

Network programming in C project

Quentin Letort Cyril Marchand Nhu-Vuong Mavie IBO 3

17 Mars 2019

Table des matières

1	Introduction	2
2	Linux ou Windows ?	2
2.1	Linux	2
2.2	Windows	2
3	Application client/serveur TCP	3
3.1	socket ()	3
3.2	bind()	3
3.3	listen()	3
3.4	connect()	3
3.5	accept()	3
3.6	read()	3
3.7	write()	4
3.8	close()	4
4	Commandes utilisateurs	4
4.1	#Help	4
4.2	#listU	4
4.3	#ListF	5
4.4	#trfU	5
4.5	#trfD <[...]>	5
4.5.1	Le protocole FTP : File Transfer Protocol	6
4.6	#private <[...]>	6
4.7	#public	7
4.8	#ring<[...]>	7
5	Communication client/serveur ou client/client	7

1 Introduction

Dans ce projet, nous allons créer un serveur/client en langage C dans lequel les deux entités vont pouvoir communiquer et s'échanger des fichiers. Notre programme est portable sur les systèmes d'exploitation Linux et Windows.

Nota bene : L'exécution du client et du serveur ne s'effectue pas de la même manière selon le système d'exploitation (OS) utilisé. Voir section plus bas.

Après avoir créé la connexion entre le serveur et vous, le client, vous pourrez accéder aux fonctionnalités suivantes :

- #Exit
- #Help
- #ListU
- #ListF
- #TrfU
- #TrfD
- #Private <user>
- #Public
- #Ring <user>

2 Linux ou Windows ?

2.1 Linux

1. Créer les bons dossiers
 - Créer deux dossiers NetworkClient et NetworkServeur.
Dans chacun de ces dossiers, on retrouvera respectivement les programmes .c serveur.c et client.c .
2. Lancer deux consoles
 - Se positionner dans le bon répertoire grâce à la commande "cd".
 - Exécuter sur chacune des consoles : "gcc serveur.c -o serveur" et "gcc client.c -o client -lpthread"
L'option "-lpthread" permet d'ajouter la bibliothèque pthread.
3. Exécuter les exécutables créés par la commande précédente.
 - Il suffira simplement d'insérer la commande suivante " ./<nom de l'exécutable> "

2.2 Windows

Si vous êtes sur Windows, les commandes sont quasi similaires aux précédentes. Cependant, il faudra ajouter l'option "-lws2_32" à la fin des vos commandes serveur et client comme le montre ci-dessous. Une

```
\Project\NetworkProject-master>gcc -o server server.c -lws2_32
```

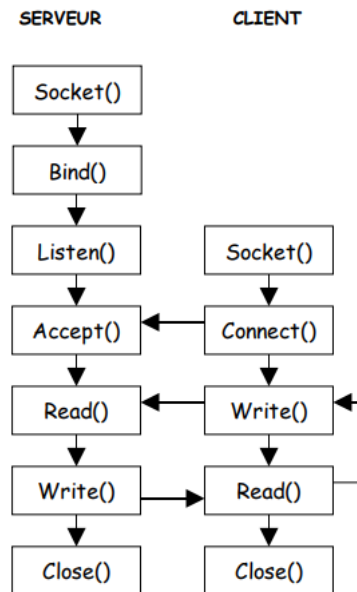
fois les commandes exécutées vous devriez obtenir ceci :

```
D:\EcoleESILVA4\S2\Advanced_net_program\Project\NetworkProject-master>server
La socket 264 est maintenant ouverte en mode TCP/IP
Listage du port 23...
```

```
D:\EcoleESILVA4\S2\Advanced_net_program\Project\NetworkProject-master>client
Bienvenue sur le chat!
```

3 Application client/serveur TCP

Nous développeront notre application en utilisant le protocole TCP. Notre application suit le schéma de communication en mode connecté suivant :



3.1 socket ()

Cette première étape consiste à créer la socket grâce à la fonction `socket()`. La fonction va renvoyer un entier `int` qui correspondra au descripteur de socket nouvellement créé. Ce descripteur sera ensuite utilisé en paramètre dans les prochaines fonctions. En cas d'erreur la fonction nous renvoie -1.

3.2 bind()

Après création du socket, il s'agit de le lier à un point de communication local défini par une adresse et un port. Pour ce faire, on utilisera la fonction `bind()`.

3.3 listen()

La fonction `listen()` va permettre à la socket nouvellement créée de se mettre en attente de connexion.

3.4 connect()

Afin d'établir une connexion avec le serveur, on utilisera la fonction `connect()`. Elle renvoie 0 dans le cas où une connexion a bien été établie, sinon elle retourne -1.

3.5 accept()

Dès lors que la socket reçoit une demande de connexion, la fonction `accept()` va permettre d'établir la connexion.

3.6 read()

Afin de lire ce que contient une socket, nous utiliserons la fonction `read()`. Elle retourne le nombre d'octets lus.

3.7 write()

Par la suite nous allons avoir besoin d'écrire dans une socket afin de transmettre des informations ou encore d'envoyer des fichiers. Pour cela, nous utiliserons la fonction `write()`. Elle renvoie le nombre d'octets envoyés.

3.8 close()

La fonction `close()` permet de mettre fin à une connexion.

4 Commandes utilisateurs

Pour que le serveur puisse reconnaître, il est nécessaire de commencer la commande par un "#". Dans le cas où le serveur ne reconnaît pas la commande, il renvoie au client un message d'erreur.

```
> #rien
la commande specifie n'est pas reconnu. Veuillez utiliser la commande #help pour plus d'informations
```

4.1 #Help

La commande `#Help` va permettre à l'utilisateur d'accéder à la liste des commandes disponibles et leur description.

```
> #help
Voici la liste des commandes disponibles sur ce serveur:

#listU : renvoie la liste des autres utilisateurs connectes (autres clients).
        Si il n'y en a aucun, le serveur renvoie : 'Aucun utilisateur actuellement en ligne.'
#listF : renvoie la liste des fichiers presents s
ur le serveur.
#trfU <> : permet de telecharger un fichier sur le serveur en mettant le nom de ce fichier a la place de
s <>.
#trfD <> : permet de tetecharger un fichier depuis le serveur en mettant le nom de ce fichier a la place
des <>.
#private <> : permet d'activer la conversa
tion prive avec un autre utilisateur. Pour se faire, il faut renseigner le nom de cet autre utilisateur
a la place des <>.
#public : permet de repasser la conversation en mode publique si elle etait en prive.
#ring <> : suivi du nom de l'utilisateur concerne a la place des <> agi
t comme une commande ping, et permet de savoir si l'autre utilisateur est connecte.
Si la commande n'est pas reconnu, un message d'erreur sera renvoye.
```

4.2 #listU

La commande `#ListU` permet à l'utilisateur d'obtenir à la liste de l'ensemble des utilisateurs qui sont connectés au même serveur que lui.

Dans le cas où il y a plusieurs clients connectés au même serveur :

```
> #listU
List users:
User: 264
User: 268
>
```

Dans le cas où l'utilisateur est seul :

Pour ce faire, le serveur parcourt la liste des noms des clients connectés qui sont stockés dans la liste "client_socket". À chaque fois qu'il trouve dans cette liste un client différent de celui qui a envoyé la requête, il met à jour la string de réponse.

```
> #listU
List users:
Aucun utilisateur actuellement en ligne
>
```

4.3 #ListF

La commande #ListF permet de lister l'ensemble des fichiers disponibles dans le serveur.

```
mavler@mavler-N751JX:~/Documents/Ne
Bienvenue sur le chat!
> #listF
...server1test.txtserver1.c
>
```

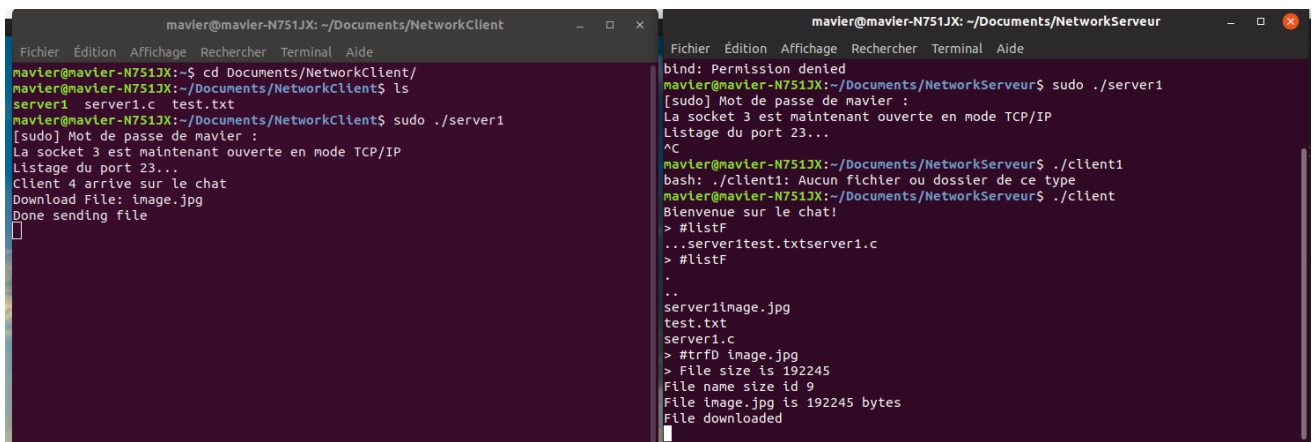
Le serveur va parcourir, si il existe, l'ensemble de ses fichiers disponibles et renvoie le nom du fichier à travers la socket qui le relie avec le client grâce à la fonction `send()`.

```
else if(strcmp(cmd, "listF") == 0)
{
    DIR *d;
    struct dirent *dir;
    d = opendir(".");
    if (d)
    {
        while ((dir = readdir(d)) != NULL)
        {
            // printf("%s\n", dir->d_name);
            send(sd, dir->d_name, strlen(dir->d_name), 0);
            // send(sd, "\n", 1, 0);
        }
        closedir(d);
    }
}
```

4.4 #trfU

4.5 #trfD <[...]>

La commande #trfD permet à l'utilisateur de demander au serveur le fichier qu'il souhaite télécharger. Il peut s'aider de la commande #listF pour avoir l'ensemble des fichiers disponibles. L'utilisateur doit indiquer juste après la commande le nom du fichier : #trfD <[nomDuFichier]>.



```
mavler@mavler-N751JX: ~/Documents/NetworkClient
Fichier Édition Affichage Rechercher Terminal Aide
mavler@mavler-N751JX:~$ cd Documents/NetworkClient/
mavler@mavler-N751JX:~/Documents/NetworkClient$ ls
server1 server1.c test.txt
mavler@mavler-N751JX:~/Documents/NetworkClient$ sudo ./server1
[sudo] Mot de passe de mavler :
La socket 3 est maintenant ouverte en mode TCP/IP
Listage du port 23...
Client 4 arrive sur le chat
Download File: image.jpg
Done sending file
█

mavler@mavler-N751JX: ~/Documents/NetworkServeur
Fichier Édition Affichage Rechercher Terminal Aide
bind: Permission denied
mavler@mavler-N751JX:~/Documents/NetworkServeur$ sudo ./server1
[sudo] Mot de passe de mavler :
La socket 3 est maintenant ouverte en mode TCP/IP
Listage du port 23...
^C
mavler@mavler-N751JX:~/Documents/NetworkServeur$ ./client1
bash: ./client1: Aucun fichier ou dossier de ce type
mavler@mavler-N751JX:~/Documents/NetworkServeur$ ./client
Bienvenue sur le chat!
> #listF
...server1test.txtserver1.c
> #listF
.
..
server1image.jpg
test.txt
server1.c
> #trfD image.jpg
> File size is 192245
File name size id 9
File image.jpg is 192245 bytes
File downloaded
█
```

Ici, nous téléchargeons le fichier "image.jpg". A la suite de ce téléchargement, ce fichier devrait apparaître dans le dossier du client.



4.5.1 Le protocole FTP : File Transfer Protocol

Voici les étapes du protocole FTP que nous avons établis :

— Côté serveur :

1. Récupérer le nom du fichier envoyé par le client
2. Vérifier l'existence de ce fichier grâce à l'appel *stat* qui nous permettra d'accéder aux informations du fichier. Dans le cas où le fichier n'existe pas, la structure retourne une erreur.
3. Stocker dans *file_size* la taille du fichier à envoyer. Récupérer également le nom du fichier.
4. Ces deux informations précédentes seront premièrement envoyées au client à travers la socket qui les relie grâce à la fonction *send*.
5. Une fois les informations envoyées au client, le serveur envoie par petit bout le fichier.

— Côté client :

1. Le client reçoit les informations envoyées par la socket et les stocke.
2. Le client attend l'ensemble du fichier à recevoir. Pour cela, au fur et à mesure de l'envoi des "morceaux" du fichier, il compare la taille totale du fichier avec la somme des fichiers déjà reçus.

```
// recevoir fichier
void * buffer = NULL;
int r = 0, w = 0, total = 0;
buffer = malloc(BUFFER_SIZE);

while(total < file_size) {
    r = recv(sock, buffer, BUFFER_SIZE, 0);
    w = fwrite(buffer, 1, r, local_file);
    total += r;
    if(w < 0) {
        perror("Writing local file");
        free(buffer);
        fclose(local_file);
        exit(EXIT_FAILURE);
    }
}
printf("File downloaded\n");
```

4.6 #private <[...]>

La commande `#private` permet de rendre privée une connexion. Pour cela, l'utilisateur doit renseigner le nom du client à la suite de la commande `#private <[nomDuClient]>`. Le serveur met à jour les informations relatives au client ayant effectué la demande. Dans «client_socket», il passe le booléen `.private` de ce client à «true», et affecte le nom du client avec qui la communication doit être privée au champ `.client_P`. Ces deux informations auront de l'importance lors des prochains envois de messages. Un message est finalement envoyé au client pour l'informer que la commande a bien pris effet.

```
> #private
Veuillez specifier l'utilisateur avec qui vous voulez passer en mode prive.
>
```

```
> #private 12
Veuillez specifier un utilisateur actuellement connecte.
>
```

```
> #private 268
vous etes maintenant en communication prive avec 268
>
```

Ce que reçoit le client de la connexion privée :

```
(private)USER:260> bonjour
>
```

On peut voir sur le serveur avec qui nous sommes en connexion privée :

```
USER:260> en communication prive avec 268
```

4.7 #public

La commande `#public` permet de emettre les champs du client demandeur relatifs à la commande précédente à leur valeur par défaut, `.private` revient donc à `false` et `.client_P` à 0 considéré comme valeur par défaut. Un message est envoyé au client demandeur pour l'informer que ses communications sont désormais publiques.

```
> #public
Retour en communication public
```

4.8 #ring<[...]>

La commande `#ring` permet à l'utilisateur de "sonner" chez un autre client, dans le but de l'interpeller. Pour cela, l'utilisateur doit indiquer le nom du client à la suite de la commande `#ring <[nomDuClient]>`.

```
> #ring
Veuillez specifier l'utilisateur que vous voulez sonner.
```

```
> #ring 24
l'utilisateur que vous essayez de sonner n'est pas connecte.
```

```
> #ring 268
l'utilisateur 268 est connecte et a reçu le ring.
```

Si un client est bien renseigné, le serveur va alors chercher dans la liste «`client_socket`» si un des clients connecté porte ce nom. Si ce n'est pas le cas, un message est envoyé. Le client à qui est envoyé le ring reçoit un message sur sa console.

```
l'utilisateur 260 vous a envoye un ring.
```

5 Communication client/serveur ou client/client

Dans le cas où l'utilisateur n'utilise pas de commande `"#"` mais envoie un simple message. La première chose que le serveur doit vérifier est le type de communication de ce client (privé ou publique). Pour se faire, il va chercher l'information dans «`client_socket`», le `.private`, si ce dernier est `"true"`, alors la communication est privée, sinon, elle est publique.

Côté client :

```
> bonjour  
>
```

Côté serveur :

```
USER:260> bonjour
```

Dans le cas d'une communication privée, le serveur va chercher dans la liste « `client_socket` » le `client_Pdu` client qui envoi le