Projet Protocoles des services Internet

12 décembre 2017

1 Architecture du serveur

Le serveur attend les connexions des clients sur le port 1027 comme spécifié dans le protocole. Le serveur accepte les connexions jusqu'à ce que le nombre de limite de connexions ouvertes soit atteint. Par défaut, on a fixé arbitrairement cette limite à 100 mais on a la possibilité de faire varier ce nombre en lançant le serveur avec un argument en ligne de commande (voir le fichier `README`).

Une fois la connexion entrante acceptée, le serveur lance un *thread* (voir la classe `server.ServerThread`) qui va traiter les requêtes envoyés par ce client. Le serveur peut donc (et heureusement) gérer plusieurs clients à la fois.

Pour stocker les annonces sur le serveur, on utilise une base de données *in-memory* (classe singleton `server.DB`). Cette base de données vise simplement à maintenir une association entre une adresse IP et un utilisateur (modélisé par la classe `common.User`). Cette base de données pouvant être utilisée de manière concurrente par les différents *threads* du serveur, on a pris soin d'en protéger les accès.

2 Architecture du client

Le programme client est composé d'une boucle interactive principale qui exécutent les commandes du client : se *logger*, lister les annonces, poster une annonce, supprimer une annonce, envoyer un message à un autre client, lister les messages reçus et se déconnecter.

La gestion des messages échangés entre client est faite de manière similaire à une boîte mail. La réception des messages provenant des autres clients est faite de manière asynchrone, c'est-à-dire qu'on a un *thread* en arrière plan qui s'occupe de gérer la réception des messages UDP des clients (voir la classe `Mailbox.Inbox`).

3 Analyse de sécurité

Le serveur est *a priori* toujours disponible... tant qu'il a suffisamment de mémoire puisqu'on utilise une base de données en mémoire.

Les clients sont traités de manière complètement équitable puisqu'on attribue un *thread* par client.

Une des gros points faibles de notre application est qu'on ne garantit rien du tout sur les messages échangés entre les clients que ce soit en termes de confidentialité ou même simplement sur le fait que les messages envoyés arrivent bien à destination.

Pour éviter des attaques de type déni de service sur le serveur, on limite le nombre de connexions acceptées par le serveur.