

# Devoir libre

MODE CONNECTE (TCP)

Miri Mohamed | 12 Novembre 2021

# Introduction

Ce rapport d’écrit l’implémentation d’un serveur multiprocesseurs en mode flot de données en utilisant des sockets en langage C.

Ce projet avait pour objectif de permettre au client connecté au serveur d’envoyer un fichier de commandes et recevoir l’état d’exécution de ces commandes.

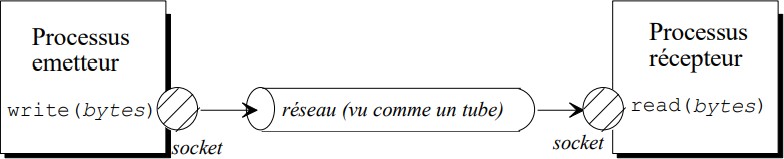
Vous pouvez accéder au code de toutes les étapes sur mon répertoire GITHUB : https://github.com/Bouddha-ctrl/Master-slave

## C’est quoi une socket

### DEFINITION

Une socket est connue comme une interface de communication logicielle qui agit comme un point d'extrémité qui fonctionne en établissant une liaison de communication réseau bidirectionnelle entre l'extrémité du serveur et le programme de réception du [client.](https://www.speedcheck.org/fr/wiki/client/)

Via cette communication logicielle une application peut envoyer/recevoir des données.



### MODES DE COMMUNICATION

Le mode connecté :

* Utilisant le protocole TCP qui contrôle si le paquet est arrivé à destination, si ce n’est pas le cas il le renvoie. Et établie une connexion durable entre les deux processus

Le mode non connecté :

* Utilisant le protocole [UDP](https://fr.wikipedia.org/wiki/User_Datagram_Protocol) qui ne vérifie pas si le paquet est bien arrivé.

## Contexte du projet

### L’OBJET DU PROJET

Ce projet est pour le but d'implémenter les concepts de la programmation réseau par l'utilisation d’API socket, et aussi de mettre en place les notions de programmation C.

### PRESENTATION DU PROJET

* L’environnement client/serveur ?

L'environnement client/serveur désigne un mode de communication à travers un réseau entre plusieurs programmes : l'un qualifié de **client**, envoie des requêtes l'autre qualifié de **serveur**, attendent les requêtes des clients et répond.

Caractéristique du **Serveur** :

* Il attend une connexion entrante
* A la connexion d’un client sur le port en écoute, il ouvre une socket local
* Suite à la connexion, le serveur communique avec le client Caractéristique du **Client** :
* Il établit une connexion au serveur avec une adresse IP et un port
* Lorsque la connexion est acceptée par le serveur, il communique comme prévoit la couche applicative du modèle OSI

Le client et le serveur doivent bien sur utiliser le même protocole de communication.

* Les bibliothèques utilisées ? Le package

<sys/socket.h>

Ce package fournit la structure **sockaddr\_in**, et les fonctions pour la mise en œuvre des application réseau, comme :

* **La structure Socket** : permet de créer une socket.
* **La structure inet\_pton / Bind :** permet de définir @ip et le port
* **La structure connect :** permet de se connecter à un serveur
* **La structure listen :** permet de se met à l’écoute
* **La fonction Send**: permet d’envoyer un message dans un Stream
* **La fonction Recv :** permet de recevoir un message dans un Stream

Le package

<arpa/inet.h>

Ce package fournit la structure **in\_addr**, qui nous permet de définir l’adresse du serveur, qui sera par la suite liée au socket créer.

Le package

<pthread.h>

Ce package nous permet de créer des threads à travers la fonction **pthread\_create**

* Architecture ?

L’implémentation de ce projet va suivre l’architecture suivante :

**Server**

**Server Esclave**

Server

**Thread 1**

3

Création des Threads

4

1

Socket d’écoute

2

Esclave.txt

9

Boucle infinie

10

88

Boucle infinie

11

Distribution des

Commandes

**Thread i**

Receive

Execute

Response

5

Socket d’écoute

12

Command.txt

7

**Client**

6

Socket Client

Read / Write

Cette architecture suit des étapes :

1. Lancement des servers esclaves sur l'écoute (l’attend une connexion)
2. Le serveur Maitre lie le fichier esclave.txt qui contient la liste des serveur esclave
3. Le serveur Maitre crée au tant de threads que les serveurs esclaves
4. A la création d'un thread il se connecte directement au serveur esclave

designer par le serveur maitre

1. Le serveur Maitre se met à l'écoute.
2. Le client lie le fichier Command.txt et conserve sont contenue dans une variable
3. Le client se connecte au serveur Maitre et envoie les commandes.
4. Le serveur Maitre reçoit les commandes et il envoie à chaque thread une partie des commandes en utilisant une pipe déjà établie hors de la création du thread.
5. Le thread reçoit la commande et il l'envoie à son tour au serveur esclave.
6. Le serveur esclave reçoit la commande, il l'exécute et renvoie au thread le résultat.
7. Le thread envoie le résultat au client directement.

Quelle mode de connexion :

Afin de réaliser cette architecture, on est obligé d’implémentée un mode de communication connecté, car le server/client/serveurEsclave exécute plusieurs taches successives qui nécessite une connexion durable.

On peut établir le mode TCP avec l’instruction suivante :

socket(AF\_INET, SOCK\_STREAM, 0))

AF\_INET : type d’adressage IPv4

SOCK\_STREAM : mode de communication TCP

A savoir :

1. A la création des threads par le serveur Maitre, une structure est passée en argument pour être utiliser par le thread :

typedef struct{

    pthread\_mutex\_t \*mutex;

    int port;

    int fd[2];

}DATAServer;

Port : le port du serveur esclave.

Mutex : pour empêcher les threads d’envoyer des réponses au client au même temps.

fd : le pipe de communication entre le process main et le thread (pour l’envoie des commandes.

La création :

    for(i=0;i<nbServers;i++){

        dServer[i].mutex = &mutex;

        dServer[i].port = Ports[i];

        pipe(dServer[i].fd);

        pthread\_create(&thr[i],NULL,ThreadFunct,&dServer[i]);

    }

1. Le client envoie les commandes dans une structure :

typedef struct{

    char fileName[50];

    char file[1024];

    int sock=0;

}DATA;

1. Le serveur reçoit cette structure, et créer une nouvelle avec le même **fileName**, mais réduit le contenue du **file** à une seul commande, attribue la socket du client a la variable **sock**.

## Implémentation

Ce projet est divisé en 3 parties, partie serveur, serveur esclave et client.

### PARTIE CLIENT :

// Client side C/C++ program to demonstrate Socket programming

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

typedef struct{

    char fileName[50];

    char file[1024];

    int sock;

}DATA;

int main(int argc, char \*argv[])

{

    //int PORT = atoi(argv[1]);

    int PORT = 7001;

    int sock = 0, valread;

    struct sockaddr\_in serv\_addr;

    char buffer[1024] = {0};

    if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

    {

        printf("\n Socket creation error \n");

        return -1;

    }

    serv\_addr.sin\_family = AF\_INET;

    serv\_addr.sin\_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form

    if(inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)<=0)

    {

        printf("\nInvalid address/ Address not supported \n");

        return -1;

    }

    if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0)

    {

        printf("\nConnection Failed \n");

        return -1;

    }

    //read data from file

    char\* content = NULL;

    FILE\* fp = fopen("command.txt","r");

    if(fp==NULL) return -1;

    size\_t len;

    ssize\_t bytes\_read = getdelim( &content, &len, '\0', fp);

    if ( bytes\_read == -1) return -1;

    //send data to server

    DATA d = {

        .fileName = "File Name",

        .sock=0

    };

    strcpy(d.file,content);   //copy "content" to file  , (pointer to table)

    int i;

    printf("file Name : %s, file : %s\n",d.fileName,d.file);

    //send data to server

    valread=send(sock , &d ,sizeof(d),0);

    printf("send : %d\n",valread);

    int \*result;

    //receive from server

    while(valread = recv(sock , result, sizeof(result),0)){

        printf("receive : %d, result: %d\n",valread,\*result);

    }

    close(sock);

    return 0;

}

### PARTIE SERVEUR :

#include <stdio.h>

#include <sys/socket.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#include <sys/types.h>

#include <pthread.h>

typedef struct{

    char fileName[50];

    char file[1024];

    int sock=0;

}DATA;

typedef struct{

    pthread\_mutex\_t \*mutex; //pour envoyer le résultat eu client, un par un

    int port;               //port du serveur esvlave

    int fd[2];              //pipe entre Main process et le Thread

}DATAServer;

void sending(int sock,int len, void\* arg){   //sending function

    int h=send(sock , arg,len,0);

    printf("message sent : %d\n",h);

}

void \*ThreadFunct(void\* arg){

    //get data

    DATAServer \*dServer = (DATAServer\*)arg;

    /////

    //connecte to slave

    int PORT\_slave = dServer->port, valread, valsend;

    int slave\_server = 0;

    struct sockaddr\_in serv\_addr;

    char buffer[1024] = {0};

    if ((slave\_server = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

    {

        printf("\n Socket creation error \n");

        return NULL;

    }

    serv\_addr.sin\_family = AF\_INET;

    serv\_addr.sin\_port = htons(PORT\_slave);

    // Convert IPv4 and IPv6 addresses from text to binary form

    if(inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)<=0)

    {

        printf("\nInvalid address/ Address not supported \n");

        return NULL;

    }

    if (connect(slave\_server, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0)

    {

        printf("\nConnection Failed \n");

        return NULL;

    }

    //slave server Connected

    DATA d;

    int \*result=malloc(sizeof(int));

    while(read(dServer->fd[0], &d, sizeof(DATA)) > 0)  //waiting data from pipe

    {

        //send to slave

        sending(slave\_server,sizeof(d),&d);

        //receive from slave

        valread = recv(slave\_server , result, sizeof(int\*),0);

        if(valread==0) break;                                  //if we failed to send data to slave(disconnected)

        printf("Thread. Receive : %d, Result from slave : %d\n",valread,\*result);

        //send to client

        pthread\_mutex\_lock(dServer->mutex);

        sending(d.sock,sizeof(result),result);

        pthread\_mutex\_unlock(dServer->mutex);

    }

    printf("Slave Server %d disconected.\n",PORT\_slave);

    return NULL;

}

int main(int argc, char \*argv[])

{

    //mutex && thread

    pthread\_mutex\_t mutex;

    pthread\_mutex\_init(&mutex, NULL);

    //read slave servers file

    FILE \*file  = fopen("LSlave.txt", "r");

    int nbServers, i;                        //number of servers

    if (file == NULL)                        // test for files not existing.

    {

        perror("file");

        exit(EXIT\_FAILURE);

    }

    fscanf(file,"%d\n",&nbServers);          //extract first line of file

    /////

    /////create thread and bind server data

    pthread\_t \*thr = malloc(nbServers\*sizeof(pthread\_t));     //liste des threads

    DATAServer \*dServer =malloc(nbServers\*sizeof(DATAServer)); //liste data envoyer au threads

    int Ports[nbServers];

    for(i=0;i<nbServers;i++){                                   //copy ports from file to table Ports[], we dont copy names (%\*s)

        fscanf(file,"%\*s %d\n",&P[i]);

    }

    fclose(file);

    for(i=0;i<nbServers;i++){                                   //Threads creation

        dServer[i].mutex = &mutex;

        dServer[i].port = Ports[i];

        pipe(dServer[i].fd);

        pthread\_create(&thr[i],NULL,ThreadFunct,&dServer[i]);

    }

    pthread\_mutex\_destroy(&mutex);

    ///////////////////Create main server

    //int PORT = atoi(argv[1]);

    int PORT = 7001;

    int serverSocket, client\_socket, valread;

    int opt = 1;

    struct sockaddr\_in address;

    int addrlen = sizeof(address);

    char buffer[1024] = {0};

    // Creating socket file descriptor

    serverSocket = socket(PF\_INET, SOCK\_STREAM, 0);

    if (serverSocket==0){

        perror("socket failed");

        exit(EXIT\_FAILURE);

    }

    // Forcefully attaching socket to the port

    if (setsockopt(serverSocket, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT,

                                                  &opt, sizeof(opt)))

    {

        perror("setsockopt");

        exit(EXIT\_FAILURE);

    }

    //Bind - Forcefully attaching socket to the port

    address.sin\_family = AF\_INET;

    address.sin\_addr.s\_addr = INADDR\_ANY;

    address.sin\_port = htons( PORT );

    if (bind(serverSocket, (struct sockaddr \*)&address, addrlen)==-1)

    {

        perror("bind failed");

        exit(EXIT\_FAILURE);

    }

    //Listen

    if (listen(serverSocket, 3) < 0)

    {

        perror("listen");

        exit(EXIT\_FAILURE);

    }

    //////Infinite LOOP

    while(1){

        printf("Waiting ! client\n");

        //accept

        if ((client\_socket = accept(serverSocket, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen))<0)

            {

                perror("accept");

                exit(EXIT\_FAILURE);

            }

        printf("Client : %d accepted.\n",client\_socket);

        //receive from client

        DATA d;

        valread = recv(client\_socket , &d, sizeof(DATA),0);

        printf("file Name : %s, file : %s\n",d.fileName,d.file);

        printf("Main receive : %d\n",valread);

        //send to thread -> slave

        char \* token = strtok(d.file, "\n");

        int i = 0;

        // loop through the string to extract all commandes seperated by "\n"

        while( token != NULL ) {

            DATA d2;

            d2.sock = client\_socket;

            strcpy(d2.file, token);                     //copy just one commande

            write(dServer[i].fd[1], &d2,sizeof(d2));    //send data to thread i

            token = strtok(NULL,"\n");

            i++;

            i=i%nbServers;

        }

    }

    for(i=0;i<nbServers;i++){  //test

        pthread\_join(thr[i],NULL);

    }

    close(serverSocket);

    close(client\_socket);

    return 0;

}

### Partie Server Esclave:

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#include <sys/types.h

#include <arpa/inet.h>

typedef struct{

    char fileName[50];

    char file[1024];

    int sock;

}DATA;

int main(int argc, char \*argv[])

{

    int PORT = atoi(argv[1]);

    //int PORT = 7007;

    int mySocket, Main\_socket,valread;

    struct sockaddr\_in address;

    int opt = 1;

    int addrlen = sizeof address;

    char buffer[1024] = {0};

    char \*hello = "Hello from slave server";

    // Creating socket file descriptor

    mySocket = socket(PF\_INET, SOCK\_STREAM, 0);

    if (mySocket==0){

        perror("socket failed");

        exit(EXIT\_FAILURE);

    }

    // Forcefully attaching socket to the port 8080

    if (setsockopt(mySocket, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT,

                                                  &opt, sizeof(opt)))

    {

        perror("setsockopt");

        exit(EXIT\_FAILURE);

    }

    //Bind - Forcefully attaching socket to the port

    address.sin\_family = AF\_INET;

    address.sin\_addr.s\_addr = INADDR\_ANY;

    address.sin\_port = htons( PORT );

    if (bind(mySocket, (struct sockaddr \*)&address, addrlen)==-1)

    {

        perror("bind failed");

        exit(EXIT\_FAILURE);

    }

    //Listen

    if (listen(mySocket, 3) < 0)

    {

        perror("listen");

        exit(EXIT\_FAILURE);

    }

    //accept

    if ((Main\_socket = accept(mySocket, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen))<0)

    {

        perror("accept");

        exit(EXIT\_FAILURE);

    }

    ///Server Connected

    DATA d;

    while(1){

        printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

        //receive data

        valread = recv(Main\_socket , &d, sizeof(DATA),0);

        printf("Receive : %d || file Name : %s, file : %s\n",valread,d.fileName,d.file);

        //process

        int \*result;

        int check = system(d.file);

        result = &check;

        printf("resulat : %d\n",check);

        //send data

        valread=send(Main\_socket , result ,sizeof(result),0);

        printf("message sent : %d\n",valread);

    }

    close(mySocket);

    close(Main\_socket);

}

**Fichier ListEsclave.txt :**

2

slave1 7007

slave2 7008

**Fichier Command.txt :**

ls

lssss

ls -lh

### Test:

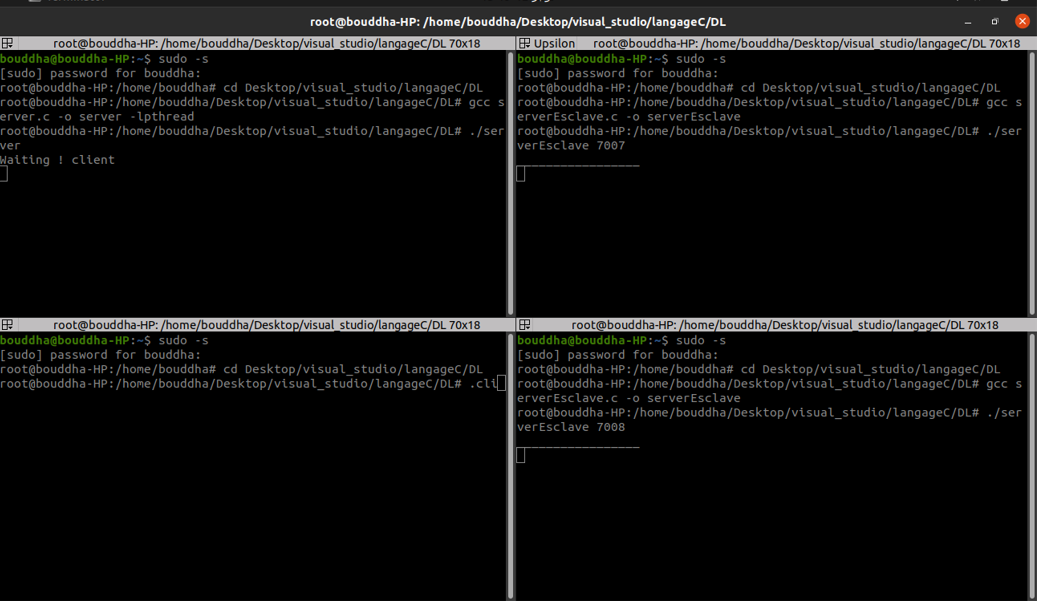
On lance les serveurs esclaves :

./serverEsclave 7007

./serverEsclave 7008

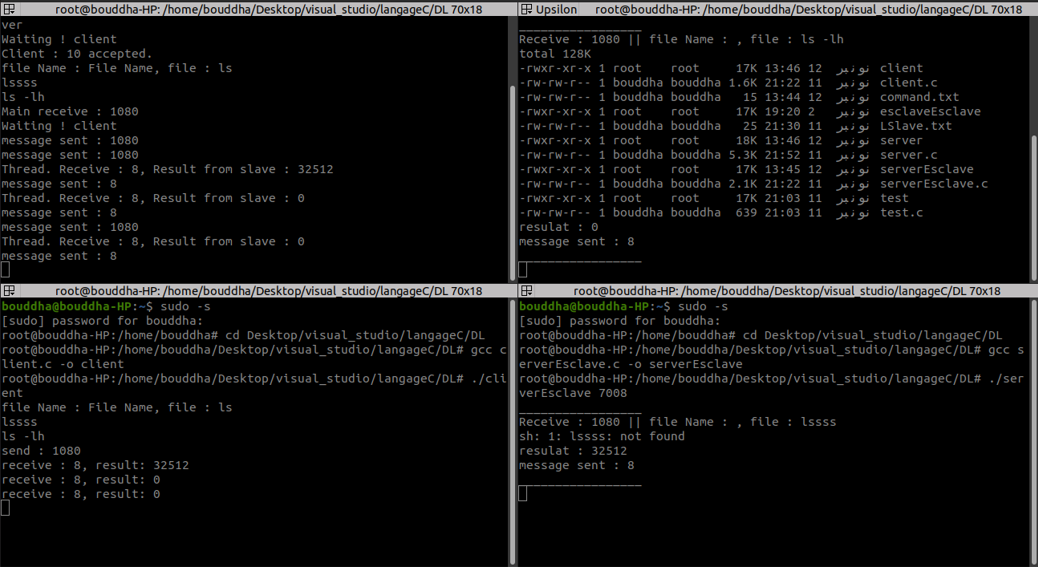
On lance le serveur Maitres :

./server



On lance le client :

./client



On remarque que le premier serveur esclave a exécuté deux commandes et le deuxième a exécuté qu’une seul, car on a 3 commandes et deux serveurs Esclave.

# Conclusion

Durant projet, ma mission consistait à créer une application d’architecture Master/slave d’exécution des commandes pour qu'un client peut se connecter et envoyer ses commandes puis attendre le bilan d’exécution, en utilisant les sockets en C.

Au cours de la période, on a eu l'opportunité de mettre en évidence les différentes connaissances acquises et d'acquérir de nouveaux concepts savoir Socket.h et Pthread.h . De plus j'ai eu l'occasion d'appliquer mes connaissances en C, et les principes du système client/serveur en réseau, ainsi le rôle principal des sockets.

La partie que j'ai développé correspond aux objectifs de départ, Mais ceci n'empêche pas d'améliorer l'application au futur.