

Projet Reseau

Nathan Seignole Mathis Vermeren

December 31, 2023

1 Introduction

Ce code est un système bancaire fonctionnant en TCP et UDP, en multi-clients. Voici comment l'utiliser : Faire "make" dans ./UDP ou ./TCP

Lancer le serveur avec :
./server.out localhost PORT

Lancer le client avec :
./client.out localhost PORT username

Les comptes déjà créés s'affichent en console :

- Un utilisateur avec ID : 1 et mot de passe : chaton123
 - Deux comptes 101 et 102.
(changer dans le main.c en cas de besoins).
- On peut quitter le programme avec exit.

2 Gestionnaire de base de données (bank.c, structBank.h)

La base de données bancaire est définie dans le fichier `structBank.h`. Les différentes structures utilisées sont les suivantes :

```
typedef enum Operation_type{
    AJOUT,
    RETRAIT
} Operation_t;

typedef struct Operation
{
    char date[64];
    Operation_t type;
    float somme;
} Operation;

typedef struct {
    int account_id;
    float balance;
    Operation operations[MAX_OPERATIONS];
    int num_operations;
} Account;

typedef struct {
    SOCKET sock;
    char name[BUF_SIZE];
} Client;
```

```

typedef struct {
    int user_id;
    char password[MAX_PASSWORD_LENGTH];
    Account accounts[MAX_ACCOUNTS];
    int num_accounts;
} User;

```

La partie `bank.c` est identique en UDP et en TCP et réalise une interface entre le serveur et sa base de données. Les fonctions suivantes sont disponibles :

```

void initialize_user(User *users, int user_id, const char *password) : Ajoute un utilisateur.
void add_account(User *users, int user_id, int account_id, const char *password) : Ajoute un compte à un utilisateur.

```

Les quatre commandes (ajout, retrait, solde, opérations) ont chacune leur fonction respective prenant comme argument le tableau `User *users`, qui représente la base de données.

Il existe également une fonction `add_operation` qui, entre autres, prend un `Operation_t` (AJOUT ou RETRAIT) en argument pour l'ajouter à l'historique des opérations, représenté par le tableau `Operation operations[MAX_OPERATIONS]`.

Enfin, le fichier `server.c` utilise uniquement la fonction :

```

void process_command(User *users, char *buffer, char *res), qui prend la commande saisie par le client en entrée, la redirige vers la bonne fonction, et renvoie dans res un résultat ou une erreur.
Voici le corps de la fonction process_command :

```

```

void process_command(User *users, char *buffer, char *res) {
    char command[20];
    int user_id, id_compte;
    char password[MAX_PASSWORD_LENGTH];
    double somme = 0.0;
    int result = sscanf(buffer, "%s %d %d %s %lf", command, &user_id, &id_compte, password, &somme);
    printf("Scanned : %s %d %d %s %f\n", command, user_id, id_compte, password, somme);
    printf("Command : %s\n", command);
    if (result < 3) {
        strncpy(res, "KO : Le format de la commande est invalide", 255);
    }
    if (strcmp(command, "AJOUT") == 0 || strcmp(command, "RETRAIT") == 0) {
        if (result < 5) {
            strncpy(res, "KO : Pas suffisamment d'arguments pour AJOUT ou RETRAIT", 255);
        }
    }
    // Traiter la commande en fonction du type
    if (strcmp(command, "AJOUT") == 0) {
        ajout(users, user_id, id_compte, password, res, somme);
    } else if (strcmp(command, "RETRAIT") == 0) {
        retrait(users, user_id, id_compte, password, res, somme);
    } else if (strcmp(command, "SOLDE") == 0) {
        solde(users, user_id, id_compte, password, res);
    } else if (strcmp(command, "OPERATIONS") == 0) {
        operations(users, user_id, id_compte, password, res);
    } else {
        strncpy(res, "KO : Commande inconnue", 255);
    }
    printf("Response : %s\n", res);

    return;
}

```

3 Mise en place de la communication réseau

3.1 Initialisation de la connexion côté serveur

Que ça soit en UDP ou TCP, le programme commence par une fonction `void init(void)` qui nous sert juste à rendre le code compatible avec windows avec : `WSAStartup(MAKEWORD(2, 2), &wsa);`. Elle permet d'initialiser une DLL permettant d'utiliser les sockets. Pareillement, `void end(void)` permet de libérer cette même DLL avec `WSACleanup();`.

On entre ensuite dans `int init_connection(int PORT)`, elle crée un socket, configure son adresse et son port, le lie à l'adresse et au port spécifiés, puis le met en mode écoute (seulement en TCP) pour accepter les connexions entrantes. On créer le socket :

```
SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);
```

On utilise un protocole internet IPV4 (AF_INET), SOCK_STREAM est un type de communication TPC, on utilise SOCK_DGRAM en UDP, et 0 correspond au protocol par défaut.

La structure `sockaddr_in` (sin) est configurée pour spécifier l'adresse IP et le port auquel le serveur sera lié.

Configuration de la Structure `sockaddr_in` :

```
sin.sin_addr.s_addr = htonl(INADDR_ANY);  
sin.sin_port = htons(PORT);  
sin.sin_family = AF_INET;
```

`sin.sin_addr.s_addr` : L'adresse IP est configurée en utilisant `htonl` pour convertir l'adresse IP en format réseau.

`sin.sin_port` : Le numéro de port est configuré en utilisant `htons` pour convertir le numéro de port en format réseau.

`sin.sin_family` : La famille d'adresses est spécifiée comme étant IPv4.

Liaison du Socket

```
bind(sock, (SOCKADDR *) &sin, sizeof sin);
```

Cette étape associe une adresse IP et un numéro de port au socket créé.

Mise en Mode Écoute (TCP)

```
listen(sock, MAX_CLIENTS);
```

Le socket est mis en mode écoute pour accepter les connexions entrantes, avec `MAX_CLIENTS` spécifiant le nombre maximal de connexions en attente dans la file d'attente.

3.2 Fonctions clés de la communication réseau :

La fonction `accept` est utilisée pour accepter une nouvelle connexion TCP. Elle prend le socket d'écoute (`sock`), crée un nouveau socket dédié à la communication avec le client (`csock`), et récupère les informations du client telles que son adresse IP et le port utilisé.

```
int csock = accept(sock, (SOCKADDR *)&csin, &sinsize);
```

La fonction `send` est utilisée pour envoyer des données sur un socket TCP. Elle prend en argument le descripteur de socket (`sock`), le tampon contenant les données (`buffer`), la longueur des données à envoyer (`length`), et des options supplémentaires (`flags`).

```
send(sock, buffer, length, flags);
```

La fonction `sendto` est utilisée pour envoyer des données sur un socket UDP avec spécification du destinataire. Elle prend en argument le descripteur de socket (`sock`), le tampon contenant les données (`buffer`), la longueur des données à envoyer (`length`), des options supplémentaires (`flags`), et les informations sur le destinataire (`to`).

```
sendto(sock, buffer, length, flags, (SOCKADDR *)&to, sizeof to);
```

La fonction `recv` est utilisée pour recevoir des données depuis un socket TCP. Elle prend en argument le descripteur de socket (`sock`), le tampon où stocker les données reçues (`buffer`), la longueur maximale des données à recevoir (`length`), et des options supplémentaires (`flags`).

```
recv(sock, buffer, length, flags);
```

La fonction `recvfrom` est utilisée pour recevoir des données depuis un socket UDP avec spécification de l'expéditeur. Elle prend en argument le descripteur de socket (`sock`), le tampon où stocker les données reçues (`buffer`), la longueur maximale des données à recevoir (`length`), des options supplémentaires (`flags`), et les informations sur l'expéditeur (`from`).

```
recvfrom(sock, buffer, length, flags, (SOCKADDR *)&from, &fromlen);
```

3.2.1 Gestion des Entrées/Sorties dans un Serveur TCP

Pour gérer les entrées/sorties (E/S) dans un serveur TCP, la fonction `select` et les macros `FD_SET` et `FD_ISSET` sont couramment utilisées.

La fonction `FD_SET` permet d'ajouter des descripteurs de fichiers à un ensemble. Dans le contexte d'un serveur, on peut ajouter le descripteur du clavier (`STDIN_FILENO`) et le descripteur du socket (`sock`) à l'ensemble `rdfs`.

```
FD_SET(STDIN_FILENO, &rdfs);
FD_SET(sock, &rdfs);
```

La fonction `select` permet d'attendre que l'un des descripteurs de fichiers spécifiés soit prêt en lecture ou en écriture. Elle prend en compte trois ensembles : `readfds`, `writelfds`, et `exceptfds`. Dans le contexte du serveur, on s'intéresse principalement à `readfds`, donc `select` surveille les descripteurs en lecture.

```
select(sock + 1, &rdfs, NULL, NULL, NULL);
```

La macro `FD_ISSET` permet de tester si un descripteur de fichier spécifique est prêt en lecture ou en écriture dans un ensemble donné. Dans le contexte du serveur, on teste si l'activité provient du clavier (`STDIN_FILENO`) ou du socket (`sock`).

```
if (FD_ISSET(STDIN_FILENO, &rdfs)) {
    // Activité sur le clavier
}

if (FD_ISSET(sock, &rdfs)) {
    // Activité sur le socket
}
```

Lorsqu'un nouveau client se connecte, son descripteur de socket (`csock`) peut être ajouté à l'ensemble `rdfs` pour surveiller son activité.

```
FD_SET(csock, &rdfs);
```

Cela permet de détecter si le client envoie des données.

3.3 Différence entre l'UDP et le TCP

UDP n'utilisant pas de connexion, on ne trouvera de fonction `listen` ou `accept`. Cependant, on ne peut plus utiliser les fonctions `send` ou `recv`. Dans notre base de données server, on ne stocke plus les descripteurs de socket des clients (`SOCKET` dans le code) mais leurs `SOCKADDR_IN`, qui des IP et des ports. C'est pourquoi il a fallu modifier une ligne de notre `structBank.h`, qui devient :

```

typedef struct {
    SOCKADDR_IN sin;
    char name[BUF_SIZE];
} Client;

```

On utilise alors d'autres fonctions d'envoie et de reception qui prennent SOCKADDR_IN qui sont `sendto` ou `recvfrom`.

3.4 Communication reseau côté Client

L'initialisation de la communication est quasiment la même que côté serveur. Voici la fonction qui permet de lire et d'envoyer des commandes au serveur.

```

void command_input(SOCKET sock){
    char input[256];
    printf("Enter a command : ");
    memset(input,0,256);
    fgets(input,255,stdin);

    //exit command
    if(!strcmp(input, "exit\n"))
        return;
    write_server(sock, input);
    memset(input,0,256);
    read_server(sock, input);
    printf("Read : %s\n", input);
}

```

`Command_input` est mis dans une boucle `while(1)` où on écoute les entrées clavier avec `FD_ISSET(STDIN_FILENO, &rdfs)` ou les paquets arrivant avec `FD_ISSET(sock, &rdfs)`. Cette fonction est la même en TCP et UDP. Comme côté Client on n'accepte pas de connexion et on n'écoute pas de port, on n'utilise ni `accept`, ni `listen` et seules les fonctions `wirte_client`, `read_client`, et le protocole de communication à l'initialisation de la socket (`SOCK_DGRAM` et `SOCK_STREAM`) diffèrent.

4 Captures d'écrans

```

Enter a command : ^C
mathis@mathis-HP-EliteBook-820-G2:/media/mathis/disque dur externe/Cloud/Documents/polytech/Annee_4/Reseau/ProjetReseau/repoGit2/ProjetReseau/TCP$ ./client.
out localhost 20001 Julien
argv[2] : 20001Connected to the server.
WelCome to the chat server!
Use these commands :
AJOUT <id_client id_compte password somme>
RETRAIT <id_client id_compte password somme>
SOLDE <id_client id_compte password>
OPERATIONS <id_client id_compte password>

Enter a command : AJOUT 1 101 chaton123 1000
Read : OK : Opération réussie

Enter a command : ^C
mathis@mathis-HP-EliteBook-820-G2:/media/mathis/disque dur externe/Cloud/Documents/polytech/Annee_4/Reseau/ProjetReseau/repoGit2/ProjetReseau/TCP$ ./client.
out localhost 20001 Mehdi
argv[2] : 20001Connected to the server.
WelCome to the chat server!
Use these commands :
AJOUT <id_client id_compte password somme>
RETRAIT <id_client id_compte password somme>
SOLDE <id_client id_compte password>
OPERATIONS <id_client id_compte password>

Enter a command : AJOUT 1 101 chaton123 42.5
Read : OK : Opération réussie

Enter a command : SOLDE 1 101 chaton123
Read : RES_SOLDE 1042.50 Sun Dec 31 00:09:14 2023

Enter a command : ^C
mathis@mathis-HP-EliteBook-820-G2:/media/mathis/disque dur externe/Cloud/Documents/polytech/Annee_4/Reseau/ProjetReseau/repoGit2/ProjetReseau/TCP$ ./server.out localhost 20002
Client créé avec succès : users[0].user_id = 1
Entering add_account
Compte créé avec succès
Entering add_account
Compte créé avec succès
Comptes 101 et 102 créés pour user_id = 1
bind(): Address already in use
mathis@mathis-HP-EliteBook-820-G2:/media/mathis/disque dur externe/Cloud/Documents/polytech/Annee_4/Reseau/ProjetReseau/repoGit2/ProjetReseau/TCP$ ./server.out localhost 20001
Client créé avec succès : users[0].user_id = 1
Entering add_account
Compte créé avec succès
Entering add_account
Compte créé avec succès
Comptes 101 et 102 créés pour user_id = 1
Server listening on port 20001...
Scanned : AJOUT 1 101 chaton123 42.500000
Command : AJOUT
Translated : AJOUT 1 101 chaton123 42.500000
ADD_OPERATION de 42.50
Response : OK : Opération réussie
Scanned : AJOUT 1 101 chaton123 1000.000000
Command : AJOUT
Translated : AJOUT 1 101 chaton123 1000.000000
ADD_OPERATION de 1000.00
Response : OK : Opération réussie
Scanned : SOLDE 1 101 chaton123 0.000000
Command : SOLDE
Response : RES_SOLDE 1042.50 Sun Dec 31 00:09:14 2023

```

Figure 1: Le TCP Fonctionne bien en Multi-Clients

```

mathis@mathis-HP-EliteBook-820-G2:/media/mathis/disque dur externe/Cloud/Documents/polytech/Annee_4/Reseau/ProjetReseau/repoGit2/ProjetReseau/UDP$ cd ../../..../repoGit0/ProjetReseau/UDP/
mathis@mathis-HP-EliteBook-820-G2:/media/mathis/disque dur externe/Cloud/Documents/polytech/Annee_4/Reseau/ProjetReseau/repoGit0/ProjetReseau/UDP$ ./client.
out localhost 20002 Julien
WelCome to the chat server!
Use these commands :
AJOUT <id_client id_compte password somme>
RETRAIT <id_client id_compte password somme>
SOLDE <id_client id_compte password>
OPERATIONS <id_client id_compte password>

Enter a command : AJOUT 1 101 chaton123 65.7
Read : OK : Opération réussie
out localhost 20001 Jean-claude
argv[2] : 20001Enter a command : ^C
mathis@mathis-HP-EliteBook-820-G2:/media/mathis/disque dur externe/Cloud/Documents/polytech/Annee_4/Reseau/ProjetReseau/repoGit2/ProjetReseau/UDP$ cd ../../..../repoGit0/ProjetReseau/UDP/
mathis@mathis-HP-EliteBook-820-G2:/media/mathis/disque dur externe/Cloud/Documents/polytech/Annee_4/Reseau/ProjetReseau/repoGit0/ProjetReseau/UDP$ ./client.
out localhost 20002 Jean-claude
WelCome to the chat server!
Use these commands :
AJOUT <id_client id_compte password somme>
RETRAIT <id_client id_compte password somme>
SOLDE <id_client id_compte password>
OPERATIONS <id_client id_compte password>

Enter a command : SOLDE 1 101 chaton123
Read : RES_SOLDE 65.70 Sun Dec 31 04:20:06 2023

Ready to read Client
Ready to read Client
Ready to read Client
Scanned : AJOUT 1 101 chaton123 65.700000
Command : AJOUT
Translated : AJOUT 1 101 chaton123 65.700000
ADD_OPERATION de 65.70
Response : OK : Opération réussie
Ready to read Client
Scanned : SOLDE 1 101 chaton123 0.000000
Command : SOLDE
Response : RES_SOLDE 65.70 Sun Dec 31 04:20:06 2023

```

Figure 2: Le UDP Fonctionne bien en Multi-Clients

The image shows two terminal windows side-by-side against a scenic background of rolling hills and a sunset.

Terminal Window 1 (Left):

```
Enter a command : AJOUT 1 101 chaton123 300
Read : OK : Opération réussie

Enter a command : AJOUT 1 101 chaton123 700
Read : OK : Opération réussie

Enter a command : AJOUT 1 101 chaton123 400
Read : OK : Opération réussie

Enter a command : RETRAIT 1 101 chaton123 1400
Read : OK : Opération réussie

Enter a command : AJOUT 1 101 chaton123 600
Read : OK : Opération réussie

Enter a command : AJOUT 1 101 chaton123 42
Read : OK : Opération réussie

Enter a command : AJOUT 1 101 chaton123 1000
Read : OK : Opération réussie

Enter a command : OPERATIONS 1 101 chaton123
Read :
RES_OPERATIONS
RETRAIT 500.00 € Sun Dec 31 00:00:41 2023
AJOUT 13.00 € Sun Dec 31 00:01:06 2023
AJOUT 516.14 € Sun Dec 31 00:02:36 2023
AJOUT 300.00 € Sun Dec 31 00:02:56 2023
AJOUT 700.00 € Sun Dec 31 00:04:01 2023
AJOUT 400.00 € Sun Dec 31 00:04:14 2023
RETRAIT 1400.00 € Sun Dec 31 00:04:27 2023
AJOUT 600.00 € Sun Dec 31 00:04:34 2023
AJOUT 42.00 € Sun Dec 31 00:04:50 2023
AJOUT 1000.00 € Sun Dec 31 00:05:04 2023

Enter a command : █
```

Terminal Window 2 (Right):

```
mathis@mathis-HP-EliteBook-820-G2: /media/mathis/disque dur externe/Cloud/Docum... ~ %
Response : OK : Opération réussie
Scanned : RETRAIT 1 101 chaton123 1400.000000
Command : RETRAIT
Translated : RETRAIT 1 101 chaton123 1400.000000
ADD_OPERERATION de 1400.00
Response : OK : Opération réussie
Scanned : AJOUT 1 101 chaton123 600.000000
Command : AJOUT
Translated : AJOUT 1 101 chaton123 600.000000
ADD_OPERERATION de 600.00
Response : OK : Opération réussie
Scanned : AJOUT 1 101 chaton123 42.000000
Command : AJOUT
Translated : AJOUT 1 101 chaton123 42.000000
ADD_OPERERATION de 42.00
Response : OK : Opération réussie
Scanned : AJOUT 1 101 chaton123 1000.000000
Command : AJOUT
Translated : AJOUT 1 101 chaton123 1000.000000
ADD_OPERERATION de 1000.00
Response : OK : Opération réussie
Scanned : OPERATIONS 1 101 chaton123 0.000000
Command : OPERATIONS
RES_OPERATIONS type_opération date_opération montant_operation
Response :
RES_OPERATIONS
RETRAIT 500.00 € Sun Dec 31 00:00:41 2023
AJOUT 13.00 € Sun Dec 31 00:01:06 2023
AJOUT 516.14 € Sun Dec 31 00:02:36 2023
AJOUT 300.00 € Sun Dec 31 00:02:56 2023
AJOUT 700.00 € Sun Dec 31 00:04:01 2023
AJOUT 400.00 € Sun Dec 31 00:04:14 2023
RETRAIT 1400.00 € Sun Dec 31 00:04:27 2023
AJOUT 600.00 € Sun Dec 31 00:04:34 2023
AJOUT 42.00 € Sun Dec 31 00:04:50 2023
AJOUT 1000.00 € Sun Dec 31 00:05:04 2023
```

Figure 3: Seulement 10 opérations s'affichent.