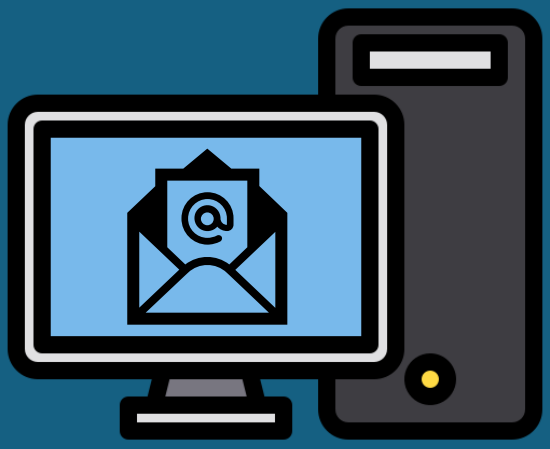
PROJET FAR



GIVRE Zolan / SEMPERE Lou / SAUVAGE Léo

POLYTECH MONTPELLIER 2023 / 2024

Table des matières

Partie Technique

Description générale de l'application

L'application de messagerie permet à plusieurs clients de se connecter à un serveur centralisé et d'échanger des messages textuels en utilisant le protocole TCP. Chaque client se connecte au serveur, choisit un pseudo, et peut envoyer des messages à tous les autres clients connectés ou à un client spécifique via des messages privés.

Description de l'architecture des programmes client et serveur

Serveur

Main

Le rôle principal du serveur est de gérer les connexions des clients et de maintenir une communication fluide et sécurisée entre eux. Voici les principales responsabilités du serveur :

- Accepter les connexions entrantes : Le serveur écoute en permanence sur un port spécifique pour les nouvelles connexions clients. Lorsqu'un client tente de se connecter, le serveur accepte la connexion, établissant ainsi un canal de communication bidirectionnel entre le client et le serveur.
- Initialisation et lancement des threads de gestion des clients : Pour chaque client connecté, le serveur crée un nouveau thread dédié à la gestion des communications avec ce client. Ces threads permettent au serveur de gérer plusieurs clients simultanément de manière concurrente, assurant une gestion fluide des messages entrants et sortants.

Threads de client

Chaque client connecté au serveur est géré par un thread dédié. Les threads de gestion des clients ont plusieurs responsabilités clés :

- Réception des messages des clients : Chaque thread attend les messages envoyés par le client qu'il gère. Lorsqu'un message est reçu, le thread lit le contenu du message et le traite en conséquence.
- Relais des messages : Les messages reçus peuvent être de deux types : publics ou privés (/mp). Les messages publics sont relayés à tous les autres clients connectés, tandis que les messages privés sont envoyés uniquement au destinataire spécifié. Cette distinction permet une communication flexible et ciblée entre les clients.
- Gestion de la déconnexion propre des clients : Lorsqu'un client souhaite se déconnecter (par exemple, en envoyant la commande /fin), le thread gère la fermeture propre de la connexion. Cela inclut la mise à jour du tableau des clients pour refléter la déconnexion et la notification des autres clients de cette déconnexion.

Client

Main

Le programme principal côté client gère l'interaction utilisateur et la communication avec le serveur. Voici ses principales fonctions :

- Interface utilisateur : Le programme principal gère la saisie des commandes et des messages par l'utilisateur. Les utilisateurs peuvent entrer des messages publics, des messages privés, ou des commandes spéciales comme /fin pour se déconnecter.
- Envoi des messages au serveur : Les messages et commandes saisies par l'utilisateur sont envoyés au serveur via le canal de communication établi. Le programme principal gère la connexion TCP et envoie les données de manière fiable au serveur.

Thread de réception

En plus du programme principal, un thread de réception est utilisé pour gérer la réception des messages provenant du serveur. Ce thread fonctionne indépendamment de la saisie utilisateur, assurant une communication asynchrone.

- Réception des messages : Le thread de réception attend en permanence les messages envoyés par le serveur. Cela inclut les messages publics envoyés par d'autres clients, les messages privés destinés spécifiquement à l'utilisateur, et les notifications de déconnexion d'autres clients.
- Affichage des messages : Les messages reçus sont affichés à l'écran pour l'utilisateur. Cela permet à l'utilisateur de voir en temps réel les messages des autres clients et de répondre rapidement sans interrompre la saisie de nouveaux messages.

Description des fonctionnalités

Envoi et réception de messages

Envoi de messages broadcast à tous les clients

Lorsque les clients envoient des messages, le serveur les reçoit et les relaie à tous les autres clients connectés pour assurer une communication en temps réel. Lorsqu'un client envoie un message, le serveur lit ce message à partir du socket du client et le diffuse à tous les autres clients actifs, à l'exception de l'expéditeur. Cette diffusion est réalisée en parcourant la liste des clients connectés et en envoyant le message via leurs sockets respectifs. Cela permet à tous les utilisateurs de recevoir instantanément les messages et de participer à la conversation de manière continue et fluide.

Envoi de messages privés via la commande /mp [pseudo] [message]

Le serveur prend également en charge l'envoi de messages privés, permettant à un client d'envoyer un message directement à un autre client sans que les autres ne puissent le voir. Pour ce faire, le client utilise la commande /mp suivie du pseudo du destinataire et du contenu du message. Lorsque le serveur reçoit cette commande, il extrait le pseudo du destinataire et recherche ce client dans la liste des utilisateurs connectés. Si le destinataire est trouvé, le serveur lui envoie le message directement. Si le pseudo du

destinataire n'existe pas, le serveur informe l'expéditeur de l'erreur. Cette fonctionnalité assure des communications privées et confidentielles entre les utilisateurs.

Commandes spéciales

- /fin : Permet à un client de se déconnecter proprement du serveur. Lorsqu'un client envoie cette commande, le serveur met à jour ses informations pour indiquer que le client s'est déconnecté et clôt la connexion de manière ordonnée. Cette commande assure une déconnexion propre sans perturber les autres clients connectés.
- /kill : Une commande pour arrêter le serveur. Lorsqu'un client envoie cette commande, le serveur envoie un message de fin à tous les clients connectés, informant de l'arrêt imminent du serveur. Ensuite, le serveur ferme toutes les connexions actives et s'arrête. Cette commande est utile pour les administrateurs souhaitant arrêter le serveur de manière contrôlée.
- /man : Affiche la liste des commandes disponibles et leur utilisation. Cette commande aide les utilisateurs à naviguer et à utiliser efficacement les fonctionnalités de l'application. Elle fournit un guide des commandes que les clients peuvent utiliser pour interagir avec le serveur et les autres clients.
- /mp [pseudo] [message] : Permet à un client d'envoyer un message privé à un autre client spécifié par son pseudo. Lorsqu'un client envoie cette commande, le serveur extrait le pseudo du destinataire et recherche ce client dans la liste des utilisateurs connectés. Si le destinataire est trouvé, le serveur lui envoie le message directement. Si le pseudo du destinataire n'existe pas, le serveur informe l'expéditeur de l'erreur. Cette commande permet des communications privées et confidentielles entre les utilisateurs.
- /request : Permet aux utilisateurs d'afficher la liste de tous les fichiers disponibles sur le serveur. En utilisant "/request", les clients peuvent rapidement explorer les ressources partagées et décider des fichiers à récupérer. Cette fonctionnalité ajoute une couche de convivialité à notre application, offrant aux utilisateurs un moyen simple et efficace d'accéder aux fichiers disponibles sur le serveur.

Déconnexion propre

La déconnexion d'un client est gérée proprement pour maintenir l'intégrité du système. Lorsqu'un client se déconnecte volontairement en envoyant la commande /fin, ou lorsque la connexion est interrompue de manière inattendue, le serveur met à jour le tableau des clients pour refléter cette déconnexion. Les autres clients sont informés de la déconnexion afin qu'ils sachent que le client n'est plus disponible.

Envoi et réception de fichiers

Au-delà de la simple messagerie textuelle, l'application permet également l'envoi et la réception de fichiers. Les clients peuvent utiliser /file une commande spécifique pour transférer des fichiers au serveur, qui les stocke et les redistribue aux clients destinataires via des canaux de communication dédiés. Cela permet de maintenir les messages textuels séparés des transferts de fichiers, améliorant ainsi la gestion et la performance du système.

Explication des protocoles

Protocole de communication

La communication entre le client et le serveur se fait principalement via le protocole TCP, garantissant une transmission fiable et ordonnée des messages.

Pour optimiser la transmission des messages textuels, l'application utilise un protocole en deux étapes :

- Envoi de la taille du message : Avant d'envoyer le contenu du message, le client envoie d'abord un entier fixe représentant la taille du message en octets. Cela permet au serveur de savoir exactement combien d'octets lire pour le message suivant.
- Envoi du message lui-même : Une fois la taille envoyée, le client envoie le contenu du message. Le serveur utilise la taille précédemment reçue pour lire le message complet sans erreurs de fragmentation. Cette méthode assure que les messages sont reconstitués correctement et entièrement avant d'être traités.

Partie Synthèse

Introduction

Ce projet de messagerie en C a pour objectif de développer une application permettant à plusieurs clients de communiquer via un serveur centralisé, en utilisant des fonctionnalités avancées telles que les messages privés et l'envoi/réception de fichiers. Le projet a été découpé en plusieurs sprints, chacun visant à ajouter des fonctionnalités spécifiques tout en améliorant les fonctionnalités existantes et en assurant la robustesse et la sécurité de l'application.

Objectif

L'objectif principal est de créer une application de messagerie fiable et extensible, capable de gérer plusieurs clients de manière concurrente, tout en offrant une expérience utilisateur fluide et intuitive.

Gestion de projet et outils utilisés

La gestion de projet et l'utilisation d'outils adéquats sont essentielles pour assurer le bon déroulement du développement de l'application de messagerie.

Gestion de version

Nous avons choisi d'utiliser GitHub comme plateforme de gestion de version pour notre projet. Cela nous a permis de collaborer efficacement, de suivre les modifications apportées au code et de gérer les problèmes rencontrés tout au long du développement.

Planification et suivi

La répartition des tâches au sein de notre équipe s'est faite de manière équilibrée tout au long des sprints. Deux membres de l'équipe étaient responsables du développement du code, tandis qu'un troisième membre se chargeait de créer le diagramme UML et de rédiger le rapport de sprint. Cette répartition des responsabilités nous a permis de travailler de manière efficace et de progresser de manière cohérente tout au long du projet.

Difficultés rencontrées

Nous avons rencontré quelques difficultés tout au long du projet, notamment lors du développement collaboratif du code. L'une des principales contraintes était liée à la nature du projet lui-même : avec seulement deux fichiers interconnectés (client et serveur), il était parfois difficile pour plusieurs membres de l'équipe de coder simultanément sans risquer des conflits de fusion ou des incohérences dans le code. Cette limitation nous a obligés à coordonner étroitement notre travail et à planifier soigneusement nos interventions pour éviter tout problème de compatibilité. Malgré ces défis, nous avons réussi à surmonter ces obstacles grâce à une communication régulière et à une organisation efficace de notre collaboration.

Résultats intermédiaires

Sprint 1 : Fondations de la communication

Au cours du premier sprint, notre équipe s'est concentrée sur la mise en place des bases de notre application de messagerie. Nous avons développé un serveur simple capable de gérer la communication entre deux clients. C'était une étape cruciale pour comprendre les concepts fondamentaux des sockets TCP et établir les mécanismes de connexion et de transmission de messages.

Sprint 2 : Gestion multi-clients et commandes spéciales

Lors du deuxième sprint, nous avons introduit le multi-threading pour permettre au serveur de gérer plusieurs clients simultanément. Cette amélioration a considérablement renforcé la réactivité de notre application, éliminant ainsi la contrainte où seul le premier client connecté pouvait initier la communication. De plus, nous avons enrichi les fonctionnalités de l'application en ajoutant des commandes spéciales telles que /fin et /kill, offrant ainsi aux utilisateurs un meilleur contrôle sur leurs interactions avec le serveur.

Sprint 3 : Fonctionnalités d'envoi et de réception de fichiers

Au cours du troisième sprint, nous avons étendu les fonctionnalités de notre application en ajoutant la capacité d'envoyer et de recevoir des fichiers. Cette fonctionnalité a considérablement enrichi l'expérience utilisateur en permettant aux utilisateurs de partager non seulement des messages textuels, mais également des fichiers de différents types. L'implémentation réussie de cette fonctionnalité a été un jalon majeur dans le développement de notre application.

Résultat final

Dans sa version finale, notre application offre un environnement complet de messagerie, permettant à plusieurs clients de se connecter, de communiquer par des messages textuels et de partager des fichiers. Avec la possibilité d'envoyer des messages privés et d'utiliser des commandes spéciales telles que /fin et /kill, notre application offre une expérience utilisateur riche et diversifiée. Ce projet nous a permis de développer nos compétences en programmation réseau et en collaboration d'équipe, tout en fournissant une solution pratique et fonctionnelle pour la communication instantanée.

Prospectives d'évolution

Salons de discussion

L'introduction de salons de discussion permettrait de créer des espaces thématiques où les utilisateurs peuvent se regrouper selon leurs centres d'intérêt ou leurs projets. Cette évolution offrirait une expérience plus personnalisée et permettrait aux utilisateurs de trouver et de rejoindre des conversations pertinentes pour eux.

Sécurité

En ajoutant une couche de chiffrement aux communications entre les clients et le serveur, nous pourrions renforcer la confidentialité et la sécurité des échanges. L'implémentation de protocoles de sécurité robustes garantirait que les données

sensibles des utilisateurs restent privées et protégées contre les tentatives de piratage ou d'interception.

Amélioration de l'Interface

Le développement d'une interface graphique utilisateur (GUI) offrirait une expérience plus conviviale et intuitive aux utilisateurs. Une interface attrayante et bien conçue faciliterait la navigation dans l'application, tout en offrant des fonctionnalités avancées telles que la gestion des salons, la visualisation des notifications et la personnalisation des préférences.

Notifications

La mise en place de notifications permettrait aux utilisateurs de rester informés des nouveaux messages et des événements importants, même lorsqu'ils ne sont pas activement connectés à l'application. Des notifications en temps réel via des pop-ups, des sons ou des alertes visuelles garantiraient que les utilisateurs ne manquent jamais une conversation importante ou une mise à jour cruciale.

Conclusion

Le développement de cette application de messagerie en C a été une expérience enrichissante et formatrice pour toute l'équipe. Nous avons pu explorer et maîtriser différents aspects de la programmation réseau, en mettant en œuvre des fonctionnalités avancées telles que le multi-threading pour gérer efficacement les connexions multiples et la communication simultanée entre les clients.

En travaillant sur ce projet, nous avons été confrontés à divers défis techniques, notamment la gestion des connexions et des déconnexions des clients, la synchronisation des threads et la sécurisation des échanges de données. Ces défis nous ont permis d'approfondir nos connaissances et nos compétences en programmation en C, tout en développant des solutions innovantes pour répondre aux exigences du projet.

Grâce à une collaboration étroite et à une planification efficace, nous avons réussi à surmonter ces défis et à livrer une application fonctionnelle et performante. L'application offre une plateforme de communication fiable et flexible, permettant aux utilisateurs d'échanger des messages textuels et des fichiers de manière sécurisée et conviviale.

En conclusion, ce projet nous a non seulement permis d'acquérir de nouvelles compétences techniques, mais aussi de renforcer notre capacité à travailler en équipe et à relever des défis complexes. Nous sommes fiers du résultat final et confiants dans le potentiel d'évolution et d'amélioration continue de notre application de messagerie en C.