```
//client
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include<unistd.h>

#include<sys/socket.h>
#include<sys/select.h>
#include<sys/time.h>

#include<arpa/inet.h>
#include<netinet/in.h>
#include<errno.h>

//===== COSTANTI ==================
#define MAX_DIM_CMD     12  //comando piu' lungo: !disconnect (11 chars)
+ \0
#define N_CMD           7   //ci sono 7 comandi
#define MAX_LENGTH      25  //max lunghezza username
#define HELP_MENU "Sono disponibili i seguenti comandi:\n * !help -->
mostra l'elenco dei comandi disponibili\n * !who --> mostra l'elenco dei
client connessi al server\n * !connect nome_client --> avvia una partita
con l'utente nome_client\n * !disconnect --> disconnette il client
dall'attuale partita intrapresa con un altro peer\n * !quit -->
disconnette il client dal server\n * !show_map --> mostra la mappa di
gioco\n * !hit num_cell --> marca la casella num_cell (valido solo quando
e' il proprio turno)\n"
//=================================

//===== VARIABILI =================
//socket
int server_sd,
    client_sd;

//configurazione
struct sockaddr_in  server_addr,
                    client_addr,
                    my_addr;

//dati miei
char            my_username[MAX_LENGTH];
unsigned long   my_IP;
unsigned short  my_UDP_port;    //da 0 a 65535
char            my_mark;

//dati avversario
char            enemy_username[MAX_LENGTH];
unsigned long   enemy_IP;
unsigned short  enemy_UDP_port; //da 0 a 65535
```

```
char            enemy_mark;

int my_turn = 1;

char commands[N_CMD][MAX_DIM_CMD] = {
    "!help",
    "!who",
    "!quit",
    "!connect",
    "!disconnect",
    "!show_map",
    "!hit"
};

char    game_grid[9];
int     empty_cells;

char    shell;          // '>' = shell comandi, '#' = shell partita
int     show_shell;     //0=don't print shell character, 1=print

//set per select
fd_set  master,         //master file descriptor list
        tmp_fd;         //temp file descriptor list for select
int     max_fd;

//timer
struct timeval timer;

//================================

//===== FUNZIONI APPOGGIO =========

//----- valid_port ---------
int valid_port(int port) {
    if(port<1024 || port>=65536)
        return 0;
    return 1;
}

//------ check_win ----- //ritorna simbolo vincitore (or '-'), 'N'
altrimenti
char check_win() {
    if(game_grid[6]==game_grid[7] && game_grid[7]==game_grid[8]) return
game_grid[6]; //tris prima riga
    if(game_grid[3]==game_grid[4] && game_grid[4]==game_grid[5]) return
game_grid[3]; //tris seconda riga
    if(game_grid[0]==game_grid[1] && game_grid[1]==game_grid[2]) return
game_grid[0]; //tris terza riga
    if(game_grid[0]==game_grid[3] && game_grid[3]==game_grid[6]) return
```

```
game_grid[0]; //tris prima colonna
    if(game_grid[1]==game_grid[4] && game_grid[4]==game_grid[7]) return
game_grid[1]; //tris seconda colonna
    if(game_grid[2]==game_grid[5] && game_grid[5]==game_grid[8]) return
game_grid[2]; //tris terza colonna
    if(game_grid[0]==game_grid[4] && game_grid[4]==game_grid[8]) return
game_grid[0]; //tris diagonale
    if(game_grid[2]==game_grid[4] && game_grid[4]==game_grid[6]) return
game_grid[2]; //tris diagonale
    return 'N';
}

//------ reset ----- //inizializza i parametri per l'inizio di una nuova
partita
void reset() {
    int i;
    for(i=0; i<9; i++)
        game_grid[i] = '-';
    shell = '#';
    empty_cells = 9;
    if(my_mark=='X')
        my_turn=1;
    else
        my_turn=0;
    //aggiorno il timer
    timer.tv_sec = 60;
    timer.tv_usec = 0;
    return;
}

//------ resolve_command ----- restituisce l'indice del comando relativo
a cmd nell'array "commands"
int resolve_command(char *cmd) {
    int i;
    for(i=0; i<N_CMD; i++) {
        if(strcmp(cmd, commands[i])==0) // 0 se uguali
            return i;
    }
    return -1;
}

//------ print_line -----
void print_line(int i) {
    int j = i+3;
    do {
        printf(" %c ", game_grid[i]);
        i++;
    } while (i<j);
    return;
```

```
}
//================================

//===== FUNZIONI ==================

//------ cmd_who ----- la lista dei client e' nel server, qui mando la
richiesta
void cmd_who() {
    int ret;
    char cmd = 'w'; //da mandare al server

    ret = send(server_sd, (void *)&cmd, sizeof(char), 0);
    if(ret==-1) {
        printf("cmd_who error: errore nella send\n");
        exit(1);
    }
    //receive fatta in get_from_server()
    return;
}

//------ cmd_connect -----
void cmd_connect() {
    int     ret,
            length;
    char    cmd = 'c';

    //controllo se username e' il mio
    if(strcmp(my_username, enemy_username)==0) { //giocare contro me
stesso
        printf("non puoi giocare contro te stesso!!\n");
        return;
    }

    //controllo se sono gia' occupato in altre attivita'
    if(shell=='#') {
        cmd = 'b';
        ret = send(server_sd, (void *)&cmd, sizeof(cmd), 0);
        if (ret==-1)
        {
            printf("cmd_connect error: errore nell'invio del comando
occupato al server\n");
            exit(1);
        }
        return;
    }

    //mando al server il comando che individua la connect
    ret = send(server_sd, (void *)&cmd, sizeof(cmd), 0);
    if (ret==-1)
```

```
    {
        printf("cmd_connect error: errore nell'invio del comando al
server\n");
        exit(1);
    }
    //invio al server la lunghezza dell'enemy_username
    length = strlen(enemy_username);
    ret = send(server_sd, (void *)&length, sizeof(length), 0);
    if (ret==-1 || ret<sizeof(ret))
    {
        printf("cmd_connect error: errore nell'invio dimensione
enemy_username al server\n");
        exit(1);
    }
    //invio al server l'enemy_username
    ret = send(server_sd, (void *)enemy_username, length, 0);
    if (ret==-1 || ret<length)
    {
        printf("cmd_connect error: errore nell'invio di enemy_username al
server\n");
        exit(1);
    }

    //la ricezione della risposta del server e' in get_from_server()
    return;
}

//------ cmd_disconnect ----- end=1 indica che la partita e' finita,
end=0 indica che il giocatore ha abbandonato,
                              //end=2 non serve notificare al server
perche' l'ha gia' fatto l'avversario
void cmd_disconnect(int end) {
    int ret;
    char cmd = 'd'; //da mandare al server e al client

    if(end==0) {        //abbandono, devo informare l'avversario
        ret = sendto(client_sd, (void *)&cmd, sizeof(char), 0, (struct
sockaddr *)&client_addr, (socklen_t)sizeof(client_addr)); //informo
l'avversario
        if(ret==-1) {
            printf("cmd_disconnect error: errore nel notificare
disconnessione all'avversario!\n");
            exit(1);
        }
        printf("Disconnessione avvenuta con successo: TI SEI ARRESO\n");
    }

    if(end==1) //partita finita
        printf("Disconnesso da %s...\n", enemy_username);
```

```
    if(end!=2) {
        ret = send(server_sd, (void *)&cmd, sizeof(char), 0); //informo
il server
        if(ret==-1) {
            printf("cmd_disconnect error: errore nella send al server
\n");
            exit(1);
        }
    }

    shell = '>';

    //reset parametri avversario
    memset(&client_addr, 0, sizeof(client_addr));

    return;
}

//------ cmd_quit -----
void cmd_quit() {
    int     ret;
    char    cmd;

    cmd = 'q';

    ret = send(server_sd, (void *)&cmd, sizeof(char), 0);
    if(ret==-1) {
        printf("cmd_quit error: errore nella send\n");
        exit(1);
    }
    if(shell=='#') //sto giocando
        cmd_disconnect(0);

    close(client_sd);
    close(server_sd);
    printf("Client disconnesso correttamente\n");
    exit(0);
}

//------ cmd_show_map ----- se cella vuota mette '-'
void cmd_show_map() {
    print_line(6);
    printf("\n");
    print_line(3);
    printf("\n");
    print_line(0);
    printf("\n");
    return;
```

```
}

//------ cmd_hit -----
void cmd_hit(int cell) {
    int     ret;
    char    cmd = 'h';

    if(shell=='>') {
        printf("comando valido solo in partita!\n");
        return;
    }
    if(!my_turn) {
        printf("comando valido solo durante il proprio turno!\n");
        return;
    }
    if(game_grid[cell] != '-') {    //cella non vuota
        printf("la cella %d e' gia' occupata!\n", cell+1);
        return;
    }

    //da qui in poi significa che e' tutto ok
    game_grid[cell] = my_mark;  //segno sul campo di gioco
    empty_cells --;
    my_turn = 0;                //il turno passa all'avversario

    //informo l'avversario che gli mando la coordinata
    ret = sendto(client_sd, (void *)&cmd, sizeof(char), 0, (struct
sockaddr *)&client_addr, (socklen_t)sizeof(client_addr));
    if(ret==-1) {
        printf("cmd_hit error: errore nell'invio all'avversario che ho
fatto hit\n");
        exit(1);
    }
    //mando coordinate all'avversario
    cell = htonl(cell); //converto in formato di rete
    ret = sendto(client_sd, (void *)&cell, sizeof(int), 0, (struct
sockaddr *)&client_addr, (socklen_t)sizeof(client_addr));
    if(ret==-1) {
        printf("cmd_hit error: errore nell'invio delle coordinate
all'avversario\n");
        exit(1);
    }

    //aggiorno timer
    timer.tv_sec = 60;
    timer.tv_usec = 0;

    //controllo se la partita e' finita
    if(check_win()==my_mark) { //ho vinto
```

```
        printf("HAI VINTO!!\n");
        cmd_disconnect(1);  //1 per indicare che non ho abbandonato la
partita
    }
    if(check_win()!=my_mark && check_win()!=enemy_mark && empty_cells==0)
{ //pareggio
        printf("PAREGGIO!!\n");
        cmd_disconnect(1);  //1 per indicare che non ho abbandonato la
partita
    }

    printf("E' il turno di %s\n", enemy_username);
    return;
}

//------ cmd_hit_received ----
void cmd_hit_received(int cell) {
    game_grid[cell] = enemy_mark;
    empty_cells--;
    my_turn = 1;
    printf("%s ha marcato la cella numero %d\n", enemy_username, cell+1);
    cmd_show_map(); //stampo il campo di gioco
    printf("E' il tuo turno:\n");

    //controllo se la partita e' finita
    if(check_win()==enemy_mark) { //ho perso
        printf("HAI PERSO!!\n");
        cmd_disconnect(1);  //1 per indicare che non ho abbandonato la
partita
    }
    if(check_win()!=my_mark && check_win()!=enemy_mark && empty_cells==0)
{ //pareggio
        printf("PAREGGIO!!\n");
        cmd_disconnect(1);  //1 per indicare che non ho abbandonato la
partita
    }
    return;
}

//------ get_input -----
void get_input() {
    char cmd[MAX_DIM_CMD];
    int cell;

    scanf("%s", cmd);
    fflush(stdin);      //per eliminare dal buffer eventuali altri
caratteri
    switch(resolve_command(cmd)) {
        case 0: { //help
```

```
                    printf("%s", HELP_MENU);
                    break;
        }
        case 1: { //who
                    cmd_who();
                    break;
        }
        case 2: { //quit
                    cmd_quit();
                    break;
        }
        case 3: { //connect nome_utente
                    if(shell=='#')
                        printf("stai gia' giocando!\n");
                    else {
                        scanf("%s", enemy_username);
                        cmd_connect(); //controlli effettuati in
cmd_connect()
                    }
                    break;
        }
        case 4: { //disconnect
                    //controllo se sono in una partita
                    if(shell=='#')
                        cmd_disconnect(0);
                    else
                        printf("non sei connesso a nessun giocatore!!
\n");
                    break;
        }
        case 5: { //show_map
                    if(shell=='#')
                        cmd_show_map();
                    else
                        printf("non stai giocando con nessuno!\n");
                    break;
        }
        case 6: { //hit num_cella
                        scanf("%d", &cell);
                        if(cell<1 || cell>9) {
                            printf("numero cella non valido! must be in
[1, 9]\n");
                            return;
                        }
                        //controllo fatto in cmd_hit
                        cmd_hit(cell-1);
                        break;
        }
        default: printf("comando inesistente! digitare \"!help\" per la
```

```
lista dei comandi disponibili\n");
    }
    return;
}

//------ connect_to_server -----
void connect_to_server(char *addr, int port) {
    int ret;

    memset(&server_addr, 0, sizeof(server_addr));

    //controllo indirizzo
    ret = inet_pton(AF_INET, addr, &server_addr.sin_addr.s_addr);
    if(ret==0) {
        printf("indirizzo non valido!\n");
        exit(1);
    }
    //controllo porta
    if(!valid_port(port)) {
        printf("porta non valida! (must be in [1025, 65535])\n");
        exit(1);
    }
    server_addr.sin_port = htons(port);
    server_sd = socket(AF_INET, SOCK_STREAM, 0);
    if(server_sd==-1) {
        printf("errore nella creazione del socket (server)\n");
        exit(1);
    }
    server_addr.sin_family = AF_INET;
    ret = connect(server_sd, (struct sockaddr *)&server_addr, sizeof
(server_addr));
    if(ret==-1) {
        printf("errore nella connect\n");
        exit(1);
    }
    return;
}

//------- log_in_server ------
void log_in_server() {
    int     length,
            ret;
    char    cmd;
    unsigned short port_tmp;
    char    UDP[7];

    memset(UDP, 0, 7);

    printf("\nInserisci il tuo nome: ");
```

```
    scanf("%s", my_username);
    do {
        printf("Inserisci la porta UDP di ascolto: ");
        scanf("%s", UDP);
        my_UDP_port = atoi(UDP);
        if(!valid_port(my_UDP_port))
            printf("porta non valida! (must be in [1025, 65535])\n");
    }while(!valid_port(my_UDP_port));

    //invio lunghezza username al server
    length = strlen(my_username);
    ret = send(server_sd, (void *)&length, sizeof(length), 0);
    if(ret==-1 || ret<sizeof(length)) {
        printf("log_in_server error: errore in invio lunghezza nome\n");
        exit(1);
    }

    //invio username al server
    ret = send(server_sd, (void *)my_username, length, 0);
    if(ret==-1 || ret<length) {
        printf("log_in_server error: errore in invio username\n");
        exit(1);
    }

    //invio porta al server
    port_tmp = htons(my_UDP_port);
    ret = send(server_sd, (void *)&port_tmp, sizeof(port_tmp), 0);
    if(ret==-1 || ret<sizeof(port_tmp)) {
        printf("log_in_server error: errore in invio porta UDP\n");
        exit(1);
    }

    //controllo cosa ha risposto il server
    ret = recv(server_sd, (void *)&cmd, sizeof(cmd), 0);
    if(ret==-1) {
        printf("log_in_server error: errore in ricezione risposta dal
server\n");
        exit(1);
    }
    if(cmd=='e') {  //username gia' esistente
        printf("lo username scelto e' gia' esistente!\n");
        exit(2);    //esce anziche' richiedere il nome
    }
    /*
    if(cmd=='@') //connessione ok!
    */
    return;
}
```

```
//------- print_client_list ----
void print_client_list(int tot) {
    int     i,
            ret,
            length,
            status;     //remember 0=free 1=busy
    char    client_name[MAX_LENGTH],
                stat[5];

    for(i=0; i<tot; i++) {
        memset(client_name, 0, MAX_LENGTH); //reset di client_name

        //ricevo lunghezza nome
        ret = recv(server_sd, (void *)&length, sizeof(int), 0);
        if(ret==-1) {
            printf("print_client_list error: errore in ricezione
lunghezza nome client dal server!\n");
            exit(1);
        }
        //ricevo il nome del client
        ret = recv(server_sd, (void *)client_name, length, 0);
        if(ret==-1) {
            printf("get_from_server error: errore in ricezione nome
client dal server!\n");
            exit(1);
        }
        client_name[length] = '\0';
        //ricevo lo stato del client
        ret = recv(server_sd, (void *)&status, sizeof(int), 0);
        if(ret==-1) {
            printf("get_from_server error: errore in ricezione stato
client dal server!\n");
            exit(1);
        }

        if(status==0)
            strncpy(stat, "free", sizeof(stat)-1);
        else
            strncpy(stat, "busy", sizeof(stat)-1);
        stat[4] = '\0';

        printf("\t%d)  %s (%s)\n", i, client_name, stat);
    }
    return;
}

//------- start_game ------
void start_game() {
    int ret;
```

```
    my_mark = 'X';
    enemy_mark = 'O';
    printf("%s ha accettato la partita\n", enemy_username);
    printf("Partita avviata con %s\n", enemy_username);
    printf("Il tuo simbolo e': %c\n", my_mark);
    printf("E' il tuo turno:\n");

    //ricevo dal server la porta di enemy
    ret = recv(server_sd, (void *)&enemy_UDP_port, sizeof
(enemy_UDP_port), 0);
    if(ret==-1) {
        printf("start_game error: errore nel ricevere la porta su cui e'
in ascolto l'avversario!\n");
        exit(1);
    }

    //ricevo dal server l'IP di enemy
    ret = recv(server_sd, (void *)&enemy_IP, sizeof(enemy_IP), 0);
    if(ret==-1) {
        printf("start_game error: errore nel ricevere l'indirizzo
dell'avversario!\n");
        exit(1);
    }

    //inizializzo parametri avversario
    memset(&client_addr, 0, sizeof(client_addr));
    client_addr.sin_family = AF_INET;
    client_addr.sin_port = enemy_UDP_port;
    client_addr.sin_addr.s_addr = enemy_IP;

    enemy_UDP_port = ntohs(enemy_UDP_port);

    //aggiorno timer
    timer.tv_sec = 60;
    timer.tv_usec = 0;

    reset();

    return;
}

//------- manage_request ------- richiesta di gioco da parte del client
void manage_request() {
    int     ret,
            length;
    char    cmd,
            res;
```

```
    //ricevo lunghezza nome di chi mi chiede di connettersi
    ret = recv(server_sd, (void *)&length, sizeof(int), 0);
    if(ret==-1) {
        printf("manage_request error: errore in ricezione lunghezza nome
client!\n");
        exit(1);
    }

    memset(enemy_username, 0, MAX_LENGTH);
    //ricevo il nome del client che mi chiede di giocare
    ret = recv(server_sd, (void *)enemy_username, length, 0);
    if(ret==-1) {
        printf("manage_request error: errore in ricezione nome client!
\n");
        exit(1);
    }
    enemy_username[length] = '\0';

    if(shell=='#') { //occupato
        //notifico al server che sono gia' impegnato
        cmd = 'b';
        ret = send(server_sd, (void *)&cmd, sizeof(cmd), 0);
        if(ret==-1) {
            printf("manage_request error: errore in invio stato occupato
al server!\n");
            exit(1);
        }
        return;
    }

    printf("%s ti ha chiesto di giocare!\n", enemy_username);

    do {
        scanf("%c", &res);  //metto questo perche' se no stampa 2 volte
"accettare..." perche' legge senza aspettare che digiti la prima volta!
        printf("accettare la richiesta? (y|n): ");  //PERCHE' LA PRIMA
VOLTA LEGGE SENZA ASPETTARE CHE DIGITI?? (risolto con scanf sopra)
        fflush(stdin);
        scanf("%c", &res);
    }while(res!='y' && res!='Y' && res!='n' && res!= 'N');

    if(res=='y' || res=='Y') {
        my_mark = 'O';
        enemy_mark = 'X';
        my_turn = 0;
        printf("Richiesta di gioco accettata\n");
        printf("Il tuo simbolo e': %s\n", &my_mark);
        printf("E' il turno di %s\n", enemy_username);
        cmd = 'a';
```

```
        ret = send(server_sd, (void *)&cmd, sizeof(cmd), 0);
        if(ret==-1) {
            printf("manage_request error: errore in invio richiesta
accettata al server!\n");
            exit(1);
        }
        reset();
    }
    else {  //richiesta rifiutata
        printf("Richiesta di gioco rifiutata\n");
        cmd = 'r';
        ret = send(server_sd, (void *)&cmd, sizeof(cmd), 0);
        if(ret==-1) {
            printf("manage_request error: errore in invio richiesta
rifiutata al server!\n");
            exit(1);
        }
    }
    return;
}
//------- get_from_server ------//mi e' arrivata la risposta del server,
devo controllare a cosa
void get_from_server() {
    int     ret,
            tot_clients;
    char    cmd;

    ret = recv(server_sd, (void *)&cmd, sizeof(char), 0);
    if(ret==-1) {
        printf("get_from_server error: errore in ricezione comando dal
server!\n");
        exit(1);
    }

    if(ret==0) { //server si e' disconnesso
        printf("Il server ha chiuso la connessione!\n");
        fflush(stdout);
        exit(2);
    }

    switch(cmd) {
        case 'w': { //who
                    printf("Client connessi al server:\n");
                    //ricevo quanti sono i client
                    ret = recv(server_sd, (void *)&tot_clients, sizeof
(int), 0);
                    if(ret==-1) {
                        printf("get_from_server error: errore in
ricezione numero client dal server!\n");
```

```
                        exit(1);
                    }
                    print_client_list(tot_clients);
                    break;
        }

        case 'c': { //connect
                    //ricevo dal server la risposta di enemy
                    ret = recv(server_sd, (void *)&cmd, sizeof(char), 0);
                    if(ret==-1) {
                        printf("get_from_server error: errore in
ricezione \"accettato\" dal server!\n");
                        exit(1);
                    }
                    switch(cmd) {
                        case 'i': { //enemy_username non esiste!
                                    printf("Impossibile connettersi a %s:
utente inesistente!\n", enemy_username);
                                    break;
                        }
                        case 'a': { //enemy ha accettato
                                    start_game();
                                    break;
                        }
                        case 'r': { //enemy ha rifiutato
                                    printf("Impossibile connettersi a %s:
l'utente ha rifiutato la partita!\n", enemy_username);
                                    break;
                        }
                        default : { //non dovrebbe succedere
                                    printf("Risposta dell'utente
incomprensibile!\n");
                                    break;
                        }
                    }
                    break;
        }

        case 'o': { //richiesta di giocare
                    manage_request();
                    break;
        }

        default : { //niente

                    break;
        }
    }
    return;
```

```
}

//------- get_from_client ------
void get_from_client() {
    int     ret,
            addrlen,
            cell;
    char    cmd;

    addrlen = sizeof(client_addr);
    ret = recvfrom(client_sd, (void *)&cmd, sizeof(cmd), 0, (struct
sockaddr *)&client_addr, (socklen_t *)&addrlen);
    if(ret==-1) {
        printf("get_from_client error: errore in ricezione comando dal
client!\n");
        exit(1);
    }
    if(ret==0) {
        printf("get_from_client error: nessun dato ricevuto dal client!
\n");
        return;
    }

    switch(cmd) {
        case 'd' :  {   //disconnect
                        printf("l'avversario si e' arreso!\nHAI VINTO!!
\n");
                        cmd_disconnect(2);
                        break;
        }
        case 'h' :  {   //hit
                        ret = recvfrom(client_sd, (void *)&cell, sizeof
(int), 0, (struct sockaddr *)&client_addr, (socklen_t *)&addrlen);
                        if(ret==-1) {
                            printf("get_from_client error: errore in
ricezione coordinate dal client!\n");
                            exit(1);
                        }
                        cell = ntohl(cell); //cell e' gia' decrementata
di 1!
                        cmd_hit_received(cell);
                        break;
        }
        default :   {
                        break;
        }
    }
    return;
}
```

```
//================================

//------- main ------
int main(int num, char* args[]) {   //remember: il primo arg e' ./client

    int ret,
        i;
    struct timeval *time;

    //controllo numero parametri
    if(num!=3) {
        printf("numero di parametri errato!\n");
        return -1;
    }

    //prova a connettersi al server
    connect_to_server(args[1], atoi(args[2]));
    printf("connessione al server %s (porta %s) effettuata con successo",
args[1], args[2]);

    //visualizza help
    printf("\n%s\n", HELP_MENU);

    //chiedo nome e UDP port
    log_in_server();

    //apro UDP socket
    client_sd = socket(AF_INET, SOCK_DGRAM, 0);
    if(client_sd==-1) {
        printf("errore nella creazione del socket UDP\n");
        exit(1);
    }

    //configurazione
    memset(&my_addr, 0, sizeof(my_addr));
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(my_UDP_port);
    my_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    //bind
    ret = bind(client_sd, (struct sockaddr*)&my_addr, sizeof(my_addr));
    if(ret==-1){
        printf("errore bind\n");
        exit(1);
    }

    //set
    FD_ZERO(&master);
```

```
    FD_SET(server_sd, &master);
    FD_SET(client_sd, &master);
    FD_SET(0, &master);  //std in
    max_fd = (server_sd>client_sd)?server_sd:client_sd;

    //reset();
    shell = '>';
    timer.tv_sec = 60;
    timer.tv_usec = 0;
    show_shell = 1;

    while(1) {
        tmp_fd = master;

        time = &timer;

        if(shell=='>')
            time = NULL;

        if(show_shell) {
            printf("%c", shell);
            fflush(stdout);
        }

        ret = select(max_fd+1, &tmp_fd, NULL, NULL, time); //attivare
timer
        if(ret==-1) {
            printf("errore select\n");
            exit(1);
        }
        if(ret==0) { //il timer e' scaduto
            if(my_turn)
                printf("il tempo e' scaduto! HAI PERSO!\n");
            else
                printf("il tempo dell'avversario e' scaduto! HAI VINTO!
\n");
            cmd_disconnect(1);  //fine partita
            continue;
        }
        for(i=0; i<=max_fd; i++) {
            if(FD_ISSET(i, &tmp_fd)) { //c'e' descrittore pronto

                if(i==0) {//descrittore pronto e' stdin
                    //leggo comando in input
                    get_input();
                    //aggiorno il timer
                    timer.tv_sec = 60;
                    timer.tv_usec = 0;
                }
```

```
                if(i==server_sd) {//descrittore pronto e' server
                    //ricevo dal server
                    get_from_server();
                }

                if(i==client_sd) {//descrittore pronto e' client
                    //ricevo da altro peer
                    get_from_client();
                    //aggiorno il timer
                    timer.tv_sec = 60;
                    timer.tv_usec = 0;
                }
            }
        }
    }
    return 1;
}
```