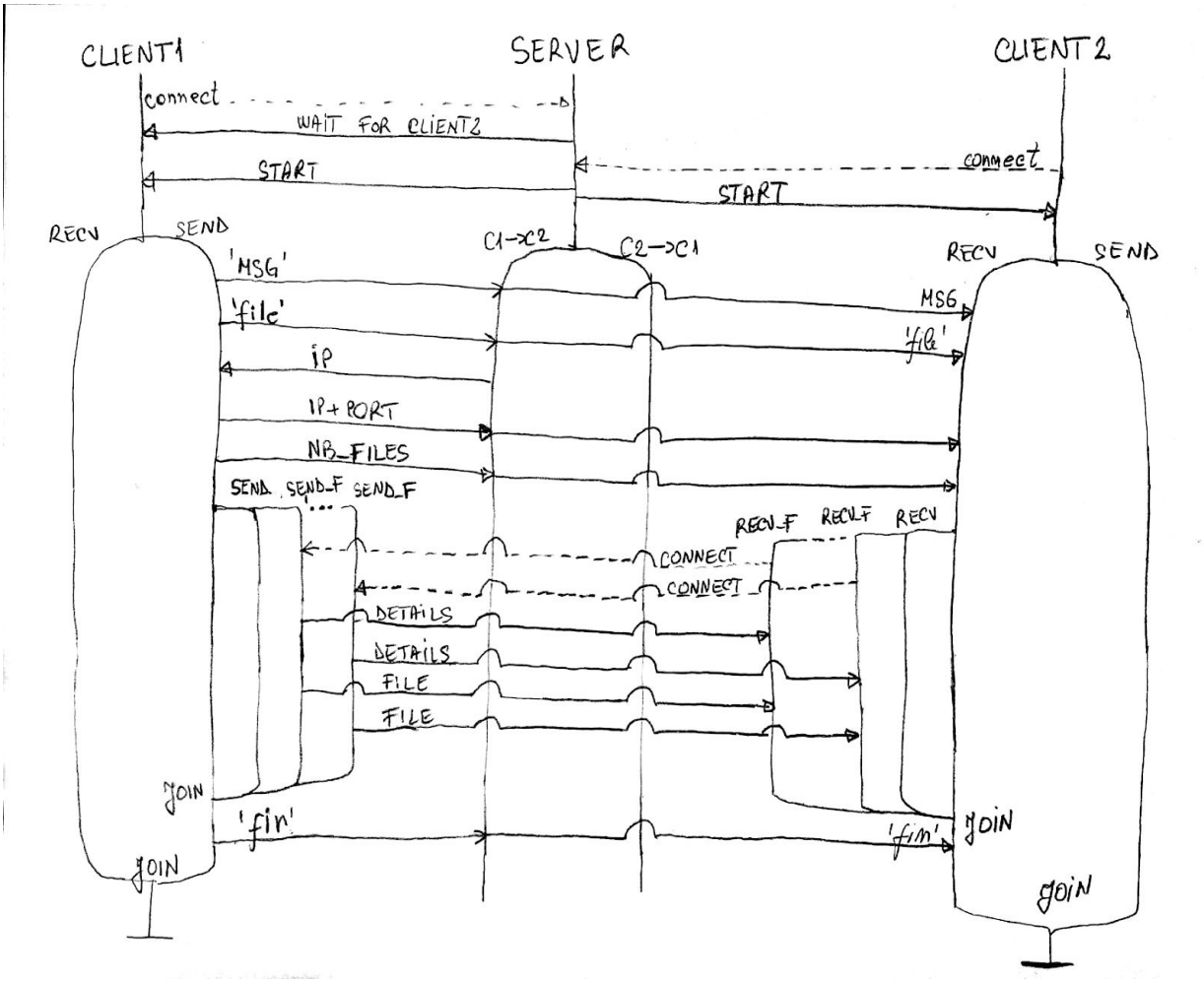
Marian Aldescu
Florine Pratlong

Projet FAR - Livrable 3

Application de messagerie instantanée

I) Protocol

V1



Dans la suite, nous utiliserons le terme ‘Client1’ à énoncer celui qui envoie les fichiers et ‘Client2’ celui qui reçoit les fichiers.

Le Client1 demande la connexion au serveur, puis le serveur accepte ou refuse et lui transmet la réponse en lui disant qu'il attend qu'au moins un autre client se connecte. Le client2 demande la connexion au serveur, le serveur accepte ou refus. Ensuite il prévient les clients que la communication peut démarrer et les deux clients peuvent s'envoyer des messages.

Chaque client crée 2 threads, 1 thread à réceptionner des messages et l’autre à envoyer des messages.

Quand un client, par exemple Client1, envoie le message ‘file’, le serveur lui envoie l'adresse IP et puis, le Client 1 envoie au Client2 l’adresse IP et le port, parce qu’il se connectera par une connection peer-to-peer au Client1.

Client1 envoie le nombre de fichiers à transmettre au Client2.

Le Client1 ouvre Nb_files+1 threads, 1 thread pour continuer à envoyer des messages et Nb_files à envoyer les fichiers, 1 thread pour 1 fichier.

Le Client2 ouvre Nb_files+1 threads, 1 thread pour continuer à receptioner des messages et Nb_files à recevoir les fichiers, 1 thread pour 1 fichier.

Chaque thread du Client2 se connecte à un thread du Client2.

Les threads du Client2 envoient les détails et puis le contenu des fichiers.

Quand le serveur a reçu le message ‘fin’, il l’envoie et puis fini les 2 threads. Le serveur attend des autres connections.

II)

L’architecture d’application:

Le program serveur est celui de l'étape précédente.

Le program client implémente les deux versions pour ce livrable.

V1 Si on envoie un seul fichier, on ne peut pas envoyer des autres jusqu'à la fin de la soumission du fichier.

V2 Si on veut envoyer plusieurs fichiers, on sera créé des threads, un thread pour un fichier.

Le program client est constitué par:

- 1 thread avant de se connecter au serveur
- 2 threads actives(1 à envoyer, 1 à recevoir des messages) après il se connecte au serveur, et 1 thread qui fait le ‘join’ des premiers deux
- 2 threads actives(1 à envoyer, 1 à recevoir des messages) et nb_fichiers threads à envoyer pour le Client1 et nb_fichiers threads à recevoir pour le Client2 après l’utilisateur introduce le mot ‘file’, et 1 thread qui fait le ‘join’ des premiers deux

On a utilisé l’alternative **‘peer-to-peer’** pour la soumission des fichiers

- le Client1 envoie les port et ip dont il attend des connections par le serveur au Client2
- le Client2 se connecte au Client1 et puis il reçoit par les plusieurs connections tcp les fichiers

Les ressources partagées par les threads sont déclarées comme variables globales.

On a eu besoin de synchronisation quand le Client1 envoie les port et IP au Client2, donc on a utilisé une semaphore à attendre la réception de l’IP du Serveur.

III) Nous avons rencontrés des difficultés parce que l’architecture de l’application c’est un peu complexe, donc on a eu besoin faire attention à tous les cas possibles.

IV) On a travaillé ensemble au client.

v) Compiler et exécuter le code

Nous avons écrit un fichier Makefile pour automatiser la compilation et l’execution.

Aussi, si on veut changer **l'adresse IP**, **le port** ou **le nombre de clients** du serveur, on modifie dans ce fichier les deux variables: **PORT, IP_SERVER, NB_CLIENTS**.

Pour compiler:

```
make
```

ou

```
make client_v_1_2
make server
```

Pour tester:

V1: On lance premierement un serveur avec:

```
make run_server
```

puis, on lance 2 clients avec:

```
make run_client_v_1_2
```

Pour effacer les executables:

```
make clean
```