

DOCUMENTATION TECHNIQUE

Serveur de chat multiclients en C

(TCP / Threads)

Mini-projet de Kyliann CLARET-LAVAL et Lucas MARROT

USSI07 - Système et Architecture des Machines

07/01/2025

Table des matières

1. Présentation du projet	3
1.1. Nom du projet	3
1.2. Objectif du projet	3
1.3. Fonctionnalités principales.....	3
2. Architecture technique.....	3
2.1. Organisation générale.....	3
2.2. Architecture du serveur (server.c)	3
2.2.1. Sockets réseau	3
2.2.2. Gestion des clients par threads.....	4
2.2.3. Synchronisation.....	4
2.2.4. Journalisation.....	4
2.3. Architecture du client (client.c)	4
2.3.1. Connexion au serveur	4
2.3.2. Gestion asynchrone.....	5
2.3.3. Commandes utilisateur	5
3. Instructions d'utilisation	5
3.1.1. Dépendances	5
3.1.2. Compilation	5
3.1.3. Exécution	5
3.1.4. Utilisation	5
4. Tests effectués	6
5. Résultats des tests.....	6
6. Conclusion.....	7

1. Présentation du projet

1.1. Nom du projet

Serveur de chat multilients

1.2. Objectif du projet

L'objectif de ce projet est de développer une application de chat réseau en C permettant à plusieurs clients de se connecter simultanément à un serveur central et d'échanger des messages en temps réel. Le projet met en œuvre les principaux concepts de programmation système, notamment la communication réseau par sockets TCP, la gestion du parallélisme à l'aide de threads et la synchronisation des accès concurrents.

1.3. Fonctionnalités principales

- Connexion simultanée de plusieurs clients à un serveur unique
- Envoi et réception de messages en temps réel
- Diffusion automatique des messages à l'ensemble des clients connectés
- Gestion de commandes côté client (/help, /quit, /switch)
- Journalisation des messages côté serveur
- Gestion propre des connexions et déconnexions

2. Architecture technique

2.1. Organisation générale

Le projet est structuré autour de deux programmes principaux :

- **server.c** : implémente le serveur de chat
- **client.c** : implémente le client permettant aux utilisateurs de se connecter au serveur

L'architecture repose sur un modèle **client/serveur** utilisant le protocole **TCP** pour garantir la fiabilité des échanges.

2.2. Architecture du serveur (server.c)

2.2.1. Sockets réseau

Le serveur :

- crée un socket TCP (socket)
- l'attache à un port (bind)

- se met en écoute (`listen`)
- accepte les connexions clientes (`accept`)

Chaque nouvelle connexion est traitée indépendamment.

2.2.2. Gestion des clients par threads

Pour chaque client connecté :

- un thread dédié est créé (`pthread_create`)
- le thread gère la réception des messages du client
- les messages reçus sont diffusés aux autres clients connectés

Ce modèle permet de gérer plusieurs clients simultanément sans bloquer le serveur.

2.2.3. Synchronisation

Un mutex global est utilisé afin de protéger les ressources partagées :

- liste des clients connectés
- diffusion des messages
- journalisation

Cela garantit la cohérence des données en cas d'accès concurrent par plusieurs threads.

2.2.4. Journalisation

Chaque message reçu est :

- horodaté
- associé à son expéditeur
- enregistré dans un fichier de log

Cette fonctionnalité permet de conserver une trace complète des échanges.

2.3. Architecture du client (`cclient.c`)

2.3.1. Connexion au serveur

Le client :

- crée un socket TCP
- se connecte au serveur via son adresse IP et son port
- reste connecté tant que l'utilisateur ne quitte pas explicitement

2.3.2. Gestion asynchrone

Le client utilise :

- un thread de réception pour afficher les messages entrants
- le thread principal pour lire les entrées clavier de l'utilisateur

Cela permet à l'utilisateur d'écrire tout en recevant des messages simultanément.

2.3.3. Commandes utilisateur

Certaines commandes sont traitées localement :

- /help : affiche la liste des commandes disponibles
- /quit : ferme proprement la connexion
- /switch : changement de canal

Les commandes non reconnues déclenchent un message d'erreur côté client.

3. Instructions d'utilisation

3.1.1. Dépendances

- Système Linux ou macOS
- Compilateur gcc
- Bibliothèque pthread (POSIX threads)

3.1.2. Compilation

Depuis le répertoire du projet :

```
gcc server.c -o server -pthread  
gcc client.c -o client -pthread
```

3.1.3. Exécution

Lancer d'abord le serveur :

```
./server
```

Puis lancer un ou plusieurs clients dans d'autres terminaux :

```
./client
```

3.1.4. Utilisation

- Saisir un message pour l'envoyer

- Utiliser /help pour afficher l'aide
- Utiliser /quit pour quitter proprement le chat

4. Tests effectués

Les tests suivants ont été réalisés afin de valider le bon fonctionnement du serveur de chat multiclients :

- Connexion de plusieurs clients
- Connexions et déconnexions successives de clients
- Déconnexion brutale d'un client en cours de communication
- Diffusion correcte des messages à tous les clients connectés
- Exclusion de l'expéditeur lors de la diffusion d'un message
- Test des commandes utilisateur (/help, /quit, /switch)
- Gestion des commandes inconnues avec message d'erreur
- Création dynamique des channels lors du premier accès
- Création et écriture des fichiers de logs
- Vérification de l'horodatage et du format des messages enregistrés
- Envoi de messages vides ou de grande taille
- Test de charge avec envoi rapide de messages
- Dépassement des limites du nombre de channels par serveur et du nombre de personnes par channel

5. Résultats des tests

Ces tests confirment la stabilité, la robustesse et le bon fonctionnement global du projet.

Exemples en image :

- Serveur actif avec 2 clients (Personne1 et Personne2) sur le channel 'test'



```
Mini_Projet git:(main) * ./output/serveur
Serveur en écoute sur le port 12345...

Bienvenue dans le channel 'test' !
[test] (06/01/2026 18:33:40) Moi : Bonjour, je suis Personne1 et toi ?
Personne2 a rejoint le channel 'test'... (2/100)
[test] (06/01/2026 18:34:15) Personne2 : Enchanté je suis Personne2 !

Envoyer un message : 
```

- Serveur actif avec 10 clients dans le même channel

[illegible]

- Message vide et message long

```
[test] (06/01/2026 21:51:06) Moi :  
[test] (06/01/2026 21:52:41) Moi : Ceci est un test d'un message très long qui a pour but de tes  
ter l'affichage et l'envoi d'un message qui soit de très grande taille et c'est pour ça que je f  
ais ce message pour vérifier que je peux bien envoyer un message de grande taille donc je vérifi  
e et je teste avec cet exemple voilà je suis personnel et je fais un message de très grande tail  
le.
```

6. Conclusion

Ce projet met en œuvre les concepts fondamentaux de la programmation système : sockets TCP, threads, synchronisation et gestion des erreurs. Il constitue une application fonctionnelle, robuste et extensible, répondant pleinement aux objectifs pédagogiques du mini-projet FIP1.

Nous sommes fiers de notre programme qui nous a permis de comprendre les concepts clés de la programmation système et de nous améliorer en C.