Une image contenant texte, capture d’écran, Police, Graphique

Description générée automatiquement

**Rapport du projet Terminal 2023-2024**

**Sujet : Outil de chat entre plusieurs client**

**Par C et Java**

**Réalisé par :**

**M’BARKI Mohammed**

**ELOGRI Adnane**

**Année scolaire : 2023-2024**

Contents

[Chapitre 1 : OUTILS ET CONCEPTS FONDAMENTAUX DU PROJET 3](#_Toc165614963)

[1. Objectifs du Projet : 4](#_Toc165614964)

[2. Pourquoi ces outils (Java, C) : 4](#_Toc165614965)

[Chapitre 2 : Mise en Œuvre et Manipulation 5](#_Toc165614966)

[1. Hiérarchie générale du projet dans notre implémentation 6](#_Toc165614967)

[1. Les défis rencontrés en créant le projet 8](#_Toc165614968)

[Chapitre 3 : Conclusion 9](#_Toc165614969)

# Chapitre 1 : OUTILS ET CONCEPTS FONDAMENTAUX DU PROJET

## Objectifs du Projet :

Ce projet de Système distribué et Système d'exploitation a été élaboré dans le but principal de créer un outil de chat intégré qui facilite la communication entre plusieurs utilisateurs connectés simultanément. Le projet a pour objectifs particuliers les suivants :

1. Création d'une interface fluide pour l’utilisateur en **Terminal** : Créer une application de chat qui offre aux utilisateurs la possibilité de s'inscrire, de se connecter, d'envoyer des messages et de consulter les messages des autres utilisateurs.
2. Mise en place de la communication en temps réel : Garantir la réception immédiate et l'affichage de tous les messages envoyés par un utilisateur par les autres utilisateurs connectés.
3. Gestion des comptes utilisateurs : Créer un système solide de gestion des comptes, comprenant un enregistrement sécurisé des identifiants et des mots de passe, ainsi que la capacité de créer et de supprimer des comptes.
4. Intégration des concepts étudié en TD et TP et cours de systèmes distribués et des systèmes d'exploitation : Utiliser les concepts des systèmes distribués et d'exploitation afin de gérer de manière efficace les communications réseau et les interactions entre les différents processus.

Ces objectifs jouent un rôle crucial non seulement dans la réussite du projet, mais également dans l'acquisition de compétences pratiques en matière de systèmes distribués et de systèmes d'exploitation. Dans ce projet, nous avons tous les deux acquis de nombreuses connaissances en programmation, en réseaux, en systèmes d'exploitation et distribués, ainsi que dans d'autres domaines.

## Pourquoi ces outils (Java, C) :

Une image contenant Graphique, Police, graphisme, affiche

Description générée automatiquementLes langages Java et C ont été sélectionnés pour ce projet en raison de leurs caractéristiques distinctives et de leur complémentarité dans la gestion des divers aspects de l'application de chat.

Une image contenant Graphique, cercle, capture d’écran, logo

Description générée automatiquement Java : Java a principalement été employé en raison de sa portabilité entre diverses plateformes, ce qui est crucial pour développer des interfaces utilisateur et des fonctionnalités de gestion des comptes. En outre, Java propose une multitude de bibliothèques pour le développement, en particulier pour les communications réseau via RMI (Remote Method Invocation), ce qui simplifie la mise en place des fonctionnalités de gestion des comptes et de communication client-serveur dans un environnement distribué.

C : Le langage C a été employé en raison de ses performances et de sa capacité à contrôler les ressources système à un niveau bas, ce qui a une importance capitale pour la gestion des communications internes du serveur et la manipulation directe des sockets TCP pour la transmission des messages. Il est également possible d'avoir une interaction plus précise avec le système d'exploitation, ce qui est essentiel pour améliorer les processus de communication et gérer les ressources système de manière efficace.

Et pour collaborer entre eux, plusieurs méthodes ont été utilisées pour étudier les interactions entre ces deux langages :

TCP : La communication directe entre le client (Java) et le serveur (C) a pourras être effectuer grâce aux sockets TCP. Cette approche garantit une communication rapide et fiable entre les utilisateurs.

UDP : qu’ils garantissent une communication bien plus rapide que TCP mais moins fiable.

# Chapitre 2 : Mise en Œuvre et Manipulation

## Hiérarchie générale du projet dans notre implémentation

Avant de débuter l'implémentation du projet, nous avons dû le structurer de manière adéquate afin de le construire de manière efficace et de répondre à tous ses besoins. Dans cette architecture, plusieurs composants interagissent pour offrir une expérience de chat fluide et sécurisée. Un aperçu global de l'architecture et de la fonction de chaque élément est présenté ci-après :

1. La partie Client : Dans cette partie, il y a deux fichiers principaux qui illustrent le client. Le fichier client\_chat\_plus.c regroupe les fonctions accessibles aux utilisateurs comme la création de compte, la suppression de compte et la communication des messages. Les messages reçus par tous les clients connectés sont affichés dans le fichier affichage\_message.c.
2. La partie Centralisée : Dans cette partie où on a beaucoup de services de communication qui on mises en place, dans cette partie on avait 2 fichiers :

’server\_chat\_amlr.c’ qui représente le serveur , et ‘gestion\_requette.c’ qui géré la communication entre la partie centralisée et la partie suivante par des sockets UDP, le fichier ’server\_chat\_amlr.c’ et la porte de cette partie , il reçois les donnée (messages et requêtes des clients) et il les traite par différents fonctions comme la fonction ‘recevoirDonnee’ qui reçoit les à l’aide des socket TCP et après on peut gérer ses données reçus par le client.

Et le fichier ‘gestion\_requette.c’ s’est comme un lien qui lie entre le serveur et le client RMI à l’aide des sockets UDP , la connexion entre ces deux fichiers se fait à l’aide des pipe nommés (une pour recevoir de serveur vers le gestion\_requettes et l’autre inverse) on a utilisé ainsi des sémaphores comme outil pour accéder la mémoire partage et organiser le problème d’accès concurrent.

1. Description du Client RMI et du Gestionnaire de Comptes ..  
   **GestionCompte.java** : Il s'agit de la classe principale qui implémente l'interface ICompte. Elle fournit les fonctionnalités pour gérer les comptes utilisateurs. Elle a des méthodes pour créer, supprimer et se connecter aux comptes. Elle lit également et écrit les données de compte dans un fichier nommé liste.txt. La méthode main dans cette classe configure un service distant pour gérer les comptes.

**ICompte.java** : Il s'agit d'une interface distante qui étend java.rmi.Remote. Elle déclare trois méthodes que toute classe implémentant cette interface doit définir : creerCompte (créer un compte), supprimerCompter (supprimer un compte) et connexion (connexion). Ces méthodes lancent une RemoteException qui est une superclasse commune pour un certain nombre d'exceptions liées à la communication qui peuvent survenir lors de l'exécution d'un appel de méthode à distance.

**ClientRMI**.java est une classe qui agit comme un client dans une application Java RMI (Remote Method Invocation). Elle se connecte à un service distant, en l'occurrence le service de gestion de comptes (ServiceCompte) qui est exposé par le serveur.

Une image contenant texte, diagramme, ligne, Police

Description générée automatiquement

Et voici un aperçu des différents outils que nous avons utilisé :

* Pipes : qui sont des canaux de communication unidirectionnels utilisés pour passer des informations d'un processus à un autre. Dans notre projet, les pipes nommés ont été utilisés pour communiquer entre les processus serveur et la gestion des requêtes et aussi entre le client et affichage de message , permettant une séparation claire et une gestion modulaire des fonctionnalités.
* Mémoire partagée : elle permet à différents processus d'accéder à un même espace de mémoire. Nous avons utilisé cette technique pour stocker des informations fréquemment utilisées, comme la liste des utilisateurs connectés, afin de minimiser les coûts de communication inter-processus et accélérer l'accès aux données communes.
* Sémaphores : Les sémaphores sont des mécanismes de synchronisation qui aident à contrôler l'accès à des ressources partagées par plusieurs processus. Dans notre système, les sémaphores ont été essentiels pour gérer l'accès concurrentiel à la mémoire partagée et pour coordonner les opérations entre les processus serveur et gestionnaire de requêtes, assurant ainsi l'intégrité et la cohérence des données.
* Sockets : Les sockets fournissent les points de connexion nécessaires pour la communication réseau entre des processus s'exécutant potentiellement sur des machines différentes. Nous avons utilisé les sockets TCP pour une communication fiable entre le client et le serveur, et les sockets UDP pour des transferts de données moins sensibles entre les composants serveur.

## Les défis rencontrés en créant le projet

Au cours de l'implémentation de ce projet de système de chat, nous avons rencontré pas mal des défis techniques mais qui ont été significatifs, Ces défis nous ont donné vraiment des occasions précieuses pour approfondir notre compréhension pratique et notre aptitude à mettre en œuvre des solutions novatrices et à cause de temps vaste qu'on avait pour réaliser ce projet, on a pus tester beaucoup des techniques et de savoir les avantages et les inconvénients de chacune d'eux. Voici quelques-uns des principaux défis auxquels nous avons été confrontés et les approches que nous avons prises pour les résoudre :

1. Gestion de la concurrence sur les ressources partagées :

Problème : Quand plusieurs clients cherchaient à accéder en même temps à des ressources communes, telles que l'envoi de messages, cela provoquait des conflis et des incohérences dans les données.

Solution : Nous avons implémenté des mécanismes de synchronisation tels que l'utilisation de sémaphores et de verrous, afin de gérer l'accès concurrent aux ressources. Cela a rendu l'accès aux parties essentielles du code plus sérieux et garantit l'intégrité des données.

1. Problème des tâches en arrière-plan et utilisation des threads :

Problème : Notre système nécessitait la gestion de multiples tâches en arrière-plan, telles que la surveillance des connexions des utilisateurs et les messages reçus et autres.  
Solution : Afin de relever cette difficulté, nous avons mis en place un modèle multi-threading, où différentes tâches en arrière-plan sont exécutées en même temps dans des threads distincts.

# Chapitre 3 : Conclusion

Conclusion

Notre projet a réussi à mettre en œuvre une plateforme de chat distribuée qui intègre des systèmes d'exploitation avancés et des technologies de systèmes distribués, similaires à celles étudiées lors de ce semestre 6, afin de fournir une solution de communication sécurisée et performante, avec des comptes personnalisés pour chaque client. Les objectifs initiaux, incluant le développement d'une interface utilisateur interactive, la gestion sécurisée des comptes utilisateurs, et la communication en temps réel entre les clients, ont été atteints avec succès. Grâce à l'utilisation judicieuse des technologies telles que Java, C et ses riches bibliothèques, les sockets, les threads, et les mécanismes de synchronisation et de gestion de la mémoire, le système peut gérer de manière fluide et stable les interactions simultanées de plusieurs utilisateurs.

Ce projet a renforcé notre compréhension des principes fondamentaux des systèmes d'exploitation et des systèmes distribués. Nous avons appris que la planification détaillée et une architecture solide sont cruciales dans le développement de logiciels, en particulier pour les systèmes en temps réel et distribués. Parmi les défis majeurs, la gestion des erreurs et la synchronisation des processus se sont révélés être des aspects particulièrement complexes. De plus, travailler en binôme nous a enseigné comment gérer le stress et améliorer notre collaboration.