Victor LEFEVRE
Bastien MALLET
3A INFO

Compte-rendu de projet Réseau

Introduction

Dans le cadre de ce projet, nous avons développé une application Java permettant de créer un système de chat en réseau, reposant sur une architecture client/serveur et utilisant des paquets datagrammes UDP. L'objectif principal de ce projet était de concevoir une solution de communication permettant l'échange de messages entre différents utilisateurs connectés sur un même réseau.

Utilisation

Pour utiliser le chat, il convient de définir préalablement certains paramètres. Pour le serveur, il faut définir le port sur lequel on se place avec la variable port_serveur.
Du côté du client, il faut donner l'adresse IP et le port du serveur.
Les clients et le serveur doivent se trouver sur le même réseau.
Pour un client se trouvant sur le même appareil que le serveur, il faut utiliser l'adresse IP localhost 127.0.0.1

On peut alors démarrer le serveur, puis le client. Au lancement du client, l'utilisateur doit choisir un pseudo.

Pour envoyer un message privé à uniquement un destinataire, le message doit contenir le pseudo du destinataire et le texte du message séparé par un "/", sous la forme :

pseudoDestinataire/ texte du message

Dans le cas contraire, le message est envoyé à tous les clients connectés actuellement au serveur.

Structure du projet

Classe Client

La classe Client instancie un DatagramSocket afin de pouvoir communiquer avec le serveur ainsi qu'un DatagramPacket afin de stocker et transmettre les données à envoyer. Elle instancie également un thread d'écoute.
Le premier message envoyé sert à la connexion au serveur en renseignant son pseudo. Les messages suivants sont également transmis au serveur.

Classe ClientReceiveProcess

Cette classe est nécessaire pour faire de l'exécution de tâches en parallèle. Elle implémente Runnable afin de lancer le processus dans un thread séparé. Cette classe écoute en continue le client et vérifie s'il reçoit des messages. Elle affiche les messages quand ils sont reçus.

Classe Server

La classe serveur écoute en continue si des messages lui sont envoyés, lorsque c'est le cas, elle crée un thread qui va s'occuper de la gestion du message.
La classe contient deux Map :

- Une map qui associe un port à un pseudo
- Une map qui associe un port à une ip (utile pour la communication entre plusieurs machines.

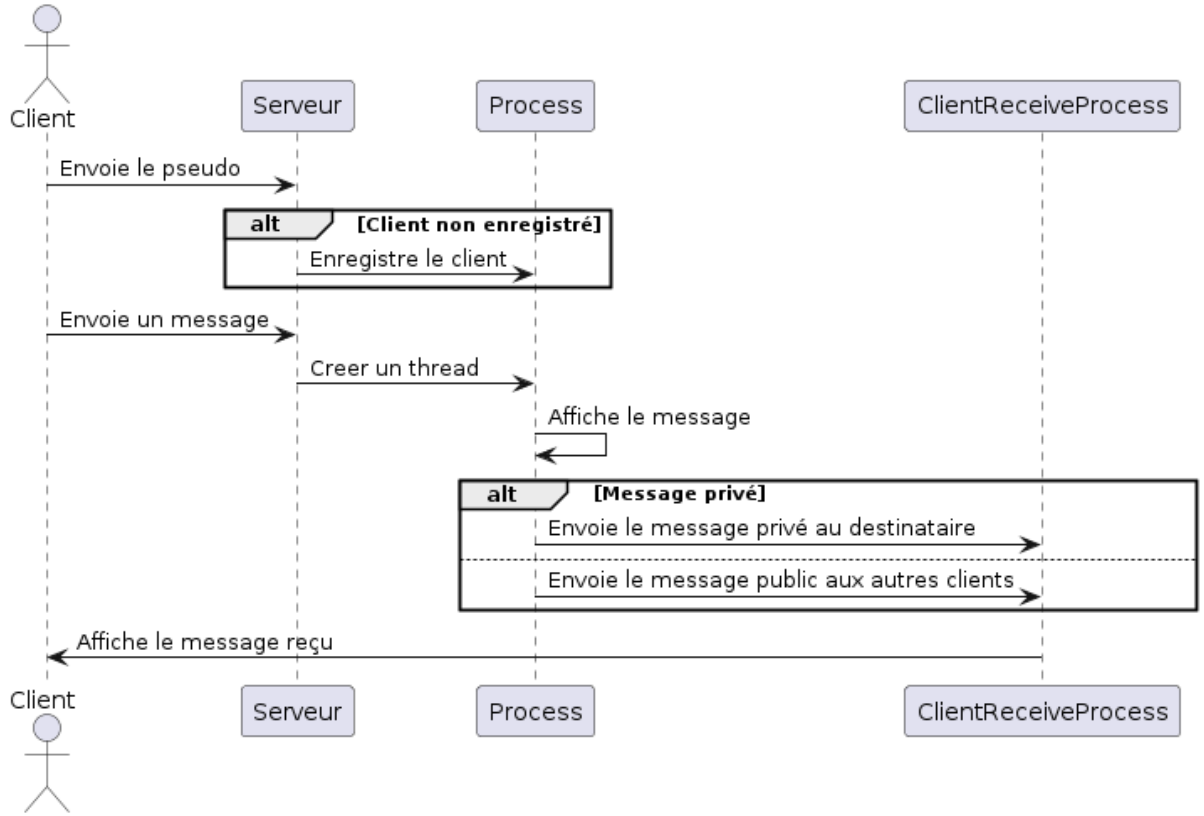
Classe Process

Cette classe ajoute à nos deux maps l'ip, le port et le pseudo de chaque client non connu. Elle affiche les messages envoyés et reçus par les clients du côté du serveur. Elle vérifie si le message contient un /.

Si le message contient un / la fonction message privé est exécutée : cette fonction parcourt tous les pseudo de la map port/pseudo afin de trouver le port correspondant au destinataire. Si le pseudo existe elle envoie le paquet au client trouvé sinon elle affiche un message d'erreur.

Sinon le message est message de groupe, alors tous les ports de la map sont parcourus et le message est envoyé à chaque client.

Diagramme de séquence



Fonctionnalités supplémentaires

Nous avons fait en sorte que notre ChatBot puisse fonctionner sur plusieurs machines différentes connectées à un même réseau.
Il faut lancer le serveur sur une machine, ainsi qu'aucun ou bien plusieurs clients, et lancer des clients sur les autres machines.
Pour ce faire, il faut être vigilant sur l'adresse ip à laquelle les clients envoient les Datagram Packets en spécifiant l'ip de la machine où le serveur est logé.

Amélioration

Pour parfaire ce projet, nous aurions pu ajouter/modifier certaines fonctionnalités, telles que :

- Créer un thread par client plutôt qu'un thread par message.
- Faire un système de déconnexion de client.
- Faire une discussion de groupe limité (pas avec tous les clients connectés).
- Afficher une liste de clients connectés afin de voir qui est disponible à la discussion.