Polytech Sorbonne Université

Réseaux EI4

Système client-serveur via socket en C

Rapport

Gestion de Comptes Bancaires

Élève(s) :
Daniel FERREIRA LARA

Enseignant(s) :
Maria POTOP-BUTUCARU
Francesca FOSSATI

8 janvier 2025

Table des matières

1 Introduction

Ce rapport décrit le projet développé pour le cours de réseaux dans le cadre du programme EISE4. Pour une meilleure compréhension, j'expliquerai l'architecture générale du projet, puis je décrirai le code TCP et UDP respectivement, et je présenterai les résultats.

Ce développement a été documenté et est disponible sur github, à partir du lien ci-dessous :

1. **Repo GitHub :** Project Repo

Il convient de noter que le code développé est basé sur les références suivantes : [4], [1], [2], [3]. Pour corriger les bogues et les erreurs de compilation, j'ai utilisé des forums pour les résoudre.

2 Définition de l'Architecture

L'architecture logicielle proposée est une relation client-serveur, avec une relation N-1, c'est-à-dire plusieurs clients pour le même serveur. Les fonctionnalités proposées dans la feuille de route ont été mises en œuvre et, en plus, des mesures de sécurité ont été créées, telles que l'option de validation de l'origine de la demande.

Les 4 méthodes décrites ci-dessous ont été développées :

1. **AJOUT :** <id_client id_compte password somme>
2. **RETRAIT :** <id_client id_compte password somme>
3. **SOLDE :** <id_client id_compte password>
4. **OPERATIONS :** <id_client id_compte password>

Afin de créer une application solide, j'ai développé 3 bibliothèques : bank, serveur et client. La première concerne les fonctions et les modèles de données qui sont partagés par les autres. Voir la structure des fichiers :

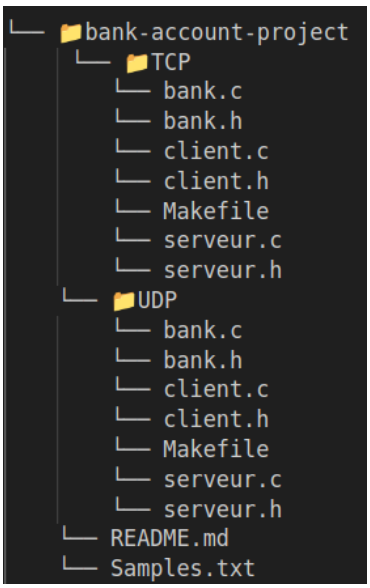

FIGURE 1 – Structure des dossiers

Quant à la bibliothèque Bank, elle définit la manière correcte d'appeler les fonctions de socket pour les deux protocoles. Parmi ses définitions, on peut citer les modèles de client, de compte et d'opérations, ainsi que les fonctions générales read_socket, write_socket et disconnect_socket. Il décrit également les méthodes d'opération et les types de réponse, ainsi que le nombre maximum de clients, le port du serveur et une fonction de date et d'heure locales. Pour les traiter comme des nombres entiers, toutes les valeurs de solde sont traitées x100 pour utiliser deux décimales plus simplement.

Les fonctions génériques correspondent aux structures suivantes des deux protocoles, TCP à gauche et UDP à droite :

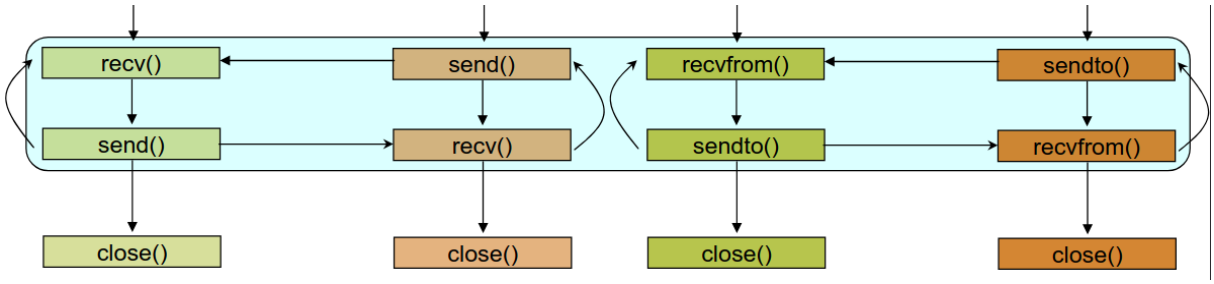

FIGURE 2 – Figure de la page 30 de [4]

Les comptes clients suivants ont été créés en standard :

```
int num_comptes = 5;
Compte comptes[5] = {
    {"JeanDupont", "00112345", "mdpJean", 150000,
        {{"AJOUT", "01/01/2023 10:00:00", 150000}}
        , 1
    },
    {"MarieCurie", "00223456", "mdpMarie", 200000,
        {{"AJOUT", "02/02/2023 11:00:00", 200000}, {"RETRAIT", "03/03/2023 12:00:00", -50000}}
        , 2
    },
    {"PierreDurand", "00334567", "mdpPierre", 300000,
        {{"AJOUT", "04/04/2023 13:00:00", 300000}, {"RETRAIT", "05/05/2023 14:00:00", -100000}, {"AJOUT", "06/06/2023 15:00:00", 50000}}
        , 3
    },
    {"LucieMartin", "00445678", "mdpLucie", 250000,
        {{"AJOUT", "07/07/2023 16:00:00", 250000}, {"RETRAIT", "08/08/2023 17:00:00", -75000},
         {"AJOUT", "09/09/2023 18:00:00", 100000}, {"RETRAIT", "10/10/2023 19:00:00", -50000}}
        , 4
    },
    {"DanielLara", "00556789", "mdpDaniel", 100000,
        {{"AJOUT", "11/11/2023 20:00:00", 100000}, {"RETRAIT", "12/12/2023 21:00:00", -25000},
         {"AJOUT", "13/01/2024 22:00:00", 50000}, {"RETRAIT", "14/02/2024 23:00:00", -10000}, {"AJOUT", "15/03/2024 00:00:00", 20000}}
        , 5
    }
};
```

FIGURE 3 – Comptes

Après avoir défini tout ce qui était commun, je suis passé à l'implémentation du serveur et du client pour chaque protocole. Voici les sections qui expliquent leur fonctionnement et la manière dont je les ai implémentés.

3 Mise en œuvre du protocole TCP

3.1 Processus

Le protocole TCP est basé sur une connexion fiable entre le client et le serveur, qui repose sur une "poignée de main" entre les deux. Dans cette optique, il est nécessaire

que la connexion soit acceptée avant que les messages ne soient envoyés, comme nous le verrons plus loin. En outre, chaque client dispose de son propre socket pour communiquer avec le serveur.

L'image ci-dessous illustre les étapes nécessaires, ainsi que les fonctions C existantes que j'ai utilisées.

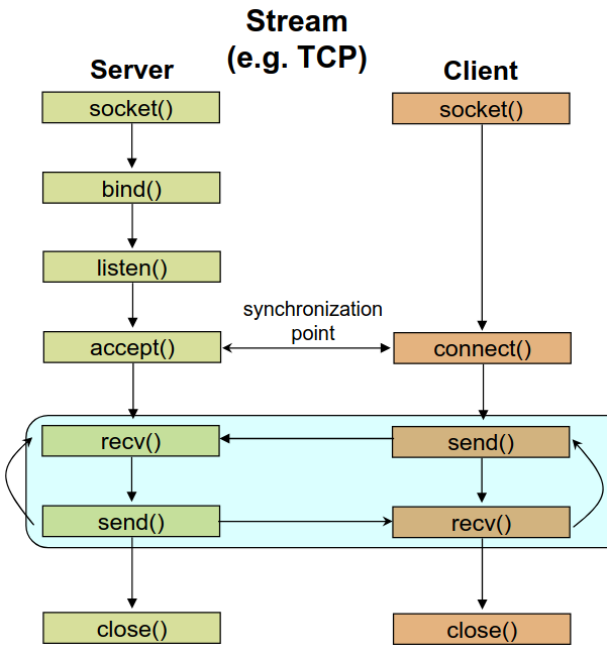

FIGURE 4 – Figure de la page 30 de [4]

Côté serveur :

1. Le serveur crée un socket d'écoute avec `socket()`
2. Il associe le socket à une adresse IP et un port via `bind()`
3. Il se met en écoute des connexions avec `listen()`
4. Il accepte les nouvelles connexions avec `accept()` qui crée un nouveau socket dédié à chaque client
5. Il peut alors échanger des données avec `send()` et `recv()`

Côté client :

1. Le client crée un socket avec `socket()`
2. Il se connecte au serveur avec `connect()` en spécifiant l'IP et le port
3. Une fois connecté, il peut échanger des données avec `send()` et `recv()`

La fermeture de connexion peut être initiée par l'un ou l'autre avec `close()`.

3.2 Serveur

Le serveur TCP possède d'autres fonctions qui sont regroupées dans la fonction run_server, qui est responsable de ce qui suit :

`run_server(Compte *comptes, int num_comptes)` : Basé sur les comptes clients existants, le serveur se connecte au port défini 1234, crée des variables pour stocker les

clients, les tampons, etc. Il entre ensuite dans une boucle infinie, utilisant la fonction `select()` pour gérer plusieurs clients simultanément. Il commence par initialiser un ensemble de descripteurs de fichiers (`rdfs`) pour tous les sockets clients existants. Si une nouvelle connexion est détectée sur le socket principal (`FD_ISSET(_socket, &rdfs)`), le serveur accepte la connexion, initialise un nouveau client avec les informations de connexion, lui envoie un message de bienvenue avec les commandes disponibles, et diffuse l'arrivée du nouveau client aux autres clients. Si une activité est détectée sur un socket client existant (`FD_ISSET(clients[j]._socket, &rdfs)`), le serveur lit le message reçu et soit traite la déconnexion du client (en le retirant de la liste et en informant les autres), soit traite la commande reçue via `handle_command()`. La boucle continue jusqu'à ce qu'une entrée soit détectée sur STDIN (pour arrêter le serveur), après quoi elle nettoie toutes les connexions et termine proprement.

Voici comment les fonctions ont été mises en œuvre :

- `server_connect()` : Établit la connexion du serveur, crée un socket, si le socket est valide, il libère la réception de n'importe quelle adresse, fait le `bind()` et commence à observer avec `listen()`
- `server_disconnect()` : Ferme la connexion du socket principale
- `read_client()` et `write_client()` : Gèrent les E/S avec les clients, simplement utilisent les fonctions `read_socket()` et `write_socket()`
- `broadcast_message()` : Diffuse les messages aux clients connectés avec un loop et la fonction `write_socket()`
- `remove_client()` : Supprime un client de la liste à partir d'une décalage de pointier
- `clear_clients()` : Nettoie la liste des clients en train de déconnecter tous
- `handle_command()` : Cette fonction est chargée de valider ou non la demande. Elle commence par séparer le tampon en différents éléments : la demande, l'identifiant du client, l'identifiant du compte, le mot de passe et la valeur. S'il est nécessaire de vérifier l'origine, le client qui envoie le message doit être le même que celui envoyé dans la demande, sinon il sera bloqué. Enfin, il vérifie si le compte existe et si le mot de passe est correct ; si c'est le cas, la demande sera traitée comme suit :
  - `AJOUT et RETRAIT` : Compte tenu de la valeur de l'opération, nous utilisons dans les deux cas la fonction auxiliaire add_operation pour ajouter les données à la liste des 10 dernières opérations et modifier le solde de ce compte en fonction de la requête.
  - `SOLDE` : Cette méthode prend la date de la dernière transaction et envoie le message avec le solde actuel du compte.
  - `OPERATIONS` : Cette fonction passe en revue toutes les fonctions du vecteur des dernières fonctions et imprime leurs informations, telles que l'index, le type, la date et la valeur.

3.3 Client

La fonction principale c'est `create_client(const char *address, const char *name)` : Si le serveur fonctionne, nous lui envoyons immédiatement le nom de l'utilisateur afin qu'il puisse enregistrer l'application du client dans sa liste.Si une entrée utilisateur est détectée (`FD_ISSET(STDIN_FILENO, &rdfs)`), elle est lue et envoyée au serveur. Si des données arrivent du serveur (`FD_ISSET(_socket, &rdfs)`), elles sont lues et affichées, sauf si la lecture retourne 0, indiquant une déconnexion du serveur. La boucle se poursuit jusqu'à

ce que le serveur se déconnecte, après quoi le client ferme proprement sa connexion et termine.

Les autres fonctions sont les suivantes :

— `client_connect()` : Établit la connexion du serveur, crée un socket, si le socket est valide, il définit l'adresse de l'hôte pour le socket et lance la connexion à l'aide de la méthode `connect()` de C.

— `client_disconnect()` : Ferme la connexion du socket principale avec `disconnect_socket()`

— `read_serveur()` et `write_serveur()` : Gèrent les E/S avec les serveurs, simplement utilisent les fonctions `read_socket()` et `write_socket()`

3.4 Tests

Voici les tests TCP. Test de toutes les méthodes sur une connexion d'un seul client au serveur :

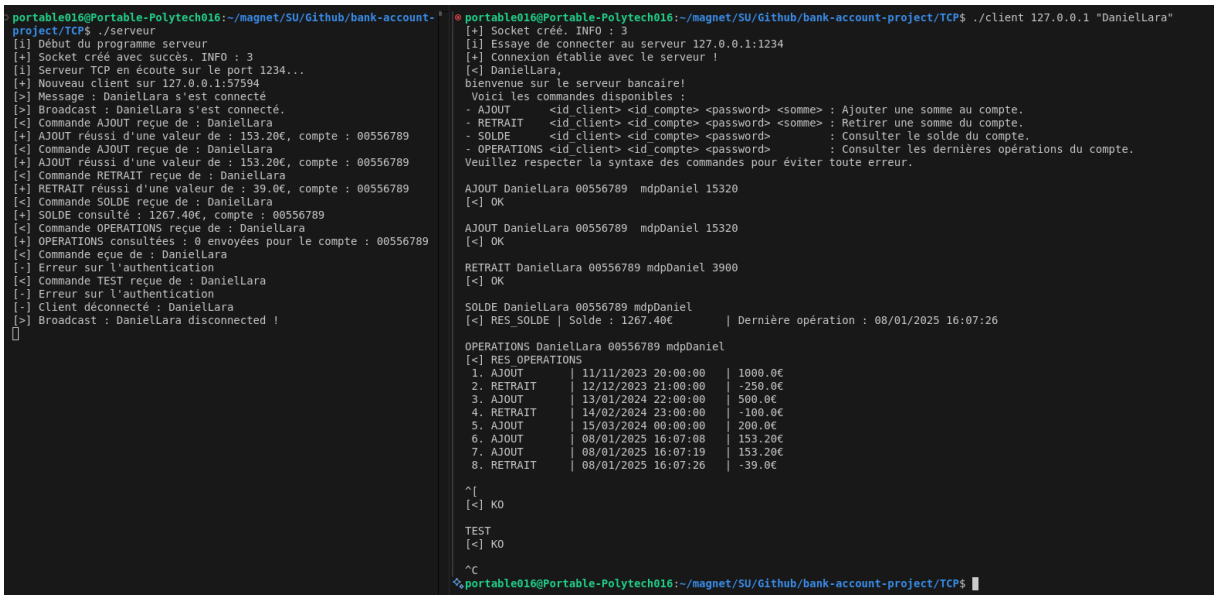

FIGURE 5 – Individuel

Test de gestion multi-clients avec connexions/déconnexions multiples :

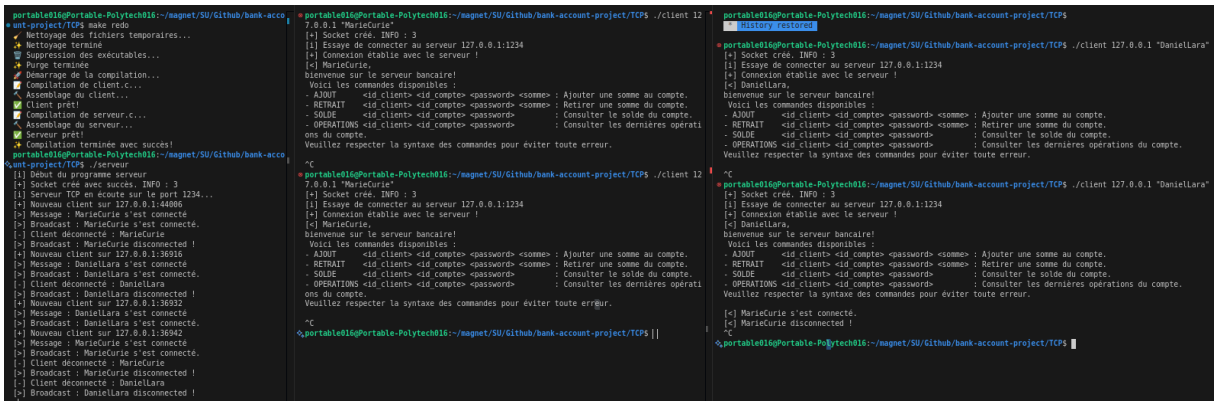

FIGURE 6

Côté serveur multi-connecté :

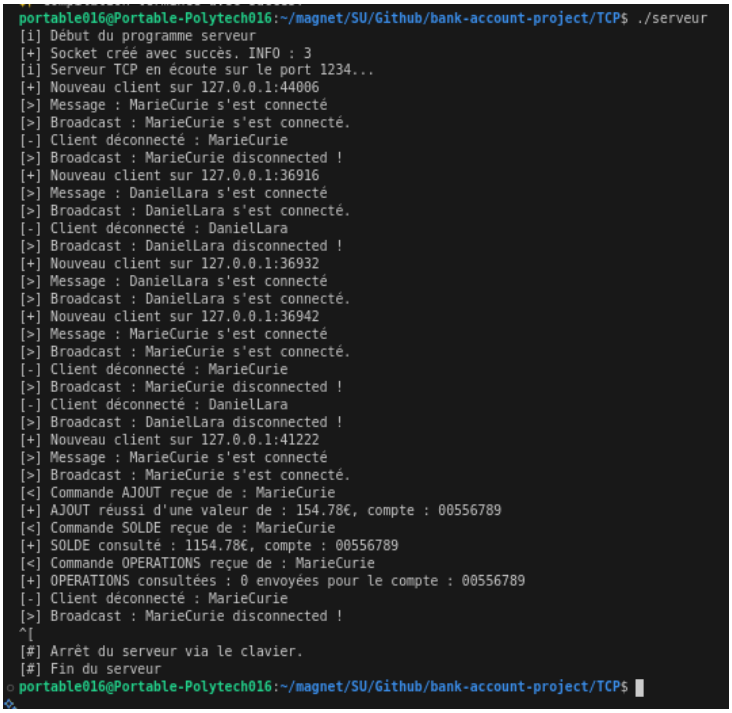

FIGURE 7 – Multi conn coté serveur

Fonctionnalité de validation de l'origine :

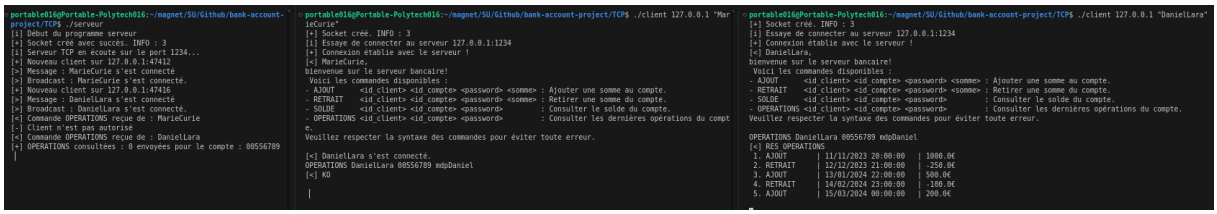

FIGURE 8 – Multi avec check$_{o}rigin$

4 Mise en œuvre du protocole UDP

4.1 Processus

Le protocole UDP est basé sur une communication sans connexion entre l'émetteur et le récepteur, ne nécessitant aucune "poignée de main" préalable. Dans cette optique, les datagrammes peuvent être envoyés directement sans établissement de connexion, comme nous le verrons plus loin. Les sockets UDP sont généralement partagés entre tous les clients communicant avec le serveur.

UDP fonctionne sur un modèle de "fire and forget" (envoyer et oublier), où chaque datagramme est indépendant et peut emprunter des chemins différents sur le réseau. Il n'y a pas de garantie de livraison, d'ordre ou d'intégrité des messages, contrairement à TCP.

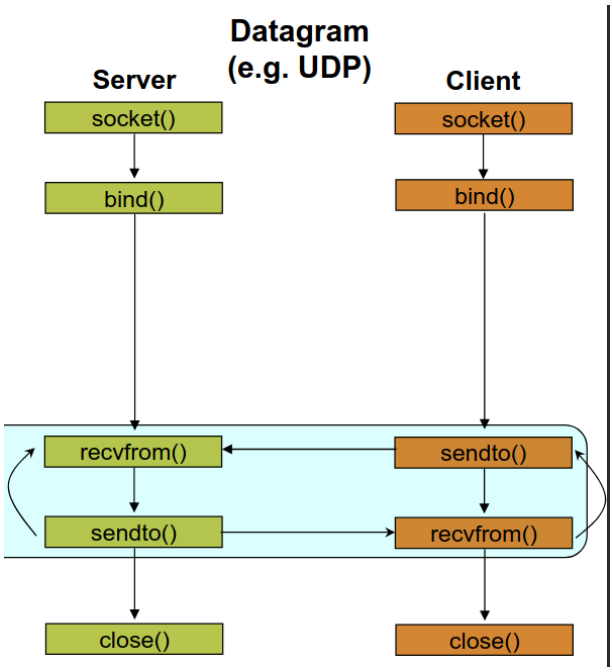

FIGURE 9 – Figure de la page 30 de [4]

Côté Client UDP :

1. Crée un socket avec `socket()` en mode UDP (SOCK_DGRAM)
2. Associe une adresse et un port avec `bind()`
3. Il va directement envoyer des datagrammes avec `sendto()` qui inclut l'adresse de destination
4. Peut recevoir des datagrammes avec `recvfrom()`
5. Aucune connexion n'est établie

Côté Serveur UDP :

1. Crée un socket avec `socket()` en mode UDP
2. Associe une adresse et un port avec `bind()`
3. Il va recevoir des datagrammes avec `recvfrom()`
4. Peut directement envoyer des datagrammes avec `sendto()` qui inclut l'adresse de destination

La fermeture de connexion peut être commencée par l'un ou l'autre avec `close()`.

4.2 Serveur

Pour éviter des répétitions explicatives inutiles qui ne font que faire perdre du temps au lecteur, je vais expliquer en quoi la mise en œuvre d'UDP diffère de celle de TCP, puisque cette dernière est basée sur ma mise en œuvre toute faite de la première.

— `run_server(Compte *comptes, int num_comptes)` : Le plus grand écart c'est qui quand une nouvelle connexion est détectée sur le socket principal (`FD_ISSET(_socket, &rdfs)`), il vais traiter toujours pour voir si c'est client il exist ou n'existe pas. Si la réponse de `client_exists()` est fausse, le client est créé et le message de bienvenue est envoyé, ainsi qu'une annonce de l'arrivée en broadcast. S'il existe déjà,

nous trouvons son index avec `get_client_index()`, si ce client est dans la liste, je traite sa commande. A la fin de la boucle, c'est la même chose.
— `server_connect()` : Établit la connexion du serveur, crée un socket, si le socket est valide, il libère la réception de n'importe quelle adresse, fait le `bind()` et c'est tout
— `read_client()` et `write_client()` : La différence est que le message est maintenant envoyé directement à l'adresse du client, qui a déjà été enregistrée dans l'objet du client, j'utilise toujours `read_socket()` et `write_socket()`
— `get_client_index()` : Il parcourt simplement la liste des clients et si les attributs de l'adresse du client sont identiques à l'adresse envoyée, il renvoie l'index
— `client_exists()` : Il parcourt simplement la liste des clients et si les attributs de l'adresse du client sont les mêmes que ceux de l'adresse envoyée, il renvoie True, sinon False.

4.3 Client

La différence sur le client est encore plus petite, la seule différence étant l'utilisation des méthodes `read_socket()` et `write_socket()` pour UDP, l'utilisation de datagrammes au lieu de streams et le fait que dans ce cas nous n'utilisons plus `connect()`.

4.4 Tests

Voici les tests UDP. Test de toutes les méthodes sur une connexion d'un seul client au serveur :

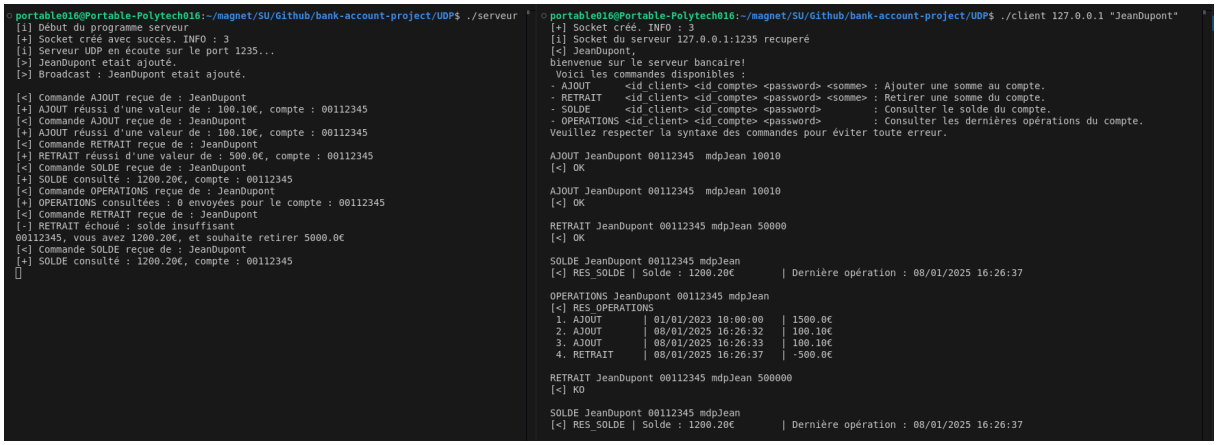

FIGURE 10 – Individuel

Serveur multi-connecté :

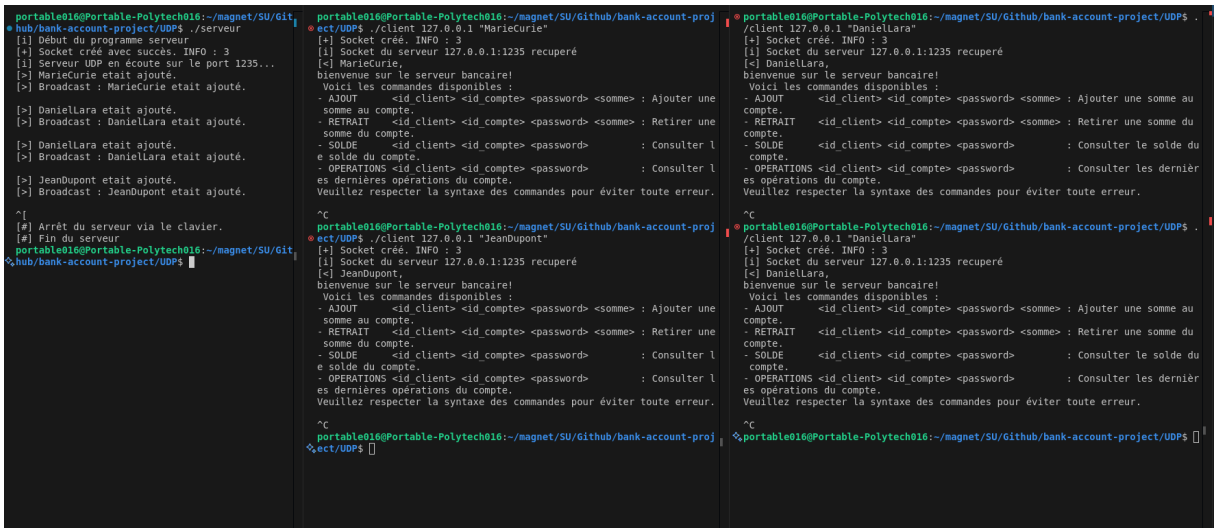

FIGURE 11 – Multi connections

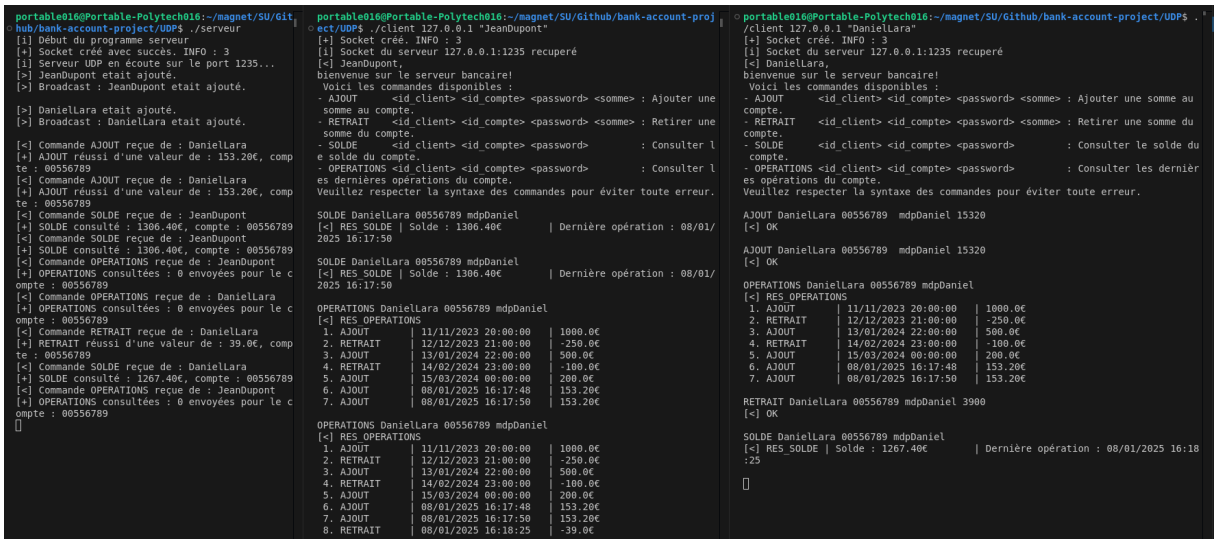

FIGURE 12 – Multi connections avec operations sans check$_{o}rigin$

Fonctionnalité de validation de l'origine avec plusiers utilisateurs :

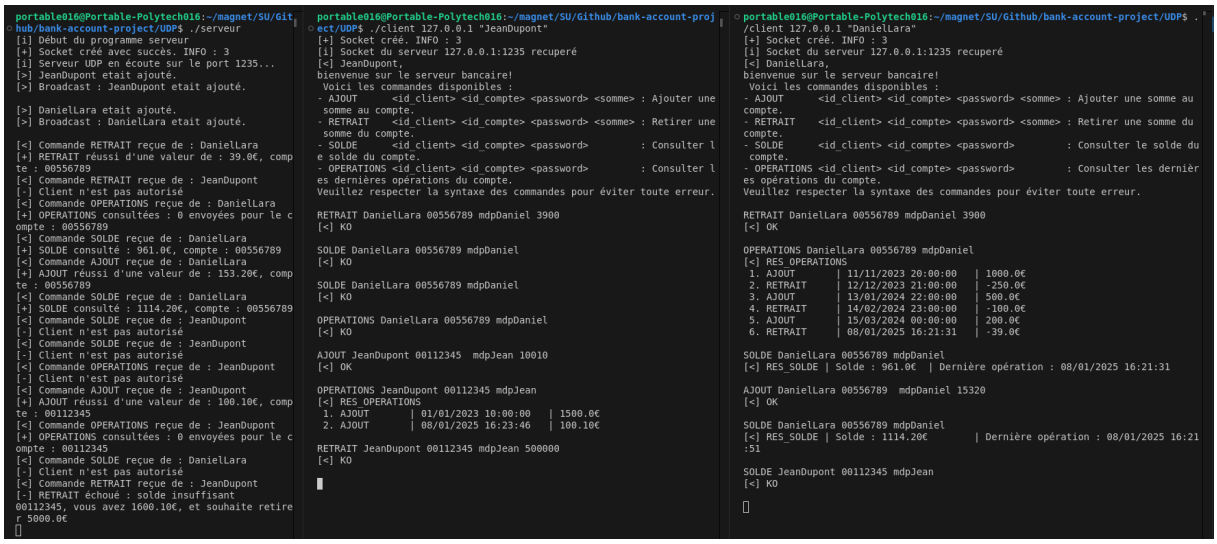

FIGURE 13 – UDP avec check origin

5 Conclusion

Ce travail a été très productif pour la mémorisation des concepts C, l'apprentissage des sockets et des réseaux à un niveau inférieur, ainsi que la manipulation des bibliothèques C. Je donc vous remercie d'avoir eu l'occasion de le développer.

Références

[1] Bogotobogo. Sockets serveur et client, 2020. Consulté le 04/01/2025.

[2] Broux. Sockets en c. https://broux.developpez.com/articles/c/sockets/#LV, 2007. Consulté le 04/01/2025.

[3] Emmanuel Delahaye. La gestion du temps en c. https://emmanuel-delahaye.developpez.com/tutoriels/c/notes-langage-c/?page=lttime-hgt-la-gestion-du-temps, 2009. Consulté le 04/01/2025.

[4] Eleftherios Kosmas. Presentation programation de socket en c. Technical report, Université de Crète. Consulté le 04/01/2025.