SEGRETERIA STUDENTI

LABORATORIO DI SISTEMI OPERATIVI – A.A. 2017/2018

De Sanctis Fabrizio N86/1708
Di Finizio Daniele N86/1801
UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

1- Guida all'uso

1.1 Compilazione ed esecuzione

Le istruzioni da riga di comando per la compilazione del progetto sono:

- Server: gcc -o server server.c AuxFunctions.c -pthread
- Client: gcc -o client client.c AuxFunctions.c

Successivamente, per eseguire i programmi sarà sufficiente digitare:

- Server: ./server <PORTA> <NOME FILE>
- Client: ./client <IP> <PORTA> <NOME FILE>

Nota: Le parentesi "<" e ">" non vanno inserite.

1.2 Guida d'uso

Per eseguire il server devono essere passati in ordine i seguenti parametri da riga di comando:

1. Il numero della porta su cui mettersi in ascolto.
2. Il nome del file dal quale bisogna leggere e sui cui bisogna scrivere.

Per eseguire il client devono essere passati in ordine i seguenti parametri da riga di comando:

1. IP a cui collegarsi.
2. Porta a cui collegarsi.
3. Nome di un file.

Nota: Il numero di porta del client deve essere uguale a quella fornita dal server per l'ascolto.

2- Descrizione e implementazione

SERVER:

Strutture dati e variabili globali:

int listen_port	La porta su cui mettersi in ascolto.
int sock	Socket.
char file_name[30]	Nome del file.
RecordingsList Registro	Lista che contiene gli esami già registrati.
VoteSumList SommaVoti	Lista che contiene la somma dei voti.
pthread_mutex_t print_lock	Semaforo per la scrittura sul file.
pthread_mutex_t registro_lock	Semaforo per l'inserimento in lista esami.
pthread_mutex_t sva_lock	Semaforo per l'inserimento in lista somma voti.

struct TRecordings {
uint16_t matr;
char exam[5];
uint16_t year;
uint16_t vote;
};
	Struttura dati che conterrà le informazioni riguardanti gli esami registrati.

struct TListRecordings {
Recordings r;
struct TListRecordings* next;
};
	Struttura dati che rappresenta una lista di esami. Il campo informazione si riferisce al tipo precedentemente descritto.

struct TVoteSum {
uint16_t matr;
uint16_t sum;
};
	Struttura dati che conterrà la somma di tutti gli esami di una particolare matricola.

struct TVoteSumList {
VoteSum i;
struct TVoteSumList* next;
};
	Struttura dati che rappresenta una lista di matricole e la relativa somma degli esami. Il campo informazione si riferisce al tipo precedentemente descritto.

Dopo aver effettuato gli opportuni controlli sui parametri in ingresso, i valori di questi ultimi vengono salvati nelle variabili globali apposite.

A questo punto si stampa a video la porta su cui si è in ascolto e si procede così:

- *FASE 1.* Lettura operazioni da file.
- *FASE 2.* Creazione e connessione thread.
- *FASE 3.* Operazioni da svolgere.
- *FASE 4.* Scrittura su file.

Fase 1: Lettura operazioni da file

RecordingsList ReadToFile(RecordingsList L,char file_name[]);

È la funzione principale in questa fase, si occupa di leggere da file tutti gli esami già registrati e di inserirli in una lista.
In particolare, prende in input una lista in cui verranno inseriti gli esami registrati ed il nome del file dal quale prelevarli. La funzione si occuperà di leggere da file le informazioni ed inserirle correttamente nei giusti campi del record.
Al termine di questa prima fase, la funzione ritornerà un puntatore alla testa di una lista.

RecordingsList Operation(int file_descriptor,RecordingsList L,char matr[]);

La funzione "Operation" aiuta la funzione "ReadoToFile" a compiere le operazioni descritte in precedenza.

RecordingsList initNodeListR(Recordings r);

E' una funzione ausiliare che permette alla funzione "ReadToFile" di portare a termine la Fase 1 del programma. Questa funzione si occupa di inizializzare un nuovo record che conterrà le informazioni su un esame già registrato.

RecordingsList appendNodeListR(RecordingsList L, Recordings r);

Altra funzione ausiliare. In questo caso, la funzione si occupa di inserire il nodo alla fine della lista che conterrà tutti gli esami registrati. In particolare, passiamo la lista e un nodo. La funzione scorrerà tutta la lista per poi inserire il nuovo nodo. Se la lista è vuota, lo inserisce direttamente. Ritorna la testa della lista.

Fase 2: Creazione e connessione thread

void Connection_creation();

Questa procedura ha il compito di dare il via alla seconda fase dell'esecuzione, cioè creare e stabilire una connessione. La "Connecion_creation" non fa altro che creare un thread che invocherà la procedura incaricata di creare la connessione.

void *Connect(void*);

Si occupa di inizializzare tutte le informazioni relative alla creazione del socket di comunicazione. Rende attivo il servizio e si mette in continua attesa di nuovi client. Quando un nuovo utente viene accettato, si occupa di lanciare il thread che gestirà le operazioni da svolgere.
La procedura si occuperà di gestire le informazioni da passare al client e stamperà a video un messaggio in cui avvertirà di essere in attesa di nuovi client.

void *NewClient();

E' sicuramente la funzione più importante di questa fase. Viene lanciata ogni volta che un client viene accettato e di conseguenza viene creato un nuovo thread. Questa procedura si occupa di ricevere e decodificare le operazioni che il client vuole compiere. Innanzitutto decodifica l'operazione, salvandone in una struttura dati il tipo e le informazioni aggiuntive. Successivamente si occupa di gestirla, dando vita alla "FASE 3" , lanciando l'apposita routine a seconda del tipo di operazione. Una volta terminata la gestione dell'operazione, il client viene avvertito dell'esito e passa alla successiva. La funzione termina quando tutte le operazioni sono terminate oppure se arriva un'operazione "CLO", prima però si occupa di effettuare una scrittura su file di tutti gli esami registrati (FASE 4). Essa è responsabile anche della gestione della sincronizzazione acquisendo e rilasciando i lock dei semafori opportunamente.

Fase 3: Operazioni da svolgere

RecordingsList Registra (int channel,Info coming);

E' la funzione che si occupa di gestire i compiti da svolgere quando il client chiede di effettuare un'operazione "REG". Prende in input la variabile che contiene l'identificativo del client e un record che contiene tutte le informazioni necessarie per svolgere il lavoro. La funzione verifica innanzitutto che il voto sia compreso tra 18 e 30 per poterlo inserire. Se si supera questo controllo allora la funzione si occupa di scorrere tutta la lista per controllare se tale esame, associato a tale matricola, è già presente nella lista o meno. Se il voto è corretto e il nodo non è presente, allora lo inseriamo e inviamo al client un messaggio di successo. In tutti gli altri casi, invece inviamo al client un messaggio di operazione fallita, senza inserire il nodo nella lista. In ogni caso, ritorniamo la testa della lista.

VoteSumList SommaVotiAnno (VoteSumList L , RecordingsList R , uint16_t matr , uint16_t year);

E' la funzione che si occupa di gestire i compiti da svolgere quando il client chiede di effettuare un'operazione "SVA". Questa funzione si occupa di sommare tutti i voti dello stesso anno relativi ad una matricola, salvandoli localmente. Alla prossima chiamata della funzione con la stessa matricola, la nuova somma sarà sommata alla precedente calcolata se il nodo è già presente, altrimenti crea un nuovo nodo e lo inserisce in lista. Ovviamente, se il nodo è appena stato creato, la precedente somma sarà 0. Ritorna la testa della lista.

void InviaSommaVoti (int channel , VoteSumList L , uint16_t matr);

E' la funzione che si occupa di gestire i compiti da svolgere quando il client chiede di effettuare un'operazione "IMV". Invia al client la somma dei voti calcolata dalla funzione precedente, relativa alla matricola passata in input. La funzione si occupa di scorrere la lista per verificare se la matricola è presente. Se lo è, salva il valore in una variabile, altrimenti passerà al client 0. A questo punto si occupa di prendere la variabile in cui è contenuto il risultato, formattarlo per bene in una stringa e di inviarlo al client in modo che possa visualizzarlo correttamente a video.

void CalcolaeInviaSommaVoti (int channel , RecordingsList R , uint16_t matr);

E' la funzione che si occupa di gestire i compiti da svolgere quando il client chiede di effettuare un'operazione "CIV". La funzione scorre tutta la lista in cui sono contenuti tutti gli esami registrati e manda al client la somma di tutti gli esami relativa alla matricola passata in input. Se la matricola non è presente al client sarà inviato 0. A questo punto la procedura formatta il valore in una stringa e lo invia al client in modo che possa visualizzarlo correttamente a video.

int Close(int channel);

E' la funzione che si occupa di gestire i compiti da svolgere quando il client chiede di effettuare un'operazione "CLO". La funzione si occupa di chiudere il canale di comunicazione con il client.

void Sleep (int channel, uint16_t time);

E' la funzione che si occupa di gestire i compiti da svolgere quando il client chiede di effettuare un'operazione "SLE". La funzione si occupa di mettere in pausa il processo per time secondi.

Fase 4: Scrittura su file

void PrinttoFile (RecordingsList R , char file_name[]);

Questa funzione viene chiamata ogni volta che un client termina le operazioni da eseguire. E' la funzione che da il via alla FASE 4 ed è anche l'unica a farne parte. La funzione si occupa di aprire il file in scrittura e

ricopiare all'interno di esso tutti gli esami registrati facendo attenzione alla formattazione del documento. Una volta ricopiata l'intera lista, chiude il file. A questo punto la "vita" di un client può ritenersi conclusa.

CLIENT:

Strutture dati e variabili globali:

int num_oper	Numero di operazioni lette.

struct TInfo {
 	
char operation[4];
uint16_t matr;
char exam[5];
uint16_t year;
uint16_t vote;
uint16_t time;
};
	Struttura che contiene le informazioni che riguardano le operazioni.

struct TList {
 	
Info i;
struct TList* next;
};
	Struttura di tipo lista con il campo informazioni relativo alla struttura descritta in precedenza.

Come successo già per il server, anche in questo caso la prima cosa che si fa è effettuare gli opportuni controlli per quanto riguarda i parametri in ingresso.

A questo punto si stampa a video un messaggio di avvenuta connessione e si procede con la seguente funzione:

List ReadToFile(List L,char file_name[]);

È la funzione principale per quanto riguarda il lato client. Si occupa di aprire il file in sola lettura e di decodificare le varie operazioni scritte su di esso, inserendole successivamente in un nodo. Nella prima fase della lettura la funzione si occuperà di riconoscere il tipo di operazione. A questo punto lancia la giusta routine, a seconda del tipo di operazione, che si occuperà di caricare il nodo correttamente, facendo bene attenzione ai campi da riempire e alla formattazione del documento. Una volta riempito, il nodo verrà inserito in coda ad una lista. La procedura continua finchè c'è qualcosa da leggere sul file, aggiornando anche il parametro "num_oper" volta per volta. Una volta finita la lettura , chiuderà il file e ritornerà la lista al main.

Una volta terminata questa fase, il client si occuperà di inizializzare tutti i parametri necessari alla connessione. Solo successivamente, inizierà a scorrere la lista ed inviare le operazioni al server. Una volta inviata una operazione, il client rimane "in attesa" di una risposta dal server sull'esito

della operazione. A questo punto si procede con l'invio della seconda operazione da effettuare. Per chiarezza, il client stamperà a video l'operazione che ha appena inviato al server facendosi aiutare dalla seguente procedura:

void PrintBuffer (int indice , Info i) ;

Questa procedura non fa altro che occuparsi di formattare una stringa con tutte le informazioni relative alla prossima operazione da effettuare e di mandarla a video.

Una volta terminato tutto l'elenco di operazioni da effettuare e aver ricevuto risposta per ognuna di loro, il client chiude il canale di comunicazione e termina.

3- Protocollo per le comunicazioni

Client to server

Il client, una volta stabilita la connessione con il server, invia innanzitutto un valore che indica il numero di operazioni che vorrà effettuare. Successivamente, inizia ad inviare la prima operazione che il server dovrà compiere. Solo dopo aver ricevuto dal server un messaggio di conferma, il client procederà ad inviare la successiva operazione da svolgere. Questo ciclo termina soltanto quando il client ha inviato l'ultima operazione al server e riceve risposta dal server.

Server to client

L'unica fase in cui il server comunica con i client è durante la FASE 2.
Durante questa fase, il server, una volta stabilita la connessione con il client, innanzitutto si occupa di accettare un nuovo client in attesa e di lasciare la sua gestione ad un nuovo thread. A questo punto il thread si occupa di leggere dal canale di comunicazione il numero di operazioni che il client vuole leggere, salvandolo in una variabile. Successivamente, inizia a leggere quella che sarà la prossima operazione da compiere, decodificandola e salvando tutte le informazioni ricevute dal client negli appositi campi di un record. Il server riconosce l'operazione e la gestisce. Solo dopo aver avvertito il client dell'esito dell'operazione, si prepara a leggere e decodificare la prossima operazione in entrata. Le comunicazioni tra un client ed il server vengono interrotte quando le operazioni da effettuare terminano oppure quando il server riceve un'operazione "CLO".

4- Dettagli implementativi

- *Struct Info* e comunicazione:
  La struttura Info è definita come segue:

```
struct TInfo {

        char operation[4];
        uint16_t matr;
        char exam[5];
        uint16_t year;
        uint16_t vote;
        uint16_t time;
};
```

  I campi della struttura Info (ma anche quelli della struttura Recordings e VoteSum) e gli altri interi inviati/ricevuti tramite socket sono di tipo uint16_t perché indipendenti dall'architettura della macchina.

- Terminazione del thread *Connect*:
  Durante la Fase 2 attendiamo la terminazione del thread *Connect* utilizzando la system call *pthread_join* in modo tale che l'intero processo non termini mai prima di aver completato tutte le operazioni e fare in modo che nuovi client possano connettersi in qualsiasi momento.

- Utilizzo di mutex:
  Dato che due o più client non possono scrivere sulle liste e sul file, abbiamo scelto di utilizzare questi 3 mutex (*pthread_mutex_t print_lock, pthread_mutex_t registro_lock, pthread_mutex_t sva_lock*) per risolvere tutti i problemi relativi alla sincronizzazione dei thread. Questi semafori fanno in modo che soltanto un client per volta possa accedere alle rispettive zone critiche.

4- Codice

Server.c

```
#include "AuxFunctions.h"
#include "server.h"

//Parametri
int listen_port; // Porta d'ascolto.
int sock; // Socket.
char file_name[30]; // Nome del file.
RecordingsList Registro = NULL; // Registro in cui salviamo tutte le operazioni.
VoteSumList SommaVoti = NULL;  // Lista in cui salviamo tutte le somme dei voti.
//Mutex
pthread_mutex_t print_lock = PTHREAD_MUTEX_INITIALIZER; // Mutex per la stampa su file.
pthread_mutex_t registro_lock = PTHREAD_MUTEX_INITIALIZER; // Mutex per l'operazione REG.
pthread_mutex_t sva_lock = PTHREAD_MUTEX_INITIALIZER; // Mutex per l'operazione SVA.

int main( int argc, char* argv[] ){
          char buff[60]; // Variabile per stampa a video.

          //Controllo dei parametri del programma
          if (argc < 3 ) PrintErrClose(ERR_NUM);
          if ( !IsUInteger(argv[1]) ) PrintErrClose(ERR_PORT);

          //Assegnazione dei parametri a variabili globali
          listen_port = atoi(argv[1]);
          strcpy(file_name,argv[2]);

          /*INIZIO PROGRAMMA*/
          CLEARSCREEN;
          sprintf(buff , "Porta: %d\n", listen_port );
          write(1, buff , strlen(buff));
          Registro = ReadToFile(Registro,file_name); //Inizio Fase 1 - Lettura operazioni da file.

          Connection_creation();       // Inizio Fase 2 - Creazione e connessione thread.

return 0;
}

/* FUNZIONI */

//FASE 1 - Lettura operazioni da file.


/*Questa funzione si occupa di inizializzare un nuovo nodo della lista. Prende in input le informazioni necessarie e restituisce una
nuova
occorrenza della lista.*/
RecordingsList initNodeListR(Recordings r) {
    RecordingsList L = (RecordingsList)malloc(sizeof(struct TListRecordings));
    L->r = r;
    L->next = NULL;
    return L;
}



/*Questa funzione si occupa di appendere un nodo direttamente alla fine della lista;
Ritorna la nuova lista. */
RecordingsList appendNodeListR(RecordingsList L, Recordings r) {
    if (L != NULL) {
        L->next = appendNodeListR(L->next,r);
    } else {
        L = initNodeListR(r);
    }
    return L;
}

/*Questa procedura libera tutta la memoria occupata dalla lista*/
void freeListR(RecordingsList L) {
    if (L != NULL) {
        freeListR(L->next);
        free(L);
    }
}

/*Questa funzione si occupa di leggere dal file tutti gli esami già registrati. Prende in input una lista in cui verranno inseriti
gli esami registrati ed il nome del file dal quale prelevarli. Ritorna la testa della lista.*/
RecordingsList ReadToFile(RecordingsList L,char file_name[]){

          char matr[5];
          int file_descriptor,read_value;
          Recordings information = (Recordings)malloc(sizeof(struct TRecordings));


          /* Apertura file */
          file_descriptor = open(file_name,O_RDONLY|O_APPEND,S_IRWXU|S_IRWXG|S_IRWXO);
          if( file_descriptor == -1 ) PrintErrClose(ERR_FILE);

        /*Lettura file*/
          read_value = read(file_descriptor,matr,4*sizeof(char));
          matr[4] = '\0';

          while(read_value != 0 && matr[0]!='\n'){ // Continua finchè non ci sono più caratteri da leggere.
                    if(read_value == -1)
                       PrintErrClose(ERR_READ);

                    L = Operation(file_descriptor,L,matr);
                    lseek(file_descriptor, (off_t)1, SEEK_CUR);  //Salta il carattere fine riga;
```

```
        /*Lettura operazione successiva*/
          read_value = read(file_descriptor,matr,4*sizeof(char));
          matr[4] = '\0';   }

          close(file_descriptor); //Chiudi file.
return L;
}

/*Questa funzione si occupa di leggere dal file le informazioni che riguardano l'esame di una determinata matricola. Ritorna la testa
della lista.*/
RecordingsList Operation(int file_descriptor,RecordingsList L,char matr[]){

        /*Allocazione struttura*/
          Recordings information = (Recordings)malloc(sizeof(struct TRecordings));
        information->matr = atoi(matr);

          char *string=malloc(dim*sizeof(char)); //Stringa acquisita;

          /*Riempimento struttura*/
                  lseek(file_descriptor, (off_t)1, SEEK_CUR);  //Salta i ":";
                    if(read(file_descriptor,information->exam,4*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci esame.
                   information->exam[4] = '\0';
                    lseek(file_descriptor, (off_t)1, SEEK_CUR); //Salta i ":";
                    if(read(file_descriptor,string,1*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci anno accademico.
                  string[1] = '\0';
                information->year = atoi(string);
                    lseek(file_descriptor, (off_t)1, SEEK_CUR); //Salta i ":";
                    if(read(file_descriptor,string,2*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci voto;
                  string[2] = '\0';
                    information->vote = atoi(string);


        L = appendNodeListR(L,information); //Inserisci in lista
return L;
}



//FASE 2 - Creazione connessione e thread.

/*Questa procedura si occupa di creare il thread che gestirà tutte le connessioni.*/
void Connection_creation(){

          pthread_t tid_connect[1];
          pthread_create( tid_connect , NULL , Connect, NULL);
          pthread_join( tid_connect[0] , NULL);
}

/*quesat procedura viene lanciata alla creazione di un nuovo thread nella funzione precedente e si occupa di inizializzare tutti i
parametri necessari per la connessione e di accettare nuovi client creando un thread ogni volta che se ne connette uno.*/
void *Connect(void* arg){

          int work = 1; // Condizione di uscita del ciclo.
          int *channel; // Identificatore del client.
          pthread_t tid[1]; // Tid del thread.
          struct sockaddr_in info; // Struttura nella quale salviamo le informazioni per la connessione.
          /*Inizio connessione.*/
          sock = socket(AF_INET,SOCK_STREAM,0);

          if (sock == -1 ) PrintErrClose(ERR_SOCKET);

          info.sin_family = AF_INET;
          info.sin_port = htons( listen_port );
          info.sin_addr.s_addr = INADDR_ANY;

          if( bind(sock, (struct sockaddr*)&info, sizeof(info)) != 0) PrintErrClose(ERR_BIND);
          if ( listen(sock,70) != 0) PrintErrClose(ERR_LISTEN);

          /*Inizio accettazione nuovi client*/
          while(work == 1){
                    channel = malloc(sizeof(int));
                    write(1,CLIENT_WAIT,strlen(CLIENT_WAIT)*sizeof(char));
                    (*channel) = accept(sock,NULL,NULL); // Nuovo client accettato.
                    pthread_create( tid , NULL , NewClient, channel);  // Creazione nuovo thread per gestione del client.
                    write(1,CLIENT_ACCEPT,strlen(CLIENT_ACCEPT)*sizeof(char));


}
          close(sock); // Chiusura della connessione.
}


/* Questa procedura viene lanciata ogni volta che si connette un client. Si occupa di leggere e completare le operazioni inviate dal
client. Prima di terminare effettua una scrittura su file.*/

void *NewClient(int *param){
          int channel=(*param); // Identificatore del client.
          int num_oper; // Numero operazioni.
          Info coming = (Info)malloc(sizeof(Info)); // Salviamo le informazioni lette dal client.
          char temp[5]; // Variabile temporanea.

                    read(channel,temp,2*sizeof(char)); // Lettura del numero di operazioni dal client.
                    temp[2] = '\0';
                    num_oper = atoi(temp); // Conversione da stringa ad intero.

                    do{
                              // Inizio lettura informazioni dal client.
                              read(channel,coming->operation,3*sizeof(char));
                              read(channel,temp,4*sizeof(char));
                              temp[4] = '\0';
                              coming->matr = atoi(temp);
```

```
                        read(channel,coming->exam,4*sizeof(char));
                        read(channel,temp,1*sizeof(char));
                        temp[1] = '\0';
                        coming->year = atoi(temp);
                        read(channel,temp,2*sizeof(char));
                        temp[2] = '\0';
                        coming->vote = atoi(temp);
                        read(channel,temp,1*sizeof(char));
                        temp[1] = '\0';
                        coming->time = atoi(temp);


                        if(strcmp(coming->operation,"REG") == 0){  //E' un'operazione "REG".
                                 pthread_mutex_lock(&registro_lock); // Blocca il mutex.
                                 Registro = Registra(channel,coming);
                                 pthread_mutex_unlock(&registro_lock); // Rilascia il mutex.
                        }

                        else if(strcmp(coming->operation,"SVA") == 0){ //E' un'operazione "SVA".
                                 pthread_mutex_lock(&sva_lock); // Blocca il mutex.
                                 SommaVoti = SommaVotiAnno(SommaVoti,Registro,coming->matr,coming->year);
                                 pthread_mutex_unlock(&sva_lock); // Rilascia il mutex.
                                 write(channel,OPERAZ_OK,strlen(OPERAZ_OK));
                                 }


                        else if(strcmp(coming->operation,"IMV") == 0) //E' un'operazione "IMV".
                                 InviaSommaVoti(channel,SommaVoti,coming->matr);


                        else if(strcmp(coming->operation,"CIV") == 0) //E' un'operazione "CIV".
                                 CalcolaeInviaSommaVoti(channel,Registro,coming->matr);

                        else if(strcmp(coming->operation,"CLO") == 0){ //E' un'operazione "CLO".
                                 num_oper = 0; // Setta a 0 il numero di operazioni ancora da effettuare.
                                 write(channel,OPERAZ_OK,strlen(OPERAZ_OK)); }

                        else if(strcmp(coming->operation,"SLE") == 0) //E' un'operazione "SLE".
                                 Sleep(channel,coming->time);

                if(num_oper != 0) // Se numero operazioni è diverso da 0.
        num_oper--; // Decrementa numero operazioni.

                }while(num_oper !=0); // Contiua finchè non sono state completate tutte le operazioni.


                Close(channel); // Chiude il canale di comunicazione con il client.
                pthread_mutex_lock(&print_lock); // Blocca il mutex.
                PrinttoFile (Registro,file_name); // Scrittura su file.
                pthread_mutex_unlock(&print_lock); // Sblocca il mutex.

        }

//FASE 3 - Operazioni da svolgere.

// OPERAZIONE REG. Questa procedura si occupa di registrare nella lista un nuovo esame inviato dal client. Ritorna la testa della
lista.
RecordingsList Registra (int channel,Info coming){

char buff[100];
int flag = 0;
Recordings info = (Recordings)malloc(sizeof(Recordings)); // Allocazione nuovo nodo Recordings.
/*Inizializzazione nodo con informazioni provenienti dal client.*/
info->matr = coming->matr;
strcpy(info->exam,coming->exam);
info->vote = coming->vote;
info->year = coming->year;
RecordingsList temp=Registro; // Puntatore per scorrere la lista.

        if(info->vote < 18 || info->vote >30) // Se il voto non è corretto.
                flag = 1; // Setta flag a 1.
        while(temp != NULL && flag == 0){ // Scorri la lista per controllare che l'esame che si vuole registrare non sia già stato
inserito.
                if(temp->r->matr == coming->matr && strcmp(coming->exam,temp->r->exam) == 0 ){ // Esame già registrato.
                         flag = 1; // Setta flag a 1.
                          }
                else
                  temp = temp->next; // Altrimenti continua a scorrere la lista.
        }
    if(flag == 0){ // Esame può essere registrato.
    Registro = appendNodeListR(Registro,info); // Aggiungi nodo alla lista.
                write(channel,OPERAZ_OK,strlen(OPERAZ_OK));
    }
    else write(channel,ERR_REG,strlen(ERR_REG)); // Esame già presente, operazione fallita.

return Registro;
}

// OPERAZIONE SVA.

/*Questa funzione si occupa di inizializzare un nuovo nodo della lista. Prende in input le informazioni necessarie e restituisce una
nuova
occorrenza della lista.*/
VoteSumList initNodeListS(VoteSum i) {
    VoteSumList L = (VoteSumList)malloc(sizeof(struct TVoteSumList));
    L->i = i;
    L->next = NULL;
    return L;}



/*Questa funzione si occupa di appendere un nodo direttamente alla fine della lista;
Ritorna la nuova lista. */
VoteSumList appendNodeListS (VoteSumList L, VoteSum i) {
```

```
    if (L != NULL) {
        L->next = appendNodeListS(L->next,i);
    } else {
        L = initNodeListS(i);
    }
    return L;
}

/*Questa procedura libera tutta la memoria occupata dalla lista*/
void freeListS (VoteSumList L) {
    if (L != NULL) {
        freeListS(L->next);
        free(L);
    }
}


/*Questa funzione si occupa di sommare tutti i voti dello stesso anno relativi ad una matricola,salvandoli localmente. Alla prossima
chiamata della funzione con la stessa matricola, la nuova somma sarà sommata alla precedente calcolata.*/
VoteSumList SommaVotiAnno ( VoteSumList L , RecordingsList R , uint16_t matr , uint16_t year ){

RecordingsList Rtemp=R; // Puntatore per scorrere la lista.
uint16_t sommavoti=0; // Somma.
VoteSum somma=malloc(sizeof(VoteSum)); // Allocazione nodo.

          while (Rtemp != NULL ){ // Finchè ci sono elementi.
                    if ( Rtemp->r->matr == matr && Rtemp->r->year == year ) // Se l'esame è della matricola cercata e dell'anno
richiesto.
                              sommavoti=sommavoti+ Rtemp->r->vote; // Somma.
                    Rtemp=Rtemp->next; // Vai al successivo.
                                        }

          /*Inizializzazione nodo.*/
          somma->matr=matr;
          somma->sum=sommavoti;
          L=ScorriLista ( L, somma); // Si occupa di inserire i dati nella lista.
return L;
}


/*Questa funzione si occupa di cercare se la matricola è già presente nella lista. In tal caso addiziona la somma precedente con
quella attuale. Altrimenti inserisce il nodo in coda alla lista. Ritorna la testa della lista.*/
VoteSumList ScorriLista ( VoteSumList L ,VoteSum somma ){
          if( L != NULL){
                    if ( L->i->matr == somma->matr )
                              L->i->sum = L->i->sum + somma->sum;
                    else
                              L->next=ScorriLista(L->next,somma); }
          else
                    L=appendNodeListS(L,somma);
return L;
}

// OPERAZIONE IMV. Questa funzione si occupa di inviare al client la somma dei voti calcolata in precedenza per una data matricola.

void InviaSommaVoti ( int channel , VoteSumList L , uint16_t matr){
VoteSumList Ltemp=L; // Puntatore per scorrere la lista.
int flag = 0;
uint16_t risultato = 0;
char buffer [50];

          while ( Ltemp != NULL && flag == 0) { // Finchè ci sono elementi oppure non hai ancora trovato la matricola.
                    if ( Ltemp->i->matr == matr ) { // Matricola trovata.
                              flag=1; // Setta flag a 1.
                              risultato = Ltemp->i->sum; // Salva la somma da inviare.
                                                  }
                    else
                              Ltemp=Ltemp->next;  // Altrimenti continua a scorrere.
                                                 }


          sprintf(buffer, "[SERVER] Matr:%d. Somma:%d\n\n" ,matr, risultato);
          buffer[49]='\0';


          write (channel,buffer,strlen(buffer)); // Invio al client.



}

//OPERAZIONE CIV. Questa funzione di occupa di sommare ed inviare al client tutti i voti per una data matricola.

void CalcolaeInviaSommaVoti ( int channel , RecordingsList R , uint16_t matr ){

RecordingsList Rtemp=R; // Puntatore per scorrere la lista.
uint16_t sommavoti=0;
char buffer[50];

          while (Rtemp != NULL ){ // Finchè ci sono elementi.
                    if ( Rtemp->r->matr == matr ) // Se la matricola è quella cercata.
                              sommavoti=sommavoti+ Rtemp->r->vote; // Aggiorna la somma.
                    Rtemp=Rtemp->next; // Contiua a scorrere.
                                        }
          sprintf(buffer, "[SERVER] Matr:%d. Somma:%d\n\n" ,matr, sommavoti);
          buffer[49]='\0';


          write (channel,buffer,strlen(buffer)); // Invio al client.

}

//OPERAZIONE CLO.
```

```
int Close( int channel ){

          close(channel);
}

//OPERAZIONE SLE.

void Sleep ( int channel, uint16_t time ){

          sleep(time);
          write(channel,OPERAZ_OK,strlen(OPERAZ_OK));
}

//FASE 4 - Scrittura su file.
/*Questa funzione si occupa di scrivere su file tutti gli esami presenti sulla lista.*/
void PrinttoFile ( RecordingsList R , char file_name[]){

RecordingsList R_temp=R; // Puntatore per scorrere la lista.
char temp[20];
int file_descriptor;

          /* Apertura file */
          file_descriptor = open(file_name,O_WRONLY|O_TRUNC,S_IRWXU|S_IRWXG|S_IRWXO);
          if( file_descriptor == -1 ) PrintErrClose(ERR_FILE);

          while(R_temp != NULL){ // Finchè ci sono elementi.

          sprintf(temp,"%d:%s:%d:%d\n",R_temp->r->matr,R_temp->r->exam,R_temp->r->year,R_temp->r->vote);
          temp[15]='\0';
          write(file_descriptor,temp,strlen(temp)); // Scrivi su file.
          R_temp=R_temp->next; //Scorri al successivo.
                                    }

          close(file_descriptor); //Chiudi il file.
                                                                        }
```

Server.h

```
#include <pthread.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

#include <sys/types.h>

#include <arpa/inet.h>

#include <signal.h>

#include <fcntl.h>


#define CLIENT_WAIT  "In attesa di un client.\n\n"

#define CLIENT_ACCEPT "Nuovo client accettato.\n"

#define ERR_REG "[SERVER] Operazione fallita.\n\n"

#define OPERAZ_OK "[SERVER] Operazione eseguita.\n\n"

/* Struttura Recordings */

//Struttura utilizzata per salvare informazioni lette dal file.

struct TRecordings {


   uint16_t matr;

   char exam[5];

   uint16_t year;

   uint16_t vote;


};


typedef struct TRecordings* Recordings;



/* Lista Recordings*/

//Struttura utilizzate per creare una lista in cui verranno salvate le informazioni lette da file.

struct TListRecordings {
```

```
    Recordings r;
    struct TListRecordings* next;
};

typedef struct TListRecordings* RecordingsList;

/*Struttura VoteSum*/
//Struttura utilizzata per salvare la somma dei voti di una matricola.
struct TVoteSum {
          uint16_t matr;
          uint16_t sum;
};

typedef struct TVoteSum* VoteSum;

/* Lista VoteSum */
// Struttura utilizzata per creare una lista i cui verranno salvate le somme dei voti delle matricole.
struct TVoteSumList {

          VoteSum i;
          struct TVoteSumList* next;
};

typedef struct TVoteSumList* VoteSumList;



//FASE 1 - Lettura operazioni da file.
RecordingsList initNodeListR(Recordings r); // Inizializza un nuovo nodo Recordings.
RecordingsList appendNodeListR(RecordingsList L, Recordings r);  // Aggiunge un nodo alla fine della lista controllandone
l'esistenza. La funzione ritorna sempre la testa della lista.
void freeListR(RecordingsList L); // Dealloca la lista interamente.
RecordingsList ReadToFile(RecordingsList L,char file_name[]); // Legge da file i voti già registrati, ritorna un puntatore alla
lista.
RecordingsList Operation(int file_descriptor,RecordingsList L,char matr[]); // Legge da file il voto relativo alla matricola passata.
//FASE 2 - Creazione connessione e thread.
void Connection_creation(); // Crea un thread che si occupa della connessione.
void *Connect(void*); // Procedura lanciata da un thread che si occupa di gestire i client che si connettono.
void *NewClient(); // Procedura lanciata da un thread che si occupa di effettuare tutte le operazioni richieste da un client.
//FASE 3 - Operazioni da svolgere.
// OPERAZIONE REG.
RecordingsList Registra (int channel,Info coming); // Registra sulla lista un nuovo voto.
// OPERAZIONE SVA.
void freeListS (VoteSumList L); // Dealloca la lista interamente.
VoteSumList appendNodeListS (VoteSumList L, VoteSum i); // Aggiunge un nodo alla fine della lista controllandone l'esistenza. La
funzione ritorna la testa della lista.
VoteSumList initNodeListS(VoteSum i); // Inizilizza un nuovo nodo VoteSum.
VoteSumList ScorriLista ( VoteSumList L ,VoteSum somma ); // Cerca nella lista VoteSum se un nodo è presente,altrimenti lo aggiunge.
VoteSumList SommaVotiAnno ( VoteSumList L , RecordingsList R , uint16_t matr , uint16_t year ); // Somma i voti di una matricola
relativi ad un anno e li inserisce nella lista VoteSumList.
//OPERAZIONE IMV.
void InviaSommaVoti ( int channel , VoteSumList L , uint16_t matr); // Invia al client la somma dei voti calcolata dalla funzione
precedente.
```

```
//OPERAZIONE CIV.

void CalcolaeInviaSommaVoti ( int channel , RecordingsList R , uint16_t matr ); // Calcola ed invia al client la somma di tutti i
voti di una matricola.

//OPERAZIONE CLO.

int Close( int channel ); // Chiude il collegamento con il client.

//OPERAZIONE SLE.

void Sleep ( int channel, uint16_t time ); // Mette il processo in attesa per "time" secondi.

// FASE 4 - Scrittura su File.

void PrinttoFile ( RecordingsList R , char file_name[]); // Scrive su file tutte le operazioni presenti sulla lista RecordingList.
```

Client.c

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include "AuxFunctions.h"


/*Parametri*/
int num_oper = 0;

List ReadToFile(List L,char file_name[]);  //Legge da file.
List OperationReg(int file_descriptor,List L,char operation[]);  //Gestisce l'operazione "REG".
List OperationSva(int file_descriptor,List L,char operation[]);  //Gestisce l'operazione "SVA".
List OperationImvAndCiv(int file_descriptor,List L,char operation[]); //Gestisce le operazioni "IMV" e "CIV".
List OperationSle(int file_descriptor,List L,char operation[]); //Gestisce l'operazione "SLE".
List OperationClo(int file_descriptor,List L,char operation[]); //Gestisce l'operazione "CLO".
void PrintBuffer ( int indice , Info i );
int main(int argc,char* argv[]){

          List L = NULL;  //Lista in cui andranno salvate informazioni raccolte dal file.
          char buff_server[100];
          char temp[5];
          int i = 1;

          struct sockaddr_in mio_ind;

         /* Controlli sui parametri in ingresso */
        if(argc<4)  PrintErrClose(ERR_NUM); //Numero parametri in ingresso;
        if( inet_aton(argv[1], NULL)==0 ) PrintErrClose(ERR_IP);  //Indirizzo IP;
        if ( !IsUInteger(argv[2]) ) PrintErrClose(ERR_PORT);   //Numero porta;

        /* Lettura da file */
          L = ReadToFile(L,argv[3]);

          /*Creazione del socket*/
          mio_ind.sin_family=AF_INET;
          mio_ind.sin_port = htons(atoi(argv[2]));
          inet_aton(argv[1],&mio_ind.sin_addr);

          int sockcl=socket(AF_INET,SOCK_STREAM,0);
          if (sockcl<0) PrintErrClose(ERR_SOCKET);

          CLEARSCREEN;
          /*Connessione al server*/
          int conn=connect(sockcl,(struct sockaddr *) &mio_ind, sizeof(mio_ind));
          if (conn<0) PrintErrClose(ERR_CONNECT);
        write(1,CONNECT_OK,strlen(CONNECT_OK)*sizeof(char));
          sleep(2);


          /*Invio al server*/
          sprintf(temp,"%d",num_oper);
                    temp[2] = '\0';
                    write(sockcl,temp,2*sizeof(char));
          while(L != NULL) {
                    write(sockcl,L->i->operation,3*sizeof(char));
                    sprintf(temp,"%d",L->i->matr);
                    temp[4] = '\0';
                    write(sockcl,temp,4*sizeof(char));
                    write(sockcl,L->i->exam,4*sizeof(char));
                    sprintf(temp,"%d",L->i->year);
                    temp[1] = '\0';
                    write(sockcl,temp,1*sizeof(char));
                    sprintf(temp,"%d",L->i->vote);
                    temp[2] = '\0';
                    write(sockcl,temp,2*sizeof(char));
                    sprintf(temp,"%d",L->i->time);
                    temp[2] = '\0';
                    write(sockcl,temp,1*sizeof(char));
                    PrintBuffer(i,L->i);
                    read(sockcl,buff_server,31*sizeof(char));
                    sleep(2);
                    write(1,buff_server,strlen(buff_server));

                    L = L->next;
                    i++;
```

```
          }

          //Terminare (chiude il socket)
          close(sockcl);

          freeList(L);

return 0;
}



/*Questa funzione prende in input una lista ed una stringa contenente il nome di un file. Si occupa di aprire il file(se esiste)
in sola lettura. Legge i primi 3 caratteri per riconoscere l'operazione da eseguire e lancia la routine che la riguarda. Termina
alla fine del file,chiudendolo e ritornando la lista al chiamante.*/
List ReadToFile(List L,char file_name[]){

          char operation[4];
          int file_descriptor,read_value;

          /* Apertura file */
          file_descriptor = open(file_name,O_RDONLY|O_APPEND,S_IRWXU|S_IRWXG|S_IRWXO);
          if( file_descriptor == -1 ) PrintErrClose(ERR_FILE);

        /*Lettura file*/
          read_value = read(file_descriptor,operation,3*sizeof(char));
          operation[3] = '\0';

          while(read_value != 0){
                    if(read_value == -1)
                       PrintErrClose(ERR_READ);

        if(strcmp(operation,"REG") == 0)  //E' un'operazione "REG".
             L = OperationReg(file_descriptor,L,operation);

          else if(strcmp(operation,"SVA") == 0) //E' un'operazione "SVA".
             L = OperationSva(file_descriptor,L,operation);

          else if(strcmp(operation,"IMV") == 0) //E' un'operazione "IMV".
             L = OperationImvAndCiv(file_descriptor,L,operation);

          else if(strcmp(operation,"CIV") == 0) //E' un'operazione "CIV".
           L = OperationImvAndCiv(file_descriptor,L,operation);

          else if(strcmp(operation,"CLO") == 0){ //E' un'operazione "CLO".
             L = OperationClo(file_descriptor,L,operation);
             lseek(file_descriptor, (off_t)1, SEEK_CUR); }

          else if(strcmp(operation,"SLE") == 0) //E' un'operazione "SLE".
             L = OperationSle(file_descriptor,L,operation);

        else PrintErrClose(ERR_OPERATION); //Operazione non riconosciuta.Nessuna delle precedenti.

          lseek(file_descriptor, (off_t)1, SEEK_CUR);  //Salta il carattere fine riga;
          num_oper++;
        /*Lettura operazione successiva*/
          read_value = read(file_descriptor,operation,3*sizeof(char));
        operation[3] = '\0';   }

          close(file_descriptor); //Chiudi file.
return L;
}


/* Funzione di routine per l'operazione "REG". Si occupa di allocare una struttura in grado di contenere le informazioni,aggiungerle
e infine passarle alla lista.*/
List OperationReg(int file_descriptor,List L,char operation[]){

        /*Allocazione struttura*/
          Info information = (Info)malloc(sizeof(struct TInfo));
        strcpy(information->operation,operation);

          char *string=malloc(dim*sizeof(char)); //Stringa acquisita;

          /*Riempimento struttura*/
          lseek(file_descriptor, (off_t)1, SEEK_CUR); //Salta i ":";
                    if(read(file_descriptor,string,4*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci matricola.
                  string[4] = '\0';
                information->matr = atoi(string);
                lseek(file_descriptor, (off_t)1, SEEK_CUR);  //Salta i ":";
                    if(read(file_descriptor,information->exam,4*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci esame.
                   information->exam[4] = '\0';
                    lseek(file_descriptor, (off_t)1, SEEK_CUR); //Salta i ":";
                    if(read(file_descriptor,string,1*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci anno accademico.
                  string[1] = '\0';
                information->year = atoi(string);
                    lseek(file_descriptor, (off_t)1, SEEK_CUR); //Salta i ":";
                    if(read(file_descriptor,string,2*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci voto;
                  string[2] = '\0';
                    information->vote = atoi(string);
          /*Setta valori non utili per questa operazione*/

          information->time = 0;  //Setta time a 0.

        L = appendNodeList(L,information); //Inserisci in lista
return L;
}

/* Funzione di routine per l'operazione "SVA". Si occupa di allocare una struttura in grado di contenere le informazioni,aggiungerle
e infine passarle alla lista.*/
List OperationSva(int file_descriptor,List L,char operation[]){

          /*Allocazione struttura*/
          Info information = (Info)malloc(sizeof(struct TInfo));
```

```
        strcpy(information->operation,operation);

          char *string=malloc(dim*sizeof(char)); //Stringa acquisita;

          /*Riempimento struttura*/
          lseek(file_descriptor, (off_t)1, SEEK_CUR); //Salta i ":";
                   if(read(file_descriptor,string,4*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci matricola.
                 string[4] = '\0';
               information->matr = atoi(string);
               lseek(file_descriptor, (off_t)1, SEEK_CUR);  //Salta i ":";
                   if(read(file_descriptor,string,1*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci anno accademico.
                 string[1] = '\0';
               information->year = atoi(string);

        /*Setta valori non utili per questa operazione*/
          information->vote = 0;
          information->time = 0;
          strcpy(information->exam,"nullo");

        L = appendNodeList(L,information); //Inserisci in lista
return L;
}


/* Funzione di routine per le operazione "IMV" e "CIV". Si occupa di allocare una struttura in grado di contenere le
informazioni,aggiungerle e infine passarle alla lista.*/
List OperationImvAndCiv(int file_descriptor,List L,char operation[]){

          /*Allocazione struttura*/
          Info information = (Info)malloc(sizeof(struct TInfo));
        strcpy(information->operation,operation);

          char *string=malloc(dim*sizeof(char)); //Stringa acquisita;

          /*Riempimento struttura*/
          lseek(file_descriptor, (off_t)1, SEEK_CUR); //Salta i ":";
                   if(read(file_descriptor,string,4*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci matricola.
                 string[4] = '\0';
               information->matr = atoi(string);

          /*Setta valori non utili per questa operazione*/
        information->time = 0;  //Setta time a 0.
          information->vote = 0;
          information->year = 0;
          strcpy(information->exam,"nullo");

        L = appendNodeList(L,information); //Inserisci in lista

return L;
}



/* Funzione di routine per l'operazione "CLO". Si occupa di allocare una struttura in grado di contenere le informazioni,aggiungerle
e infine passarle alla lista.*/
List OperationClo(int file_descriptor,List L,char operation[]){

          /*Allocazione struttura*/
          Info information = (Info)malloc(sizeof(struct TInfo));
        strcpy(information->operation,operation);

          /*Setta valori non utili per questa operazione*/
          information->matr = 0;
        information->time = 0;  //Setta time a 0.
          information->vote = 0;
          information->year = 0;
          strcpy(information->exam,"nullo");

          L = appendNodeList(L,information); //Inserisci in lista

}


/* Funzione di routine per l'operazione "SLE". Si occupa di allocare una struttura in grado di contenere le informazioni,aggiungerle
e infine passarle alla lista.*/
List OperationSle(int file_descriptor,List L,char operation[]){

          /*Allocazione struttura*/
          Info information = (Info)malloc(sizeof(struct TInfo));
        strcpy(information->operation,operation);

          char *string=malloc(dim*sizeof(char)); //Stringa acquisita;

          /*Riempimento struttura*/
          lseek(file_descriptor, (off_t)1, SEEK_CUR); //Salta i ":";
                   if(read(file_descriptor,string,1*sizeof(char))==-1)  PrintErrClose(ERR_READ); //Acquisisci matricola.
                 string[1] = '\0';
               information->time = atoi(string);

          /*Setta valori non utili per questa operazione*/
        information->matr = 0;
        information->vote = 0;
          information->year = 0;
          strcpy(information->exam,"nullo");

          L = appendNodeList(L,information); //Inserisci in lista

return L;
}


/*Questa procedura si occupa di formattare e mandare a video una stringa che contiene l'operazione inviata al server.*/
void PrintBuffer ( int indice , Info i ) {
```

```
char buff_client[100];

          if(strcmp(i->operation,"REG") == 0)  //E' un'operazione "REG".
            sprintf(buff_client,"[CLIENT] Operazione %d: REG:%d:%s:%d:%d. Invio Ok!\n",indice,i->matr,i->exam,i->year,i->vote);

          else if(strcmp(i->operation,"SVA") == 0) //E' un'operazione "SVA".
            sprintf(buff_client,"[CLIENT] Operazione %d: SVA:%d:%d. Invio Ok!\n",indice,i->matr,i->year);

          else if(strcmp(i->operation,"IMV") == 0) //E' un'operazione "IMV".
            sprintf(buff_client,"[CLIENT] Operazione %d: IMV:%d. Invio Ok!\n",indice,i->matr);

          else if(strcmp(i->operation,"CIV") == 0) //E' un'operazione "CIV".
           sprintf(buff_client,"[CLIENT] Operazione %d: CIV:%d. Invio Ok!\n",indice,i->matr);

          else if(strcmp(i->operation,"CLO") == 0) //E' un'operazione "CLO".
            sprintf(buff_client,"[CLIENT] Operazione %d: CLO:. Invio Ok!\n",indice);

          else if(strcmp(i->operation,"SLE") == 0) //E' un'operazione "SLE".
            sprintf(buff_client,"[CLIENT] Operazione %d: SLE:%d. Invio Ok!\n",indice,i->time);

          write(1,buff_client,strlen(buff_client));
}
```

AuxFunctions.c

```
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include "AuxFunctions.h"

int IsDigit(char c){
          return (c>='0' && c<='9');
}

/*Questa funzione previene che vengano inseriti dall'utente valori non interi.*/
int IsUInteger(char *s){
          int res=1, i = 0;

          while( s[i] != '\0' && IsDigit(s[i])) i++;

          if(i==0 || s[i]!='\0') res = 0;

          return res;
}


/*Questa funzione si occupa di stampare su standard output un messaggio relativo ad un errore.*/
void PrintErrClose(char *s){
          write(2, s , strlen(s));
          exit(1);
}



/*Questa funzione si occupa di inizializzare un nuovo nodo della lista. Prende in input le informazioni necessarie e restituisce una
nuova
occorrenza della lista.*/
List initNodeList(Info i) {
    List L = (List)malloc(sizeof(struct TList));
    L->i = i;
    L->next = NULL;
    return L;
}



/*Questa funzione si occupa di appendere un nodo direttamente alla fine della lista;
Ritorna la nuova lista. */
List appendNodeList(List L, Info i) {
    if (L != NULL) {
        L->next = appendNodeList(L->next,i);
    } else {
        L = initNodeList(i);
    }
    return L;
}


/*Questa procedura libera tutta la memoria occupata dalla lista*/
void freeList(List L) {
    if (L != NULL) {
        freeList(L->next);
        free(L);
    }
}
```

AuxFunctions.h

```
#include <stdint.h>
#include <stdio.h>

/* Errori */
#define ERR_NUM "Numero di parametri non sufficiente.\n"
#define ERR_PORT "Il numero di porta deve essere un intero non negativo.\n"
#define ERR_IP "IP non corretto.\n"
#define ERR_SOCKET "Errore creazione Socket.\n"
#define ERR_BIND "Errore Bind.\n"
#define ERR_LISTEN "Errore Listen.\n"
#define ERR_CONNECT "Errore Connect.\n"
#define ERR_FILE "Il file scelto non esiste.\n"
#define ERR_READ "Errore lettura da file.\n"
#define ERR_OPERATION "Errore lettura operazione.\n"
#define CLEARSCREEN system("clear")
#define CONNECT_OK "Connessione OK.\n\n"
#define dim 5

void PrintErrClose(char *s);  //Scrive s su standard output e chiude il programma con exit(1).
int IsUInteger(char *s);      //ritorna 1 se la stringa s e' un intero senza segno, 0 altrimenti.
int IsDigit(char c);


/* Struttura Info*/

struct TInfo {

   char operation[4];
   uint16_t matr;
   char exam[5];
   uint16_t year;
   uint16_t vote;
   uint16_t time;
};

typedef struct TInfo* Info;

/* Lista */

struct TList {

    Info i;
    struct TList* next;
};

typedef struct TList* List;

List initNodeList(Info i); // Inizializza un nuovo nodo
List appendNodeList(List L, Info i);  /* Aggiunge un nodo alla fine della lista controllandone l'esistenza. La funzione ritorna
sempre la testa della lista*/
void freeList(List L); // Dealloca la lista interamente
```