous avons aussi créé un tableau de taille n qui contient un 0 si la personne correspondant à l'indice i est déconnectée, 1 sinon. De fait, lorsqu'on souhaite réaliser des traitements à grande échelle, nous regardons pour chaque indice si la personne est connectée, sinon nous ne faisons pas le traitement pour cet indice i.

2) Evolution

Tous les objectifs de l'itération 2 ont été atteints, on pourrait envisager une amélioration qui serait la création de différents salons, dont chacun pourrait contenir un nombre donné de clients. Ceci correspond à une des itérations qui sera réalisée plus tard.

IV – Répartition du travail et organisation

Concernant la charge de travail, nous voulions la répartir équitablement et de manière à ce qu'on soit le plus efficace possible. D'autre part nous voulions changer par rapport à l'itération 1 et faire en sorte qu'Etienne s'occupe en particulier du code pour la partie client et que Lauren s'occupe principalement du code pour la partie serveur.

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.