```
//server
#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_CONNECTION  10
#define MAX_LENGTH      25  //max lunghezza username
//=================================

//===== VARIABILI =================
//client
struct client {
    char            username[MAX_LENGTH];
    short int       UDP_port;
    int             socket;
    unsigned long   address;
    int             status; // 0=free, 1=busy
    struct client   *enemy;
    struct client   *next;
};

struct client *users;   //lista dei client connessi
int tot_users = 0;

struct client *client; //client che comunica con il server

//configurazione
struct sockaddr_in  server_addr,
                    client_addr;

int listener; //descrittore del socket in ascolto

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

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

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

//----- add_to_list --------
void add_to_list(struct client *elem) {
    tot_users++;
    elem->next = users;
    users = elem;
}

//----- remove_from_list --------
void remove_from_list(struct client *elem) {
    struct client *tmp = users;

    if(!tmp) { //lista vuota, potevo fare anche if(tot_users==0)
        return;
    }
    if(tmp==elem) { //elem in testa
        users = users->next;
        free(tmp);
        tot_users--;
        return;
    }
    while(tmp->next!=elem && tmp->next!=NULL)
        tmp = tmp->next;
    if(tmp->next==NULL) //elem non trovato
        return;
    tot_users--;
    tmp->next = tmp->next->next;
    return;
}

//----- find_client --------
struct client* find_client(int sd) {
    struct client *tmp = users;
    while(tmp!=NULL) {
        if(tmp->socket==sd)
            return tmp;
        tmp = tmp->next;
    }
    return NULL;
}

//----- find_by_username ---
```

```
struct client* find_by_username(char *name) {
    struct client *tmp = users;
    while(tmp!=NULL) {
        if(strcmp(tmp->username, name)==0)  //sono uguali
            return tmp;
        tmp = tmp->next;
    }
    return NULL;
}

//----- exist -------------
int exist(char *name) {
    struct client *tmp = users;
    while(tmp!=NULL) {
        if(strcmp(tmp->username, name)==0)  //sono uguali
            return 1;
        tmp = tmp->next;
    }
    return 0;
}

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

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

//----- cmd_who ---------------
void cmd_who() {
    int     ret,
            length;
    char    cmd = 'w';
    struct client *tmp = users;

    //invio al client che e' la risposta al comando who
    ret = send(client->socket, (void *)&cmd, sizeof(cmd), 0);
    if(ret==-1) {
        printf("cmd_who error: errore in invio comando\n");
        exit(1);
    }

    ret = send(client->socket, (void *)&tot_users, sizeof(tot_users), 0);
    if(ret==-1 || ret<sizeof(tot_users)) {
        printf("cmd_who error: errore in invio tot utenti!\n");
        exit(1);
    }
    while(tmp) {    //finche' ci sono client, ossia tot_user volte
        length = strlen(tmp->username);

        //mando la lunghezza di username al client
        ret = send(client->socket, (void *)&length, sizeof(int), 0);
```

```
        if(ret==-1 || ret<sizeof(length)) {
            printf("cmd_who error: errore in invio lunghezza nome client!
\n");
            exit(1);
        }

        //mando il nome del client
        ret = send(client->socket, (void *)tmp->username, length, 0);
        if(ret==-1 || ret<length) {
            printf("cmd_who error: errore in invio nome client!\n");
            exit(1);
        }

        //mando lo stato del client
        ret = send(client->socket, (void *)&tmp->status, sizeof(int), 0);
        if(ret==-1 || ret<sizeof(int)) {
            printf("cmd_who error: errore in invio stato client!\n");
            exit(1);
        }

        //potrei implementare di mandare anche con chi e' occupato!

        tmp = tmp->next;
    }
}

//----- cmd_disconnect---------
void cmd_disconnect() {
    //controllo se client e' effettivamente in una partita, controllo
gia' fatto sul client, non dovrebbe servire
    //if(client->status==1) {
        printf("%s si e' disconnesso da %s\n", client->username, client-
>enemy->username);
        client->status = 0;
        client->enemy->status = 0;
        printf("%s e' libero\n", client->username);
        printf("%s e' libero\n", client->enemy->username);
        client->enemy=NULL;
    /*}
    else {
        printf("%s non e' impegnato in nessuna partita!\n", client-
>username);
    }*/
}

//----- cmd_connect------------
void cmd_connect() {
    int     ret,
            length;
```

```
    char    cmd;
    char    usr[MAX_LENGTH];
    struct client *client2;
    short   int port;

    //ricevo lunghezza username a cui client si vuole connettere
    ret = recv(client->socket, (void *)&length, sizeof(int), 0);
    if(ret==-1) {
        printf("cmd_connect error: errore in ricezione lunghezza username
\n");
        exit(1);
    }

    //ricevo username a cui client si vuole connettere
    ret = recv(client->socket, (void *)usr, length, 0);
    if(ret==-1) {
        printf("cmd_connect error: errore in ricezione username\n");
        exit(1);
    }
    usr[length] = '\0';

    //informo client che rispondo alla sua connect
    cmd = 'c'; //connect
    ret = send(client->socket, (void *)&cmd, sizeof(char), 0);
    if(ret==-1) {
        printf("cmd_connect error: errore in invio risposta a connect a
client\n");
        exit(1);
    }

    if(!exist(usr)) {   //username non esiste tra i client connessi
        cmd = 'i';
        ret = send(client->socket, (void *)&cmd, sizeof(char), 0);
        if(ret==-1) {
            printf("cmd_connect error: errore in invio username non
esistente\n");
            exit(1);
        }
        return;
    }

    client2 = find_by_username(usr);
    //inoltro la richiesta di gioco a client2
    cmd = 'o';
    //invio comando che identifica richiesta
    ret = send(client2->socket, (void *)&cmd, sizeof(char), 0);
    if(ret==-1) {
        printf("cmd_connect error: errore in invio richiesta di gioco a
client2\n");
```

```
        exit(1);
    }
    //invio lunghezza username del client richiedente
    length = strlen(client->username);
    ret = send(client2->socket, (void *)&length, sizeof(int), 0);
    if(ret==-1) {
        printf("cmd_connect error: errore in invio lunghezza username
client a client2\n");
        exit(1);
    }
    //invio username di client
    ret = send(client2->socket, (void *)client->username, length, 0);
    if(ret==-1) {
        printf("cmd_connect error: errore in invio username client a
client2\n");
        exit(1);
    }

    //ricevo risposta da client2
    ret = recv(client2->socket, (void *)&cmd, sizeof(char), 0);
    if(ret==-1) {
        printf("cmd_connect error: errore in ricezione risposta
client2\n");
        exit(1);
    }

    switch(cmd) {
        case 'b': { //client2 e' gia' occupato
                    //invio che client2 e' occupato a client
                    ret = send(client->socket, (void *)&cmd, sizeof
(char), 0);
                    if(ret==-1) {
                        printf("cmd_connect error: errore in invio
client2 occupato a client\n");
                        exit(1);
                    }
                    //client2->status = 1; //occupato
                    client->status = 0; //libero
                    client->enemy = NULL;
                    break;
        }
        case 'r': { //client2 ha rifiutato
                    //invio che client2 ha rifiutato a client
                    ret = send(client->socket, (void *)&cmd, sizeof
(char), 0);
                    if(ret==-1) {
                        printf("cmd_connect error: errore in invio
client2 rifiuta a client\n");
                        exit(1);
```

```
                    }
                    client->status = 0; //libero
                    client->enemy = NULL;
                    break;
        }
        case 'a': { //client2 ha accettato
                    //invio che client2 ha accettato a client
                    ret = send(client->socket, (void *)&cmd, sizeof
(char), 0);
                    if(ret==-1) {
                        printf("cmd_connect error: errore in invio
client2 accetta a client\n");
                        exit(1);
                    }
                    //invio porta di client2 a client
                    port = htons(client2->UDP_port);
                    ret = send(client->socket, (void *)&port, sizeof
(port), 0);
                    if(ret==-1) {
                        printf("cmd_connect error: errore in invio porta
client2 a client\n");
                        exit(1);
                    }
                    //invio IP di client2 a client
                    ret = send(client->socket, (void *)&client2->address,
sizeof(client2->address), 0);
                    if(ret==-1) {
                        printf("cmd_connect error: errore in invio IP
client2 a client\n");
                        exit(1);
                    }

                    client2->status = 1; //occupato
                    client->status = 1; //occupato
                    client->enemy = client2;
                    client2->enemy = client;
                    printf("%s si e' connesso a %s\n", client->username,
client->enemy->username);
                    break;
        }
        default:  { //risposta incomprensibile, non dovrebbe mai
succedere!
                    cmd = '$';  //un qualunque carattere non riconosciuto
da "get_from_server()" del client
                    ret = send(client->socket, (void *)&cmd, sizeof
(char), 0);
                    if(ret==-1) {
                        printf("cmd_connect error: errore in invio
risposta incomprensibile a client\n");
```

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

//----- quit ------------------
void cmd_quit(int x) {
    if(x==0)
        printf("Il client %s si e' disconnesso dal server\n", client-
>username);
    else
        printf("Il client ha usato uno username non valido!\n");
    close(client->socket);
    FD_CLR(client->socket, &master);
    remove_from_list(client);
}

//----- manage_client ---------
void manage_client(int sd) {
    int     ret;
    char    cmd;

    client = find_client(sd);
    if(!client) {
        printf("manage_client error: client non trovato!\n");
        exit(1);
    }

    ret = recv(sd, (void *)&cmd, sizeof(cmd), 0);
    if(ret==-1) {
        printf("manage_client error: errore in ricezione comando\n");
        exit(1);
    }
    if(ret==0) { //client disconnected
        cmd_quit(0);
        return;
    }
    switch(cmd) {
        case 'w': { //who
                    cmd_who(); //deve inviare la lista dei client
                    break;
        }
        case 'd': { //disconnect
                    cmd_disconnect();
                    break;
        }
        case 'c': { //connect
```

```
                    cmd_connect();
                    break;
        }
        case 'q': { //quit
                    cmd_quit(0);
                    break;
        }
    }
}

//----- add_client ---------
void add_client(int sd) {
    int     ret,
            length;
    char    cmd;

    struct client *new_client = malloc(sizeof(struct client));

    length = sizeof(client_addr);
    memset(&client_addr, 0, length);
    getpeername(sd, (struct sockaddr *)&client_addr, (socklen_t *)
&length); //trovo l'indirizzo del client che si e' connesso

    new_client->address = client_addr.sin_addr.s_addr;
    new_client->socket  = sd;
    new_client->status  = 0;
    new_client->enemy   = NULL;

    //ricezione lunghezza nome
    ret = recv(sd, (void *)&length, sizeof(length), 0);
    if(ret==-1) {
        printf("add_client error: errore in ricezione lunghezza nome\n");
        exit(1);
    }

    //ricezione nome
    ret = recv(sd, (void *)new_client->username, length, 0);
    if(ret==-1) {
        printf("add_client error: errore in ricezione nome\n");
        exit(1);
    }
    new_client->username[length] = '\0';

    //ricezione porta UDP
    ret = recv(sd, (void *)&new_client->UDP_port, sizeof(int), 0);
    if(ret==-1) {
        printf("add_client error: errore in ricezione porta UDP\n");
        exit(1);
    }
```

```
    //visto che lavoro in locale dovrei controllare che sulla stessa
porta non ci siano gia' altri client

    new_client->UDP_port = ntohs(new_client->UDP_port);

    if(exist(new_client->username)) {
        cmd = 'e'; //exists
        ret = send(sd, (void *)&cmd, sizeof(cmd), 0);
        if(ret==-1) {
            printf("add_client error: errore invio exists\n");
            exit(1);
        }
        //chiudo connessione con il client
        client = new_client;
        cmd_quit(1);
        return;
    }
    //connessione accettata, devo mandare qualcosa al client? per
sicurezza mando @ che non significa niente
    cmd = '@';
    ret = send(sd, (void *)&cmd, sizeof(cmd), 0);
    if(ret==-1) {
        printf("add_client error: errore invio connessione ok\n");
        exit(1);
    }

    //inserimento in lista client connessi
    add_to_list(new_client);
    printf("%s si e' connesso\n", new_client->username);
    printf("%s e' libero\n", new_client->username);
}

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

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

    int ret,
        i,
        addrlen,
        new_client_sd;  //socket del nuovo client

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

```
    //controllo indirizzo
    ret = inet_pton(AF_INET, args[1], &server_addr.sin_addr.s_addr);
    if(ret==0) {
        printf("indirizzo non valido!\n");
        exit(1);
    }
    //controllo porta
    ret = atoi(args[2]);
    if(!valid_port(ret)) {
        printf("porta non valida! (must be in [1025, 65535])\n");
        exit(1);
    }

    //configurazione
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(ret);
    inet_pton(AF_INET, args[1], &server_addr.sin_addr.s_addr); //lo
rifaccio perche' per ordine ho inizializzato server_addr dopo il
controllo IP

    printf("Indirizzo: %s (Porta: %s)\n", args[1], args[2]);

    //apro socket TCP
    listener = socket(AF_INET, SOCK_STREAM, 0);
    if(listener==-1) {
        printf("errore nella creazione del socket (server)\n");
        exit(1);
    }

    //--> se voglio mettere opt: ret = setsockopt(listener, SOL_SOCKET,
SO_REUSEADDR, &optval, sizeof(optval));

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

    //listen
    ret = listen(listener, MAX_CONNECTION);
    if(ret==-1){
        printf("errore listen\n");
        exit(1);
    }

    //set
```

```
    FD_ZERO(&master);
    FD_ZERO(&tmp_fd);
    FD_SET(listener, &master);
    max_fd = listener;                  //e' il fd massimo per ora

    while(1) {
        tmp_fd = master;
        ret = select(max_fd+1, &tmp_fd, NULL, NULL, NULL);
        if(ret==-1) {
            printf("errore select\n");
            exit(1);
        }
        for(i=0; i<=max_fd; i++) {
            if(FD_ISSET(i, &tmp_fd)) {  //descrittore pronto
                if(i==listener) {                   //un nuovo client
vuole connettersi
                    addrlen = sizeof(client_addr);
                    new_client_sd = accept(listener, (struct sockaddr *)
&client_addr, (socklen_t *)&addrlen);
                    if(new_client_sd==-1) {
                        printf("errore accept\n");
                        exit(1);
                    }
                    printf("Connessione stabilita con il client\n");
                    FD_SET(new_client_sd, &master); //aggiungo il fd del
client ai socket da controllare
                    if(new_client_sd>max_fd)
                        max_fd = new_client_sd;
                    add_client(new_client_sd);          //aggiungo client
alla lista
                }
                else {      //e' un client gia' connesso
                    manage_client(i);   //gestisce i dati inviati dal
client
                }
            }
        }
    }
    return 0;
}
```