UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

RELAZIONE PROGETTO L.S.O. "Traccia C"

Docente: Alberto Finzi
Componente: Gianluca Panzuto (N86/1099)

Capitolo 0: Introduzione

Prima di spiegare come funziona l'intero progetto, ecco alcune indicazioni relative al funzionamento e alla comprensione del programma:

1) Nella registrazione utente, quando dobbiamo inserire delle stringhe, è preferibile evitare spaziature ed utilizzare caratteri come “_” oppure “-”;

2) Rispettare il tipo di input stabilito. Esempio: in fase di registrazione, alla richiesta dell'età dell'utente da registrare, non bisogna inserire caratteri alfanumerici, ma solo numeri;

3) I dati relativi agli utenti, ai voti dei vari hotel e ai commenti dei vari hotel sono registrati tutti rispettivamente nei file “***login.txt***”, “**voti.txt**” e “***commenti.txt***”;

4) Il file dove il server effettuerà il logging delle attività principali è il file “***cronologia.txt***”;

5) La media dei voti ai vari hotel è di tipo intero, quindi verranno effettuato un troncamento nel caso di divisioni con resto diverso da zero;

6) All'inizio tutti i file di testo sono vuoti. Quindi per effettuare il login, bisogna prima registrarsi.

Capitolo 1: GUIDA D'USO DEL CLIENT

Il client è composto da due menù:

- Quello iniziale, dove è possibile effettuare una registrazione di un nuovo utente, il login di un utente già registrato oppure l'abbandono del programma;

- Quello utente dopo un login avvenuto con successo, dove è possibile effettuare una valutazione di un hotel, di visualizzare i voti medi dei vari hotel e di visualizzare i commenti effettuati dai vari utenti.

La registrazione avviene nel seguente modo: l'utente inserisce i dati (nome,cognome,età,indirizzo,username e password) e li invia al server. Se esiste un utente già registrato con username uguale a quello inserito dall'utente per la registrazione, allora essa fallisce e verrà notificato al client un messaggio di fallimento. Invece se la registrazione va a buon fine, il client riceverà un messaggio di successo.

Il login avviene nel seguente modo: l'utente inserisce l'username e la password e, in caso di successo, accede al menù di valutazione degli hotel. In caso di fallimento del login, l'utente ritorna al menù iniziale.

Per valutazione di hotel, si intende che l'utente può dare un voto, espresso come un valore numerico compreso tra -5 e 5, oppure può commentare un hotel, inserendo massimo 120 caratteri.
Se un utente ha già effettuato una valutazione (sia tramite voto, sia tramite commento) ad un determinato hotel, viene stampato in STDOUT un messaggio di fallimento dell'operazione. Altrimenti viene visualizzato un messaggio di successo.

Una volta effettuato il login, l'utente entra nel secondo menù e può effettuare le seguenti operazioni:

- Valutazione di un hotel;
- Visualizzare i voti medi dei vari hotel;
- Visualizzare i commenti che gli utenti hanno effettuato ai vari hotel;
- Visualizzare eventuali notifiche;
- Effettuare un logout.

Se l'utente effettua un logout, esso ritorna al menù principale.

Se l'utente effettua una valutazione dell'hotel, esso sceglie l'hotel da valutare, il tipo di valutazione e il giudizio; se voto, verrà salvato nel file

“***voti.txt***” nel seguente modo: “***<utente>;<hotel>;<voto>;***”, altrimenti verrà salvato nel file “***commenti.txt***” nel seguente modo: “***<utente>;<hotel>;<commento>;***”.

Nella visualizzazione dei voti medi e dei commenti bisogna notificare alcune cose:

- La visualizzazione dei voti medi ai vari hotel verrà visualizzata nel seguente modo: “***<hotel> : <voto_medio>***”;

- Se uno di essi non ha ricevuto voto, allora affianco all'hotel verrà visualizzata una stringa “nessun voto”;

- La visualizzazione dei commenti ai vari hotel verrà visualizzata nel seguente modo: “***l'utente <utente> ha commentato il seguente hotel: <hotel> –--> <commento***>”.

- Se l'utente sceglie di visualizzare i commenti dei vari hotel e nessun hotel ha ricevuto commento, allora verrà visualizzata la stringa “***nessun commento***”.

Nel caso il server si chiude all'improvviso, tramite il comando “CTRL+C”, e il client invia un messaggio al server, viene generato un segnale SIGPIPE che notifica la chiusura del server.

Nel caso il client viene chiuso tramite il comando “CTRL+C”, viene generato un segnale SIGINT che notifica la chiusura improvvisa del client.

Per quanto riguarda la notifica, sarà il server a verificare se sono state effettuate delle valutazioni di hotel valutato dal client richiedente.

La compilazione e l'esecuzione del client su terminale avviene nel seguente modo:

*** COMPILAZIONE ***

gcc client.c -o client

*** ESECUZIONE ***

./client <indirizzo_server> <numero_di_porta>

N.B. nel caso in cui vengono eseguiti client e server nella stessa macchina, allora <indirizzo_server> sarà “localhost” oppure “127.0.0.1”.

Capitolo 2: GUIDA D'USO DEL SERVER

Il server è di tipo **multiconcorrente multithread**, cioè più client si possono connettere al server e viene creato un thread per ogni client che si connette ad esso.

Il server non invia alcun output su **STDOUT** e non riceve nessun input da **STDIN**.
Può solo inviare output su **STDERR** solo in caso di terminazione.

Una volta accettata la connessione di un client e creato un apposito thread, il server riceve da client una stringa con tante parole separate da un “;”.
La prima parola indica l'operazione che deve effettuare il server.
Il server verifica che tipo di operazione deve effettuare e, una volta che l'operazione è stata eseguita, invia al client una stringa di risposta.
La stringa di risposta dipende dal tipo di operazione che ha scelto di effettuare il client; essa può essere:

- **Una valutazione (registrazione,login,voto,commento)**: invia una stringa al client dove comunica l'esito dell'operazione;

- **Una visualizzazione (voti medi, commenti e notifiche):** invia al client una stringa con tutti i dati richiesti dal client.

Se una valutazione di un hotel effettuata dal client è ritenuta corretta, il server la salva sull'apposito file (se voto “voti.txt”, se commento “commenti.txt”, se registrazione “login.txt”).

Il server registra le operazioni di valutazione effettuate dai vari client in un apposito file chiamato “cronologia.txt”. Esso è composto nel seguente modo:

Data: <DATA>
Ora: <ORA>
Operazione: <OPERAZIONE>
Utente: <UTENTE>, se l'operazione effettuata non è una registrazione.

Per fare in modo che due o più client non possano scrivere contemporaneamente sullo stesso file, ci sta la struttura a semaforo (mutex) con zona critica la scrittura dei file e sulla modifica di variabili globali.

Se un client ha effettuato una determinata valutazione ad un hotel e quell'hotel è stato già valutato da uno o più client, essi dovranno essere notificati. Per fare ciò, utilizziamo la chiamata signal(SIGUSR,<handler>) dove nella funzione handler, mettiamo una flag di notifica a 1 per gli utenti che devono ricevere la notifica. Successivamente ai client con la flag di notifica a 1 gli verrà notificato l'ingresso di una nuova valutazione.

Nel caso il server si chiude all'improvviso, tramite il comando “CTRL+C”, verrà sollevato il segnale SIGINT e il server stamperà su STDERR la chiusura del server.

Nel caso un client connesso al server si chiude in maniera inaspettata (esempio: CTRL+C), viene generato un SIGPIPE. La sua funzione handler provvederà all'eliminazione del thread associato e a stampare su STDERR il messaggio di eliminazione di tale thread.

La compilazione e l'esecuzione del client su terminale avviene nel seguente modo:

***** COMPILAZIONE *****

gcc server.c -o server -lpthread

***** ESECUZIONE *****

./server <numero_di_porta>

Capitolo 3: PROTOCOLLI UTILIZZATI

Per il progetto, ho scelto di utilizzare la seguente famiglia di protocolli: ***AF_INET*** (Internet Address Family).

Grazie ad essa, la comunicazione tra client e server non avviene solo in locale, ma anche tramite internet (vers. 4).

Il tipo di trasmissione del messaggio è il seguente: ***SOCK_STREAM***. Con esso, il tipo di trasmissione è a flusso, sequenziale ed affidabile.

Capitolo 4: DETTAGLI IMPLEMENTATIVI

1) Semafori (MUTEX):

/* creazione nuovo mutex */

pthread_mutex_t lock;

/* inizializzo mutex */

pthread_mutex_init(&lock, NULL);

/* blocco il semaforo. Inizio zona critica */

pthread_mutex_lock(&lock);

/* sblocco il semaforo. Fine zona critica */

pthread_mutex_unlock(&lock);

2) Socket TCP:

/* len = memorizza il client; portno = numero di porta; s_fd e c_fd = connessioni rispettivamente del server e del client */

int len,portno,s_fd,c_fd;

/* Composizione dell'indirizzo */

struct sockaddr_in sin;

/* Creo il socket TCP */

socket(AF_INET, SOCK_STREAM, 0);

/* assegno una famiglia di protocolli */

sin.sin_family = AF_INET;

/* Host TO Network Short --- conversione formato numerico per il network */

sin.sin_port = htons(portno);

/* Host TO Network Short --- conversione formato numerico per il

network */

sin.sin_addr.s_addr = htonl(INADDR_ANY);

/* Host TO Network Long --- INADDR_ANY --- accetta connessioni generiche */

len = sizeof(client);

/* Associazione del socket TCP all'indirizzo */

bind(s_fd, (struct sockaddr *) &sin, sizeof(sin));

/* Rimane in ascolto per le richieste di connessione del client */

listen(s_fd,5);

/* Accetta la connessione del client */

accept(s_fd, (struct sockaddr *)&client, &len);

/* Converte un indirizzo in una stringa */

inet_ntop(AF_INET, &client.sin_addr, buffer, sizeof(buffer));

/* *restituisce una struttura di tipo "hostent" in cui mette l'indirizzo ip* */

gethostbyname(argv[1]);

/* Connetti il client al server */

connect(c_fd, (struct sockaddr *) &sin, sizeof(sin));

/* Chiudi la connessione */

close(s_fd);

3) Pthread:

/* creazione nuovo thread */

pthread_create(&tid, NULL, gestisci, (void *) c_fd);

/* Lo spazio allocato per un thread può essere deallocato/riutilizzato quando il thread muore */

pthread_detach(tid);

/* chiusura del thread */

pthread_exit ();

4) System Call I/O:

/* file descriptor */

int file;

/* apro il file <*nome_file*> in modalità <*modalità_apertura*> con i permessi <*permessi*>. Restituisce -1 in caso di errore */

file = open (“<nome_file>”,<modalità_apertura>,<permessi>);

/* Leggo il file <*nome_file*> e memorizzo il tutto su “stringa”. Memorizza al massimo 9000 caratteri. Restituisce -1 in caso di errore */

read (file,stringa,9000);

/* scrivo sul file <nome_file> il contenuto della stringa “stringa” Restituisce -1 in caso di errore */

write (file,stringa,strlen(stringa));

/* chiudo il file descriptor */

close (file);

5) Segnali

/* imposto l'handler <funzione> per il segnale <segnale> */

signal (<segnale>,<funzione>);

/* invio al thread con tid <tid> il segnale <signal> */

pthread_kill (<tid>,<segnale>);
Segnali utilizzati:

- **SIGINT** : Se si riceve un segnale causato dalla seguente combinazione di tasti: ***"CTRL+C"***;

- **SIGUSR1** : segnale definito dall'utente;

- **SIGPIPE** : Se un processo che dovrebbe leggere da una pipe termina inaspettatamente, questo segnale viene inviato al programma che dovrebbe scrivere sulla pipe in questione.

CODICE SORGENTE

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>
#include <netinet/in.h>
#include <netdb.h>
#define N 9000

// STRUTTURA NOTIFICA
typedef struct el
{
   char hotel[100];
   char op[100];
   char utente[100];
   int inserito;
}notifica;

// PROTOTIPI
void login (int);
void menu (int,char[]);
/* funzioni handler */
void rottura (int);
void invia (int);

//FUNZIONE PRINCIPALE

int main(int argc, char *argv[])
{
   int i,c_fd,portno,y;
   /*
      segnali utilizzati.
      SIGPIPE, nel caso il server si disconnette
      SIGINT nel caso di chiusura forzata del client
   */
   signal (SIGPIPE,rottura);
   signal (SIGINT,invia);
   // struttura indirizzo
   struct sockaddr_in sin;
   struct hostent *hp;
   char buffer[N], buf[N];
   /* Controllo sul numero di argomenti */
   if (argc != 3)
   {
      printf("Illegal number of arguments");
      exit(1);
   }
   /* Ottengo l'indirizzo del server tramite nome */
   if ((hp = gethostbyname(argv[1])) == 0)
   {
      perror("tom: gethostbyname");
      exit(1);
   }
   /* Creo il socket TCP */
   c_fd = socket(AF_INET, SOCK_STREAM, 0);
   if (c_fd < 0)
   {
```

```
        perror("socket");
        return 1;
    }
    portno = atoi(argv[2]); // numero di porta
    sin.sin_family = AF_INET; // assegno la famiglia di protocolli
    memcpy((char *)&sin.sin_addr,(char *)hp->h_addr,hp->h_length);
    sin.sin_port = htons(portno);
    // Mi connetto al server
    if(connect(c_fd, (struct sockaddr *) &sin, sizeof(sin)) < 0)
    {
        printf("connect() failed\n");
        return 1;
    }
    // se la connessione è andata a buon fine, allora vado al menù principale
    login (c_fd);
}

// MENU INIZIALE DEL CLIENT

void login (int c_fd)
{
    int val = 1,scelta,err,n=0,y, eta,k;
    char c, stringa[1000],utente[N],nome [20], cognome [20],indirizzo [100],nickname [100],pwd [100], et[3];
    do
    {
        for (k = 0; k < 8; k++)
            printf ("\n\n\n\n\n\n\n\n\n\n");
        printf ("\n\nLOGIN\n\n'0' = Registrazione Utente\n\n'1' = Login Utente\n\n'2' = Uscita dal programma\n\nScelta: ");
        err = scanf ("%d",&scelta);
        fflush(stdin);
        if (err == 0)
            printf("valore errato");
        else if (scelta == 0)
        {
            strcpy(stringa,"");
            strcpy(stringa,"registrazione;");
            printf ("Inserire nome: ");
            scanf ("%s",nome);
            strcat(stringa,nome);
            strcat(stringa,";");
            printf ("Inserire cognome: ");
            scanf ("%s",cognome);
            strcat(stringa,cognome);
            strcat(stringa,";");
            printf ("Inserire eta': ");
            scanf ("%d",&eta);
            sprintf(et,"%d",eta);
            strcat(stringa,et);
            strcat(stringa,";");
            printf ("Inserire indirizzo: ");
            scanf ("%s",indirizzo);
            strcat(stringa,indirizzo);
            strcat(stringa,";");
            printf ("Inserire nickname: ");
            scanf ("%s",nickname);
            strcat(stringa,nickname);
            strcat(stringa,";");
            printf ("Inserire password: ");
            scanf ("%s",pwd);
            strcat(stringa,pwd);
            strcat(stringa,";\0");
            write(c_fd,stringa,strlen(stringa));
            strcpy(stringa,"");
            y = read (c_fd,stringa,N);
            stringa[y] = '\0';
            printf ("*** %s ***\n",stringa);
            sleep(3);
```

```
        }
        else if (scelta == 1)
        {
           strcpy(stringa,"");
           strcpy(stringa,"login;");
           printf ("Nome Utente: ");
           scanf ("%s",utente);
           printf ("Password: ");
           scanf ("%s",pwd);
           strcat(stringa,utente);
           strcat(stringa,";");
           strcat(stringa,pwd);
           strcat(stringa,";");
           write (c_fd,stringa,strlen(stringa));
           strcpy(stringa,"");
           y = read (c_fd,stringa,N);
           stringa[y] = '\0';
           y = 0;
           while (stringa[y] != ';')
           {
              nome[y] = stringa[y];
              ++y;
           }
           nome[y] = '\0';
          if (strcmp (nome,"login_corretto") == 0)
           {
              printf ("Login corretto\n");
              ++y;
              k = 0;
              while (stringa[y] != ';' && stringa[y] != '\0')
              {
                 et[k] = stringa[y];
                 ++y;++k;
              }
              et[k] = '\0';
              menu(c_fd,et);
           }
            else
              printf ("login fallito\n");
        }
        else if (scelta == 2)
        {
           printf ("Arrivederci\n\n");
           strcpy(stringa,"");
           strcpy(stringa,"uscita;");
           write(c_fd,stringa,strlen(stringa));
        }
        else
        {
           printf ("Errore!!!\nPremi INVIO per continuare...");
           scanf ("%c",&c);
        }
     }while (scelta != 2);
}

//INTERFACCIA MENU' DOPO IL LOGIN

void menu (int c_fd,char utente[])
{
    signal (SIGPIPE,rottura);
    char hotel[5][100]={"Holiday_Inn","Excelsior","Star_Hotel","Royal_Palace","Miramare"};
    int choice,error,scelta,voto,scelta1,i,j,k,nutenti,y,x;
    char stringa[N],commento[120],valore[2],hot[100],ut[200],c;
    notifica not [100];
    for (i = 0; i < 100; i++)
       not[i].inserito = -1;
    do
```

```
        {
    for (i = 0; i < 8; i++)
      printf ("\n\n\n\n\n\n\n\n\n\n");
                printf ("  _____________________________________________________________  \n");
                printf (" |                                                             | \n");
                printf (" |            ***   VALUTAZIONE HOTEL  ***                    | \n");
                printf (" |                  BENVENUTO                          | \n");
                printf ("  _____________________________________________________________  \n");
    printf ("\nUtente -----> %s ",utente);
                printf ("\n\n'0' = LOGOUT\n\n'1' = EFFETTUA UNA VALUTAZIONE DELL'HOTEL\n\n");
                printf ("'2' = VISUALIZZA VALUTAZIONE HOTEL\n\n'3' = VISUALIZZA COMMENTI\n\n'4' =
VISUALIZZA NOTIFICHE\n\nscelta: ");
                error = scanf ("%d", &choice);
          if (error == 0)
                        printf ("Errore input\n");
    else if (choice < 0 || choice > 4)
    {
      printf ("\n **** SCELTA NON VALIDA **** \n\nPremi INVIO per continuare...");
      scanf ("%c",&c);
    }
    else
    {
      switch (choice)
      {
        // SCELTA '0': LOGOUT UTENTE
        case 0:
          strcpy(stringa,"");
          strcpy(stringa,"logout;");
          strcat(stringa,utente);
          strcat(stringa,";");
          write(c_fd,stringa,strlen(stringa));
          strcpy (stringa,"");
          y = read (c_fd,stringa,N);
          stringa[y] = '\0';
          break;

        // SCELTA '1': VALUTAZIONE RISTORANTE CON VOTO O COMMENTO
        case 1:
          printf ("\n\n*** AVETE SCELTO DI EFFETTUARE UNA VALUTAZIONE DI UN HOTEL ***\n\n");
          printf("\n\n'0' = Holiday_Inn\n'1' = Excelsior\n'2' = Star_Hotel\n'3' = Royal_Palace\n'4' = Miramare\n\nscelta: ");
          scanf("%d",&scelta1);
          while (scelta1 < 0 || scelta1 > 4)
          {
            printf ("Errore!!!\nInserire un valore tra 0 e 4\n\nscelta: ");
            scanf("%d",&scelta1);
          }
          printf ("Scegliere il tipo di valutazione:\n'1' = voto numerico\n'2' = commento\n\nscelta: ");
          scanf("%d",&scelta);
          // INSERIMENTO NUOVO VOTO
          if (scelta == 1)
          {
            printf ("Inserire voto: ");
            scanf ("%d",&voto);
            while (voto < -5 || voto > 5)
            {
              printf ("Attenzione: voto non valido.\nBisogna rispettare il seguente intervallo: (-5) - (+5)\n\nInserire voto: ");
              scanf("%d",&voto);
            }
            sprintf(valore,"%d",voto);
            strcpy(stringa,"");
            strcpy(stringa,"voto;");
            strcat(stringa,utente);
            strcat(stringa,";");
            strcat(stringa,hotel[scelta1]);
            strcat(stringa,";");
            strcat(stringa,valore);
            strcat(stringa,";");
```

```
        write (c_fd,stringa,strlen(stringa));
        strcpy(stringa,"");
        y = read (c_fd,stringa,N);
        stringa[y] = '\0';
        printf ("%s\n",stringa);
    }
    // INSERIMENTO NUOVO COMMENTO
    else if (scelta == 2)
    {
        printf ("Inserire commento: ");
        scanf ("%s",commento);
        strcpy(stringa,"");
        strcpy(stringa,"commento;");
        strcat(stringa,utente);
        strcat(stringa,";");
        strcat(stringa,hotel[scelta1]);
        strcat(stringa,";");
        strcat(stringa,commento);
        strcat(stringa,";");
        write (c_fd,stringa,strlen(stringa));
        y = read (c_fd,stringa,N);
        stringa[y] = '\0';
        printf ("%s\n",stringa);
    }
    else
        printf ("\nERRORE SCELTA\n\n");
    break;

// SCELTA '2': VISUALIZZAZIONE VOTI HOTEL
case 2:
    printf ("\n\n*** AVETE SCELTO DI VISUALIZZARE LE VALUTAZIONI DEI VARI HOTEL ***\n\n");
    strcpy(stringa,"");
    strcpy(stringa,"visualizza_voti;");
    write (c_fd,stringa,strlen(stringa));
    y = read (c_fd,stringa,N);
    stringa[y] = '\0';
    j = 0;
    for (i=0; i < 5; i++)
    {
        printf ("%s : ",hotel[i]);
        k = 0;
        while (stringa[j] != ';')
        {
            valore[k] = stringa[j];
            ++j; ++k;
        }
        valore[k] = '\0';
        ++j;
        if (strcmp(valore,"-999") == 0)
            printf ("nessun voto\n\n");
        else
            printf ("%s\n\n",valore);
    }
    break;

// SCELTA '3': VISUALIZZAZIONE COMMENTI RISTORANTI
case 3:
    printf ("\n\n*** AVETE SCELTO DI VISUALIZZARE DEI COMMENTI ***\n\n");
    strcpy(stringa,"");
    strcpy(stringa,"visualizza_commenti;");
    write (c_fd,stringa,strlen(stringa));
    strcpy(stringa,"");
    y = read (c_fd,stringa,N);
    stringa[y] = '\0';
    if (strcmp(stringa,”null”)= 0)
        printf ("\nNon sono stati effettuati commenti\n");
    else
```

```
                {
                    y = 0;
                    k = 0;
                    while (stringa[y] != '\0')
                    {
                        k = 0;
                        while (stringa[y] != ';')
                        {
                            ut[k] = stringa[y];
                            ++k;++y;
                        }
                        ut[k] = '\0';
                        ++y;
                        k = 0;
                        while (stringa[y] != ';' && stringa[y] != '\0')
                        {
                            hot[k] = stringa[y];
                            ++k;++y;
                        }
                        hot[k] = '\0';
                        ++y;
                        k = 0;
                        while (stringa[y] != ';' && stringa[y] != '\0')
                        {
                            commento[k] = stringa[y];
                            ++k;++y;
                        }
                        commento[k] = '\0';
                        ++y;
                        printf ("\nL'utente %s ha commentato il seguente hotel : %s --> %s\n",ut,hot,commento);
                    }
                }
                break;

            // CENTRO NOTIFICHE
            case 4:
                strcpy(stringa,"");
                strcpy(stringa,"notifica;");
                write (c_fd,stringa,strlen(stringa));
                y = read (c_fd,stringa,N);
                stringa[y] = '\0';
                if (strcmp(stringa,"null") != 0)
                    printf (" -------> ATTENZIONE: %s <-------\n",stringa);
                else
                    printf ("NESSUNA NOTIFICA\n\n");
                break;
        }
        if (choice != 0)
        {
            do
            {
                printf ("\n\n\nLOGOUT? [0/*]: ");
                error = scanf ("%d",&choice);
                if (error == 0)
                                        printf ("errore input\n");
                else if (choice == 0)
                    printf ("\n\nLOGOUT\n");
            } while (error == 0);
        }
    }
        } while (error == 0 || choice != 0);
}


// HANDLER SIGPIPE

void rottura (int segnale)
```

```
{
   signal (segnale,SIG_IGN);
   perror ("\nIL SERVER E' STATO CHIUSO INASPETTATAMENTE\n");
   signal (segnale,rottura);
}

// HANDLER SIGINT
void invia (int segnale)
{
   printf ("\nCHIUSURA INASPETTATA DEL CLIENT\n");
   exit(1);
}
```

< ---------- >

server.c

```
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <pthread.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <time.h>
#define N 9000


// STRUTTURA UTENTI

typedef struct utenti
{
   char nome[50];
   char cognome[50];
   int eta;
   char indirizzo[100];
   char user[1000];
   char password[1000];
   int inserito;
   int notificato;
}dati_utente;

// STRUTTURA DEI VOTI E COMMENTI

typedef struct st1
{
   char utente[50];
   char hotel[100];
   char commento[120];
   int voto;
   int voto_inserito; // VERIFICA SE E' STATO VOTATO QUELL'HOTEL (VALORE POSITIVO) OPPURE NO (VALORE -1)
   int com_inserito; // VERIFICA SE E' STATO COMMENTATO QUELL'HOTEL (VALORE POSITIVO) OPPURE NO (VALORE
-1)
   int inserito;
}commenti;
```

```
typedef struct elemento
{
   unsigned int tid;
   char utente[20];
   int notificato;
   struct elemento * next;
}lista_utenti;

// MUTEX
pthread_mutex_t lock;

// ARRAY DI UTENTI REGISTRATI
dati_utente ut [N];

// ARRAY DI VALUTAZIONI
commenti com [N];

// LISTA UTENTI LOGGATI
lista_utenti * T = NULL;

// PROTOTIPI DELLE FUNZIONI

int azione (char []);
int memorizzazione ();
int registrazione (char []);
int suddividi (char []);
int suddividi_voti (char []);
int memorizza_voti ();
int inserisci_voto (char []);
int suddividi_commenti (char []);
int memorizza_commenti ();
int inserisci_commento (char []);
int media (char []);
void * gestisci (void *); // FUNZIONE THREAD
// Funzioni Handler
void notifica (int);
void rottura (int);
void rottura1 (int);

// METODO MAIN

int main(int argc, char *argv[])

{
   int s_fd, c_fd, portno, len, err, err1,i,n_utenti = 0, file;
   pthread_t tid;
   if (pthread_mutex_init(&lock, NULL) != 0)
   {
      perror ("mutex");
      return 1;
   }
   char buffer[N], stringa [N];
   /* CONTROLLO SUL NUMERO DI PARAMETRI */
   if (argc != 2)
   {
      printf ("Errore parametri\n");
      return 0;
   }
   for (i = 0; i < 90; i++)
      ut[i].inserito = -1;
   /* Composizione dell'indirizzo */
   struct sockaddr_in sin, client;
   /* Creazione del socket TCP */
   s_fd = socket(AF_INET, SOCK_STREAM, 0);
   if (s_fd == -1)
   {
```

```
        perror ("socket");
        return 1;
    }
    portno = atoi(argv[1]);
    sin.sin_family = AF_INET; // assegno una famiglia di protocolli
    sin.sin_port = htons(portno); // Host TO Network Short --- conversione formato numerico per il network
    sin.sin_addr.s_addr = htonl(INADDR_ANY); // Host TO Network Long --- INADDR_ANY --- accetta connessioni generiche
    len = sizeof(client); // Memorizzo le informazioni sulla struttura d'indirizzo del client
    /* Associazione del socket TCP all'indirizzo */
    bind(s_fd, (struct sockaddr *) &sin, sizeof(sin));
    /* Il server si mette in ascolto */
    listen(s_fd,5);
    /* Segnali. SIGUSR1 per la notifica. SIGINT in caso di chiusura forzata tramite CTRL+C */
    signal (SIGUSR1,notifica);
    signal (SIGINT,rottura);
    signal (SIGPIPE,rottura1);
    /* Ciclo di accettazione di client */
    while (1){
        /* Accetto la connessione di un client */
        c_fd = accept(s_fd, (struct sockaddr *)&client, &len);
        if (c_fd < 0)
        {
            perror("accept");
            return 1;
        }
        inet_ntop(AF_INET, &client.sin_addr, buffer, sizeof(buffer));
        /* Creo il thread per il client connesso */
        err = pthread_create(&tid, NULL, gestisci, (void *) c_fd);
        err1 = pthread_detach(tid);
    }
    /* Chiudo la connessione */
    close(s_fd);
    return 0;
}

// VERIFICA IL TIPO DI AZIONE CHE DEVE EFFETTUARE IL SERVER

int azione (char buf [])
{
    int i = 0;
    char verifica[N];
    while (buf[i] != ';')
    {
        verifica[i] = buf[i];
        ++i;
    }
    verifica[i] = '\0';
    if (strcmp(verifica,"registrazione") == 0)
        return 0;
   else if (strcmp(verifica,"login") == 0)
        return 1;
    else if (strcmp(verifica,"voto") == 0)
        return 2;
    else if (strcmp(verifica,"commento") == 0)
        return 3;
    else if (strcmp(verifica,"visualizza_voti") == 0)
        return 4;
    else if (strcmp(verifica,"visualizza_commenti") == 0)
        return 5;
    else if (strcmp(verifica,"logout") == 0)
        return 6;
    else if (strcmp(verifica,"notifica") == 0)
        return 7;
   else
        return 8;
}
```

```
// SPEZZA LA STRINGA PRESA DAL FILE "login.txt" IN MODO DA MEMORIZZARE GLI UTENTI

int suddividi (char buffer [])

{
   int x = 0,y = 0, i = 0, z = 0;
   pthread_mutex_lock(&lock); // INIZIO DELLA ZONA CRITICA
   for (i = 0; i < 90; i++)
      ut[i].inserito = -1;
   i = 0;
   char valore[3];
   while (buffer[y] != '\0')
   {
      if (buffer[y] == '\n')
         ++x;
      while (buffer[x] != ';' && buffer[x] != '\0')
      {
         ut[i].nome[z] = buffer [x];
         ++x;
         ++z;
      }
      ut[i].nome[z] = '\0';
      ++x;
      z = 0;
      while (buffer[x] != ';' && buffer[x] != '\0')
      {
         ut[i].cognome[z] = buffer [x];
         ++x;
         ++z;
      }
      ut[i].cognome[z] = '\0';
      ++x;
      z = 0;
      while (buffer[x] != ';' && buffer[x] != '\0')
      {
         valore[z] = buffer [x];
         ++x;
      }
      valore[z] = '\0';
      ut[i].eta = atoi (valore);
      ++x;
      z = 0;
      while (buffer[x] != ';' && buffer[x] != '\0')
      {
         ut[i].indirizzo[z] = buffer [x];
         ++x;
         ++z;
      }
      ut[i].indirizzo[z] = '\0';
      ++x;
      z = 0;
      while (buffer[x] != ';' && buffer[x] != '\0')
      {
         ut[i].user[z] = buffer [x];
         ++x;
         ++z;
      }
      ut[i].user[z] = '\0';
      ++x;
      z = 0;
      while (buffer[x] != ';' && buffer[x] != '\0')
      {
         ut[i].password[z] = buffer [x];
         ++x;
         ++z;
      }
      ut[i].password[z] = '\0';
```

```
        ++x;
        z = 0;
        y = x;
        ut[i].inserito = 1;
        ut[i].notificato = 0;
        ++i;
    }
    pthread_mutex_unlock(&lock); // FINE DELLA ZONA CRITICA
    return i;
}

// RESTITUISCE IL NUMERO DI UTENTI MEMORIZZATI

int memorizzazione ()
{
    int i,n ,y, file;
    char buffer [N];
    file = open ("login.txt",O_RDONLY);
    y = read (file,buffer,N);
    buffer[y] = '\0';
    close(file);
    n = suddividi (buffer);
    return n;
}

/ EFFETTUA LA REGISTRAZIONE DI UN UTENTE

int registrazione (char buf[])
{
    int verifica,i = 0, z = 0,utenti, file;
    utenti = memorizzazione ();
    char nome [20], cognome [20],indirizzo [100],nickname [100],pwd [100],stringa[1000],et[3],tmp;
    int eta,vero=1;
    while (buf[i] != ';')
    {
        tmp = buf[i];
        ++i;
    }
    ++i;
    while (buf[i] != ';')
    {
        nome[z] = buf[i];
        ++i;
        ++z;
    }
    nome[z] = '\0';
    z = 0;
    ++i;
    while (buf[i] != ';')
   {
        cognome[z] = buf[i];
        ++i;
        ++z;
   }
    cognome[z] = '\0';
    z = 0;
    ++i;
    while (buf[i] != ';')
    {
        et[z] = buf[i];
        ++i;
        ++z;
    }
    et[z] = '\0';
    eta = atoi(et);
    z = 0;
    ++i;
```

```
    while (buf[i] != ';')
    {
        indirizzo[z] = buf[i];
        ++i;
        ++z;
    }
    indirizzo[z] = '\0';
    z = 0;
    ++i;
    while (buf[i] != ';')
    {
        nickname[z] = buf[i];
        ++i;
        ++z;
    }
    nickname[z] = '\0';
    z = 0;
    ++i;
    while (buf[i] != ';')
    {
        pwd[z] = buf[i];
        ++i;
        ++z;
    }
    pwd[z] = '\0';
    for (i = 0; i < 90 && vero == 1 && ut[i].inserito != -1; i++)
    {
        if (strcmp(ut[i].user,nickname) == 0)
            vero = 0;
    }
    if (vero == 1)
    {
        pthread_mutex_lock(&lock); // INIZIO DELLA ZONA CRITICA
        strcpy (ut[utenti].nome,nome);
        strcpy (ut[utenti].cognome,cognome);
        strcat(stringa,ut[utenti].cognome);
        ut[utenti].eta = eta;
        strcpy (ut[utenti].indirizzo,indirizzo);
        strcpy (ut[utenti].user,nickname);
        strcpy (ut[utenti].password,pwd);
        ut[utenti].inserito = 1;
        ut[utenti].notificato = 0;
        pthread_mutex_unlock(&lock); // FINE DELLA ZONA CRITICA
    }
    return vero;
}

// SUDDIVIDE LA STRINGA OTTENUTA DAL FILE "voti.txt"

int suddividi_voti (char buffer [])
{
    int x = 0,y = 0, i = 0, z = 0;
    char valore[3];
    pthread_mutex_lock(&lock); // INIZIO DELLA ZONA CRITICA
    for (i = 0; i < N; i++)
      com[i].inserito = com[i].voto_inserito = com[i].com_inserito = -1;
    i = 0;
    while (buffer[y] != '\0')
    {
        if (buffer[y] == '\n')
            ++x;
        while (buffer[x] != ';' && buffer[x] != '\0')
        {
            com[i].utente[z] = buffer [x];
            ++x;
            ++z;
        }
```

```
        com[i].utente[z] = '\0';
        ++x;
        z = 0;
        while (buffer[x] != ';' && buffer[x] != '\0')
        {
           com[i].hotel[z] = buffer [x];
           ++x;
           ++z;
        }
        com[i].hotel[z] = '\0';
        ++x;
        z = 0;
        while (buffer[x] != ';' && buffer[x] != '\0')
        {
           valore[z] = buffer [x];
           ++x;++z;
        }
        valore[z] = '\0';
        com[i].voto = atoi (valore);
        ++x;
        z = 0;
        y = x;
        com[i].voto_inserito = 1;
        com[i].inserito = 1;
        ++i;
    }
    pthread_mutex_unlock(&lock); // FINE DELLA ZONA CRITICA
    return i;
}

// MEMORIZZAZIONE DELLE VOTAZIONI DEI VARI HOTEL

int memorizza_voti ()
{
    int i = 0, j = 0, k = 0, nvoti = 0, file, y;
    char stringa[N];
    file = open ("voti.txt",O_RDONLY);
    y = read (file,stringa,N);
    stringa[y] = '\0';
    close (file);
    nvoti = suddividi_voti (stringa);
    return nvoti;
}

// INSERIMENTO DEL VOTO

int inserisci_voto (char buf[])
{
    int i = 0,k = 0, file,nvoti,voto, si = 1;
    char ut[20],tmp,hotel[100],val[2],stringa[800];
    nvoti = memorizza_voti ();
    while (buf[i] != ';')
    {
       tmp = buf[i];
       ++i;
    }
    ++i;
    while (buf[i] != ';')
    {
       ut[k] = buf[i];
       ++i;++k;
    }
   ut[k] = '\0';
    k = 0;
    ++i;
    while (buf[i] != ';')
    {
```

```
        hotel[k] = buf[i];
        ++i;++k;
    }
    hotel[k] = '\0';
    k = 0;
    ++i;
    while (buf[i] != ';')
    {
        val[k] = buf[i];
        ++i;++k;
    }
    val[k] = '\0';
    voto = atoi(val);
    if (voto > 0 && val[0] == '-')
        voto -= (2*voto);
    for (i = 0; i < nvoti && si == 1; i++)
    {
        if (com[i].voto_inserito != -1 && strcmp(com[i].hotel,hotel) == 0 && strcmp(com[i].utente,ut) == 0)
            si = -1;
        else if (com[i].voto_inserito == -1 && strcmp(com[i].hotel,hotel) == 0 && strcmp(com[i].utente,ut) == 0)
            si = i;
    }
   strcpy(stringa,"");
    if (si == 1)
    {
        pthread_mutex_lock(&lock); // INIZIO DELLA ZONA CRITICA
        com[i].voto_inserito = 1;
        com[i].voto = voto;
        com[i].inserito = 1;
        strcpy(com[i].utente,ut);
        strcpy(com[i].hotel,hotel);
        pthread_mutex_unlock(&lock); // FINE DELLA ZONA CRITICA
    }
    else if (si != -1)
    {
        pthread_mutex_lock(&lock); // INIZIO DELLA ZONA CRITICA
        com[si].voto_inserito = 1;
        com[si].voto = voto;
        si = 1;
        pthread_mutex_unlock(&lock); // FINE DELLA ZONA CRITICA
   }
    return si;
}

// SUDDIVISIONE DELLA STRINGA OTTENUTA DAL FILE "commenti.txt"

int suddividi_commenti (char buffer [])

{
    int x = 0,y = 0, i = 0, z = 0;
    pthread_mutex_lock(&lock); // INIZIO DELLA ZONA CRITICA
    for (i = 0; i < N; i++)
        com[i].inserito = com[i].voto_inserito = com[i].com_inserito = -1;
    i = 0;
    while (buffer[y] != '\0')
    {
        if (buffer[y] == '\n')
            ++x;
        while (buffer[x] != ';' && buffer[x] != '\0')
        {
            com[i].utente[z] = buffer [x];
            ++x;
            ++z;
        }
        com[i].utente[z] = '\0';
        ++x;
        z = 0;
```

```
        while (buffer[x] != ';' && buffer[x] != '\0')
        {
           com[i].hotel[z] = buffer [x];
           ++x;
           ++z;
        }
        com[i].hotel[z] = '\0';
        ++x;
        z = 0;
        while (buffer[x] != ';' && buffer[x] != '\0')
        {
           com[i].commento[z] = buffer [x];
           ++x;
        }
        com[i].commento[z] = '\0';
        ++x;
        z = 0;
        y = x;
        com[i].com_inserito = 1;
        com[i].inserito = 1;
        ++i;
    }
    pthread_mutex_unlock(&lock); // FINE DELLA ZONA CRITICA
    return i;
}

// MEMORIZZAZIONE DEI COMMENTI

int memorizza_commenti ()
{
    int i = 0, j = 0, k = 0, ncommenti = 0, file, y;
    char stringa[N];
    file = open ("commenti.txt",O_RDONLY);
    y = read (file,stringa,N);
    stringa[y] = '\0';
    close (file);
    ncommenti = suddividi_commenti (stringa);
    return ncommenti;
}

// INSERIMENTO DI UN COMMENTO

int inserisci_commento (char buf [])
{
    int i = 0,k = 0, ncommenti, file, voto, si = 1;
    char ut[20],tmp,hotel[100],commento[120],stringa[N];
    ncommenti = memorizza_commenti ();
    while (buf[i] != ';')
    {
       tmp = buf[i];
       ++i;
    }
    ++i;
    while (buf[i] != ';')
    {
       ut[k] = buf[i];
       ++i;++k;
    }
    ut[k] = '\0';
    k = 0;
    ++i;
    while (buf[i] != ';')
    {
       hotel[k] = buf[i];
       ++i;++k;
    }
    hotel[k] = '\0';
```

```
    k = 0;
    ++i;
    while (buf[i] != ';')
    {
        commento[k] = buf[i];
        ++i;++k;
    }
    commento[k] = '\0';
    for (i = 0; i < ncommenti && si == 1 && com[i].inserito != -1; i++)
    {
        if (com[i].com_inserito != -1 && strcmp(com[i].hotel,hotel) == 0 && strcmp(com[i].utente,ut) == 0)
            si = -1;
        else if (com[i].com_inserito == -1 && strcmp(com[i].hotel,hotel) == 0 && strcmp(com[i].utente,ut) == 0)
            si = i;
    }
    if (si == 1)
    {
        pthread_mutex_lock(&lock); // INIZIO DELLA ZONA CRITICA
        com[i].com_inserito = 1;
        strcpy(com[i].commento,commento);
        strcpy(com[i].utente,ut);
        strcpy(com[i].hotel,hotel);
        com[i].inserito = 1;
        pthread_mutex_unlock(&lock); // FINE DELLA ZONA CRITICA
    }
    else if (si != -1)
    {
        pthread_mutex_lock(&lock); // INIZIO DELLA ZONA CRITICA
        com[si].com_inserito = 1;
        strcpy(com[si].commento,commento);
        pthread_mutex_unlock(&lock); // FINE DELLA ZONA CRITICA
        si = 1;
    }
    return si;
}

// VERIFICA LOGIN

int verifica_login (char buffer[])
{
    int i = 0, j = 0, si = -1,utenti;
    char user[1000],pwd[1000],tmp;
    utenti = memorizzazione();
    while (buffer[i] != ';')
    {
        tmp = buffer[i];
        ++i;
    }
    ++i;
    while (buffer[i] != ';')
    {
        user[j] = buffer[i];
        ++i;
        ++j;
    }
    user[j] = '\0';
    ++i;
    j = 0;
    while (buffer[i] != ';')
    {
        pwd[j] = buffer[i];
        printf ("%c",pwd[j]);
        ++i;
        ++j;
    }
    pwd[j] = '\0';
    for (i = 0; i < utenti && ut[i].inserito != -1 && si == -1; i++)
```

```
    {
        if (strcmp(ut[i].user,user) == 0 && strcmp (ut[i].password,pwd) == 0)
            si = i;
    }
    if (si >= 0)
        si = 1;
    return si;
}

// CALCOLO MEDIA VOTI RISTORANTE

int media (char hotel[])
{
    int conta = 0, somma = 0,i,stop = 0, nvoti = 0;
    nvoti = memorizza_voti ();
    for (i = 0; i < nvoti && com[i].inserito != -1; i++)
    {
        if (com[i].voto_inserito != -1 && strcmp(com[i].hotel,hotel) == 0)
        {
            ++conta;
            somma += com[i].voto;
        }
    }
    if (conta == 0)
        return -999;
    else
        return (somma/conta);
}

// FUNZIONE SIGNAL IN CASO DI SIGPIPE

void rottura1 (int segnale)
{
    perror ("\n\nClient Disconnesso all'improvviso. Chiudo il suo thread\n\n");
    pthread_exit(NULL);

}

// FUNZIONE SIGNAL IN CASO DI SIGPINT

void rottura (int segnale)
{
    perror ("\n\nSERVER CHIUSO INASPETTATAMENTE\n\n");
    exit(1);
}

// FUNZIONE SIGNAL IN CASO DI SIGUSR1

void notifica (int segnale)
{
    lista_utenti *L = T;
    while (L != NULL && L -> tid != pthread_self() ) L = L -> next;
    if (L != NULL)
        L -> notificato = 1;
}

// DEALLOCA LA LISTA DEGLI UTENTI CONNESSI

void dealloca_lista (lista_utenti * L)
{
    if (L == NULL)
        return;
    lista_utenti * p = L -> next;
    free(L);
    dealloca_lista(p);
}
```

```
// INVIA IL SEGNALE SIGUSR1 A TUTTI GLI ALTRI THREAD, IN MODO DA NOTIFICARLI.

void avviso (char hotel[])
{
   lista_utenti *L = T;
   int i;
   while (L != NULL)
   {
      if (L -> tid != pthread_self() )
      {
         for (i = 0; i < N && com[i].inserito != -1; i++)
         {
            if (strcmp(L->utente,com[i].utente) == 0 && strcmp (hotel,com[i].hotel) == 0)
               pthread_kill(T->tid,SIGUSR1);
         }
      }
      L = L -> next;
   }
}

// FUNZIONE THREAD

void *gestisci(void *arg)
{
   char hotel[5][100]={"Holiday_Inn","Excelsior","Star_Hotel","Royal_Palace","Miramare"};
   int x,valore,file, k, cronologia, critica, log,not, conta, check;
   time_t rawtime;
   struct tm* leggibile;
   lista_utenti *TMP = NULL, *L;
   char buf[N],verifica1[N], val[40], tmp,hot[N], ut[32],verifica[4]
[100],tmp2[N],operazione[100],ora[10],minuti[10],secondi[10],giorno[10],mese[10],anno[10], notifica[N];
   strcpy (notifica,"");
   do
   {
      not = 0;
      critica = 0;
      for (k = 0; k < 4; k++)
        verifica[k][0]='\0';
      strcpy(verifica1,"");
      x = read ((int *) arg, buf, N);
      buf[x] = '\0';
      if (x > 0)
         valore = azione (buf);
      switch (valore)
      {
         case 0:
            x = registrazione(buf);
            strcpy(verifica1,"");
            strcpy (verifica1,"REGISTRAZIONE\n\n");
            if (x == 1)
            {
               strcpy(verifica1,"registrazione_correttamente_avvenuta");
               strcpy(verifica[0],buf);
               strcpy(verifica[3],buf);
               critica = 1;
               time (&rawtime);
              leggibile = localtime(&rawtime);
            }
            else
               strcpy(verifica1,"registrazione_non_valida");
            break;

         case 1:
            x = verifica_login (buf);
            if (x != -1)
           {
               strcpy(verifica1,"");
```

```
        strcpy(verifica1,"login_corretto;");
        x = 0;
        while (buf[x] != ';')
        {
           tmp = buf[x];
           ++x;
        }
        ++x;
        k = 0;
        while (buf[x] != ';')
        {
           ut[k] = buf[x];
           ++x; ++k;
        }
        ut[k] = '\0';
        pthread_mutex_lock (&lock); //INIZIO ZONA CRITICA
        if (T == NULL)
        {
           T = (lista_utenti *) malloc (sizeof(lista_utenti));
           T -> tid = pthread_self();
           strcpy(T->utente,ut);
           T -> notificato = 0;
           T -> next = NULL;
        }
       else
        {
           TMP = (lista_utenti *) malloc (sizeof(lista_utenti));
           TMP -> tid = pthread_self();
           strcpy(TMP->utente,ut);
           TMP -> notificato = 0;
           TMP -> next = T;
           T = TMP;
        }
        pthread_mutex_unlock(&lock); // FINE ZONA CRITICA
        strcat(verifica1,ut);
        strcat(verifica1,";");
        check = 1;
        strcpy (verifica[3],buf);
        critica = 1;
        time (&rawtime);
        leggibile = localtime(&rawtime);
     }
     else
        strcpy(verifica1,"login_errato;");
     break;

case 2:
     strcpy(verifica1,"");
     x = inserisci_voto (buf);
     if (x == 1)
     {
        strcpy(verifica1,"voto_correttamente_inserito");
        strcpy(verifica[1],buf);
        strcpy(verifica[3],buf);
        k = 0;
        while (buf[k] != ';')
        {
           tmp2[k] = buf[k];
           ++k;
        }
        tmp2[k] = '\0';
        x = 0;
        ++k;
        while (buf[k] != ';')
        {
           ut[x] = buf[k];
           ++k;++x;
```

```
        }
        ut[x] = '\0';
        ++k;
        x = 0;
        while (buf[k] != ';')
        {
           hot[x] = buf[k];
           ++k;++x;
        }
        hot[x] = '\0';
        strcpy (notifica,"");
        avviso (hot);
        critica = 1;
       time (&rawtime);
        leggibile = localtime(&rawtime);
    }
    else
       strcpy(verifica1,"hai_già_votato_questo_hotel");
    break;

case 3:
    strcpy(verifica1,"");
    x = inserisci_commento(buf);
    if (x == 1)
    {
       strcpy(verifica1,"commento_correttamente_inserito");
       strcpy(verifica[2],buf);
       strcpy(verifica[3],buf);
       k = 0;
       while (buf[k] != ';')
       {
          tmp2[k] = buf[k];
          ++k;
       }
       tmp2[k] = '\0';
       x = 0;
      ++k;
       while (buf[k] != ';')
       {
          ut[x] = buf[k];
          ++k;++x;
       }
       ut[x] = '\0';
       ++k;
       x = 0;
      while (buf[k] != ';')
       {
          hot[x] = buf[k];
          ++k;++x;
       }
       hot[x] = '\0';
       avviso(hot);
       critica = 1;
       time (&rawtime);
       leggibile = localtime(&rawtime);
    }
    else
       strcpy(verifica1,"hai_già_commentato_questo_hotel");
    break;

case 4:
    strcpy(verifica1,"");
    for (x = 0; x < 5; x++)
    {
       valore = media (hotel[x]);
       sprintf(val,"%d",valore);
       strcat(verifica1,val);
```

```
            strcat(verifica1,";");
         }
         break;

      case 5:
        file = open ("commenti.txt",O_RDONLY);
         strcpy (verifica1,"");
         x = read (file,verifica1,N);
         verifica1[x] = '\0';
         close (file);
         if (strlen(verifica1) == 0)
            strcpy(verifica1,"null");
         break;

      case 6:
         strcpy(verifica[3],buf);
         L = T;
         while (L -> next != NULL && L -> tid != pthread_self())
            L = L -> next;
         if (L != NULL)
         {
            TMP = L;
            L = TMP -> next;
            free(L);
         }
         time (&rawtime);
         leggibile = localtime(&rawtime);
         critica = 1;
         strcpy(verifica1,"");
         strcpy(verifica1,"logout");
         break;

      case 7:
         if (strlen(notifica) > 0)
         {
            strcpy(verifica1,notifica);
            strcpy(notifica,"");
         }
         else
            strcpy(verifica1,"null");
         break;

      case 8:
         strcpy(verifica1,"");
         check = 0;
         strcpy(verifica1,"uscita");
         break;
   }
    /*CONTROLLO DELLE NOTIFICHE*/
    if (check == 1)
   {
      L = T;
      strcpy(notifica,"");
      while (L != NULL && L -> tid != pthread_self())
         L = L -> next;
      if (L -> notificato == 1)
      {
         strcat(notifica,"INSERITA_NUOVA_VALUTAZIONE");
         L -> notificato = 0;
      }
   }
   /* DEVO MODIFICARE DEI FILE? */
   if (critica == 1)
   {
      pthread_mutex_lock(&lock); // INIZIO DELLA ZONA CRITICA
      strcpy(tmp2,"\0");
      if (strlen(verifica[0]) > 0)
```

```
{
   cronologia = open ("login.txt",O_WRONLY,777);
   lseek (cronologia,0,SEEK_END);
   k = 0;
   while (verifica[0][k] != ';')
   {
      tmp = verifica[0][k]; k++;
   }
   ++k;
   x = 0;
   while (verifica[0][k] != '\0')
   {
      tmp2[x] = verifica[0][k];
      ++x;++k;
   }
   tmp2[x] = '\0';
   write (cronologia,tmp2,strlen(tmp2));
   close(cronologia);
}
else if (strlen(verifica[1]) > 0)
{
   cronologia = open ("voti.txt",O_WRONLY,777);
   lseek (cronologia,0,SEEK_END);
   k = 0;
   while (verifica[1][k] != ';')
   {
      tmp = verifica[1][k]; k++;
   }
   ++k;
   x = 0;
   while (verifica[1][k] != '\0')
   {
      tmp2[x] = verifica[1][k];
      ++x;++k;
   }
   tmp2[x] = '\0';
   write (cronologia,tmp2,strlen(tmp2));
   close(cronologia);
}
else if (strlen(verifica[2]) > 0)
{
   cronologia = open ("commenti.txt",O_WRONLY,777);
   lseek (cronologia,0,SEEK_END);
   k = 0;
   while (verifica[2][k] != ';')
   {
      tmp = verifica[2][k]; k++;
   }
   ++k;
   x = 0;
   while (verifica[2][k] != '\0')
   {
      tmp2[x] = verifica[2][k];
      ++x;++k;
   }
   tmp2[x] = '\0';
   write (cronologia,tmp2,strlen(tmp2));
   close(cronologia);
}
if (strlen(verifica[3]) > 0)
{
   strcpy(operazione,"");
   cronologia = open ("cronologia.txt",O_WRONLY,777);
   lseek (cronologia,0,SEEK_END);
   k = 0;
   strcpy(ut,"");
   while (verifica[3][k] != ';')
```

```
{
   operazione[k] = verifica[3][k];
   ++k;
}
operazione[k] = '\0';
if (strcmp (operazione,"registrazione") != 0)
{
   ++k;
   x = 0;
   while(verifica[3][k] != ';')
   {
      ut[x] = verifica[3][k];
      ++x;++k;
   }
   ut[x] = '\0';
}
ut[x] = '\0';
sprintf (ora,"%d",leggibile->tm_hour);
sprintf (minuti,"%d",leggibile->tm_min);
sprintf (secondi,"%d",leggibile->tm_sec);
sprintf (giorno,"%d",leggibile->tm_mday);
sprintf (mese,"%d",leggibile->tm_mon+1);
sprintf (anno,"%d",leggibile->tm_year+1900);
strcpy (tmp2,"\nORARIO --> ");
strcat (tmp2,ora);
strcat (tmp2,":");
if (strlen(minuti) == 1)
{
   strcat(tmp2,"0");
   strcat(tmp2,"\0");
}
strcat (tmp2,minuti);
strcat (tmp2,":");
if (strlen(secondi) == 1)
{
   strcat(tmp2,"0");
   strcat(tmp2,"\0");
}
 strcat (tmp2,secondi);
strcat (tmp2,"\n");
strcat (tmp2,"DATA --> ");
if (strlen(giorno) == 1)
{
   strcat(tmp2,"0");
   strcat(tmp2,"\0");
}
strcat (tmp2,giorno);
strcat (tmp2,"/");
if (strlen(mese) == 1)
{
   strcat(tmp2,"0");
   strcat(tmp2,"\0");
}
strcat (tmp2,mese);
strcat (tmp2,"/");
strcat (tmp2,anno);
strcat (tmp2,"\n");
strcat (tmp2,"OPERAZIONE --> ");
if (strcmp(operazione,"voton") == 0)
   strcpy(operazione,"voto");
 else if (strcmp(operazione,"logint") == 0)
   strcpy(operazione,"login");
strcat (tmp2,operazione);
strcat (tmp2,"\n");
if (strlen(ut) > 0)
{
   strcat (tmp2,"UTENTE --> ");
```

```
                strcat (tmp2,ut);
                strcat (tmp2,"\n");
              }
              strcat (tmp2,"\n--------\n\n");
              write (cronologia,tmp2,strlen(tmp2));
              close(cronologia);
            }
            pthread_mutex_unlock(&lock); // FINE DELLA ZONA CRITICA
          }
          write((void *)arg,verifica1,strlen(verifica1));
      }while (strcmp (verifica1,"uscita") != 0);
      dealloca_lista(T);
      close((void *) arg);
      pthread_exit (0);
}
```