ÉCOLE POLYTECHNIQUE DE MONTRÉAL
Rapport de laboratoire 4, session automne 2017

Travail présenté à Mme Jackie Phung

Par
Christophe Bourque Bédard 1692224
Justine Pepin 1789244
gr. 4

DANS LE CADRE DU COURS DE RÉSEAUX INFORMATIQUES
INF3405
25 novembre 2017

Introduction :

Les réseaux informatiques ont comme but de permettre la communication efficace entre deux nœuds (ordinateurs, serveurs, périphériques, etc.) afin d'échanger de l'information. De nos jours, une application des réseaux très utilisée consiste en l'échange de messages écrits en temps réel avec une autre personne, c'est-à-dire le clavardage. Dans ce laboratoire, nous devons justement mettre au point un système de clavardage interactif à l'aide de Winsock2, une librairie permettant d'instancier des sockets via lesquels on peut communiquer. Notre système doit avoir un côté serveur, en charge de d'authentifier les clients et de diffuser leurs messages, ainsi qu'un côté client, en charge d'interpréter les entrées d'un utilisateur et de les transmettre au serveur sous certaines conditions. Notre système doit également disposer d'une base de données répertoriant les messages envoyés et les utilisateurs s'étant authentifiés.

Présentation des travaux :

Pour ce laboratoire, nous avons adopté une formule de base similaire à celle proposée dans le laboratoire précédent. Nous sommes partis des fichiers serveurs et client et nous avons étendu les fonctionnalités de base afin d'obtenir la solution complète requise dans ce laboratoire.

Du côté client, notre système est tout d'abord responsable d'instancier un socket et de l'utiliser afin d'envoyer le message d'authentification. Si on tente de se connecter sous le même nom qu'un autre utilisateur déjà connecté ou si on tente de se connecter sous un compte déjà existant mais avec le mauvais mot de passe, un message d'erreur est retourné et le programme client termine son exécution. Dans le cas où l'authentification est approuvée, c'est-à-dire qu'on se connecte sous un nouveau compte ou qu'on se connecte sous un compte déjà existant et avec le bon mot de passe, le client reçoit les quinze derniers messages écrits sur le chat (ou moins selon la longueur d'historique disponible).

Enfin, un thread secondaire est créé par le client pour s'occuper de la réception des messages, tandis que le thread principal s'en va effectuer une fonction chargée de l'envoi de messages. Afin de terminer l'exécution du client, l'utilisateur peut envoyer un message vide en appuyant sur la touche Entrée; le serveur saura que l'usager s'est déconnecté.

Du côté serveur, notre système instancie un socket serveur afin d'entretenir plusieurs connexions avec des sockets clients. Il tente tout d'abord de vérifier l'authenticité des usagers en regardant dans sa base de données .txt s'il y a un usager déjà existant sous ce nom et si le mot de passe fourni coïncide avec celui de la première authentification. La base de données est gérée par la classe DataBase et comporte deux fichiers : le fichier usagers.txt, qui répertorie tous les pseudos et leur mots de passe (chacun sur une ligne différente, le mot de passe suivant son pseudo), et le fichier données.txt, qui répertorie tous les messages envoyés depuis toujours. Le fichier usagers.txt sert uniquement à l'étape d'authentification, tandis que le fichier données.txt est appelé à chaque fois qu'un utilisateur authentifié écrit un message.

Le serveur, s'il invalide un usager, envoie un message de refus (Enum : AuthentificationRep) et supprime le socket relié à ce client. S'il valide l'authentification, il envoie un message d'acceptation ou de création (auquel cas une écriture dans la base de données usagers.txt se fera) et créée un thread chargé de surveiller le socket afin de diffuser les messages reçus de cet usager à tous les autres usagers connectés.

Le client comme le serveur utilisent des méthodes tirées de la classe ConnectionInfos, qui gère les ports et les adresses des hôtes, et de la classe Communications, qui gère le bon format des messages partant du client/serveur et se rendant au serveur/client. Cette classe décide par exemple quel sera le type de message (Enum : TypeMessage) qui partira et mettra ce type en guise d'en-tête suivi d'un séparateur. Ainsi, son interlocuteur saura comment réagir à ce nouveau message (authentifier un client, diffuser un message, annoncer à un usager qu'il n'a pas inscrit le bon mot de passe, etc.).

Difficultés rencontrées lors de l'exécution du TP :

La plus grande difficulté que nous avons rencontrée avec ce TP était l'élaboration de la base de données. Au début, nous voulions utiliser HDF5, qui est une librairie de gestion en C, C++, python, Java et Fortran pour une base de données extensible, rapide et à très grande capacité dans des fichiers binaires. Malheureusement, HDF5 n'est pas vraiment adaptée pour des strings de tailles variables et aussi était-il très difficile de faire fonctionner cette librairie correctement dans notre cas. Nous avons donc opté pour une solution plus facile, c'est-à-dire l'utilisation de fichiers .txt, qui ne nécessite aucun téléchargement et qui rend la vérification de la solution plus aisée car en un coup d'œil on peut voir si l'ajout a été fait correctement.

Critiques et Améliorations :

Dans ce TP, nous aurions aimé que certains éléments laissés à notre discrétion soient précisés, car ils font évidemment partie d'un système de clavardage efficace. Par exemple, il n'est pas écrit dans l'énoncé qu'un même usager ne doit pas être capable de s'authentifier deux fois (être disponible dans 2 sockets différents à la fois). Pourtant, le clavardage ne devrait pas permettre qu'une personne s'écrive à elle-même, il s'agit d'une situation absurde. Un autre exemple serait l'entrée de mots de passe non camouflée; il est rare de voir un système d'authentification qui affiche sans camoufler par des étoiles ou autre chose similaire un mot de passe.

Conclusion :

Avant de ce TP, un des deux partenaires de l'équipe n'avait jamais touché à des threads ou à des mutexes. Il a pu apprendre à utiliser ces deux objets sur un cas concret, ce qui est toujours une compétence utile de programmation. Il a également appris à utiliser les sockets. Globalement, nous avons eu beaucoup de plaisir à implémenter ce TP qui n'est ni trop court, ni trop long à faire et qui permet de bien comprendre certains concepts vus en classe.