

Università degli Studi di Napoli Federico II



Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea Triennale in Informatica

Classe n. L-31

Progetto di sistemi operativi

Traccia A

Professore:
Finzi Alberto

Candidati:
Turco Mario
Matr. N8600/2503
Longobardi Francesco
Matr. N8600/2468

Anno Accademico
2019/2020

Indice

1 Istruzioni preliminari	1
1.1 Modalità di compilazione	1
2 Guida all'uso	1
2.1 Server	1
2.2 Client	1
3 Comunicazione tra client e server	2
3.1 Configurazione del server	2
3.2 Configurazione del client	3
3.3 Comunicazione tra client e server	4
3.3.1 Esempio: la prima comunicazione	4
4 Comunicazione durante la partita	5
4.1 Funzione core del server	5
4.2 Funzione core del client	5
A Codici sorgente	7
A.1 Codice sorgente del client	7
A.2 server.c	12
A.3 boardUtility.c	12
A.4 boardUtility.h	12
A.5 lsit.c	12
A.6 lsit.h	12
A.7 parser.c	12
A.8 parser.h	12

1 Istruzioni preliminari

1.1 Modalità di compilazione

Il progetto è provvisto di un file makefile il quale è in grado di compilare autonomamente l'intero progetto. Per utilizzare il makefile aprire la cartella del progetto tramite la console di sistema e digitare "make".

In alternativa è possibile compilare manualmente il client ed il server con i seguenti comandi:

```
gcc -o server server.c boardUtility.c parser.c list.c -lpthread
gcc -o client client.c boardUtility.c parser.c list.c -lpthread
```

2 Guida all'uso

2.1 Server

Una volta compilato il progetto è possibile avviare il server digitando da console il seguente comando

```
./server users
```

L'identificativo *users* si riferisce al nome del file sul quale sarà salvata la lista degli utenti e delle loro credenziali.

È possibile scegliere un nome a piacimento per il file purchè esso sia diverso da *log*.

2.2 Client

Una volta compilato il progetto è possibile avviare il client digitando da console il seguente comando:

```
./client ip porta
```

Dove *ip* andrà sostituito con l'ip o l'indirizzo URL del server e *porta* andrà sostituito con la porta del server.

Una volta avviato il client comparirà il menu con le scelte 3 possibili: accedi, registrati ed esci.

Una volta effettuata la registrazione dell'utente è possibile effettuare l'accesso al programma al seguito del quale verranno mostrate sia la mappa del gioco sia le istruzioni di gioco.

3 Comunicazione tra client e server

Di seguito verranno illustrate le modalità di comunicazione tra client e server.

3.1 Configurazione del server

Il socket del server viene configurato con famiglia di protocolli PF_INET, con tipo di trasmissione dati SOCK_STREAM e con protocollo TCP. Mostriamo di seguito il codice sorgente:

Listato 1: Configurazione socket del server

```
1 void configuraSocket(struct sockaddr_in mio_indirizzo) {
2     if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
3         perror("Impossibile creare socket");
4         exit(-1);
5     }
6     if (setsockopt(socketDesc, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) <
7         0)
8         perror("Impossibile impostare il riutilizzo dell'indirizzo ip e della "
9             "porta\n");
10    if ((bind(socketDesc, (struct sockaddr *)&mio_indirizzo,
11        sizeof(mio_indirizzo))) < 0) {
12        perror("Impossibile effettuare bind");
13        exit(-1);
14    }
```

È importante notare anche come il server riesca a gestire in modo concorrente più client tramite l'uso di un thread dedicato ad ogni client. Una volta aver configurato il socket, infatti, il server si mette in ascolto per nuove connessioni in entrata ed ogni volta che viene stabilita una nuova connessione viene avviato un thread per gestire tale connessione. Di seguito il relativo codice:

Listato 2: Procedura di ascolto del server

```
1 void startListening()
2 {
3     pthread_t tid;
4     int clientDesc;
5     int *puntClientDesc;
6     while (1)
7     {
8         if (listen(socketDesc, 10) < 0)
9             perror("Impossibile mettersi in ascolto"), exit(-1);
10        printf("In ascolto...\n");
11        if ((clientDesc = accept(socketDesc, NULL, NULL)) < 0)
12        {
13            perror("Impossibile effettuare connessione\n");
14            exit(-1);
15        }
16        printf("Nuovo client connesso\n");
17        puntClientDesc = (int *)malloc(sizeof(int));
18        *puntClientDesc = clientDesc;
19        pthread_create(&tid, NULL, gestisci, (void *)puntClientDesc);
20    }
21    close(clientDesc);
22    quitServer();
23 }
```

In particolare al rigo 19 notiamo la creazione di un nuovo thread per gestire la connessione in entrata a cui passiamo il descrittore del client di cui si deve occupare.

3.2 Configurazione del client

Il cliente invece viene configurato e si connette al server tramite la seguente funzione:

Listato 3: Configurazione e connessione del client

```
1 int connettiAlServer(char **argv) {
2     char *indirizzoServer;
3     uint16_t porta = strtoul(argv[2], NULL, 10);
4     indirizzoServer = ipResolver(argv);
5     struct sockaddr_in mio_indirizzo;
6     mio_indirizzo.sin_family = AF_INET;
7     mio_indirizzo.sin_port = htons(porta);
8     inet_aton(indirizzoServer, &mio_indirizzo.sin_addr);
9     if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0)
10        perror("Impossibile creare socket"), exit(-1);
11    else
12        printf("Socket creato\n");
13    if (connect(socketDesc, (struct sockaddr *)&mio_indirizzo,
14              sizeof(mio_indirizzo)) < 0)
15        perror("Impossibile connettersi"), exit(-1);
16    else
17        printf("Connesso a %s\n", indirizzoServer);
18    return socketDesc;
19 }
```

Si noti come al rigo 9 viene configurato il socket ed al rigo 13 viene invece effettuato il tentativo di connessione al server.

Al rigo 3 invece viene convertita la porta inserita in input (argv[2]) dal tipo stringa al tipo della porta (uint16_t ovvero unsigned long integer).

Al rigo 4 notiamo invece la risoluzione dell'url da parte della funzione ipResolver che è riportata di seguito:

Listato 4: Risoluzione url del client

```
1 char *ipResolver(char **argv) {
2     char *ipAddress;
3     struct hostent *hp;
4     hp = gethostbyname(argv[1]);
5     if (!hp) {
6         perror("Impossibile risolvere l'indirizzo ip\n");
7         sleep(1);
8         exit(-1);
9     }
10    printf("Address:\t%s\n", inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]));
11    return inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]);
12 }
```

Al rigo 4, tramite l'url o l'indirizzo ip viene riempita la struttura hostent da cui poi possiamo estrarre l'indirizzo ip presente nel campo h_addr_list che, in effetti, è un array che contiene i vari indirizzi ip associati a quell'host.

Infine, al rigo 11 decidiamo di ritornare soltanto il primo indirizzo convertito in ascii.

3.3 Comunicazione tra client e server

La comunicazione tra client e server avviene tramite write e read sul socket.

Il comportamento del server e del client è determinato da particolari messaggi inviati e/o ricevuti che codificano, tramite interi o caratteri, la richiesta da parte del client di usufruire di un determinato servizio e la relativa risposta del server.

3.3.1 Esempio: la prima comunicazione

In particolare, una volta effettuata la connessione, il server attenderà un messaggio dal client per poter avviare una delle tre possibili procedure, ovvero login, registrazione ed uscita (rispettivamente codici: 1,2,3).

Di seguito sono riportate le relative funzioni di gestione che entrano in esecuzione subito dopo aver stabilito la connessione tra client e server.

Listato 5: Prima comunicazione del server

```
1 void *gestisci(void *descriptor) {
2     int bufferReceive[2] = {1};
3     int client_sd = *(int *)descriptor;
4     int continua = 1;
5     char name[MAX_BUF];
6     while (continua) {
7         read(client_sd, bufferReceive, sizeof(bufferReceive));
8         if (bufferReceive[0] == 2)
9             registraClient(client_sd);
10        else if (bufferReceive[0] == 1)
11            if (tryLogin(client_sd, name)) {
12                play(client_sd, name);
13                continua = 0;
14            } else if (bufferReceive[0] == 3)
15                disconnettiClient(client_sd);
16        else {
17            printf("Input invalido, uscita...\n");
18            disconnettiClient(client_sd);
19        }
20    }
21    pthread_exit(0);
22 }
```

Si noti come il server riceva, al rigo 7, il messaggio codificato da parte del client e metta in esecuzione la funzione corrispondente.

Listato 6: Prima comunicazione del client

```
1 int gestisci() {
2     char choice;
3     while (1) {
4         printMenu();
5         scanf("%c", &choice);
6         fflush(stdin);
7         system("clear");
8         if (choice == '3') {
9             esciDalServer();
10            return (0);
11        } else if (choice == '2') {
12            registrati();
13        } else if (choice == '1') {
14            if (tryLogin())
15                play();
16        } else
17            printf("Input errato, inserire 1,2 o 3\n");
18    }
19 }
```

4 Comunicazione durante la partita

Una volta effettuato il login, il client potrà iniziare a giocare tramite la funzione play (Vedi List. 7 e List. 8) che rappresentano il cuore della comunicazione tra client e server.

4.1 Funzione core del server

La funzione play del server consiste di un ciclo infinito nel quale il server invia al client tre informazioni importanti:

- La griglia di gioco (Rigo 26)
- Il player con le relative informazioni (Rigo 28 a 31)
- Il timer (Rigo 32)
- Un messaggio che notifica al client se è iniziato un nuovo turno oppure no

Dopodichè il thread del server rimane in attesa di ricevere l'input del client per spostare il giocatore sulla mappa tramite la relativa funzione. (Rigo 34)

Oltre questo, la funzione play del server si occupa anche di generare la posizione del player appena entra in partita e generare la nuova posizione del player (Rigo 52 a 66) quando viene effettuato il cambio di mappa (allo scadere del tempo o alla raggiunta del massimo punteggio).

4.2 Funzione core del client

Dall'altro lato, la funzione play del client, è stata mantenuta volutamente semplice. Lo scopo del client è unicamente quello di ricevere i dati forniti dal server, stampare la mappa di gioco e leggere l'input da inviare al server. Unica eccezione sono il rigo 30 e 32 del client che non richiedono la ricezione di ulteriori dati dal server: al rigo 30 si avvia la procedura di disconnessione del client ed al rigo 32 si stampa il timer che è stato ricevuto precedentemente dal server.

Listato 7: Funzione play del server

```

1 void play(int clientDesc, char name[]) {
2     int true = 1;
3     int turnoFinito = 0;
4     int turnoGiocatore = turno;
5     int posizione[2];
6     int destinazione[2] = {-1, -1};
7     PlayerStats giocatore = initStats(destinazione, 0, posizione, 0);
8     Obstacles listaOstacoli = NULL;
9     char inputFromClient;
10    if (timer != 0) {
11        inserisciPlayerNellaGrigliaInPosizioneCasuale(
12            grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
13            giocatore->position);
14        playerGenerati++;
15    }
16    while (true) {
17        if (clientDisconnesso(clientDesc)) {
18            freeObstacles(listaOstacoli);
19            disconnettiClient(clientDesc);
20            return;
21        }
22        char grigliaTmp[ROWS][COLUMNS];
23        clonaGriglia(grigliaTmp, grigliaDiGiocoConPacchiSenzaOstacoli);
24        mergeGridAndList(grigliaTmp, listaOstacoli);
25        // invia la griglia
26        write(clientDesc, grigliaTmp, sizeof(grigliaTmp));
27        // invia la struttura del player
28        write(clientDesc, giocatore->deploy, sizeof(giocatore->deploy));
29        write(clientDesc, giocatore->position, sizeof(giocatore->position));
30        write(clientDesc, &giocatore->score, sizeof(giocatore->score));
31        write(clientDesc, &giocatore->hasApack, sizeof(giocatore->hasApack));
32        sendTimerValue(clientDesc);
33        // legge l'input
34        if (read(clientDesc, &inputFromClient, sizeof(char)) > 0)
35            numMosse++;
36        if (inputFromClient == 'e' || inputFromClient == 'E') {
37            freeObstacles(listaOstacoli);
38            listaOstacoli = NULL;
39            disconnettiClient(clientDesc);
40        } else if (inputFromClient == 't' || inputFromClient == 'T') {
41            write(clientDesc, &turnoFinito, sizeof(int));
42            sendTimerValue(clientDesc);
43        } else if (inputFromClient == 'l' || inputFromClient == 'L') {
44            write(clientDesc, &turnoFinito, sizeof(int));
45            sendPlayerList(clientDesc);
46        } else if (turnoGiocatore == turno) {
47            write(clientDesc, &turnoFinito, sizeof(int));
48            giocatore =
49                gestisciInput(grigliaDiGiocoConPacchiSenzaOstacoli,
50                    grigliaOstacoliSenzaPacchi, inputFromClient, giocatore,
51                    &listaOstacoli, deployCoords, packsCoords, name);
52        } else {
53            turnoFinito = 1;
54            write(clientDesc, &turnoFinito, sizeof(int));
55            freeObstacles(listaOstacoli);
56            listaOstacoli = NULL;
57            inserisciPlayerNellaGrigliaInPosizioneCasuale(
58                grigliaDiGiocoConPacchiSenzaOstacoli, grigliaOstacoliSenzaPacchi,
59                giocatore->position);
60            giocatore->score = 0;
61            giocatore->hasApack = 0;
62            giocatore->deploy[0] = -1;
63            giocatore->deploy[1] = -1;
64            turnoGiocatore = turno;
65            turnoFinito = 0;
66            playerGenerati++;
67        }
68    }
69 }

```

Listato 8: Funzione play del client

```
1 void play() {
2     PlayerStats giocatore = NULL;
3     int score, deploy[2], position[2], timer;
4     int turnoFinito = 0;
5     int exitFlag = 0, hasApack = 0;
6     while (!exitFlag) {
7         if (serverCaduto())
8             serverCrashHandler();
9         if (read(socketDesc, grigliaDiGioco, sizeof(grigliaDiGioco)) < 1)
10            printf("Impossibile comunicare con il server\n"), exit(-1);
11        if (read(socketDesc, deploy, sizeof(deploy)) < 1)
12            printf("Impossibile comunicare con il server\n"), exit(-1);
13        if (read(socketDesc, position, sizeof(position)) < 1)
14            printf("Impossibile comunicare con il server\n"), exit(-1);
15        if (read(socketDesc, &score, sizeof(score)) < 1)
16            printf("Impossibile comunicare con il server\n"), exit(-1);
17        if (read(socketDesc, &hasApack, sizeof(hasApack)) < 1)
18            printf("Impossibile comunicare con il server\n"), exit(-1);
19        timer = getTimer();
20        giocatore = initState(deploy, score, position, hasApack);
21        printGrid(grigliaDiGioco, giocatore);
22        char send = getUserInput();
23        write(socketDesc, &send, sizeof(char));
24        read(socketDesc, &turnoFinito, sizeof(turnoFinito));
25        if (turnoFinito) {
26            system("clear");
27            printf("Turno finito\n");
28            sleep(1);
29        } else {
30            if (send == 'e' || send == 'E')
31                printf("Disconnessione in corso...\n"), exit(0);
32            if (send == 't' || send == 'T')
33                printTimer();
34            else if (send == 'l' || send == 'L')
35                printPlayerList();
36        }
37    }
38 }
```

A Codici sorgente

Di seguito sono riportati tutti i codici sorgenti integrali del progetto.

A.1 Codice sorgente del client

```
1 #include "boardUtility.h"
2 #include "list.h"
3 #include "parser.h"
4 #include <arpa/inet.h>
5 #include <fcntl.h>
6 #include <netdb.h>
7 #include <netinet/in.h> //conversioni
8 #include <netinet/in.h>
9 #include <netinet/ip.h> //struttura
10 #include <pthread.h>
11 #include <signal.h>
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <string.h>
15 #include <sys/socket.h>
16 #include <sys/stat.h>
17 #include <sys/types.h>
18 #include <time.h>
19 #include <unistd.h>
20
21 void printPlayerList();
```

```

22 int getTimer();
23 void printTimer();
24 void play();
25 int tryLogin();
26 void printMenu();
27 int connettiAlServer(char **argv);
28 char *ipResolver(char **argv);
29 int registrati();
30 int gestisci();
31 char getUserInput();
32 void clientCrashHandler();
33 void serverCrashHandler();
34 int serverCaduto();
35 void esciDalServer();
36 int isCorrect(char);
37
38 int socketDesc;
39 char grigliaDiGioco[ROWS][COLUMNS];
40
41 int main(int argc, char **argv) {
42     signal(SIGINT, clientCrashHandler); /* CTRL-C */
43     signal(SIGHUP, clientCrashHandler); /* Chiusura della console */
44     signal(SIGQUIT, clientCrashHandler);
45     signal(SIGTSTP, clientCrashHandler); /* CTRL-Z */
46     signal(SIGTERM, clientCrashHandler); /* generato da 'kill' */
47     signal(SIGPIPE, serverCrashHandler);
48     char bufferReceive[2];
49     if (argc != 3) {
50         perror("Inserire indirizzo ip/url e porta (./client 127.0.0.1 5200)");
51         exit(-1);
52     }
53     if ((socketDesc = connettiAlServer(argv)) < 0)
54         exit(-1);
55     gestisci(socketDesc);
56     close(socketDesc);
57     exit(0);
58 }
59 void esciDalServer() {
60     int msg = 3;
61     printf("Uscita in corso\n");
62     write(socketDesc, &msg, sizeof(int));
63     close(socketDesc);
64 }
65 int connettiAlServer(char **argv) {
66     char *indirizzoServer;
67     uint16_t porta = strtoul(argv[2], NULL, 10);
68     indirizzoServer = ipResolver(argv);
69     struct sockaddr_in mio_indirizzo;
70     mio_indirizzo.sin_family = AF_INET;
71     mio_indirizzo.sin_port = htons(porta);
72     inet_aton(indirizzoServer, &mio_indirizzo.sin_addr);
73     if ((socketDesc = socket(PF_INET, SOCK_STREAM, 0)) < 0)
74         perror("Impossibile creare socket"), exit(-1);
75     else
76         printf("Socket creato\n");
77     if (connect(socketDesc, (struct sockaddr *)&mio_indirizzo,
78         sizeof(mio_indirizzo)) < 0)
79         perror("Impossibile connettersi"), exit(-1);
80     else
81         printf("Connesso a %s\n", indirizzoServer);
82     return socketDesc;
83 }
84 int gestisci() {
85     char choice;
86     while (1) {
87         printMenu();
88         scanf("%c", &choice);
89         fflush(stdin);
90         system("clear");
91         if (choice == '3') {
92             esciDalServer();
93             return (0);
94         } else if (choice == '2') {
95             registrati();

```

```

96     } else if (choice == '1') {
97         if (tryLogin())
98             play();
99     } else
100         printf("Input errato, inserire 1,2 o 3\n");
101     }
102 }
103 int serverCaduto() {
104     char msg = 'y';
105     if (read(socketDesc, &msg, sizeof(char)) == 0)
106         return 1;
107     else
108         write(socketDesc, &msg, sizeof(msg));
109     return 0;
110 }
111 void play() {
112     PlayerStats giocatore = NULL;
113     int score, deploy[2], position[2], timer;
114     int turnoFinito = 0;
115     int exitFlag = 0, hasApack = 0;
116     while (!exitFlag) {
117         if (serverCaduto())
118             serverCrashHandler();
119         if (read(socketDesc, grigliaDiGioco, sizeof(grigliaDiGioco)) < 1)
120             printf("Impossibile comunicare con il server\n"), exit(-1);
121         if (read(socketDesc, deploy, sizeof(deploy)) < 1)
122             printf("Impossibile comunicare con il server\n"), exit(-1);
123         if (read(socketDesc, position, sizeof(position)) < 1)
124             printf("Impossibile comunicare con il server\n"), exit(-1);
125         if (read(socketDesc, &score, sizeof(score)) < 1)
126             printf("Impossibile comunicare con il server\n"), exit(-1);
127         if (read(socketDesc, &hasApack, sizeof(hasApack)) < 1)
128             printf("Impossibile comunicare con il server\n"), exit(-1);
129         timer = getTimer();
130         giocatore = initStats(deploy, score, position, hasApack);
131         printGrid(grigliaDiGioco, giocatore);
132         char send = getUserInput();
133         write(socketDesc, &send, sizeof(char));
134         read(socketDesc, &turnoFinito, sizeof(turnoFinito));
135         if (turnoFinito) {
136             system("clear");
137             printf("Turno finito\n");
138             sleep(1);
139         } else {
140             if (send == 'e' || send == 'E')
141                 printf("Disconnessione in corso...\n"), exit(0);
142             if (send == 't' || send == 'T')
143                 printTimer();
144             else if (send == 'l' || send == 'L')
145                 printPlayerList();
146         }
147     }
148 }
149 void printPlayerList() {
150     system("clear");
151     int lunghezza = 0;
152     char buffer[100];
153     int continua = 1;
154     int number = 1;
155     fprintf(stdout, "Lista dei player: \n");
156     if (!serverCaduto(socketDesc)) {
157         read(socketDesc, &continua, sizeof(continua));
158         while (continua) {
159             read(socketDesc, &lunghezza, sizeof(lunghezza));
160             read(socketDesc, buffer, lunghezza);
161             buffer[lunghezza] = '\0';
162             fprintf(stdout, "%d) %s\n", number, buffer);
163             continua--;
164             number++;
165         }
166         sleep(1);
167     }
168 }
169 void printTimer() {

```

```

170     int timer;
171     if (!serverCaduto(socketDesc)) {
172         read(socketDesc, &timer, sizeof(timer));
173         printf("\t\tTempo restante: %d...\n", timer);
174         sleep(1);
175     }
176 }
177 int getTimer() {
178     int timer;
179     if (!serverCaduto(socketDesc))
180         read(socketDesc, &timer, sizeof(timer));
181     return timer;
182 }
183 int tryLogin() {
184     int msg = 1;
185     write(socketDesc, &msg, sizeof(int));
186     system("clear");
187     printf("Inserisci i dati per il Login\n");
188     char username[20];
189     char password[20];
190     printf("Inserisci nome utente(MAX 20 caratteri): ");
191     scanf("%s", username);
192     printf("\nInserisci password(MAX 20 caratteri):");
193     scanf("%s", password);
194     int dimUname = strlen(username), dimPwd = strlen(password);
195     if (write(socketDesc, &dimUname, sizeof(dimUname)) < 0)
196         return 0;
197     if (write(socketDesc, &dimPwd, sizeof(dimPwd)) < 0)
198         return 0;
199     if (write(socketDesc, username, dimUname) < 0)
200         return 0;
201     if (write(socketDesc, password, dimPwd) < 0)
202         return 0;
203     char validate;
204     int ret;
205     read(socketDesc, &validate, 1);
206     if (validate == 'y') {
207         ret = 1;
208         printf("Accesso effettuato\n");
209     } else if (validate == 'n') {
210         printf("Credenziali Errate o Login gia' effettuato\n");
211         ret = 0;
212     }
213     sleep(1);
214     return ret;
215 }
216 int registrati() {
217     int msg = 2;
218     write(socketDesc, &msg, sizeof(int));
219     char username[20];
220     char password[20];
221     system("clear");
222     printf("Inserisci nome utente(MAX 20 caratteri): ");
223     scanf("%s", username);
224     printf("\nInserisci password(MAX 20 caratteri):");
225     scanf("%s", password);
226     int dimUname = strlen(username), dimPwd = strlen(password);
227     if (write(socketDesc, &dimUname, sizeof(dimUname)) < 0)
228         return 0;
229     if (write(socketDesc, &dimPwd, sizeof(dimPwd)) < 0)
230         return 0;
231     if (write(socketDesc, username, dimUname) < 0)
232         return 0;
233     if (write(socketDesc, password, dimPwd) < 0)
234         return 0;
235     char validate;
236     int ret;
237     read(socketDesc, &validate, sizeof(char));
238     if (validate == 'y') {
239         ret = 1;
240         printf("Registrato con successo\n");
241     }
242     if (validate == 'n') {
243         ret = 0;

```

```

244     printf("Registrazione fallita\n");
245 }
246 sleep(1);
247 return ret;
248 }
249 char *ipResolver(char **argv) {
250     char *ipAddress;
251     struct hostent *hp;
252     hp = gethostbyname(argv[1]);
253     if (!hp) {
254         perror("Impossibile risolvere l'indirizzo ip\n");
255         sleep(1);
256         exit(-1);
257     }
258     printf("Address:\t%s\n", inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]));
259     return inet_ntoa(*(struct in_addr *)hp->h_addr_list[0]);
260 }
261 void clientCrashHandler() {
262     int msg = 3;
263     int rec = 0;
264     printf("\nChiusura client...\n");
265     do {
266         write(socketDesc, &msg, sizeof(int));
267         read(socketDesc, &rec, sizeof(int));
268     } while (rec == 0);
269     close(socketDesc);
270     signal(SIGINT, SIG_IGN);
271     signal(SIGQUIT, SIG_IGN);
272     signal(SIGTERM, SIG_IGN);
273     signal(SIGTSTP, SIG_IGN);
274     exit(0);
275 }
276 void serverCrashHandler() {
277     system("clear");
278     printf("Il server e' stato spento o e' irraggiungibile\n");
279     close(socketDesc);
280     signal(SIGPIPE, SIG_IGN);
281     premiEnterPerContinuare();
282     exit(0);
283 }
284 char getUserInput() {
285     fflush(stdin);
286     char c;
287     c = getchar();
288     int daIgnorare;
289     while ((daIgnorare = getchar()) != '\n' && daIgnorare != EOF) {
290     }
291     return c;
292 }

```

- A.2 server.c**
- A.3 boardUtility.c**
- A.4 boardUtility.h**
- A.5 lsit.c**
- A.6 lsit.h**
- A.7 parser.c**
- A.8 parser.h**

Listati

1	Configurazione socket del server	2
2	Procedura di ascolto del server	2
3	Configurazione e connessione del client	3
4	Risoluzione url del client	3
5	Prima comunicazione del server	4
6	Prima comunicazione del client	4
7	Funzione play del server	6
8	Funzione play del client	7