Rapport de Projet : Implémentation de Freescord

Darren Chauvet

18 mai 2025

1 Introduction

Ce rapport présente l'implémentation du projet Freescord, un service de messagerie en temps réel permettant à plusieurs utilisateurs de communiquer via un serveur central. Le projet est développé en C et utilise les sockets TCP/IP pour la communication réseau ainsi que des threads pour gérer les connexions multiples.

2 Fonctionnalités mises en œuvre

L'application Freescord comprend les fonctionnalités suivantes :
— Connexion TCP/IP entre clients et serveur
— Gestion multi-clients avec des threads
— Système d'attribution et de validation de pseudonymes
— Diffusion des messages à tous les clients connectés
— Conversion automatique entre formats de terminaison de ligne (CRLF/LF)
— Lecture bufferisée pour optimiser les performances
— Journal des événements et des messages côté serveur
— Interface utilisateur basique avec notifications visuelles
— Gestion des commandes spéciales (emojis textuels)

3 Architecture du système

Le système Freescord se compose de deux applications principales :

3.1 Serveur

Le serveur est conçu pour accepter des connexions multiples et gérer la diffusion des messages entre les clients.

— **Thread principal** : Accepte les nouvelles connexions et crée un thread dédié pour chaque client.
— **Threads clients** : Gèrent les communications individuelles avec chaque client, incluant l'attribution des pseudonymes et la réception des messages.
— **Thread répéteur** : S'occupe de la diffusion des messages reçus à tous les clients connectés.

Les messages sont transmis entre les threads via un tube (pipe) interne, ce qui assure la synchronisation entre la réception et la diffusion des messages.

3.2 Client

Le client permet à l'utilisateur de se connecter au serveur, de choisir un pseudonyme et d'échanger des messages.

— Utilise `poll()` pour gérer l'entrée utilisateur et les messages du serveur de manière non-bloquante.
— Effectue la conversion entre les formats de ligne (LF locale, CRLF pour le protocole).
— Affiche les messages des autres utilisateurs et les notifications système.

4 Protocole de communication

Le protocole Freescord implémenté comprend les étapes suivantes :

1. **Connexion initiale** : Le serveur envoie un message de bienvenue.
2. **Attribution de pseudonyme** :
   — Le client envoie une commande `/nickname <pseudo>`
   — Le serveur répond avec un code d'état :
      — 0 : Pseudonyme accepté
      — 1 : Pseudonyme déjà utilisé
      — 2 : Pseudonyme interdit (trop long ou contient des caractères interdits)
      — 3 : Format de commande invalide
3. **Communication** :
   — Les messages envoyés par un client sont préfixés avec son pseudonyme et diffusés à tous les clients.
   — L'expéditeur reçoit son propre message avec un préfixe `[vous]` pour identification.

5 Aspects techniques notables

5.1 Gestion des connexions clients

Le serveur utilise la librairie pthread pour créer un thread dédié à chaque client connecté. Ces threads sont détachés (`pthread_detach`) pour libérer automatiquement leurs ressources à la déconnexion. La liste des clients actifs est protégée par un mutex pour éviter les conditions de course lors des accès concurrents.

5.2 Buffering et traitement des lignes

Une implémentation personnalisée de buffer de lecture est utilisée pour optimiser les performances en minimisant les appels système. Le module `buffer.c` fournit des fonctions pour créer, lire et libérer des buffers associés à des descripteurs de fichiers.

5.3 Conversion CRLF/LF

Pour assurer la compatibilité entre le protocole réseau (utilisant CRLF comme terminaison de ligne) et les systèmes Unix (utilisant LF), les fonctions `crlf_to_lf` et `lf_to_crlf` ont été implémentées pour convertir automatiquement les formats.

6 Améliorations du protocole

Par rapport au protocole de base décrit dans le sujet, quelques améliorations ont été apportées :

— **Commandes spéciales** : Le client supporte des commandes textuelles pour insérer des emojis ASCII :
   — `/tableflip`
   — `/unflip`
   — `/shrug`

— **Formatage des messages** : Les messages de l'utilisateur courant sont affichés en gris pour les distinguer facilement des autres messages.
— **Système de journalisation** : Le serveur enregistre toutes les activités importantes (connexions, déconnexions, messages) dans un fichier journal avec horodatage.

7 Ressources externes

Lors de la réalisation de ce projet, plusieurs ressources en ligne ont été consultées pour mieux comprendre certains concepts et implémentations :

— Tutoriel sur la programmation réseau en C avec sockets :
`https://beej.us/guide/bgnet/`
— Explication sur la gestion des entrées/sorties non bloquantes et `poll()` :
`https://www.gnu.org/software/libc/manual/html_node/Nonblocking-I_002fO.html`

8 Conclusion

L'implémentation actuelle de Freescord fournit une base solide pour un service de messagerie en temps réel. Le code est structuré de manière modulaire et robuste, avec une attention particulière portée à la gestion des erreurs et à l'efficacité des opérations réseau.

Des améliorations futures pourraient inclure le support d'IPv6, l'ajout de salons de discussion, l'implémentation du transfert de fichiers, ou encore le chiffrement des communications.