Sublim Telegram: Un système de communication

Thibault Béziers la Fosse, Benjamin Moreau

23 mars 2016

Table des matières

1 Introduction

Dans le cadre de notre module de Réseaux en Master ALMA, nous avons eu à créer et implémenter une application Client/Serveur utilisant les protocoles *TCP/IP*, sous *UNIX*, avec le langage *C*. Afin de remplir les objectifs du projet, nous avons décidé de développer un système de communication synchrone par terminal. Il intègre un analyseur syntaxique pour détecter les insultes, un système de salles de discutions, une gestion des droits, des commandes administrateur et un fichier permettant de pérenniser des informations sur les clients. Nous détaillerons d'abord les fonctionnalités de notre application, puis quelques détails d'implémentations sur chaque fonction.

2 Fonctionnalités

Les fonctionnalités présentées ci-dessous sont listées par ordre de priorité.

2.1 Connexion Client/Serveur

Cette première étape dans la conception de notre application permet au serveur de gérer des clients se connectant en envoyant des messages de manière simultanés. Lorsqu'un utilisateur envoie un message, le serveur le retransmet à tous les clients.

2.2 Analyseur syntaxique

Une fois la connexion établie, l'envoie d'un message entraînera une analyse syntaxique de chaque mot par le serveur. Le message sera décomposé, et comparé à un dictionnaire, qui pourra à son tour censurer certaines parties du message. La chaîne est enfin reconstruite et envoyée à tous les utilisateurs.

2.3 Pérennisation des informations utilisateur

Sublim Telegram a la possibilité de pérenniser ses données. Le serveur conservera dans un fichier les adresses des utilisateurs associées au nombre d'insulte détectés par l'analyseur syntaxique.

2.4 Salles de discutions

Une amélioration importante de *Sublim Telegram* a été implémentée. L'objectif est de pouvoir créer différentes salles de discussions créés par les clients et gérées du coté serveur. Les utilisateurs peuvent donc discuter en privé dans des salles hermétiques.

2.5 Gestion des droits et hiérarchisation

Enfin, nous avons décidé de donner la possibilité aux clients de gérer leur salon. l'utilisateur créant la salle de discussion en devient son administrateur. Il est possible d'administrer le salon à l'aide de commandes commençant par le caractère @.

3 Fonctionnement du client/serveur

3.1 Logiciel Client

3.1.1 lancement du client

L'exécution du client requière 3 arguments :

1. l'adresse du serveur.
2. le pseudo de l'utilisateur.
3. et le nom du salon à rejoindre ou à créer.

un *socket* de communication est alors créé entre le client et le serveur. le pseudo et le nom du salon sont envoyés au serveur.

Deux *threads* sont alors lancés. le premier s'occupe de la réception de message en provenance du serveur. Alors que Le second s'occupe de l'envoi de message vers ce dernier.

3.1.2 Envoi et réception de messages

À l'envoi, les messages sont automatiquement formatés : ils sont datés, identifiés à l'aide du pseudo et adressé par le nom du salon choisi. Les messages réceptionnés s'affichent avec leur date et le pseudo de l'utilisateur l'ayant expédié.

3.1.3 Fermeture du client

La fermeture du client s'effectue en pressant : Ctrl + C. une fonction est alors appelée. Cette dernière s'occupe de fermer le *socket* et les *threads* puis envoie un message notifiant les autres utilisateurs du salon.

3.2 Logiciel Serveur

3.2.1 Lancement du serveur

Le serveur doit être exécuté dans le même répertoire que le dictionnaire et le fichier utilisateurs.

La programme principal contient une boucle d'acceptation client. elle permet, durant l'exécution, d'accepter les connexions avec de nouveaux clients.

3.2.2 Détection et d'un client

A la création d'un *socket* entre le nouveau client et le serveur, un *threads* est lancé. Ce code exécuté en concurrence permet de recevoir des messages du client et de les renvoyer à chaque utilisateur.

Quand un *socket* est créé, l'adresse du client est récupérée et stockée dans le fichier ulisateur.txt si cette dernière n'y est pas déjà présente. l'*id* du *socket* est alors stocké dans une structure spéciale représentant les utilisateurs présents dans chaque salons. En effet, les salles de discussions sont stockés dans un tableau et chaque salle de discussion contient un tableau d'*id* de *socket* représentant les utilisateurs du salon.

3.2.3 Réception et distribution des messages

À la réception d'un message, une analyse syntaxique est faite. Chaque mot est comparé au dictionnaire d'insulte. Pour chaque mot reconnu, ses caractères sont remplacés par des * et le second champ présent dans le fichier *utilisateur.txt*, à la ligne correspondant à l'utilisateur ayant envoyé le message, est incrémenté. Ce champ séparé de l'adresse par un *:* représente donc le nombre d'insultes exprimées par le client.

Quand le message est enfin analysé, le nom de la salle de discussion est extraite de la trame et ce dernier est envoyé à tous les utilisateurs présent dans ce salon. Ces clients sont trouvé en effectuant une simple recherche dans la structure décrite ci dessus.

3.2.4 Fermeture du serveur

La fermeture du serveur s'effectue en pressant Ctrl + C. une fonction est alors appelée. Elle ferme les *sockets* et les *threads* puis envoie un message notifiant les utilisateurs.

3.3 Structure de la trame

L'ensemble de nos trames sont composés d'un entier placé en début de chaîne indiquant le type de message. Actuellement, il existe 3 sortes de messages :

id	signification
0	Message utilisateur
1	Commande administrateur
2	Message système

3.3.1 Message utilisateur

Un message envoyé par un utilisateur au serveur, est inscrit dans le buffer sous le format suivant

0	[hh :mm :ss]	salon	pseudo :	message
int	18 char	16 char	14 char	256 char

Le message est reçu par le serveur qui s'occupe de renvoyer le message à tout les clients du salon inscrit dans le message. Ce dernier élément est enlevé de la trame juste avant l'envoi ce qui donne le message ci dessous.

0	[hh :mm :ss]	pseudo :	message
int	18 char	14 char	256 char

3.3.2 Commande administrateur

Une commande administrateur est envoyée du client au serveur sous cette forme.

1	[hh :mm :ss]	salon	pseudo :	@commande
int	18 char	16 char	14 char	6 char

la commande commence par un @ et ne fait pas plus de 6 caractères. Actuellement seul la commande *@close* est implémentée. Elle permet de fermer le salon et d'y expulser tout les utilisateurs. La commande *@close* est reçu par le serveur et un message système est envoyé à tout les utilisateurs.

3.3.3 Messages systèmes

Un message système est envoyé du serveur et permet de prévenir un client d'une anomalie ou d'une déconnexion.

2	$id_{message}$
int	int

Le client recevant un tel message va voir apparaître un message système adapté. Voici la table de correspondance des $id_{message}$ avec leur signification.

$id_{message}$	utilisation
0	fermeture salon
1	fermeture serveur
2	utilisateur kické
3	salon plein
4	salon créé
5	salon rejoins
6	commandes disponibles
7	dernier socket connecté
8	un utilisateur est expulsé
9	erreur kick

L'analyse des commandes est géré du coté serveur. Si une commande est rajoutée, seul la mise a jour du serveur est nécessaire. Des clients possèdent une version différentes de *sublim Telegram* peuvent communiquer ensembles à condition que les trames n'ai pas été modifiées.

4 Problèmes rencontrés

Au cours de la présentation de *SublimTelegram*, une erreur est survenue lors de l'envoi d'un message par un client, sur une machine différente. Une boucle infinie a interrompu la réception des trames issues du serveur. Nous n'avons malheureusement pas su reproduire ce bug.

Au niveau du serveur, la partie réseau n'a pas posé de difficultés. Cependant, la majeure partie des problemes provenait d'erreurs de programmation liées au langage *C* et plus particulièrement la gestion des pointeurs et mémoire.

5 Conclusion

Pour conclure, ce fût un projet particulièrement intéressant. Depuis le début de notre cursus nous n'avions jamais été confronté à des problèmes de réseau, et implémenter une solution logicielle comme *Sublim Telegram* nous a permis de découvrir la programmation par Sockets en C. Même si notre code est fonctionnel, nous n'avons pas pu implémenter tout ce que nous désirions. Effectivement nous voulions faire de *Sublim Telegram* une application complète, mêlant une *Sublime* interface graphique, et des activités diverses à l'intention des utilisateurs.

Cependant, La structure de nos trames nous permets aisément d'ajouter des fonctionnalités. En effet, il est possible d'intégrer de nouveaux types de messages en modifiant l'identifiant de la trame. Par exemple, l'identifiant numéro 4 pourrait correspondre à un échange de fichiers tandis que, l'identifiant numéro 5 pourrait permettre d'échanger les données d'une partie endiablée d'échecs.