LAB2-C

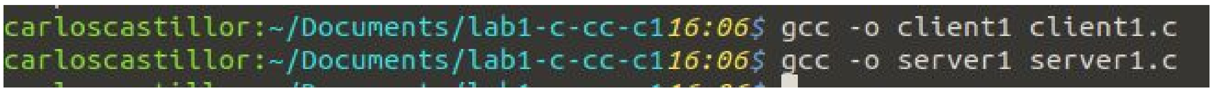
Claudio Maiorana Arvelo, Carlos Castillo Rivero
Githubs: CMaio, carloscastillor

OPERATING SYSTEMS
2020-2021
10 March 2021

Index

Implement new version Tic-Tac-Toe connection oriented

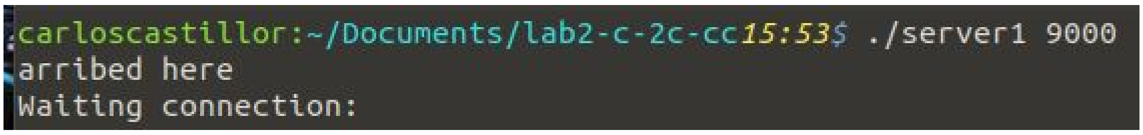
- Screenshots and explanation of the compiling process



```
carloscastillor:~/Documents/lab1-c-cc-c116:06$ gcc -o client1 client1.c
carloscastillor:~/Documents/lab1-c-cc-c116:06$ gcc -o server1 server1.c
```

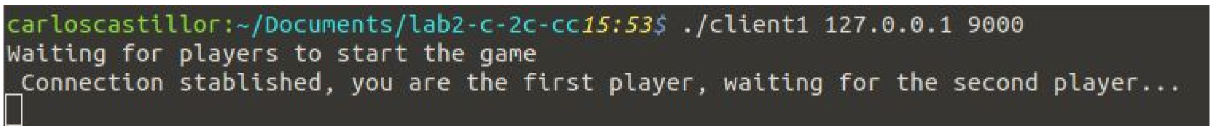
In order to compile and generate the programs, the GCC compiler is used, but in order to compile the server1.c file another option in the command is used. This other option is "-pthread", and is necessary because the compiler needs to know if this library is used in the implementation of the code.

- Screenshots and explanation of the execution process

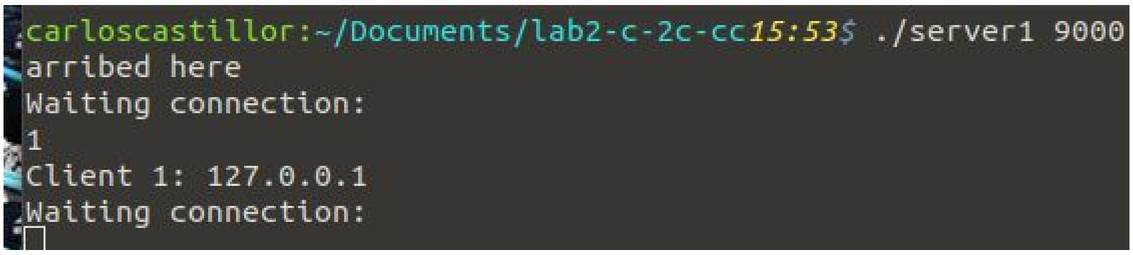


```
carloscastillor:~/Documents/lab2-c-2c-cc15:53$ ./server1 9000
arribed here
Waiting connection:
```

In order to start playing, you must execute the server1 program first, using the command "./server1" and specify a port, then the process will wait for the two clients to answer.

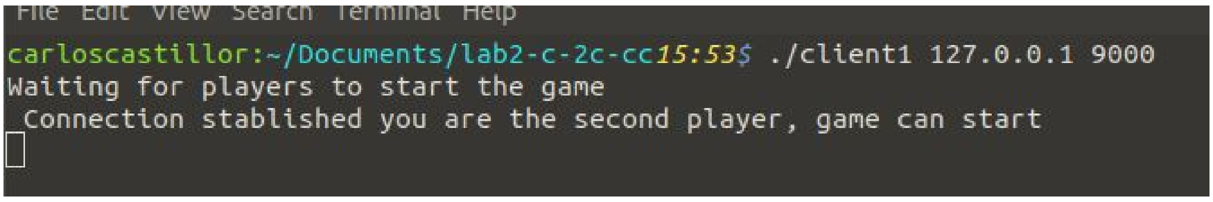


```
carloscastillor:~/Documents/lab2-c-2c-cc15:53$ ./client1 127.0.0.1 9000
Waiting for players to start the game
 Connection stablished, you are the first player, waiting for the second player...
```



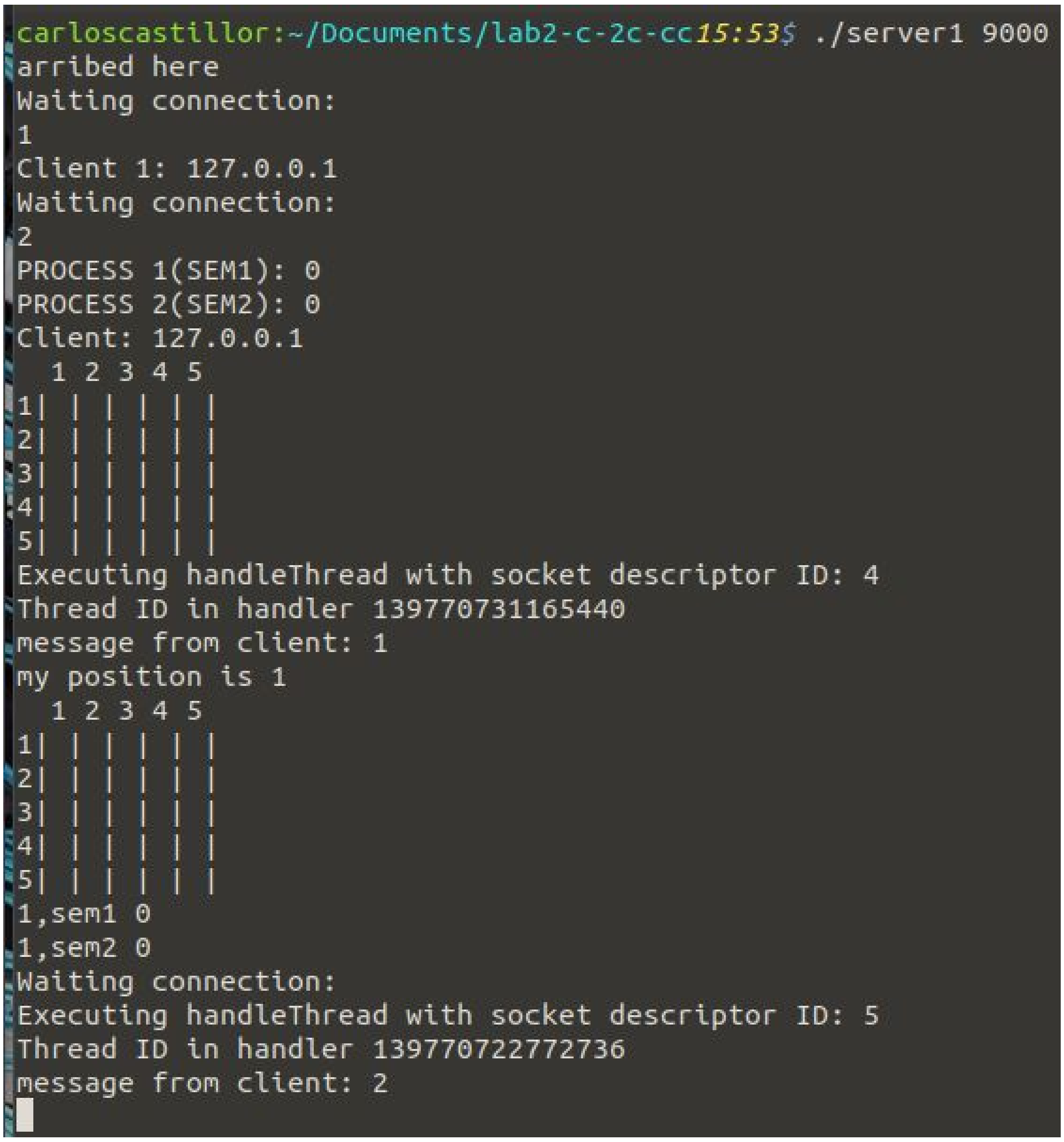
```
carloscastillor:~/Documents/lab2-c-2c-cc15:53$ ./server1 9000
arribed here
Waiting connection:
1
Client 1: 127.0.0.1
Waiting connection:
```

Once executing a client1 program using the command "./client1" you must specify an ip address and the port to connect, this will be the same specified in the server1 execution. The server will detect that connection and wait for the last client to join the game.

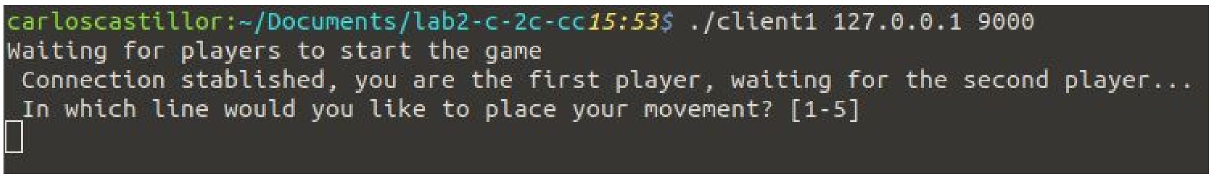


```
carloscastillor:~/Documents/lab2-c-2c-cc15:53$ ./client1 127.0.0.1 9000
Waiting for players to start the game
 Connection stablished you are the second player, game can start
```

Once the second player joins the game will start.

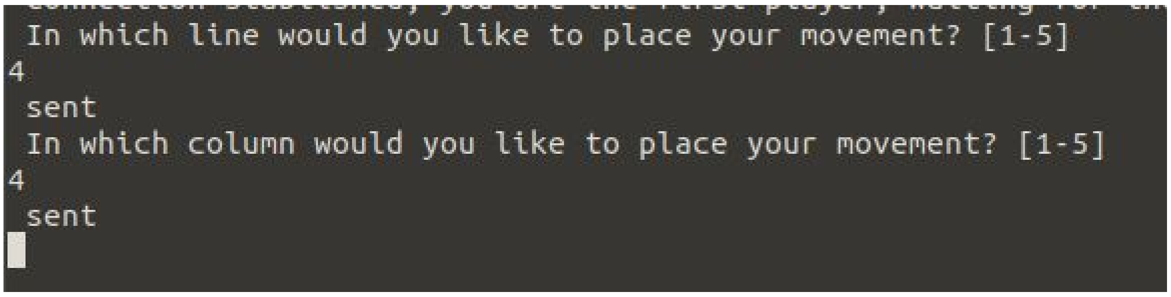


```
carloscastillor:~/Documents/lab2-c-2c-cc15:53$ ./server1 9000
arribed here
Waiting connection:
1
Client 1: 127.0.0.1
Waiting connection:
2
PROCESS 1(SEM1): 0
PROCESS 2(SEM2): 0
Client: 127.0.0.1
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | | | |
5| | | | | |
Executing handleThread with socket descriptor ID: 4
Thread ID in handler 139770731165440
message from client: 1
my position is 1
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | | | |
5| | | | | |
1,sem1 0
1,sem2 0
Waiting connection:
Executing handleThread with socket descriptor ID: 5
Thread ID in handler 139770722772736
message from client: 2
```

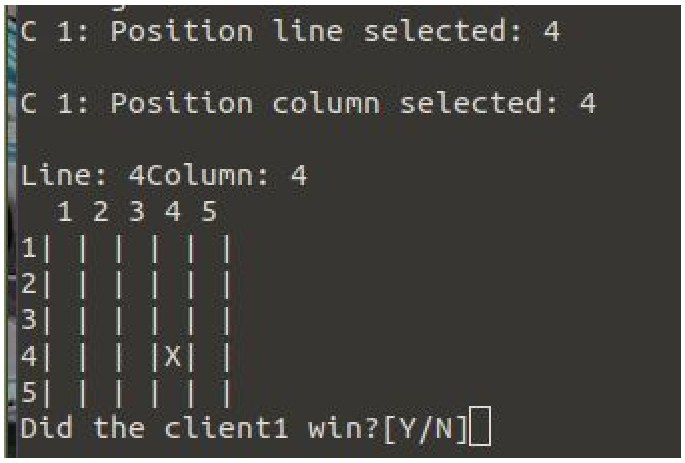


```
carloscastillor:~/Documents/lab2-c-2c-cc15:53$ ./client1 127.0.0.1 9000
Waiting for players to start the game
 Connection stablished, you are the first player, waiting for the second player...
 In which line would you like to place your movement? [1-5]
```

The server will show the board for each client, but not on their screen since they are playing blindly, and ask the first client to make a move. Each move consists of a combination of a row and a column.

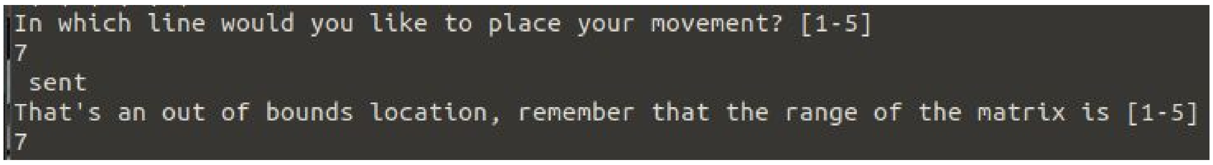


```
 In which line would you like to place your movement? [1-5]
4
 sent
 In which column would you like to place your movement? [1-5]
4
 sent
```

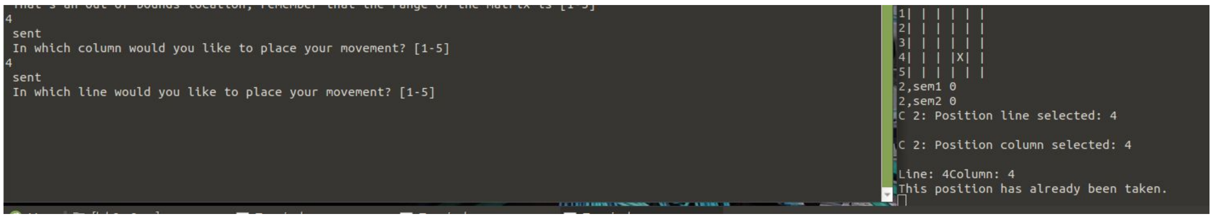


```
C 1: Position line selected: 4

C 1: Position column selected: 4

Line: 4Column: 4
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | |X| |
5| | | | | |
Did the client1 win?[Y/N]
```

If the client inserts a valid move, it will be sent to the server and it will introduce it inside the board and print it in the server terminal.



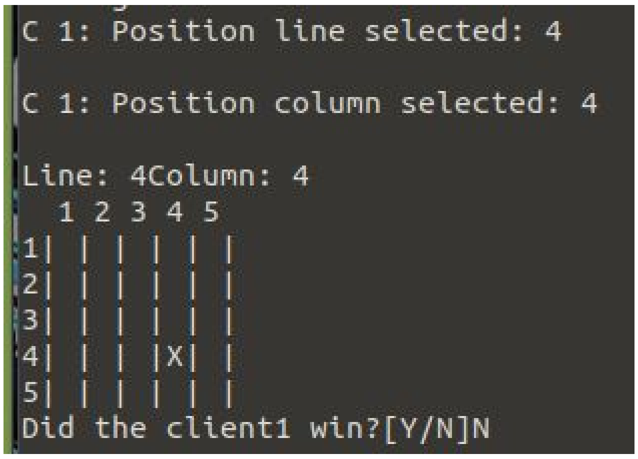
```
In which line would you like to place your movement? [1-5]
7
 sent
That's an out of bounds location, remember that the range of the matrix is [1-5]
7
```



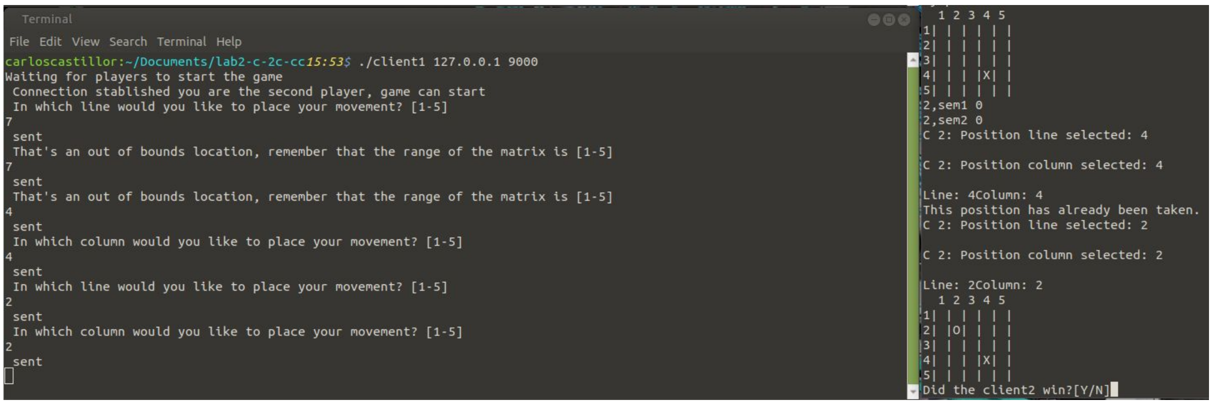
```
4
 sent
 In which column would you like to place your movement? [1-5]
4
 sent
 In which line would you like to place your movement? [1-5]
```

```
1| | | | | |
2| | | | | |
3| | | | | |
4| | | |X| |
5| | | | | |
2,sem1 0
2,sem2 0
C 2: Position line selected: 4

C 2: Position column selected: 4

Line: 4Column: 4
This position has already been taken.
```

If the move is not valid it will ask to insert a new one. If it is an out of bounds location will show image number 1, if it is an already occupied position it will show image number 2, and ask for a new move.



```
C 1: Position line selected: 4

C 1: Position column selected: 4

Line: 4Column: 4
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | |X| |
5| | | | | |
Did the client1 win?[Y/N]N
```

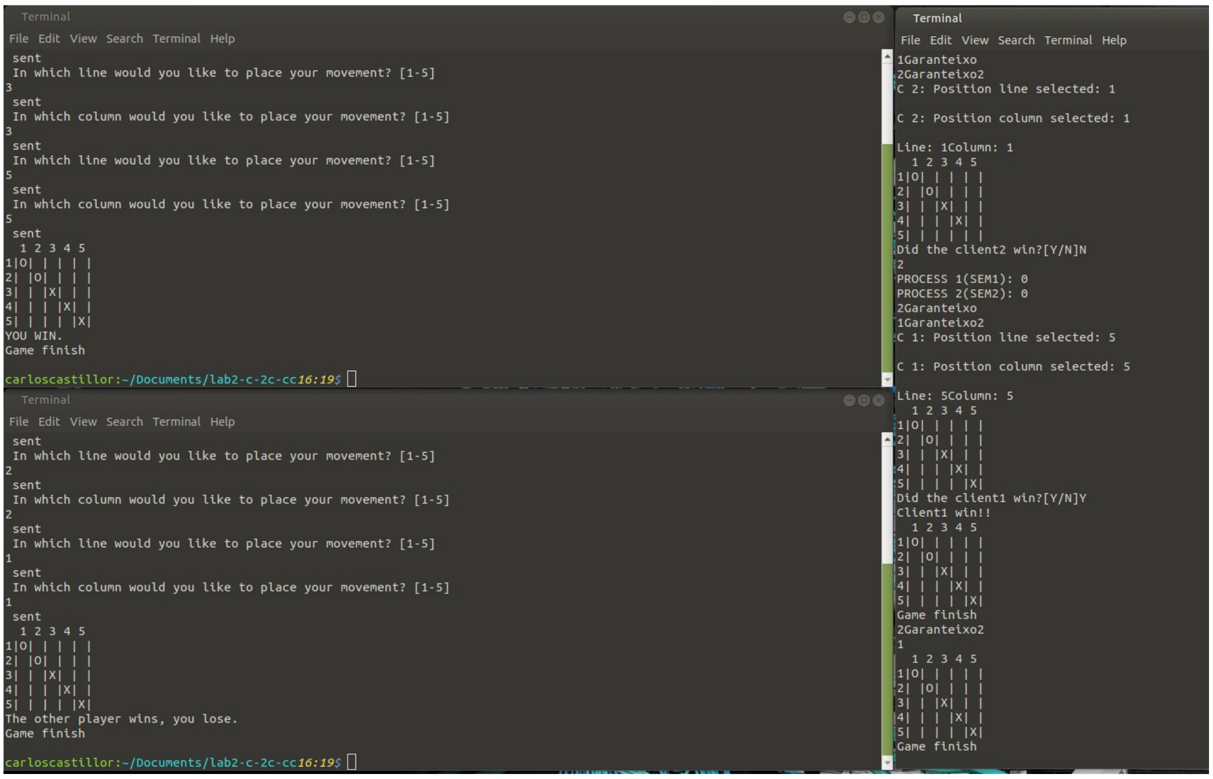


```
carloscastillor:~/Documents/lab2-c-2c-cc15:53$ ./client1 127.0.0.1 9000
Waiting for players to start the game
 Connection stablished you are the second player, game can start
 In which line would you like to place your movement? [1-5]
7
 sent
 That's an out of bounds location, remember that the range of the matrix is [1-5]
7
 sent
 That's an out of bounds location, remember that the range of the matrix is [1-5]
4
 sent
 In which column would you like to place your movement? [1-5]
4
 sent
 In which line would you like to place your movement? [1-5]
2
 sent
 In which column would you like to place your movement? [1-5]
2
 sent
```

```
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | |X| |
5| | | | | |
2,sem1 0
2,sem2 0
C 2: Position line selected: 4

C 2: Position column selected: 4

Line: 4Column: 4
This position has already been taken.
C 2: Position line selected: 2

C 2: Position column selected: 2

Line: 2Column: 2
  1 2 3 4 5
1| | | | | |
2| |O| | | |
3| | | | | |
4| | | |X| |
5| | | | | |
Did the client2 win?[Y/N]
```

Once the move is already set on the board, the server will ask the user if that client move wins the game or not. If it is not a win move it will ask the second client to introduce one itself and if it is valid will ask it if this move wins the game or not as well.

At this point the execution keeps that methodology until it is decided that a client wins.

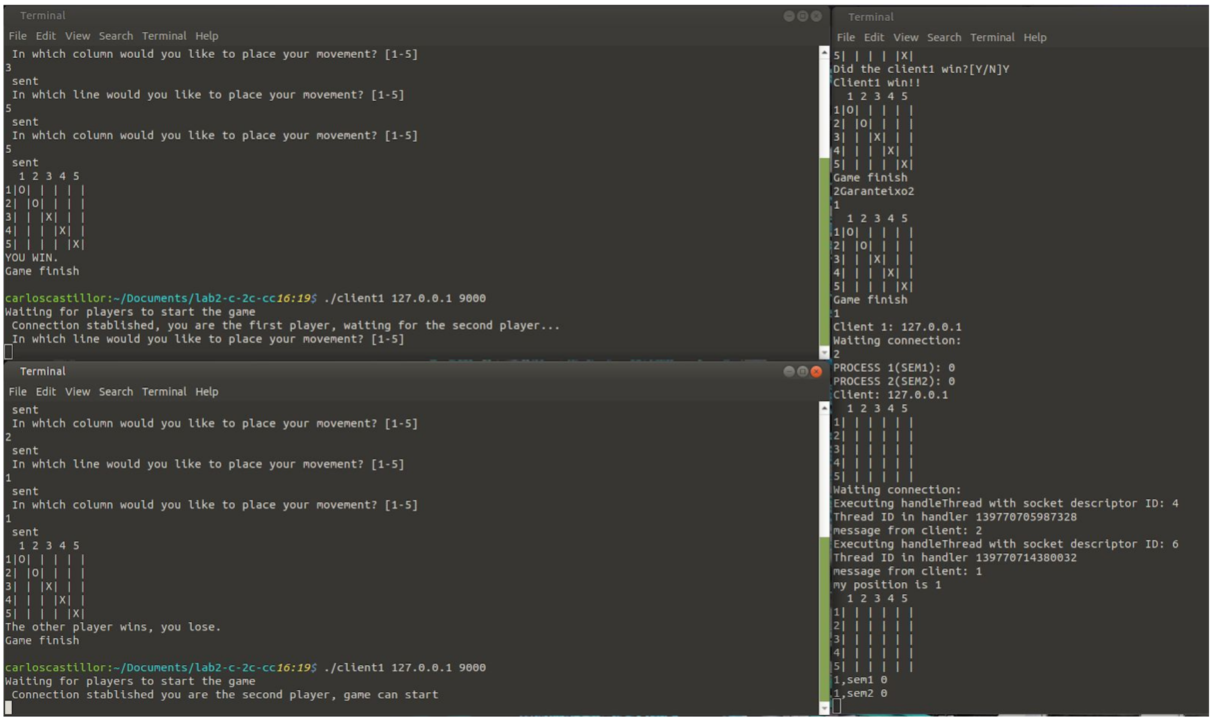

Terminal
File Edit View Search Terminal Help
Terminal
File Edit View Search Terminal Help
Terminal
File Edit View Search Terminal Help

```
 sent
 In which line would you like to place your movement? [1-5]
3
 sent
 In which column would you like to place your movement? [1-5]
3
 sent
 In which line would you like to place your movement? [1-5]
5
 sent
 In which column would you like to place your movement? [1-5]
5
 sent
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | |X|
YOU WIN.
Game finish

carloscastillor:~/Documents/lab2-c-2c-cc16:19$
```

```
 sent
 In which line would you like to place your movement? [1-5]
2
 sent
 In which column would you like to place your movement? [1-5]
2
 sent
 In which line would you like to place your movement? [1-5]
1
 sent
 In which column would you like to place your movement? [1-5]
1
 sent
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | |X|
The other player wins, you lose.
Game finish

carloscastillor:~/Documents/lab2-c-2c-cc16:19$
```

```
1Garanteixo
2Garanteixo2
C 2: Position line selected: 1

C 2: Position column selected: 1

Line: 1Column: 1
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | | |
Did the client2 win?[Y/N]N
2
PROCESS 1(SEM1): 0
PROCESS 2(SEM2): 0
2Garanteixo
1Garanteixo2
C 1: Position line selected: 5

C 1: Position column selected: 5

Line: 5Column: 5
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | |X|
Did the client1 win?[Y/N]Y
Client1 win!!
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | |X|
Game finish
2Garanteixo2
1
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | |X|
Game finish
```

If either the client1 or client2 is the winner it will show a custom message and the final board result in each terminal. And finally the execution process of those clients finishes. The server execution will remain active and waiting for another couple clients to join a game.

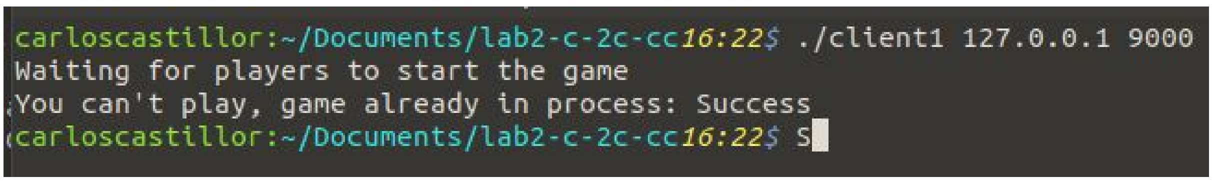

Terminal
File Edit View Search Terminal Help
Terminal
File Edit View Search Terminal Help
Terminal
File Edit View Search Terminal Help

```
 In which column would you like to place your movement? [1-5]
3
 sent
 In which line would you like to place your movement? [1-5]
5
 sent
 In which column would you like to place your movement? [1-5]
5
 sent
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | |X|
YOU WIN.
Game finish

carloscastillor:~/Documents/lab2-c-2c-cc16:19$ ./client1 127.0.0.1 9000
Waiting for players to start the game
 Connection stablished, you are the first player, waiting for the second player...
 In which line would you like to place your movement? [1-5]
```

```
 sent
 In which column would you like to place your movement? [1-5]
2
 sent
 In which line would you like to place your movement? [1-5]
1
 sent
 In which column would you like to place your movement? [1-5]
1
 sent
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | |X|
The other player wins, you lose.
Game finish

carloscastillor:~/Documents/lab2-c-2c-cc16:19$ ./client1 127.0.0.1 9000
Waiting for players to start the game
 Connection stablished you are the second player, game can start
```

```
5| | | | |X|
Did the client1 win?[Y/N]Y
Client1 win!!
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | |X|
Game finish
2Garanteixo2
1
  1 2 3 4 5
1|O| | | | |
2| |O| | | |
3| | |X| | |
4| | | |X| |
5| | | | |X|
Game finish
1
Client 1: 127.0.0.1
Waiting connection:
2
PROCESS 1(SEM1): 0
PROCESS 2(SEM2): 0
Client: 127.0.0.1
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | | | |
5| | | | | |
Waiting connection:
Executing handleThread with socket descriptor ID: 4
Thread ID in handler 139770705987328
message from client: 2
Executing handleThread with socket descriptor ID: 6
Thread ID in handler 139770714380032
message from client: 1
my position is 1
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | | | |
5| | | | | |
1,sem1 0
1,sem2 0
```

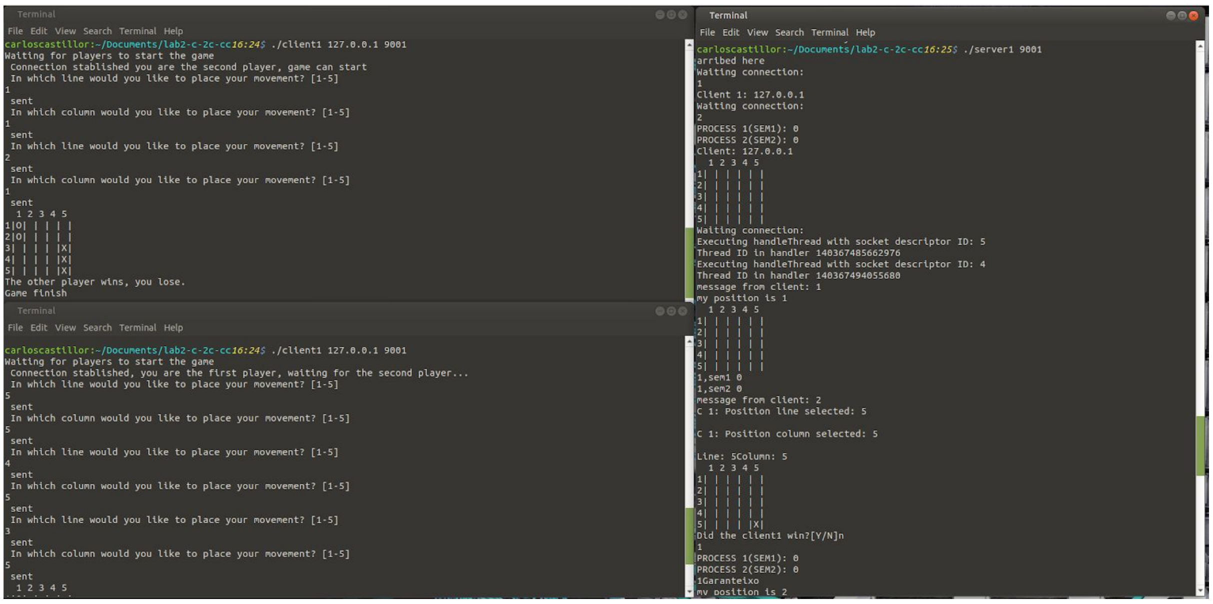
*prove that you can start a game after already one has finished.



```
carloscastillor:~/Documents/lab2-c-2c-cc16:22$ ./client1 127.0.0.1 9000
Waiting for players to start the game
You can't play, game already in process: Success
carloscastillor:~/Documents/lab2-c-2c-cc16:22$ S
```

*Prove that a client3 can not join a game.

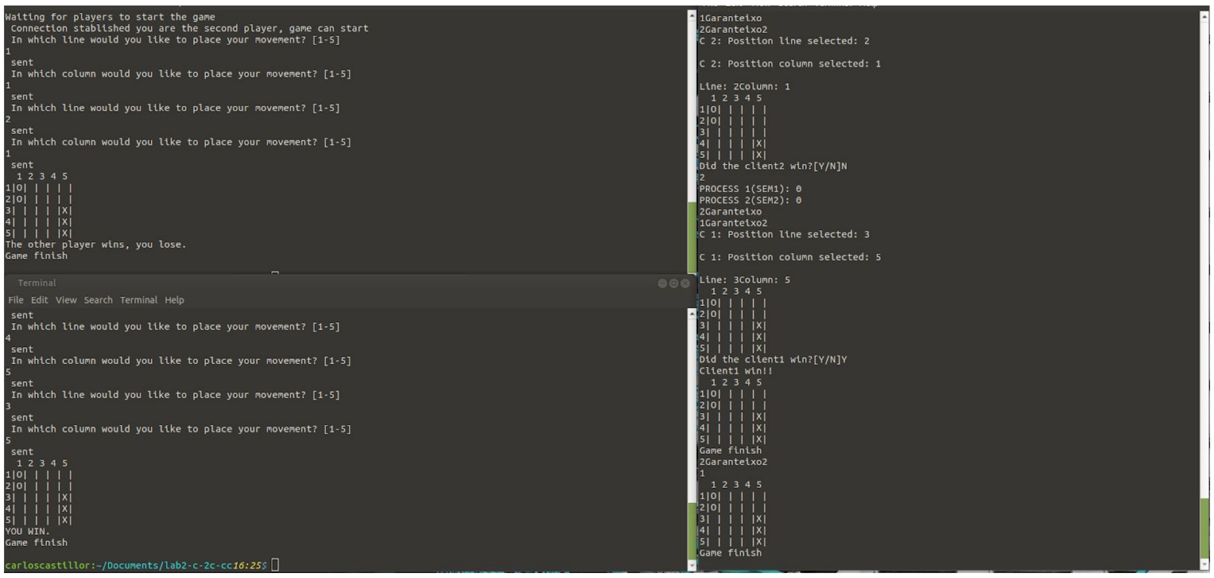
- Other example:



```
carloscastillor:~/Documents/lab2-c-2c-cc16:24$ ./client1 127.0.0.1 9001
Waiting for players to start the game
 Connection stablished you are the second player, game can start
 In which line would you like to place your movement? [1-5]
1
 sent
 In which column would you like to place your movement? [1-5]
1
 sent
 In which line would you like to place your movement? [1-5]
2
 sent
 In which column would you like to place your movement? [1-5]
1
 sent
  1 2 3 4 5
1|O| | | | |
2|O| | | | |
3| | | | |X|
4| | | | |X|
5| | | | |X|
The other player wins, you lose.
Game finish
```

```
carloscastillor:~/Documents/lab2-c-2c-cc16:24$ ./client1 127.0.0.1 9001
Waiting for players to start the game
 Connection stablished, you are the first player, waiting for the second player...
 In which line would you like to place your movement? [1-5]
5
 sent
 In which column would you like to place your movement? [1-5]
5
 sent
 In which line would you like to place your movement? [1-5]
4
 sent
 In which column would you like to place your movement? [1-5]
5
 sent
 In which line would you like to place your movement? [1-5]
3
 sent
 In which column would you like to place your movement? [1-5]
5
 sent
  1 2 3 4 5
```

```
carloscastillor:~/Documents/lab2-c-2c-cc16:25$ ./server1 9001
arribed here
Waiting connection:
1
Client 1: 127.0.0.1
Waiting connection:
2
PROCESS 1(SEM1): 0
PROCESS 2(SEM2): 0
Client: 127.0.0.1
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | | | |
5| | | | | |
Waiting connection:
Executing handleThread with socket descriptor ID: 5
Thread ID in handler 140367485662976
Executing handleThread with socket descriptor ID: 4
Thread ID in handler 140367494055680
message from client: 1
my position is 1
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | | | |
5| | | | | |
1,sem1 0
1,sem2 0
message from client: 2
C 1: Position line selected: 5

C 1: Position column selected: 5

Line: 5Column: 5
  1 2 3 4 5
1| | | | | |
2| | | | | |
3| | | | | |
4| | | | | |
5| | | | |X|
Did the client1 win?[Y/N]n
1
PROCESS 1(SEM1): 0
PROCESS 2(SEM2): 0
1Garanteixo
my position is 2
```



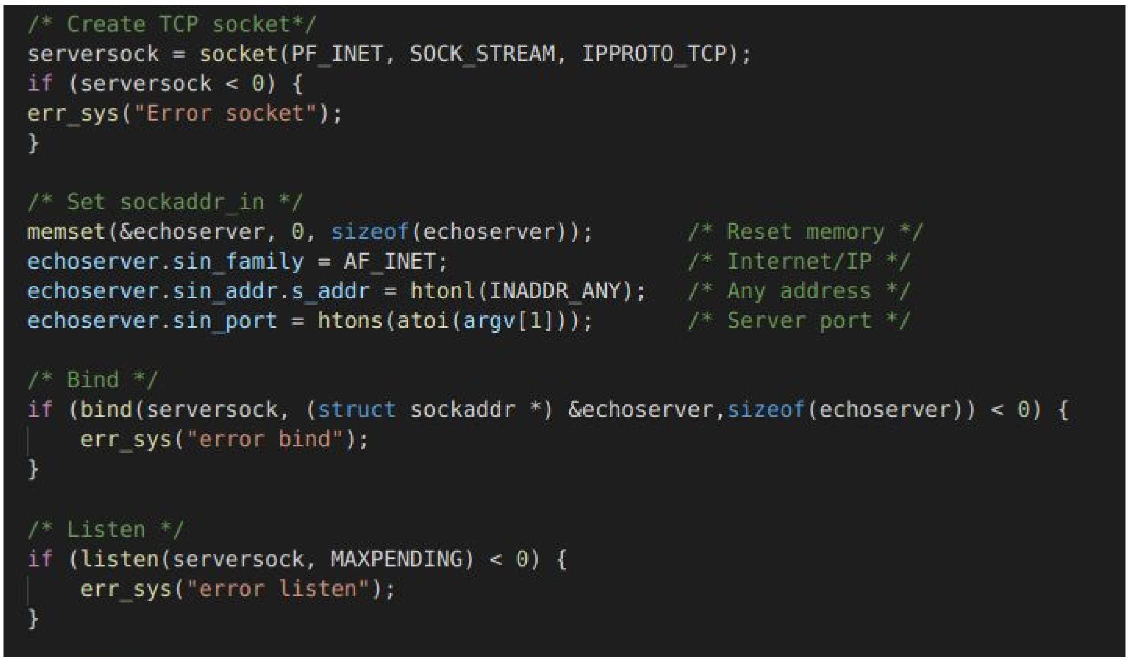
```
Waiting for players to start the game
 Connection stablished you are the second player, game can start
 In which line would you like to place your movement? [1-5]
1
 sent
 In which column would you like to place your movement? [1-5]
1
 sent
 In which line would you like to place your movement? [1-5]
2
 sent
 In which column would you like to place your movement? [1-5]
1
 sent
  1 2 3 4 5
1|O| | | | |
2|O| | | | |
3| | | | |X|
4| | | | |X|
5| | | | |X|
The other player wins, you lose.
Game finish
```

```
Terminal
File  Edit  View  Search  Terminal  Help
 sent
 In which line would you like to place your movement? [1-5]
4
 sent
 In which column would you like to place your movement? [1-5]
5
 sent
 In which line would you like to place your movement? [1-5]
3
 sent
 In which column would you like to place your movement? [1-5]
5
 sent
  1 2 3 4 5
1|O| | | | |
2|O| | | | |
3| | | | |X|
4| | | | |X|
5| | | | |X|
YOU WIN.
Game finish

carloscastillor:~/Documents/lab2-c-2c-cc16:25$
```

```
1Garanteixo
2Garanteixo2
C 2: Position line selected: 2

C 2: Position column selected: 1

Line: 2Column: 1
  1 2 3 4 5
1|O| | | | |
2|O| | | | |
3| | | | | |
4| | | | |X|
5| | | | |X|
Did the client2 win?[Y/N]N
2
PROCESS 1(SEM1): 0
PROCESS 2(SEM2): 0
2Garanteixo
1Garanteixo2
C 1: Position line selected: 3

C 1: Position column selected: 5

Line: 3Column: 5
  1 2 3 4 5
1|O| | | | |
2|O| | | | |
3| | | | |X|
4| | | | |X|
5| | | | |X|
Did the client1 win?[Y/N]Y
Client1 win!!
  1 2 3 4 5
1|O| | | | |
2|O| | | | |
3| | | | |X|
4| | | | |X|
5| | | | |X|
Game finish
2Garanteixo2
1
  1 2 3 4 5
1|O| | | | |
2|O| | | | |
3| | | | |X|
4| | | | |X|
5| | | | |X|
Game finish
```

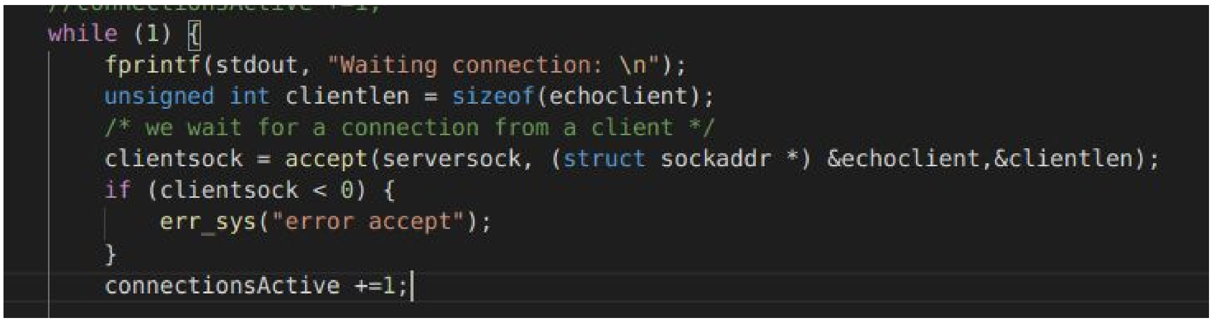
- Comments on the relevant parts of the code.

Server1



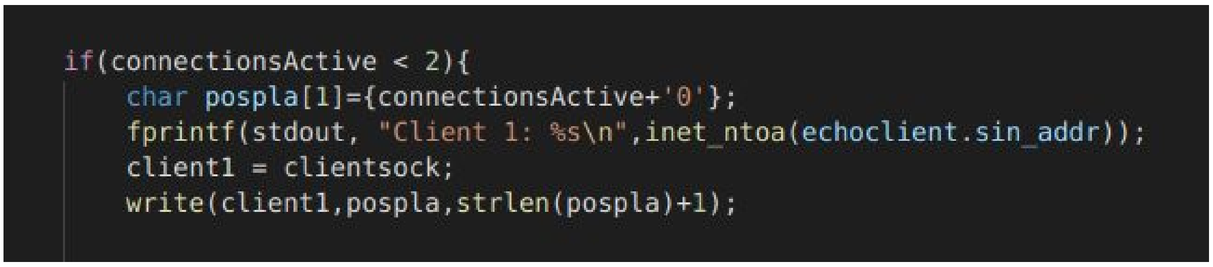
```c
/* Create TCP socket*/
serversock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (serversock < 0) {
err_sys("Error socket");
}

/* Set sockaddr_in */
memset(&echoserver, 0, sizeof(echoserver));       /* Reset memory */
echoserver.sin_family = AF_INET;                  /* Internet/IP */
echoserver.sin_addr.s_addr = htonl(INADDR_ANY);   /* Any address */
echoserver.sin_port = htons(atoi(argv[1]));       /* Server port */

/* Bind */
if (bind(serversock, (struct sockaddr *) &echoserver,sizeof(echoserver)) < 0) {
    err_sys("error bind");
}

/* Listen */
if (listen(serversock, MAXPENDING) < 0) {
    err_sys("error listen");
}
```

About the server code here we can say that a socket is created for the server. To avoid problems, we reset the memory, set the IP, specify which ips can access and the port of the server and wait for a connection .



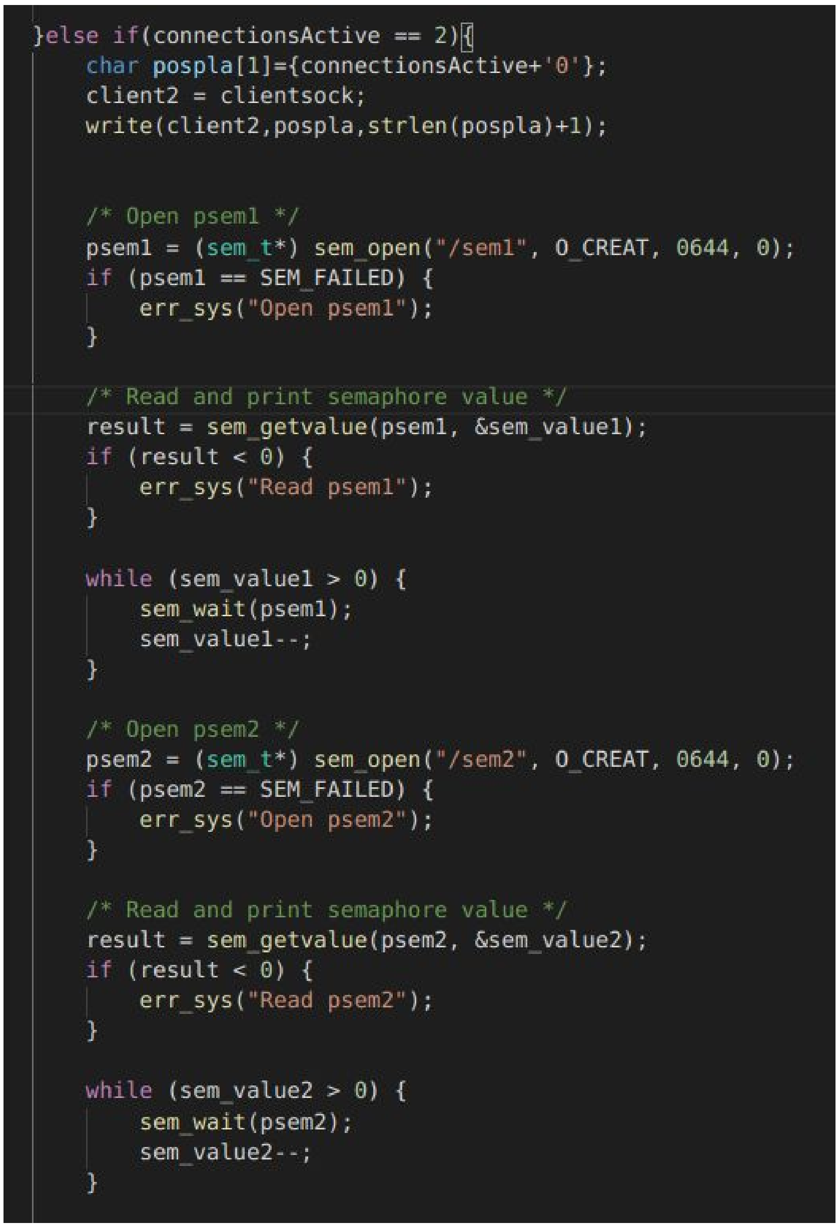
```c
while (1) {
    fprintf(stdout, "Waiting connection: \n");
    unsigned int clientlen = sizeof(echoclient);
    /* we wait for a connection from a client */
    clientsock = accept(serversock, (struct sockaddr *) &echoclient,&clientlen);
    if (clientsock < 0) {
        err_sys("error accept");
    }
    connectionsActive +=1;
```

Because we want to have to clients connected, we are using threads for every client, so here can see that the first thing we do is wait for a connection of a client and count the number of connections we already have.



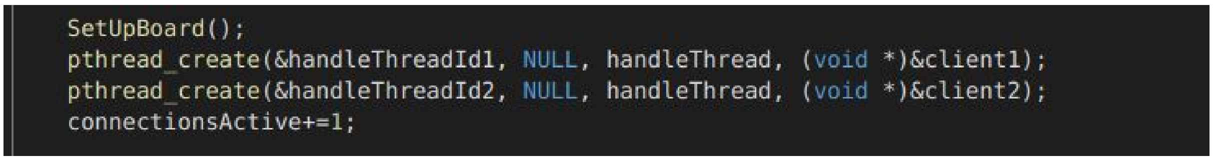
```c
if(connectionsActive < 2){
    char pospla[1]={connectionsActive+'0'};
    fprintf(stdout, "Client 1: %s\n",inet_ntoa(echoclient.sin_addr));
    client1 = clientsock;
    write(client1,pospla,strlen(pospla)+1);
```

We check the connections we have, if the number of connections is under 2 we send the position of the player to the client and save the clientsocket in a variable client and wait for another connection



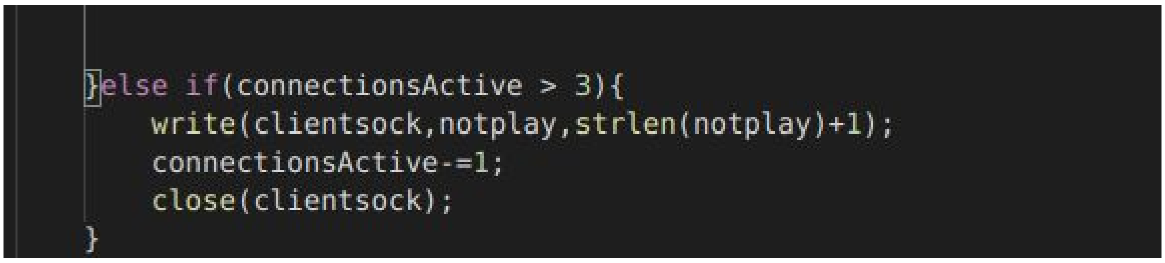
```
}else if(connectionsActive == 2){
    char pospla[1]={connectionsActive+'0'};
    client2 = clientsock;
    write(client2,pospla,strlen(pospla)+1);


    /* Open psem1 */
    psem1 = (sem_t*) sem_open("/sem1", O_CREAT, 0644, 0);
    if (psem1 == SEM_FAILED) {
        err_sys("Open psem1");
    }

    /* Read and print semaphore value */
    result = sem_getvalue(psem1, &sem_value1);
    if (result < 0) {
        err_sys("Read psem1");
    }

    while (sem_value1 > 0) {
        sem_wait(psem1);
        sem_value1--;
    }

    /* Open psem2 */
    psem2 = (sem_t*) sem_open("/sem2", O_CREAT, 0644, 0);
    if (psem2 == SEM_FAILED) {
        err_sys("Open psem2");
    }

    /* Read and print semaphore value */
    result = sem_getvalue(psem2, &sem_value2);
    if (result < 0) {
        err_sys("Read psem2");
    }

    while (sem_value2 > 0) {
        sem_wait(psem2);
        sem_value2--;
    }
```

When we receive a new connection and the number of connections are equals to 2, first of all we do the same as when the number of connections are under 2 and then create and set to 0 the semaphore that we are gonna use to control which client is playing.



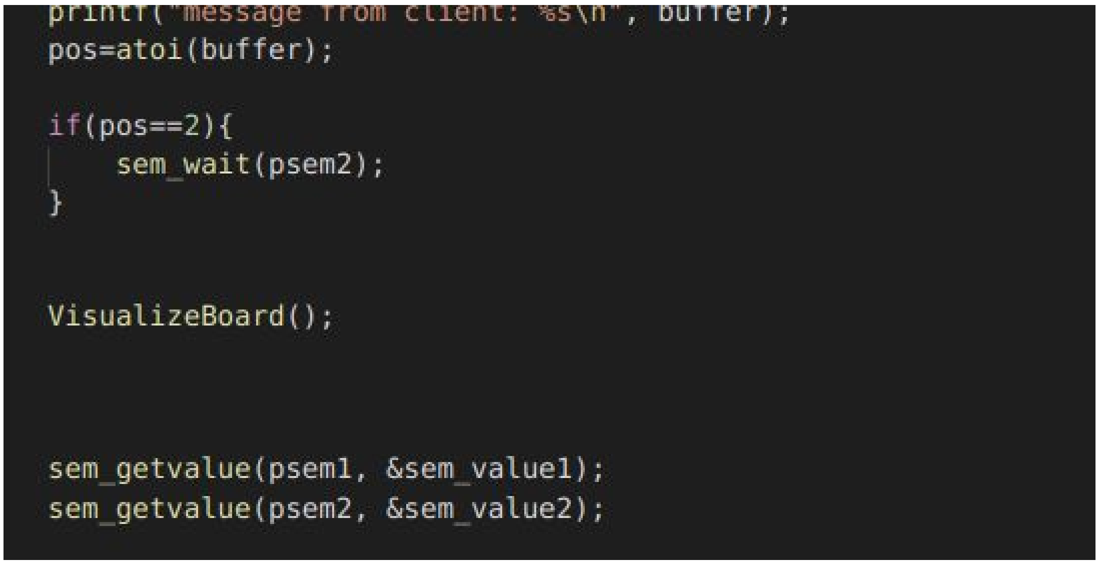
```
SetUpBoard();
pthread_create(&handleThreadId1, NULL, handleThread, (void *)&client1);
pthread_create(&handleThreadId2, NULL, handleThread, (void *)&client2);
connectionsActive+=1;
```

When all is created and there are no errors, we create the threads to start playing the game.



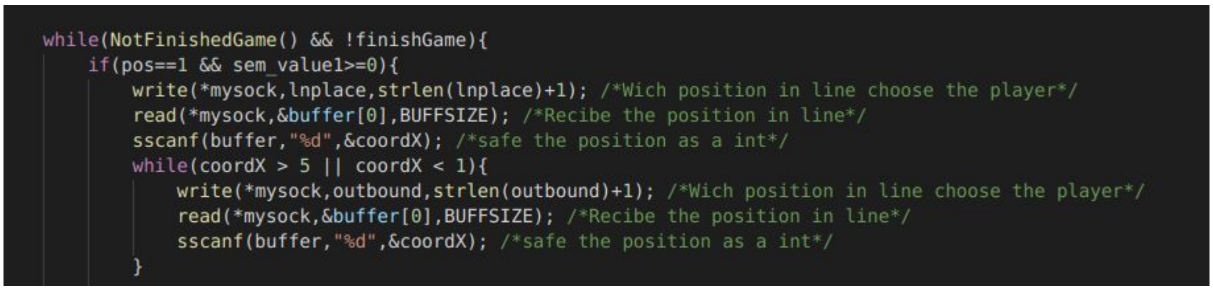
```
}else if(connectionsActive > 3){
    write(clientsock,notplay,strlen(notplay)+1);
    connectionsActive-=1;
    close(clientsock);
}
```

In case, someone wants to connect to the server but there is a game in progress the server accepts the connection and then says to the client that it's not possible to play and close the connection.



```
printf("message from client: %s\n", buffer);
pos=atoi(buffer);

if(pos==2){
    sem_wait(psem2);
}


VisualizeBoard();



sem_getvalue(psem1, &sem_value1);
sem_getvalue(psem2, &sem_value2);
```

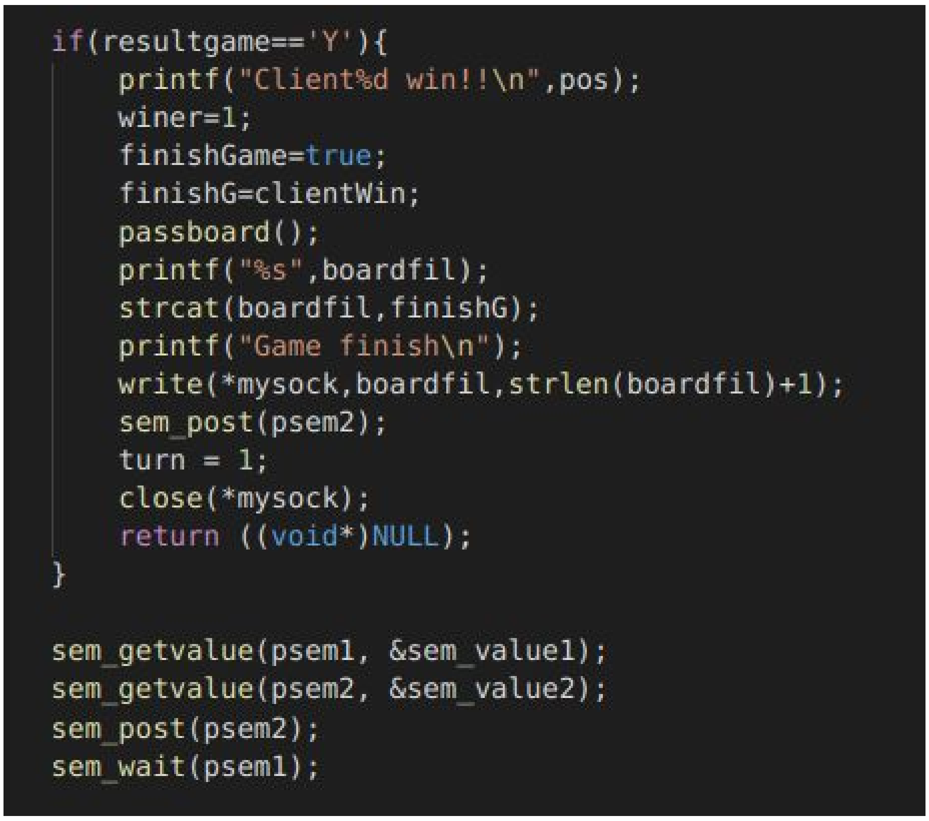
When the game start, we check the position of the player and if the position is 2, meaning that this thread is the thread of the player number 2 we make a wait in the semaphore number 2 to stop the process until the player 1 ends the turn and make a post on the semaphore of the second player and a wait on his semaphore.



```
while(NotFinishedGame() && !finishGame){
    if(pos==1 && sem_value1>=0){
        write(*mysock,lnplace,strlen(lnplace)+1); /*Wich position in line choose the player*/
        read(*mysock,&buffer[0],BUFFSIZE); /*Recibe the position in line*/
        sscanf(buffer,"%d",&coordX); /*safe the position as a int*/
        while(coordX > 5 || coordX < 1){
            write(*mysock,outbound,strlen(outbound)+1); /*Wich position in line choose the player*/
            read(*mysock,&buffer[0],BUFFSIZE); /*Recibe the position in line*/
            sscanf(buffer,"%d",&coordX); /*safe the position as a int*/
        }
```

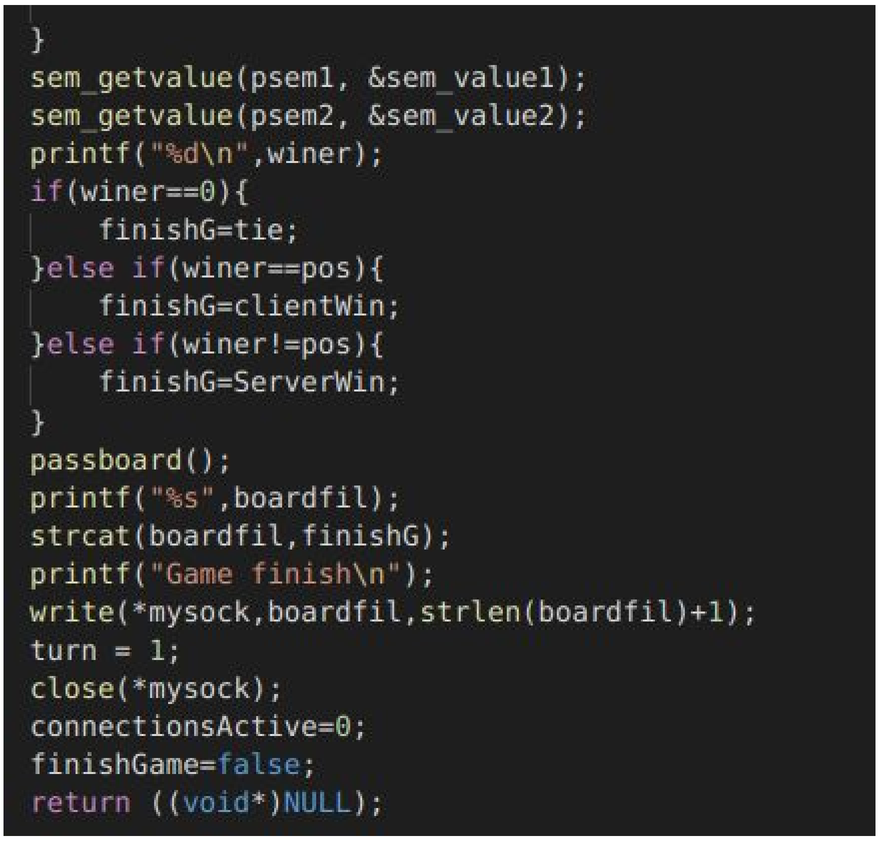
Here we can see the part of the code that has the implementation of the game. This part manages what the clients have to do with the clients when it’s their respective turns.

First we add the current board in a string, then we concatenate this string with a string that contains the line of the next movement asked and then this current board is sent to the client with the method write. Later we wait until the read method receives the line number and parse it to an int to check later if the move is correct along with the column number.



```
if(resultgame=='Y'){
    printf("Client%d win!!\n",pos);
    winer=1;
    finishGame=true;
    finishG=clientWin;
    passboard();
    printf("%s",boardfil);
    strcat(boardfil,finishG);
    printf("Game finish\n");
    write(*mysock,boardfil,strlen(boardfil)+1);
    sem_post(psem2);
    turn = 1;
    close(*mysock);
    return ((void*)NULL);
}

sem_getvalue(psem1, &sem_value1);
sem_getvalue(psem2, &sem_value2);
sem_post(psem2);
sem_wait(psem1);
```

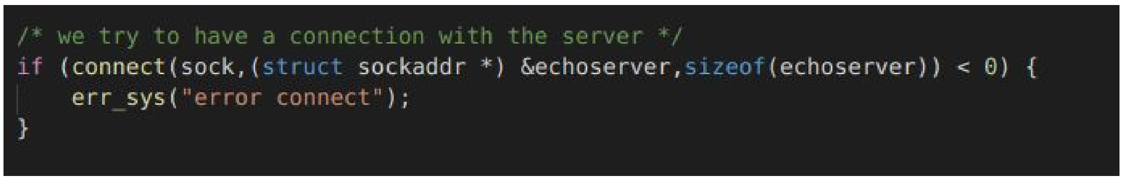
When we check if any of the players win the things we do are:
Set in finishG the string to say to the client who wins, make the method passboard to set a string with the board, added to the sentence of finishG and then activate the semaphore of the other client and close the socket of the client who wins.
If the client doesn’t win, we set the semaphore of the other server to active and put in wait the other one.



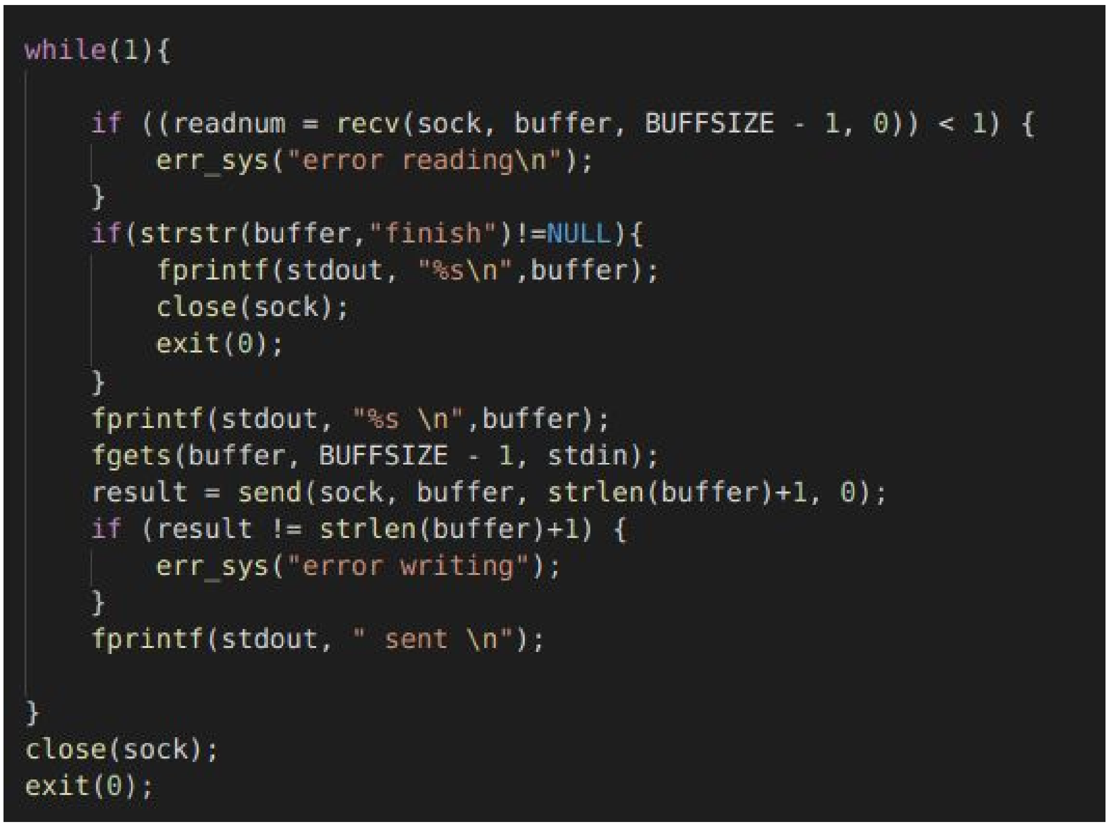
```
}
sem_getvalue(psem1, &sem_value1);
sem_getvalue(psem2, &sem_value2);
printf("%d\n",winer);
if(winer==0){
    finishG=tie;
}else if(winer==pos){
    finishG=clientWin;
}else if(winer!=pos){
    finishG=ServerWin;
}
passboard();
printf("%s",boardfil);
strcat(boardfil,finishG);
printf("Game finish\n");
write(*mysock,boardfil,strlen(boardfil)+1);
turn = 1;
close(*mysock);
connectionsActive=0;
finishGame=false;
return ((void*)NULL);
```

This part it’s gonna be used by the client who lose, here we check who is the winner for the case is a tie and then to the same as we do in the part of the client who wins.

Client1



```
/* we try to have a connection with the server */
if (connect(sock,(struct sockaddr *) &echoserver,sizeof(echoserver)) < 0) {
    err_sys("error connect");
}
```



```c
while(1){

    if ((readnum = recv(sock, buffer, BUFFSIZE - 1, 0)) < 1) {
        err_sys("error reading\n");
    }
    if(strstr(buffer,"finish")!=NULL){
        fprintf(stdout, "%s\n",buffer);
        close(sock);
        exit(0);
    }
    fprintf(stdout, "%s \n",buffer);
    fgets(buffer, BUFFSIZE - 1, stdin);
    result = send(sock, buffer, strlen(buffer)+1, 0);
    if (result != strlen(buffer)+1) {
        err_sys("error writing");
    }
    fprintf(stdout, " sent \n");

}
close(sock);
exit(0);
```

In the case of the code of the client it’s more or less the same as the server in terms of connectivity but one thing to say is that when the configuration of the connection that the client socket has it tries to connect to the server and if the client can’t it close the program , the only thing to remark is this screenshot above. This loop is for the part when the game is running. Once receiving a string, look if the string has the word “finish”, meaning that the game has ended and then the client socket can be closed, if it doesn’t contain the word “finish” then it will ask to write a number for the new move.