GRAVOUIL Achille (achille.gravouil@etud.univ-pau.fr)

GARCIA Matéo (garcia.m@etud.univ-pau.fr)

Compte rendu du Projet Systèmes Distribués et Systèmes d'Exploitation

---

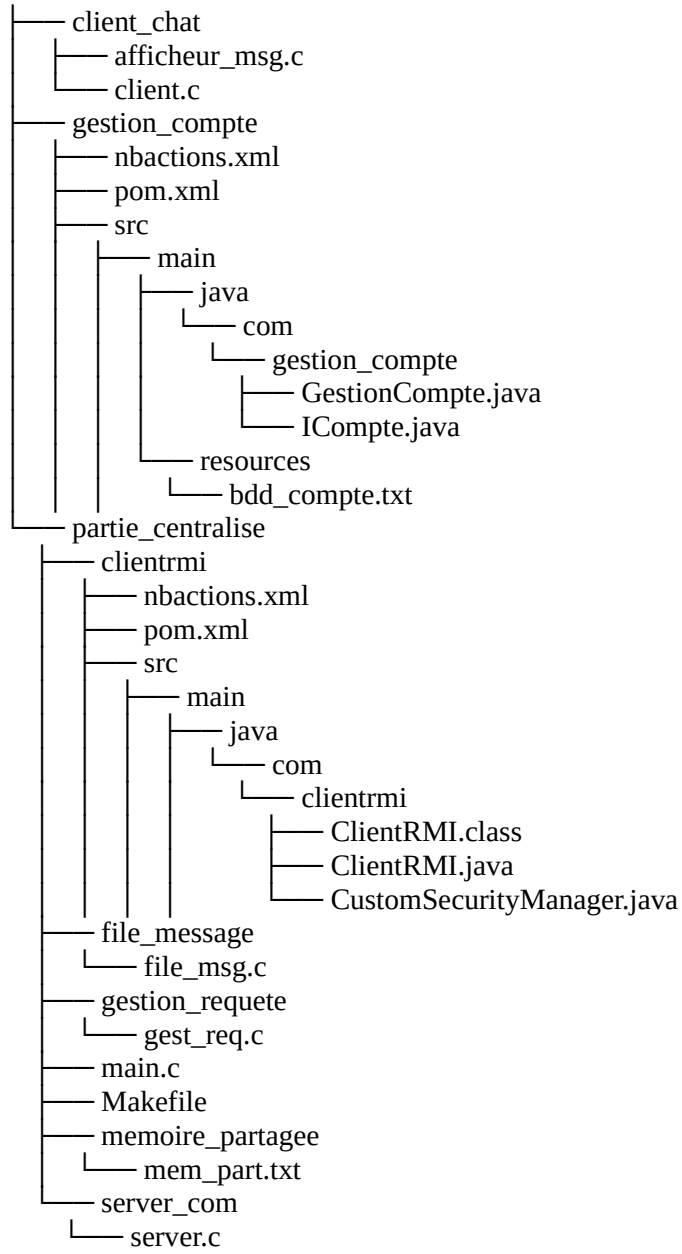
COMPILATION ET EXÉCUTION :

Organisation du dossier :

- Un makefile présent dans la partie centralise permet d'executer tout d'un coup avec la commande make run.
- Il faudra modifier dans le main les chemins absolus pour les commandes qui compilent et executent le clientRMI (Compile Classpath & Runtime Classpath de netbeans)
- La commande make run_client permet de compiler et lancer un client.
- La commande make clean_pipes permet de supprimer tous fichiers de pipes
- La commande make clean permet de supprimer les pipes et les executables.
- Le serveur GestionCompte sera exécuter depuis NetBeans.
- Lors du lancement de la partie centrale, il faudra renseigner l'adresse IP du serveur - - GestionCompte et le port de communication entre les deux.
- Lors de l'exécution d'un client, il faudra renseigner l'adresse ip du serveur et le port du serveur

---

tree :


client_chat
afficheur_msg.c
client.c
gestion_compte
nbactions.xml
pom.xml
src
main
java
com
gestion_compte
GestionCompte.java
ICompte.java
resources
bdd_compte.txt
partie_centralise
clientrmi
nbactions.xml
pom.xml
src
main
java
com
clientrmi
ClientRMI.class
ClientRMI.java
CustomSecurityManager.java
file_message
file_msg.c
gestion_requete
gest_req.c
main.c
Makefile
memoire_partagee
mem_part.txt
server_com
server.c

FONCTIONNEMENT DE L'APPLICATION :

Résumé du projet :

Le projet consiste à développer un outil de chat permettant à plusieurs utilisateurs de dialoguer. Chaque utilisateur possède un compte avec un pseudo et un mot de passe. Une fois connecté, un utilisateur peut envoyer des messages à tous les autres utilisateurs connectés. L'architecture comprend des éléments en Java pour la gestion des comptes et en C pour la communication entre les utilisateurs.

Le projet est lancé sur plusieurs machines. La paartie central comporte 4 terminaux, avec en exécution le serveur communication qui permet la communication entre le client et la partie centrale (en tcp), la file message qui fait l'intermédiare entre le serveur communication et le gestion requete, le gestion requete qui fait l'intermediaire entre file message et le client RMI (en UDP) et le client RMI qui se connecte au serveur Gestion Compte qui est sur une autre machine. Ce serveur java permettra la gestion des comptes et contiendra aussi toutes les infos sur les comptes présents.

Pour commencer parlons de la partie client_chat. Le terminal du client (client.c) affiche un menu, avec lequel l'utilisateur peut interagir pour choisir différents choix. Dans un premier temps, il peut se connecter, créer un compte, supprimer un compte existant ou quitter le client. Pour créer un compte, l'utilisateur sélectionne l'option correspondante et saisit un pseudo unique ainsi qu'un mot de passe d'au moins quatre caractères. De la même façon, il peut supprimer un compte en fournissant les bonnes informations d'identification. Aussi, une liste de comptes préexistants est stockée dans le dossier « ressources » du gestionnaire de compte (GestionCompte.java). Ce qui permet aux utilisateurs de se connecter en utilisant soit leurs identifiants déjà existants soit ceux nouvellement créés.

Une fois connecté, l'utilisateur a accès à quatre options principales : écrire un message, afficher la liste des utilisateurs connectés, se déconnecter ou quitter le client. Chaque requête effectuée par l'utilisateur est identifiée par un préfixe spécifique, ce qui facilite la vérification des données. Lorsque l'utilisateur choisit d'écrire un message, cela ouvre un afficheur de messages (afficheur_msg.c) qui affiche toutes les interactions entre les différents utilisateurs connectés. Celui-ci fonctionne grâce à un pipe nommé unique pour chaque client. Ce visualiseur se ferme lorsque l'utilisateur se déconnecte. La liste des utilisateurs connectés permet de visualiser comme son nom l'indique les utilisateurs connectés.

Dans la partie centralise de l'application, le serveur joue un rôle crucial en assurant toutes les interactions entre les clients via une socket TCP. De plus, il gère les préfixes nécessaires à l'envoi des messages vers le file message en fonction des choix effectués par l'utilisateur. On ouvrira un thread par client qui se connecte pour gérer en parralèlle plusieurs client à la fois (jusqu'à 10). Lors de la fermeture d'un socket, si il était détenu par un client connecté, le serveur s'occupera d'envoyer la demande de déconnexion (et donc de suppression de la mémoire partagé) au gestionnaire requete.

Le file message (file_msg.c), quant à lui, est constamment à l'écoute des messages provenant du serveur. Dés lors qu'il reçoit un message, il s'occupera de le transmettre et de renvoyer la réponse au bon thread du serveur en utilisant le thread id. Il agit comme un canal de communication centralisé pour la transmission des messages entre les différents composants du système et implémente une file d'attente si trop de thread du serveur envoie des messages en même temps.

Le gestionnaire de requêtes (gest_req.c) est responsable du traitement des requêtes émises par les utilisateurs et il fonctionne grâce à notre système de préfixe. De plus, comme je l'ai indiqué plus haut, le processus communication et le gestionnaire de requêtes gèrent une mémoire partagée pour maintenir une liste actualisée des utilisateurs connectés au système. Cette mémoire partagée permet une coordination efficace

entre les deux processus pour garantir que la liste des utilisateurs est toujours à jour. Le gestionnaire requete sera donc responsable d’ajouter ou de supprimer de la mémoire partagé les pseudos des utilisateurs en fonction de leurs actions et des messages reçu. De plus, le gestionnaire de requêtes est relié au client RMI par une socket UDP (port 4003).

En effet, le client RMI (ClientRMI.java) agit comme une passerelle entre l'interface utilisateur Java et les fonctionnalités fournies par le serveur gestion_requête.c au quel il se connecte en RMI. Il permet aux utilisateurs de se connecter, de créer et de supprimer des comptes.

Mais avant de parler de la dernier partie, gestion compte (GestionCompte.java), Le processus principal (main.c) est responsable de lancer tous les processus de la partie centralisée du système en ouvrant autant de terminaux que nécessaires. Sa fonction principale est d'initier et de coordonner les différents processus en créant et initialisant la mémoire partagée ainsi que la file de messages. De plus, il est chargé de garantir une fermeture ordonnée et propre de tous les éléments de la partie centralisée lorsque cela est nécessaire dont la fermeture des pipes par exemple.

Pour finir le GestionCompte, est l’implémentation de l’interface Icompte. Il permet d’accéder à la base de données contenant toutes les infos sur les comptes et est le seul à pouvoir les modifier.

En résumé, une fois le projet lancé, une infrastructure complexe de terminaux, de processus, de socket, de thread, de communication et etc est mise en place pour permettre aux utilisateurs d'interagir avec le système de chat. Chaque composant joue un rôle spécifique dans l'ensemble du système, depuis la gestion des comptes utilisateur jusqu'à la transmission des messages entre les utilisateurs connectés. La coordination entre les différents processus, qu'ils soient exécutés en C ou en Java, est essentielle pour assurer le bon fonctionnement de l'application de chat distribué.