Mar 24
2021

LAB 2-C

OPERATING SYSTEMS

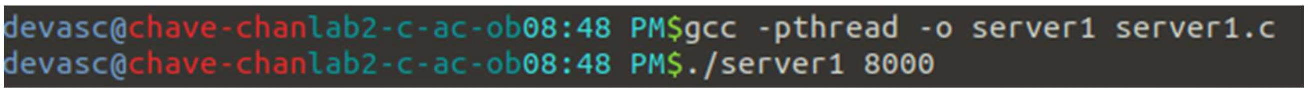
AIDA CHAVERO PÉREZ & ORIAC BONVEHÍ VILLA

Double Degree Computer Engineering and Videogames Development

INDEX

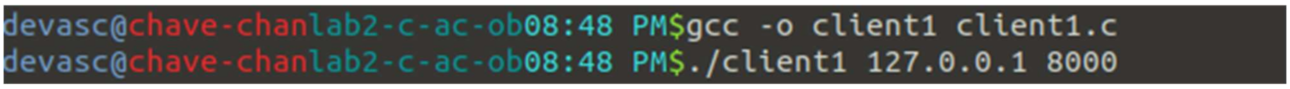
Compiling

To compile server1 you have to use gcc -pthread -o server1 server1.c to make the file. To run server1 you have to write ./server1 <port> to say that server1 is in the current directory and that you want server1 using the port number specified in the <port> section.



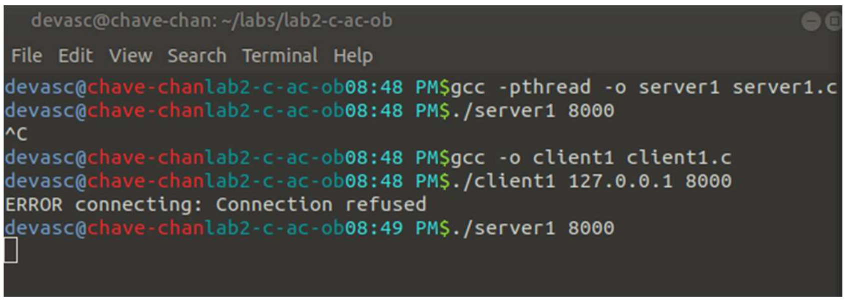
```
devasc@chave-chanlab2-c-ac-ob08:48 PM$gcc -pthread -o server1 server1.c
devasc@chave-chanlab2-c-ac-ob08:48 PM$./server1 8000
```

To compile client1 you have to use gcc -o client1 client1.c to make the file. To run client1 you have to write ./client1 <IP address> <port> to say that client1 is in the current directory and that you want client1 with IP address specified in <IP address> section (which has to be the IP address of the machine) and using the port number specified in the <port> section.

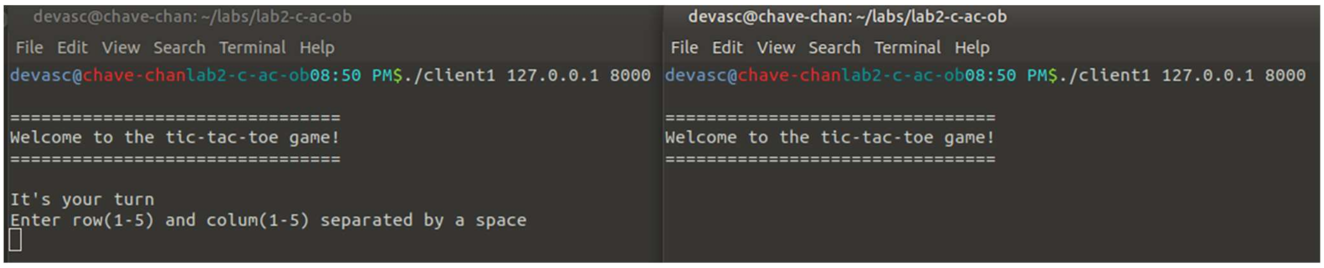


```
devasc@chave-chanlab2-c-ac-ob08:48 PM$gcc -o client1 client1.c
devasc@chave-chanlab2-c-ac-ob08:48 PM$./client1 127.0.0.1 8000
```

When server1 runs, the program waits until two clients are connected. If the clients have specified the correct IP address and port, both clients are identified, the connections are accepted and the game starts. If there is only one client, the server waits until a second client connects to start the game.



```
devasc@chave-chan: ~/labs/lab2-c-ac-ob
File Edit View Search Terminal Help
devasc@chave-chanlab2-c-ac-ob08:48 PM$gcc -pthread -o server1 server1.c
devasc@chave-chanlab2-c-ac-ob08:48 PM$./server1 8000
^C
devasc@chave-chanlab2-c-ac-ob08:48 PM$gcc -o client1 client1.c
devasc@chave-chanlab2-c-ac-ob08:48 PM$./client1 127.0.0.1 8000
ERROR connecting: Connection refused
devasc@chave-chanlab2-c-ac-ob08:49 PM$./server1 8000
```



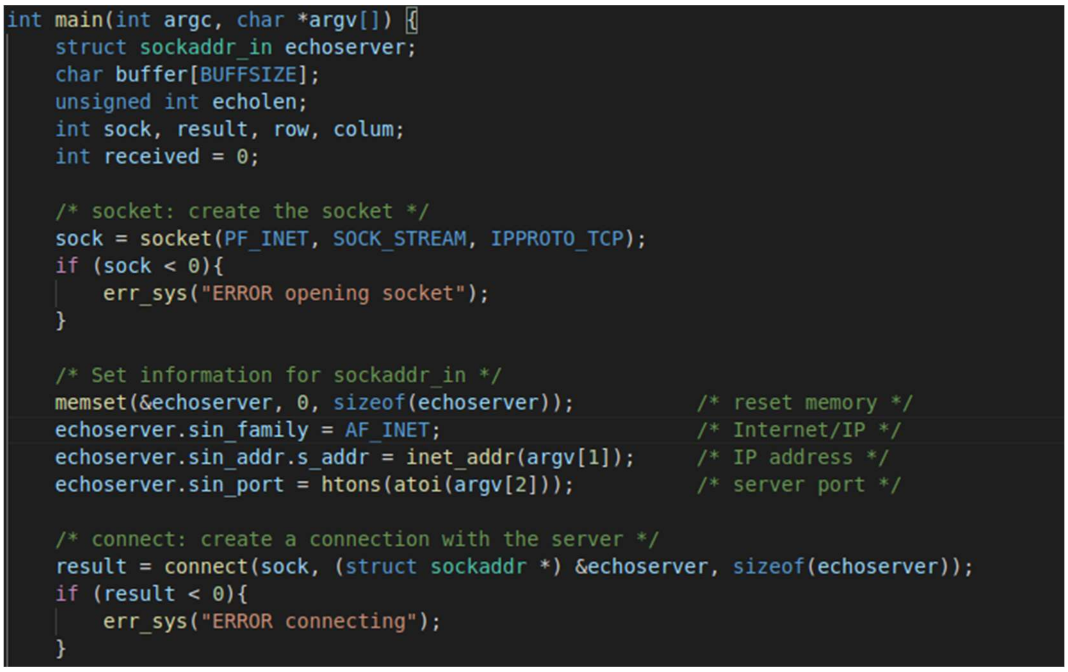
```
devasc@chave-chan: ~/labs/lab2-c-ac-ob
File Edit View Search Terminal Help
devasc@chave-chanlab2-c-ac-ob08:50 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
```

```
devasc@chave-chan: ~/labs/lab2-c-ac-ob
File Edit View Search Terminal Help
devasc@chave-chanlab2-c-ac-ob08:50 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================
```

Client Calling Process

This is client1 calling process:

1. Client creates the socket with the variables given. If the information is wrong, an error message is printed.
2. Client sets the information of the socket.
3. Client creates a connection with the server. If the connection fails, an error message is printed.

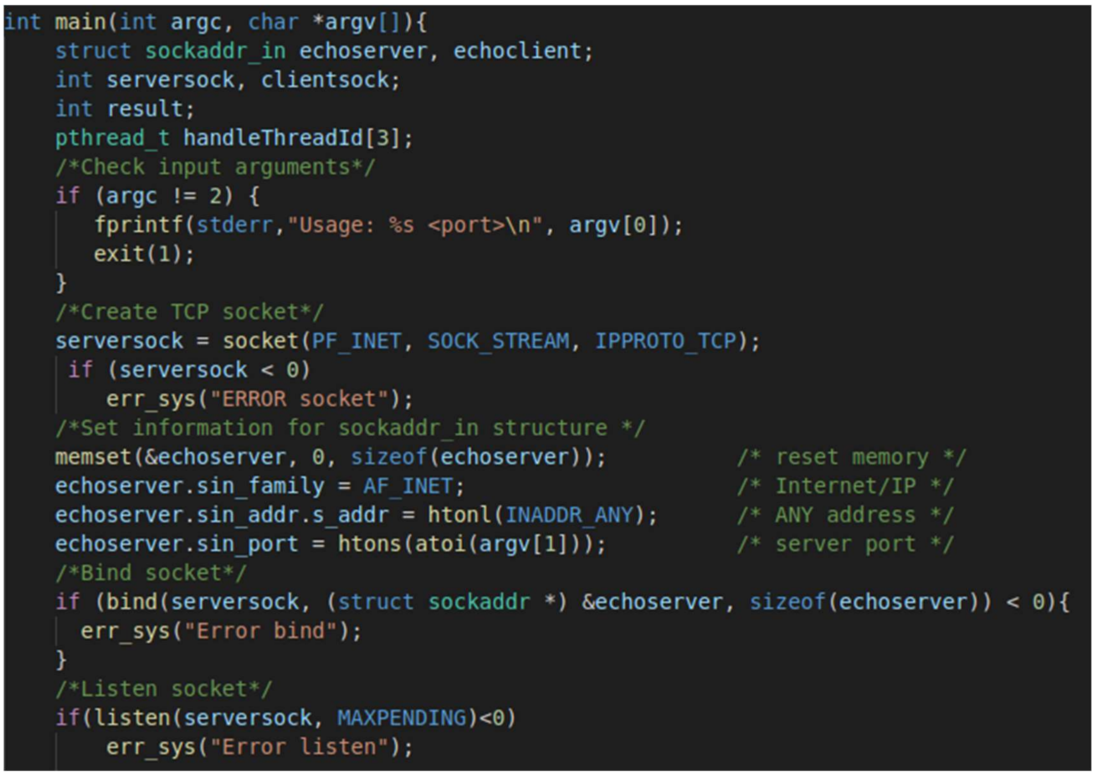


```c
int main(int argc, char *argv[]) {
    struct sockaddr_in echoserver;
    char buffer[BUFFSIZE];
    unsigned int echolen;
    int sock, result, row, colum;
    int received = 0;

    /* socket: create the socket */
    sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0){
        err_sys("ERROR opening socket");
    }

    /* Set information for sockaddr_in */
    memset(&echoserver, 0, sizeof(echoserver));          /* reset memory */
    echoserver.sin_family = AF_INET;                     /* Internet/IP */
    echoserver.sin_addr.s_addr = inet_addr(argv[1]);     /* IP address */
    echoserver.sin_port = htons(atoi(argv[2]));          /* server port */

    /* connect: create a connection with the server */
    result = connect(sock, (struct sockaddr *) &echoserver, sizeof(echoserver));
    if (result < 0){
        err_sys("ERROR connecting");
    }
```

Server Calling Process

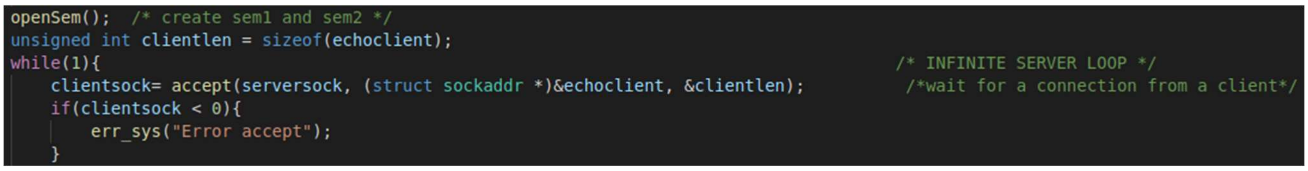
This is server1 calling process:

1. Server checks that the input arguments are correct.
2. Server creates the TCP socket.
3. Server sets the information of the socket.
4. Server bind the socket.
5. Server listens to the socket.



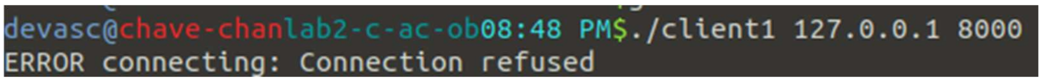
```c
int main(int argc, char *argv[]){
    struct sockaddr_in echoserver, echoclient;
    int serversock, clientsock;
    int result;
    pthread_t handleThreadId[3];
    /*Check input arguments*/
    if (argc != 2) {
       fprintf(stderr,"Usage: %s <port>\n", argv[0]);
       exit(1);
    }
    /*Create TCP socket*/
    serversock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
     if (serversock < 0)
        err_sys("ERROR socket");
    /*Set information for sockaddr_in structure */
    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 socket*/
    if (bind(serversock, (struct sockaddr *) &echoserver, sizeof(echoserver)) < 0){
      err_sys("Error bind");
    }
    /*Listen socket*/
    if(listen(serversock, MAXPENDING)<0)
        err_sys("Error listen");
```

This is how the server receives the client ID when client can play. If client cannot play or there is an error, an error message is printed.



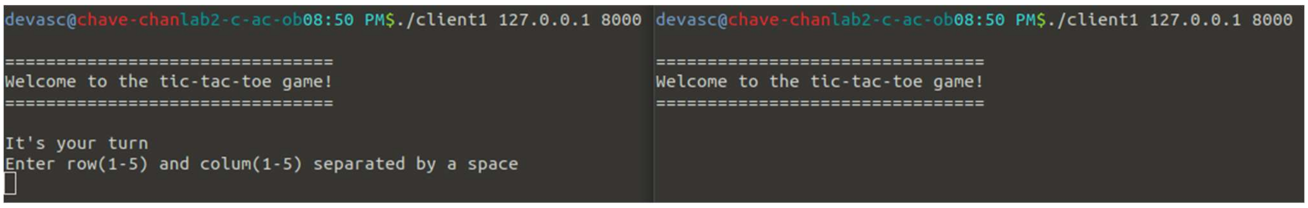
```
openSem();  /* create sem1 and sem2 */
unsigned int clientlen = sizeof(echoclient);
while(1){                                                               /* INFINITE SERVER LOOP */
    clientsock= accept(serversock, (struct sockaddr *)&echoclient, &clientlen);          /*wait for a connection from a client*/
    if(clientsock < 0){
        err_sys("Error accept");
    }
```

If the player cannot play, client receives an error message indicating that they cannot play.



```
devasc@chave-chanlab2-c-ac-ob08:48 PM$./client1 127.0.0.1 8000
ERROR connecting: Connection refused
```

If the client can play, the message "Welcome to the tic-tac-toe game!" is printed. If there are two players who can play, the game starts.

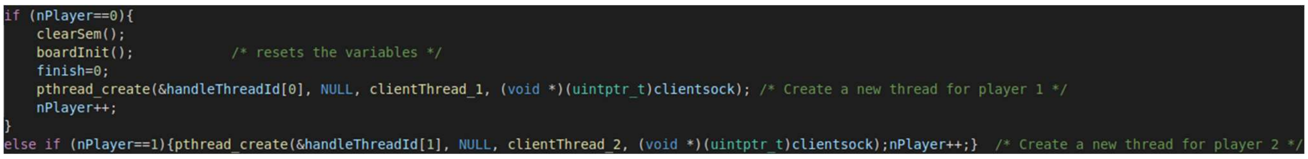


```
devasc@chave-chanlab2-c-ac-ob08:50 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
```

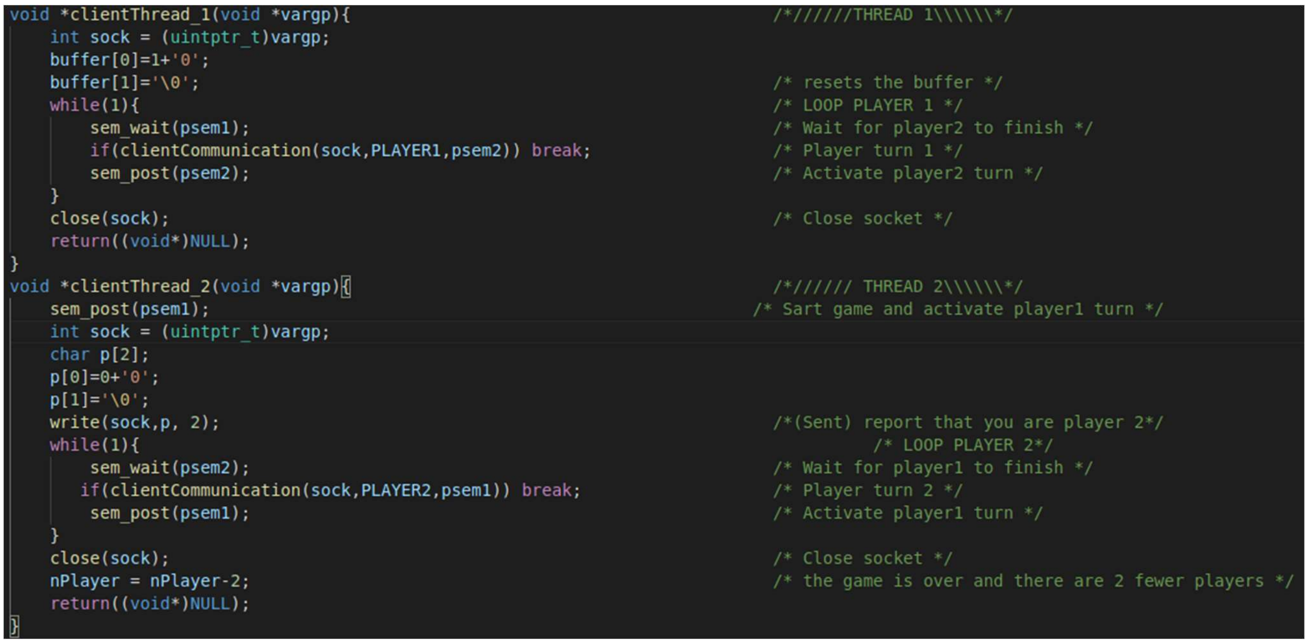
```
devasc@chave-chanlab2-c-ac-ob08:50 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================
```

Game

This is how server creates a new thread for player1 and player2 and starts a game.

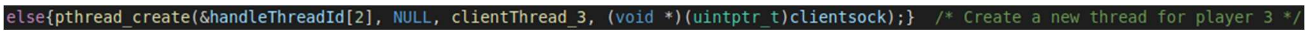


```
if (nPlayer==0){
    clearSem();
    boardInit();            /* resets the variables */
    finish=0;
    pthread_create(&handleThreadId[0], NULL, clientThread_1, (void *)(uintptr_t)clientsock); /* Create a new thread for player 1 */
    nPlayer++;
}
else if (nPlayer==1){pthread_create(&handleThreadId[1], NULL, clientThread_2, (void *)(uintptr_t)clientsock);nPlayer++;}  /* Create a new thread for player 2 */
```

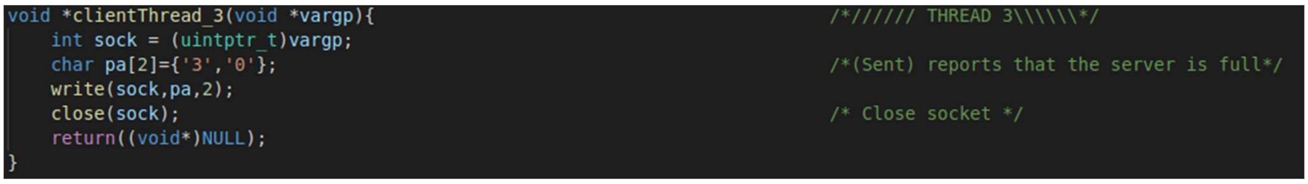


```
void *clientThread_1(void *vargp){                                    /*//////THREAD 1\\\\\\*/
    int sock = (uintptr_t)vargp;
    buffer[0]=1+'0';
    buffer[1]='\0';                                                   /* resets the buffer */
    while(1){                                                         /* LOOP PLAYER 1 */
        sem_wait(psem1);                                              /* Wait for player2 to finish */
        if(clientCommunication(sock,PLAYER1,psem2)) break;            /* Player turn 1 */
        sem_post(psem2);                                              /* Activate player2 turn */
    }
    close(sock);                                                      /* Close socket */
    return((void*)NULL);
}
void *clientThread_2(void *vargp){                                    /*////// THREAD 2\\\\\\*/
    sem_post(psem1);                                                 /* Sart game and activate player1 turn */
    int sock = (uintptr_t)vargp;
    char p[2];
    p[0]=0+'0';
    p[1]='\0';
    write(sock,p, 2);                                                 /*(Sent) report that you are player 2*/
    while(1){                                                                  /* LOOP PLAYER 2*/
        sem_wait(psem2);                                              /* Wait for player1 to finish */
       if(clientCommunication(sock,PLAYER2,psem1)) break;             /* Player turn 2 */
        sem_post(psem1);                                              /* Activate player1 turn */
    }
    close(sock);                                                      /* Close socket */
    nPlayer = nPlayer-2;                                              /* the game is over and there are 2 fewer players */
    return((void*)NULL);
}
```

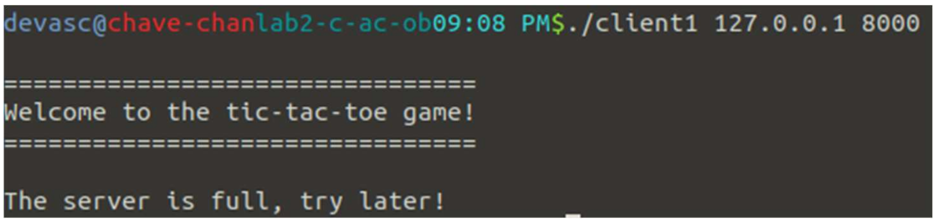
This is how server creates a new thread for player3 and prints an error message.



```
else{pthread_create(&handleThreadId[2], NULL, clientThread_3, (void *)(uintptr_t)clientsock);}  /* Create a new thread for player 3 */
```

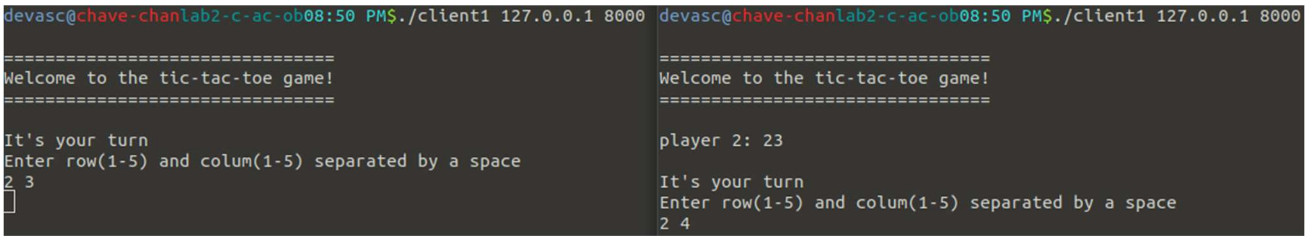


```
void *clientThread_3(void *vargp){                                      /*////// THREAD 3\\\\\\*/
    int sock = (uintptr_t)vargp;
    char pa[2]={'3','0'};                                               /*(Sent) reports that the server is full*/
    write(sock,pa,2);
    close(sock);                                                        /* Close socket */
    return((void*)NULL);
}
```



```
devasc@chave-chanlab2-c-ac-ob09:08 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================

The server is full, try later!
```

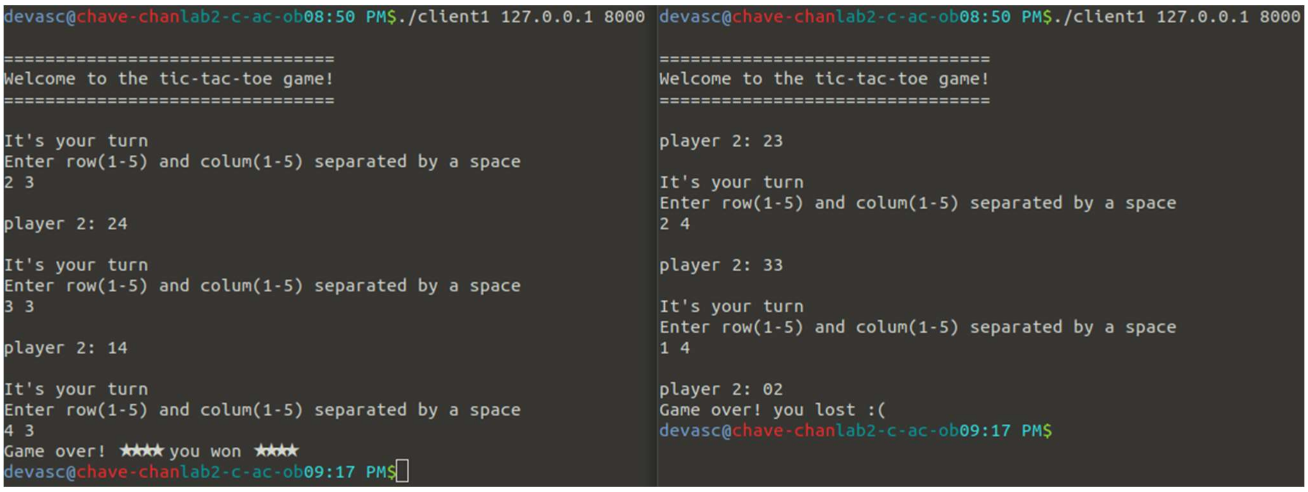
When the game starts, the first movement is made by player1. When player1 enters their movement, its turn for player2. There is no board printed because they have to play blinded.



```
devasc@chave-chanlab2-c-ac-ob08:50 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
2 3
```

```
devasc@chave-chanlab2-c-ac-ob08:50 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================

player 2: 23

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
2 4
```

This is an example of a full game played, with their respective end of game messages.

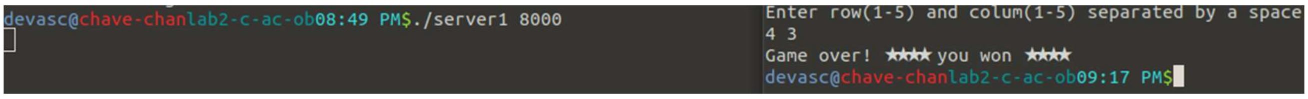


```
devasc@chave-chanlab2-c-ac-ob08:50 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
2 3

player 2: 24

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
3 3

player 2: 14

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
4 3
Game over! ★★★★ you won ★★★★
devasc@chave-chanlab2-c-ac-ob09:17 PM$
```

```
devasc@chave-chanlab2-c-ac-ob08:50 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================

player 2: 23

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
2 4

player 2: 33

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
1 4

player 2: 02
Game over! you lost :(
devasc@chave-chanlab2-c-ac-ob09:17 PM$
```

Finished game

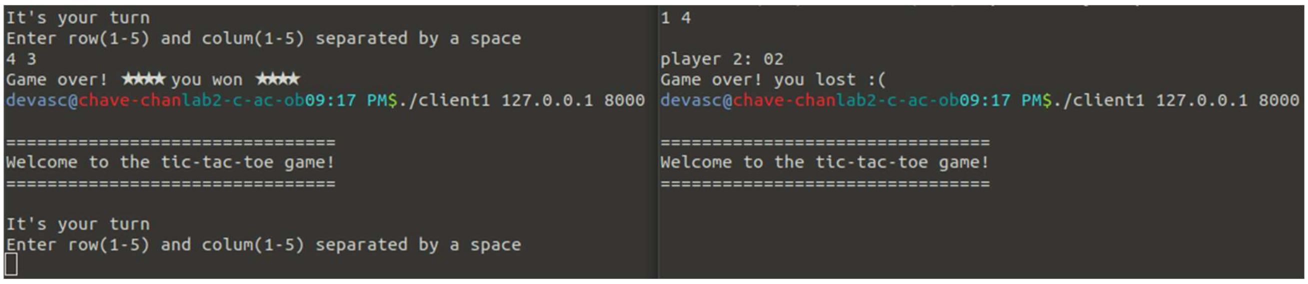
When the game finishes, the server keeps waiting for new connections, so its process does not end.



```
devasc@chave-chanlab2-c-ac-ob08:49 PM$./server1 8000
```

```
Enter row(1-5) and colum(1-5) separated by a space
4 3
Game over! ★★★★ you won ★★★★
devasc@chave-chanlab2-c-ac-ob09:17 PM$
```

After a game is finished, a new game can be played if two players connect to the server.

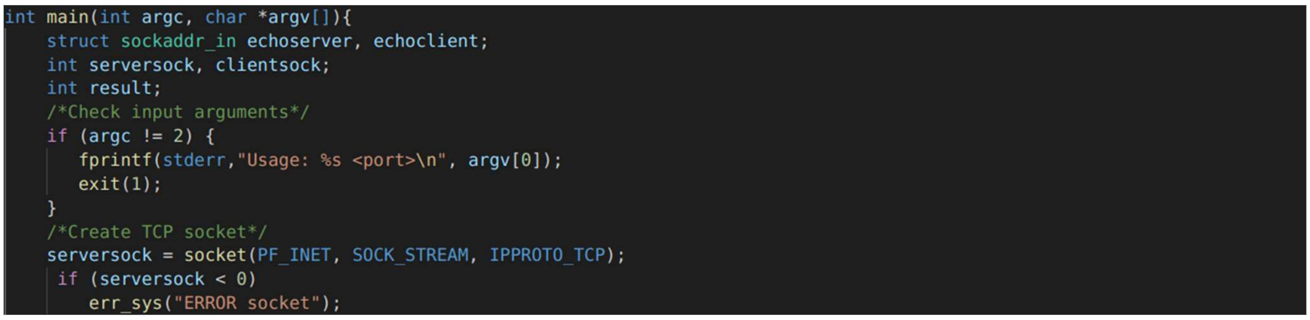


```
It's your turn
Enter row(1-5) and colum(1-5) separated by a space
4 3
Game over! ★★★★ you won ★★★★
devasc@chave-chanlab2-c-ac-ob09:17 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================

It's your turn
Enter row(1-5) and colum(1-5) separated by a space
```

```
1 4

player 2: 02
Game over! you lost :(
devasc@chave-chanlab2-c-ac-ob09:17 PM$./client1 127.0.0.1 8000

================================
Welcome to the tic-tac-toe game!
================================
```

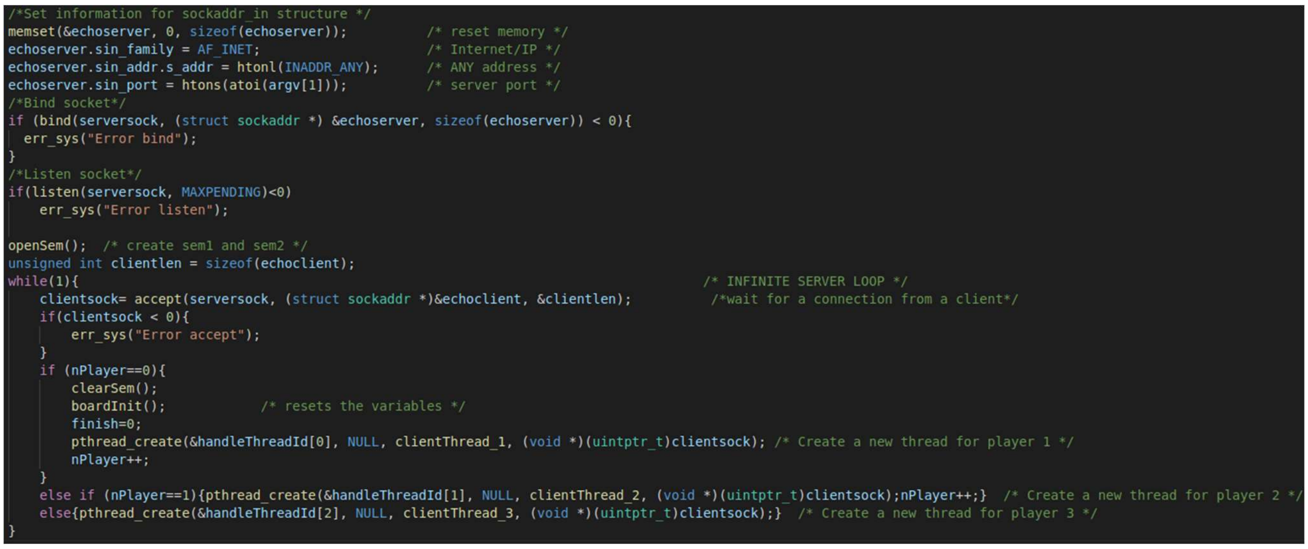
Code Server1 Explanation:

First the input arguments are checked and if they are correct, the TCP socket is created.



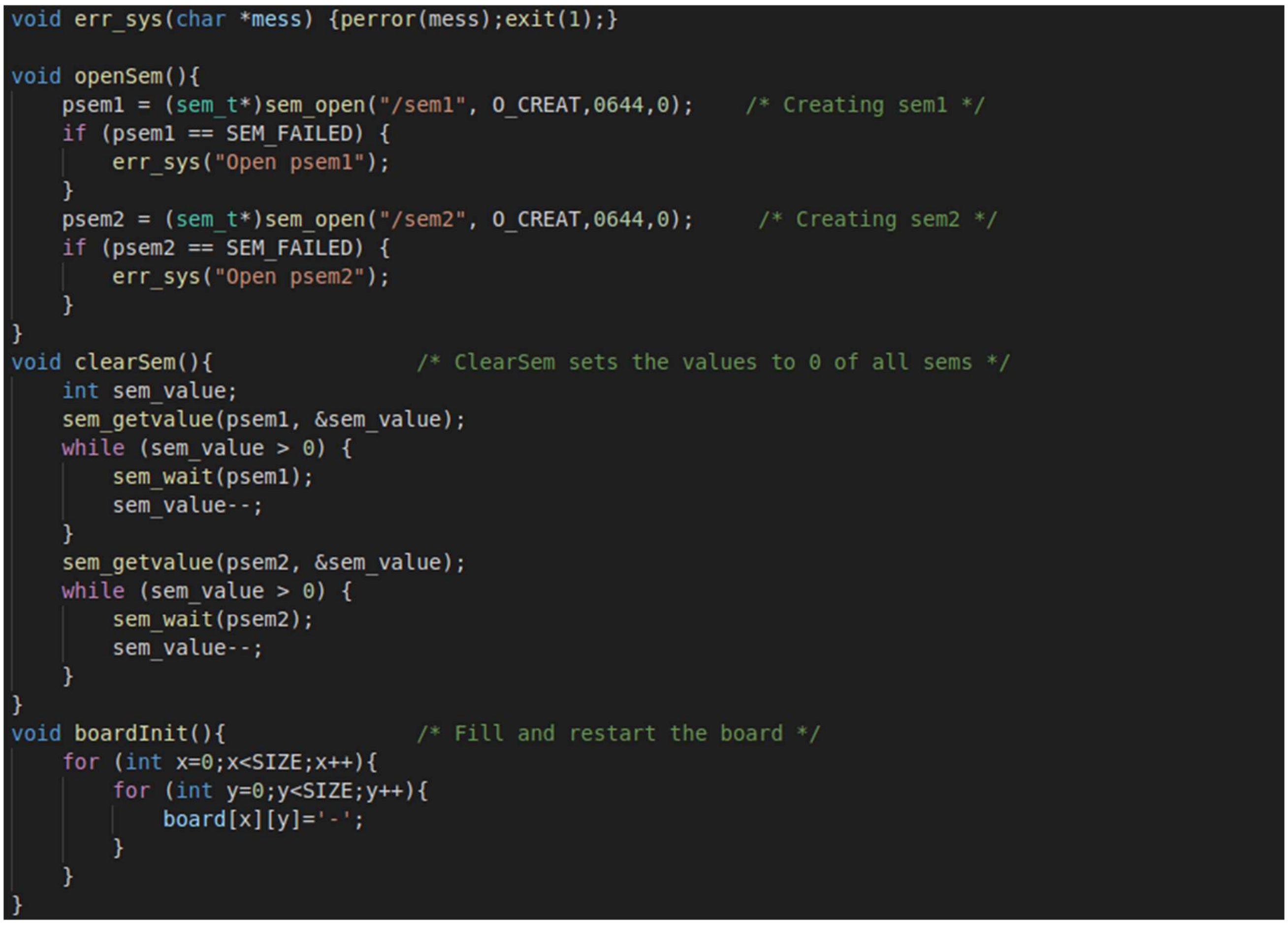
```c
int main(int argc, char *argv[]){
    struct sockaddr_in echoserver, echoclient;
    int serversock, clientsock;
    int result;
    /*Check input arguments*/
    if (argc != 2) {
       fprintf(stderr,"Usage: %s <port>\n", argv[0]);
       exit(1);
    }
    /*Create TCP socket*/
    serversock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
     if (serversock < 0)
        err_sys("ERROR socket");
```

The server sets the information of the socket, binds and listens to it. After those steps, the server opens semaphores sem1 and sem2, waits for players connections and if there are 2, the game starts.



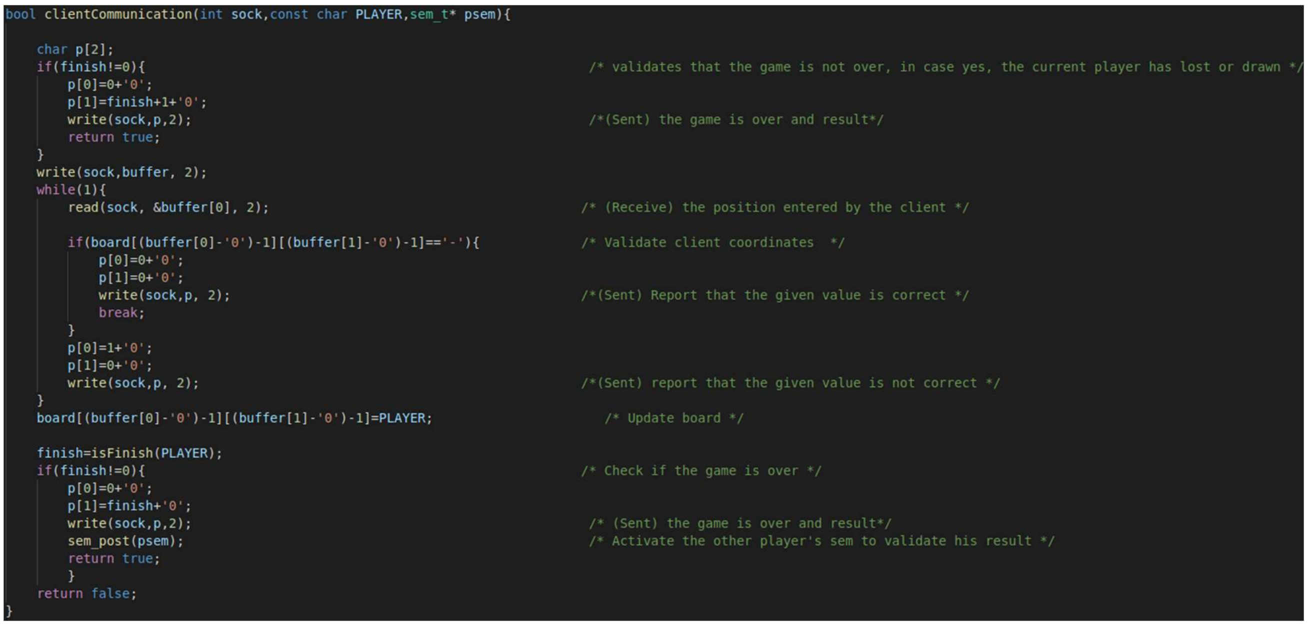
```c
/*Set information for sockaddr_in structure */
memset(&echoserver, 0, sizeof(echoserver));          /* reset memory */
echoserver.sin_family = AF_INET;                     /* Internet/IP */
echoserver.sin_addr.s_addr = htonl(INADDR_ANY);      /* ANY address */
echoserver.sin_port = htons(atoi(argv[1]));          /* server port */
/*Bind socket*/
if (bind(serversock, (struct sockaddr *) &echoserver, sizeof(echoserver)) < 0){
  err_sys("Error bind");
}
/*Listen socket*/
if(listen(serversock, MAXPENDING)<0)
    err_sys("Error listen");

openSem();  /* create sem1 and sem2 */
unsigned int clientlen = sizeof(echoclient);
while(1){                                                                  /* INFINITE SERVER LOOP */
    clientsock= accept(serversock, (struct sockaddr *)&echoclient, &clientlen);          /*wait for a connection from a client*/
    if(clientsock < 0){
        err_sys("Error accept");
    }
    if (nPlayer==0){
        clearSem();
        boardInit();            /* resets the variables */
        finish=0;
        pthread_create(&handleThreadId[0], NULL, clientThread_1, (void *)(uintptr_t)clientsock); /* Create a new thread for player 1 */
        nPlayer++;
    }
    else if (nPlayer==1){pthread_create(&handleThreadId[1], NULL, clientThread_2, (void *)(uintptr_t)clientsock);nPlayer++;}  /* Create a new thread for player 2 */
    else{pthread_create(&handleThreadId[2], NULL, clientThread_3, (void *)(uintptr_t)clientsock);}  /* Create a new thread for player 3 */
}
```

Those functions inform in case there is an error, open semaphores, clear semaphores and initialize the board, respectively.



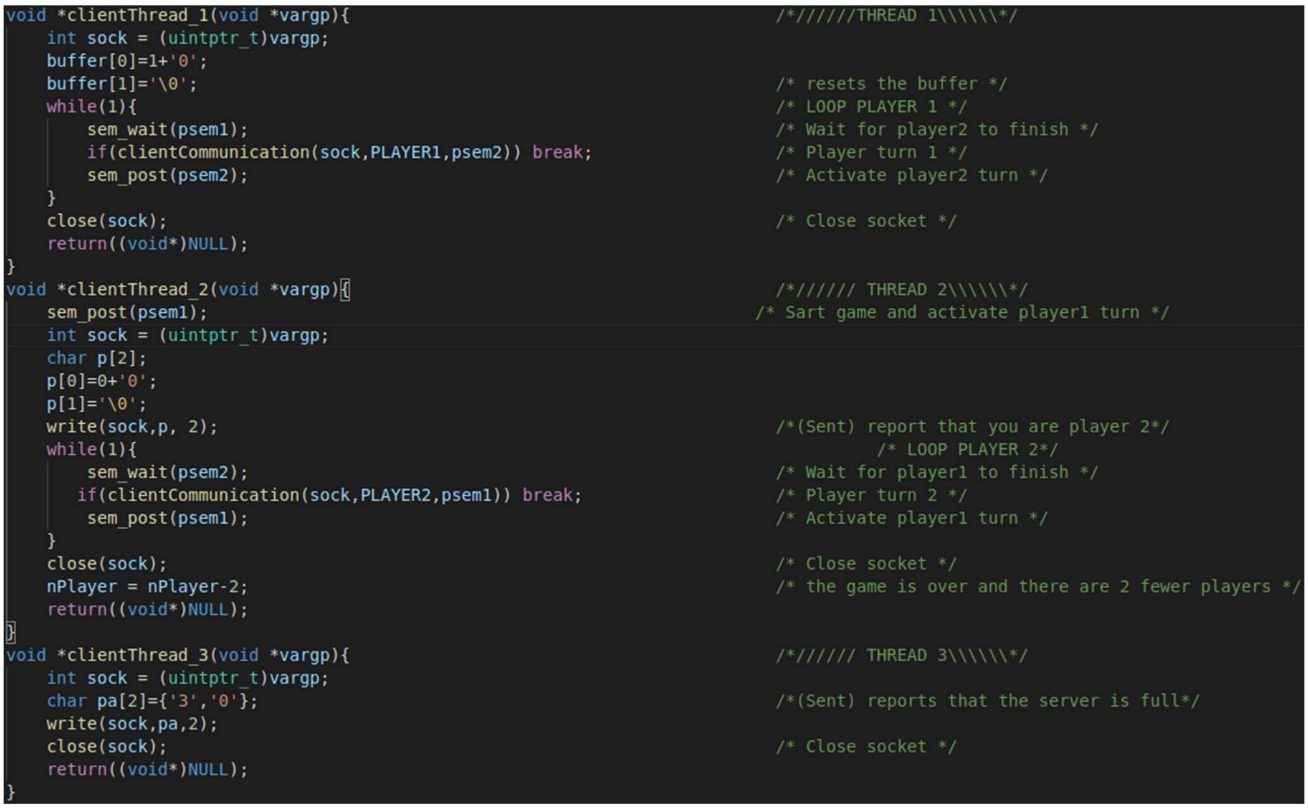
```c
void err_sys(char *mess) {perror(mess);exit(1);}

void openSem(){
    psem1 = (sem_t*)sem_open("/sem1", O_CREAT,0644,0);    /* Creating sem1 */
    if (psem1 == SEM_FAILED) {
        err_sys("Open psem1");
    }
    psem2 = (sem_t*)sem_open("/sem2", O_CREAT,0644,0);     /* Creating sem2 */
    if (psem2 == SEM_FAILED) {
        err_sys("Open psem2");
    }
}
void clearSem(){                /* ClearSem sets the values to 0 of all sems */
    int sem_value;
    sem_getvalue(psem1, &sem_value);
    while (sem_value > 0) {
        sem_wait(psem1);
        sem_value--;
    }
    sem_getvalue(psem2, &sem_value);
    while (sem_value > 0) {
        sem_wait(psem2);
        sem_value--;
    }
}
void boardInit(){               /* Fill and restart the board */
    for (int x=0;x<SIZE;x++){
        for (int y=0;y<SIZE;y++){
            board[x][y]='-';
        }
    }
}
```

This function controls the communication between both players, gets the input from they, informs in case of error in the inputs and checks the state of the board.



```c
bool clientCommunication(int sock,const char PLAYER,sem_t* psem){

    char p[2];
    if(finish!=0){                                                      /* validates that the game is not over, in case yes, the current player has lost or drawn */
        p[0]=0+'0';
        p[1]=finish+1+'0';
        write(sock,p,2);                                                /*(Sent) the game is over and result*/
        return true;
    }
    write(sock,buffer, 2);
    while(1){
        read(sock, &buffer[0], 2);                                     /* (Receive) the position entered by the client */

        if(board[(buffer[0]-'0')-1][(buffer[1]-'0')-1]=='-'){          /* Validate client coordinates  */
            p[0]=0+'0';
            p[1]=0+'0';
            write(sock,p, 2);                                          /*(Sent) Report that the given value is correct */
            break;
        }
        p[0]=1+'0';
        p[1]=0+'0';
        write(sock,p, 2);                                              /*(Sent) report that the given value is not correct */
    }
    board[(buffer[0]-'0')-1][(buffer[1]-'0')-1]=PLAYER;                   /* Update board */

    finish=isFinish(PLAYER);
    if(finish!=0){                                                     /* Check if the game is over */
        p[0]=0+'0';
        p[1]=finish+'0';
        write(sock,p,2);                                                /* (Sent) the game is over and result*/
        sem_post(psem);                                                 /* Activate the other player's sem to validate his result */
        return true;
        }
    return false;
}
```

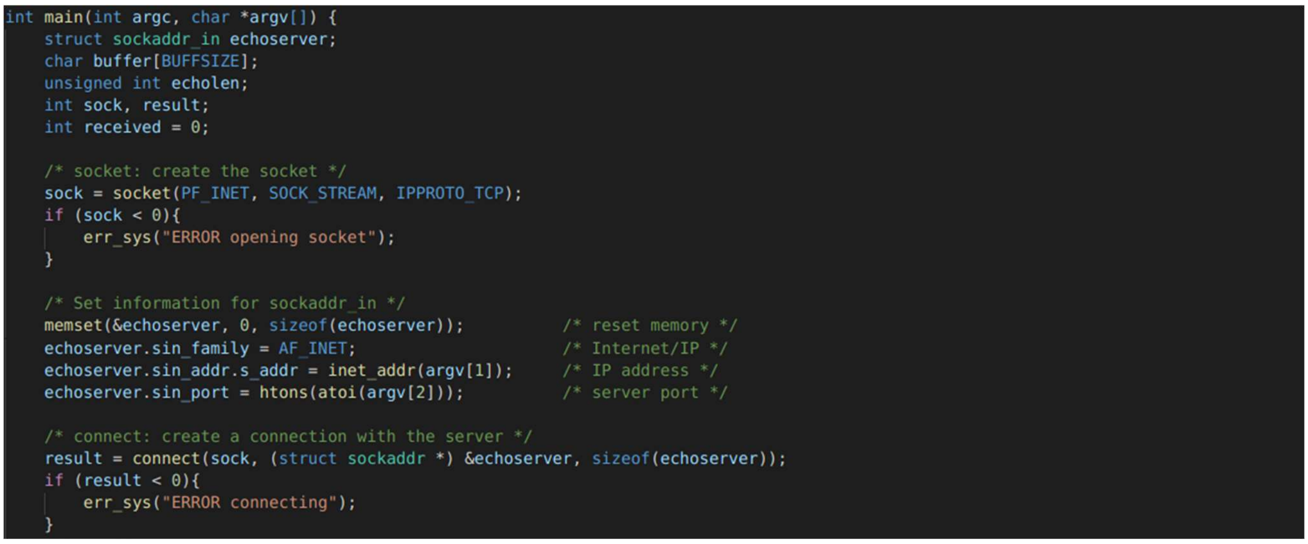
Those are the functions that creates client threads for player1, player2 and player3, respectively.



```
void *clientThread_1(void *vargp){                                         /*//////THREAD 1\\\\\\*/
    int sock = (uintptr_t)vargp;
    buffer[0]=1+'0';
    buffer[1]='\0';                                                        /* resets the buffer */
    while(1){                                                              /* LOOP PLAYER 1 */
        sem_wait(psem1);                                                   /* Wait for player2 to finish */
        if(clientCommunication(sock,PLAYER1,psem2)) break;                 /* Player turn 1 */
        sem_post(psem2);                                                   /* Activate player2 turn */
    }
    close(sock);                                                           /* Close socket */
    return((void*)NULL);
}
void *clientThread_2(void *vargp){                                         /*////// THREAD 2\\\\\\*/
    sem_post(psem1);                                                     /* Sart game and activate player1 turn */
    int sock = (uintptr_t)vargp;
    char p[2];
    p[0]=0+'0';
    p[1]='\0';
    write(sock,p, 2);                                                      /*(Sent) report that you are player 2*/
    while(1){                                                                        /* LOOP PLAYER 2*/
        sem_wait(psem2);                                                   /* Wait for player1 to finish */
       if(clientCommunication(sock,PLAYER2,psem1)) break;                  /* Player turn 2 */
        sem_post(psem1);                                                   /* Activate player1 turn */
    }
    close(sock);                                                           /* Close socket */
    nPlayer = nPlayer-2;                                                   /* the game is over and there are 2 fewer players */
    return((void*)NULL);
}
void *clientThread_3(void *vargp){                                         /*////// THREAD 3\\\\\\*/
    int sock = (uintptr_t)vargp;
    char pa[2]={'3','0'};                                                  /*(Sent) reports that the server is full*/
    write(sock,pa,2);
    close(sock);                                                           /* Close socket */
    return((void*)NULL);
}
```

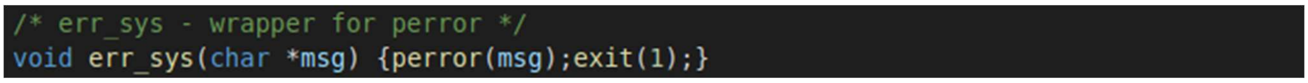
Code client1 explanation:

First the socket is created and its information is set. Then the client creates a connection with the server to communicates with.



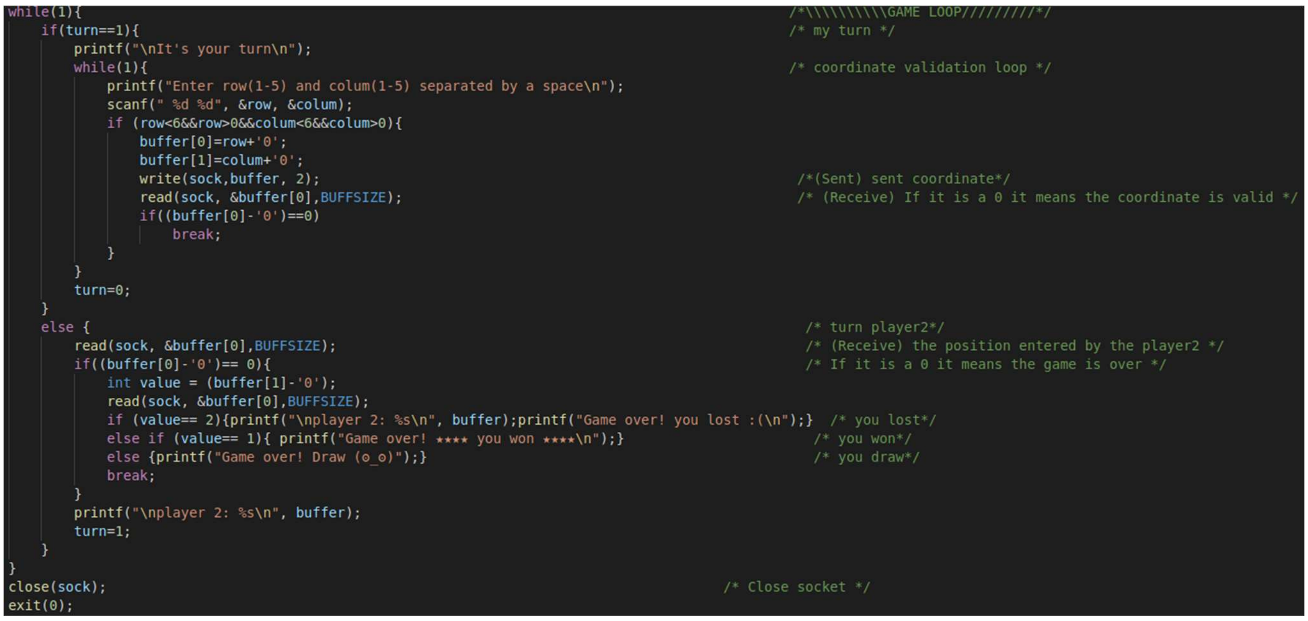
```
int main(int argc, char *argv[]) {
    struct sockaddr_in echoserver;
    char buffer[BUFFSIZE];
    unsigned int echolen;
    int sock, result;
    int received = 0;

    /* socket: create the socket */
    sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0){
        err_sys("ERROR opening socket");
    }

    /* Set information for sockaddr_in */
    memset(&echoserver, 0, sizeof(echoserver));          /* reset memory */
    echoserver.sin_family = AF_INET;                     /* Internet/IP */
    echoserver.sin_addr.s_addr = inet_addr(argv[1]);     /* IP address */
    echoserver.sin_port = htons(atoi(argv[2]));          /* server port */

    /* connect: create a connection with the server */
    result = connect(sock, (struct sockaddr *) &echoserver, sizeof(echoserver));
    if (result < 0){
        err_sys("ERROR connecting");
    }
```

This function informs in case of error.



```
/* err_sys - wrapper for perror */
void err_sys(char *msg) {perror(msg);exit(1);}
```

After the connection is established, the client is welcomed to the game and the socket is read to report if the server is full or if it is the player1 turn. The player1 is asked to enter the coordinates of the position where they want to put their piece. Those coordinates are sent to the server, who receives and validates them and updates the board. Then it is the player2 turn, who is informed of the player1 movement and asked to enter the coordinates of the position where they want to put their piece. The server receives and validates the coordinates and updates the board again. Those actions are repeated until the game finishes. The server constantly checks if the last movement has finished the game. When the game has finished, the socket is closed and both players are told if they are the winner, the loser or if there is a drawn. The processes of both players are finished, but the server keeps waiting.



```c
while(1){                                                                                /*\\\\\\\\\\GAME LOOP/////////*/
    if(turn==1){                                                                         /* my turn */
        printf("\nIt's your turn\n");
        while(1){                                                                        /* coordinate validation loop */
            printf("Enter row(1-5) and colum(1-5) separated by a space\n");
            scanf(" %d %d", &row, &colum);
            if (row<6&&row>0&&colum<6&&colum>0){
                buffer[0]=row+'0';
                buffer[1]=colum+'0';
                write(sock,buffer, 2);                                                    /*(Sent) sent coordinate*/
                read(sock, &buffer[0],BUFFSIZE);                                          /* (Receive) If it is a 0 it means the coordinate is valid */
                if((buffer[0]-'0')==0)
                    break;
            }
        }
        turn=0;
    }
    else {                                                                                /* turn player2*/
        read(sock, &buffer[0],BUFFSIZE);                                                  /* (Receive) the position entered by the player2 */
        if((buffer[0]-'0')== 0){                                                          /* If it is a 0 it means the game is over */
            int value = (buffer[1]-'0');
            read(sock, &buffer[0],BUFFSIZE);
            if (value== 2){printf("\nplayer 2: %s\n", buffer);printf("Game over! you lost :(\n");}  /* you lost*/
            else if (value== 1){ printf("Game over! **** you won ****\n");}                       /* you won*/
            else {printf("Game over! Draw (o_o)");}                                               /* you draw*/
            break;
        }
        printf("\nplayer 2: %s\n", buffer);
        turn=1;
    }
}
close(sock);                                                                    /* Close socket */
exit(0);
```