Polytech Sorbonne
Electronique et Informatique

Réseaux
COMPTE RENDU

Rapport Projet : Système Bancaire avec Sockets TCP/UDP

Nom & Prénom : IDRI Liza

Année Universitaire : 2024/2025

Contents

1 Introduction

L'objectif de ce projet est de créer une application client-serveur permettant de gérer des comptes bancaires en utilisant des sockets. Le serveur est responsable de la gestion des comptes bancaires, notamment la gestion des dépôts, retraits, consultations des soldes et des opérations récentes. Le client interagit avec le serveur en envoyant des requêtes pour effectuer ces actions et reçoit des réponses appropriées.

Deux types de sockets sont utilisés dans ce projet : les sockets TCP et UDP. Le protocole TCP offre une communication fiable et garantit que les données sont bien reçues et dans le bon ordre, tandis que le protocole UDP est plus léger et plus rapide, mais n'assure pas la fiabilité de la transmission.

2 Gestion des systèmes d'exploitation

L'application est conçue pour être compilée et exécutée sur des systèmes d'exploitation Windows et Linux. Le code utilise des directives de préprocesseur (#ifdef _WIN32 et #else) pour inclure les bibliothèques et les fonctions spécifiques à chaque système d'exploitation.

Code pour Windows :
Sur Windows, les sockets sont gérés par la bibliothèque winsock2. Avant d'utiliser les sockets, il est nécessaire d'initialiser cette bibliothèque avec la fonction WSAStartup(), et une fois le programme terminé, de la nettoyer avec WSACleanup().

Code pour Linux :
Sur Linux, les sockets sont gérés par la bibliothèque ¡sys/socket.h¿. Aucune initialisation particulière des sockets n'est requise sur Linux. Cependant, il faut utiliser la fonction close() pour fermer une socket, contrairement à closesocket() sur Windows.

3 Implémentation du système bancaire

Le système bancaire implémenté dans ce projet est basé sur une structure de données qui représente des clients et leurs comptes bancaires.

Chaque client peut effectuer des actions comme un dépôt, un retrait, consulter le solde de son compte, ou obtenir l'historique des opérations. Ces fonctions permettent de traiter les commandes depuis le serveur.

3.1 Structure des données Client

La structure Client contient toutes les informations nécessaires à la gestion d'un compte bancaire :

id_client : Un identifiant unique pour chaque client.

password : Le mot de passe du client.

solde : Le solde du compte bancaire associé au client.

operations : Un tableau à deux dimensions qui stocke les 10 dernières opérations (représentées par une chaîne de caractères).

nb_operations : Le nombre d'opérations effectuées sur le compte.

3.2 Initialisation

Lors de l'initialisation du serveur, un certain nombre de clients avec des soldes et des informations fictives sont créés. est créée.

Cette liste permet de remplir la base de données avec des clients pré-existants. Les clients peuvent alors effectuer des actions sur leurs comptes.

```
Client 1 initialise, mot de passe : password1, solde : 1000.00
Client 2 initialise, mot de passe : password2, solde : 1000.00
Client 3 initialise, mot de passe : password3, solde : 1000.00
Client 4 initialise, mot de passe : password4, solde : 1000.00
Client 5 initialise, mot de passe : password5, solde : 1000.00
Client 6 initialise, mot de passe : password6, solde : 1000.00
Client 7 initialise, mot de passe : password7, solde : 1000.00
Client 8 initialise, mot de passe : password8, solde : 1000.00
Client 9 initialise, mot de passe : password9, solde : 1000.00
Client 10 initialise, mot de passe : password10, solde : 1000.00
```

Figure 1: Initialisation des clients

3.3 Fonctions

Fonctions ajouter_somme() : Permet d'ajouter une somme spécifiée à un compte.
retirer_somme() : Permet de retirer une somme spécifiée d'un compte, à condition que le solde soit suffisant.
consulter_solde() : Permet de consulter le solde d'un compte.
consulter_operations() : Permet de consulter les 10 dernières opérations effectuées sur un compte.

Ces fonctions sont utilisées dans le serveur pour traiter les requêtes des clients.

4 Architecture du système

4.1 Serveur TCP

Le serveur TCP écoute sur un port donné en utilisant la fonction bind() pour lier la socket à l'adresse IP et au port spécifiés. Il attend les connexions entrantes via la fonction listen() et accepte les connexions avec accept().

```
Initialisation du serveur ...
Serveur initialise et en ecoute sur le port : 8080
Adresse IP serveur : 0.0.0.0
```

Figure 2: Initialisation du serveur

Une fois la connexion établie, le serveur reçoit des commandes des clients et traite les requêtes. Selon le type de commande (AJOUT, RETRAIT, SOLDE, OPERATIONS), le serveur met à jour les informations du compte bancaire ou renvoie des données comme le solde ou les opérations (via les fonctions de la lib banque.c).

Principales étapes pour le serveur TCP :
socket() : Crée la socket du serveur.
bind() : Lie la socket à un port.
listen() : Attend des connexions.
accept() : Accepte une connexion entrante.
recv() et send() : Reçoit les commandes et envoie les réponses.

4.2 Serveur UDP

Le serveur UDP écoute également sur un port donné, mais contrairement à TCP, il n'établit pas de connexion. Il utilise une socket de type SOCK_DGRAM. Le serveur reçoit les messages des clients via la fonction recvfrom() et envoie une réponse via la fonction sendto(). UDP étant sans connexion, le serveur ne doit pas appeler accept().

Les principales étapes effectuées par le serveur UDP :
socket() : Crée la socket du serveur.
bind() : Lie la socket à un port pour recevoir des messages.
recvfrom() : Reçoit les commandes des clients (Et récupère l'adresse du client pour pouvoir lui répondre).
sendto() : Envoie une réponse au client à l'adresse qu'il a utilisée pour envoyer la commande.

4.3 Client TCP

Le client TCP crée une socket en utilisant le protocole TCP, se connecte au serveur en utilisant la fonction connect() et envoie des requêtes à l'aide de send().

Le client attend ensuite une réponse du serveur via recv(). Une fois la réponse reçue, elle est affichée à l'utilisateur.

```
Initialisation du client ...
Connexion ├®tablie avec le serveur.
Entrez la commande (ex: AJOUT <id_client> <id_compte> <password> <somme>) :
```

Figure 3: Initialisation du client

Etapes effectuées par le client :
socket() : Crée la socket du client.
connect() : Établit une connexion avec le serveur en utilisant son adresse IP et son port.
send() : Envoie des commandes au serveur.
recv() : Reçoit les réponses du serveur.
close() : Fermer la socket à la fin.

4.4 Client UDP

Le client UDP envoie des messages via la fonction sendto() à une adresse IP et un port donnés. Il attend ensuite une réponse avec recvfrom(). Cependant UDP ne nécessite pas d'établir une connexion.

Etapes effectuées par le client :
socket() : Crée la socket du client (protocole UDP).
sendto() : Envoie une commande au serveur en précisant son adresse IP et son port.
recvfrom() : Reçoit la réponse du serveur (Et récupére l'adresse du serveur).
close() : Fermer la socket à la fin.

5 Commandes et Réponses

Les commandes envoyées par le client :
Ces tests ont été effectués de la même manière sur les deux protocoles TCP et UDP.

5.1 Commande AJOUT

```
Entrez la commande (ex: AJOUT <id_client> <id_compte> <password> <somme>) : AJOUT 1 1 password1 200
Requete envoyee avec succes : (24 octets)
Reponse du serveur : OK
```

Figure 4: Exemple de commande d'ajout

La réponse reçu par le serveur est bien "OK" pour informer que l'opération s'est bien déroulée.

5.2 Commande SOLDE

```
Entrez la commande (ex: AJOUT <id_client> <id_compte> <password> <somme>) : SOLDE 1 1 password1
Requete envoyee avec succes : (21 octets)
Reponse du serveur : RES_SOLDE 1200.00
```

Figure 5: Exemple de commande SOLDE

La réponse du serveur est sous forme "RES_SOLDE somme".

5.3 Commande RETRAIT

```
Entrez la commande (ex: AJOUT <id_client> <id_compte> <password> <somme>) : RETRAIT 1 1 password1 1500
Requete envoyee avec succes : (27 octets)
Reponse du serveur : KO
```

Figure 6: Exemple de commande de retrait

Dans ce cas, le solde du compte est de 1200. La commande de retrait a donc échoué.
La réponse du serveur "KO" indique que la commande n'a pas pu être effectuée.

5.4 Commande OPERATIONS

```
Entrez la commande (ex: AJOUT <id_client> <id_compte> <password> <somme>) : OPERATIONS 1 1 password1
Requete envoyee avec succes : (26 octets)
Reponse du serveur :
RES_OP - Depot : +200.00
Depot : +500.00
Retrait : -100.00
```

Figure 7: Exemple de commande OPERATIONS

Le serveur renvoie une réponse sous forme de "RES_OP" + liste des 10 dernières opérations.

6 Conclusion

Ce projet a permis d'implémenter une application client-serveur pour la gestion de comptes bancaires, utilisant des communications via des sockets TCP et UDP.

Le serveur a été conçu pour traiter différentes requêtes bancaires, telles que l'ajout ou le retrait d'argent, la consultation des soldes, et l'affichage des historiques de transactions.
Le client, quant à lui, permet de formuler ces commandes, tout en recevant des réponses appropriées du serveur en fonction de l'exécution des commandes.

L'utilisation de TCP et UDP a mis en lumière leurs avantages et inconvénients dans ce type d'architecture :
TCP (Transmission Control Protocol) :
Avantages : Fiabilité accrue grâce au contrôle d'erreurs, garantie d'une livraison ordonnée des données et retransmission automatique des paquets perdus. Cela le rend idéal pour des opérations bancaires critiques où l'intégrité des données est primordiale.
Inconvénients : Plus lent que l'UDP en raison de l'établissement préalable d'une connexion et de la surcharge liée à la gestion des accusés de réception.

UDP (User Datagram Protocol) :
Avantages : Rapidité et faible surcharge, adapté pour des environnements où les performances en temps réel sont prioritaires, et où une certaine tolérance à la perte de paquets est acceptable.
Inconvénients : Absence de garantie de livraison des données et désordre potentiel des paquets, ce qui peut poser problème pour des transactions bancaires nécessitant une haute fiabilité.