Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Classe delle Lauree Magistrali in Ingegneria Elettronica, Classe n. LM-29

Corso di Laurea Magistrale in Ingegneria Elettronica

Thesis

Title according to the official assignment

Professore:
Finzi Alberto

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

Anno Accademico
2019/2020

Indice

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 manualemente 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* ri 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 cient 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 comprarià 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 dei gioco si 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 protocoli PF_NET, con tipo di trasmissione dati SOCK_STREAM e con protocollo TCP. Mostriamo di seguito il codice sorgente:

```
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)) <
         0)
7      perror("Impossibile impostare il riutilizzo dell'indirizzo ip e della "
8             "porta\n");
9    if ((bind(socketDesc, (struct sockaddr *)&mio_indirizzo,
10             sizeof(mio_indirizzo))) < 0) {
11     perror("Impossibile effettuare bind");
12     exit(-1);
13   }
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:

```
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:

```
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:

```
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 ritoranre soltanto il primo indirizzo convertito in ascii.