Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Progetto per l'esame di Laboratorio di sistemi operativi

alla cortese attenzione del Prof. Finzi Alberto

Film Manager

Maione Michele, 566002580
Ullah Ross, N86/111

Descrizione progetto

Realizzare un sistema client server che consenta a più utenti di segnalare, commentare, e valutare film. Si utilizzi il linguaggio C su piattaforma UNIX. I processi dovranno comunicare tramite socket TCP. Corredare l'implementazione di adeguata documentazione.

Il sistema deve gestire la valutazione di film con commenti e valutazioni, anche i commentatori possono essere valutati. Ogni film potrà essere valutato con un voto da 1 a 5 e con un commento testuale con un massimo di 120 caratteri. Se più utenti valutano lo stesso film, il film sarà associato alla media delle valutazioni pesate per il voto attuale del valutatore. Ogni commento potrà essere valutato con un voto da 0 a 5 dagli altri utenti e ogni valutatore sarà valutato con un voto pari alla media dei voti ricevuti per commento.

Per accedere al servizio ogni utente dovrà prima registrarsi al sito indicando il proprio nome, il proprio cognome, l'età, l'indirizzo, la password e il nickname. Non c'è un limite a priori al numero di utenti che si possono collegare con il server.

Il client consentirà all'utente di collegarsi ad un server di comunicazione, indicando tramite riga di comando il nome o l'indirizzo IP di tale server e la porta da utilizzare. Una volta collegato ad un server l'utente potrà: registrarsi come nuovo utente o accedere al servizio come utente registrato. Il servizio permetterà all'utente di: vedere la lista dei film valutati con voto medio attuale, leggere le valutazioni di un film, scrivere una propria valutazione di un film, disconnettersi. Ad ogni utente connesso che ha commentato un film verrà notificato l'ingresso di una nuova valutazione per quel film e potrà valutare tale valutazione una sola volta e solo entro un tempo di 10 minuti.

Il server dovrà supportare tutte le funzionalità descritte nella sezione relativa al client. All'avvio del server, sarà possibile specificare tramite riga di comando la porta TCP sulla quale mettersi in ascolto.

Il server sarà di tipo concorrente, ovvero, in grado di servire più client simultaneamente. Durante il suo regolare funzionamento, il server effettuerà il logging delle attività principali in un file apposito. Ad esempio, memorizzando la data è l'ora di connessione dei client e il loro nome simbolico (se disponibile, altrimenti l'indirizzo IP), e la data e l'ora di creazione dei commenti postati.

Sommario

1. Documento di utilizzo

1.1 Installazione

Il progetto è formato da due file eseguibili, **FM_Server** e **FM_Client**, rispettivamente il programma server e il programma client, e da 2 cartelle, la cartella **DATABASE**, che contiene il DB del progetto e che deve trovarsi sul server, e la cartella Icone che contiene le icone del client.

Per installare il programma server basta copiare **FM_Server** e la cartella **DATABASE**.
Per installare il programma client basta copiare **FM_Client** e la cartella **Icone**.

1.2 Avvio

1.2.1 Avvio del programma server

Per avviare il programma server bisogna eseguire **./FM_Server [NumeroPorta]** (NumeroPorta se non utilizzato verrà settato automaticamente a 50000)

1.2.2 Avvio del programma client

Per avviare il programma client bisogna eseguire **./FM_Client**.

2. Documento di progettazione

2.1 Strumenti utilizzati

Il progetto è stato realizzato per sistemi operativi Linux, utilizzando NetBeans IDE 8 per la stesura del codice, e QT Designer 4.8 per la realizzazione delle GUI del client.

2.2 Database

Il progetto utilizza un DB basato sui file.

Nella cartella **DATABASE** verranno inserite, per logica gerarchica, delle cartelle per ogni tipo di struttura dati utilizzata. Per ogni struttura dati che verrà salvata su disco verrà creata, nella cartella del tipo di struttura dati, una cartella con l'ID della chiave della struttura: un guid univoco calcolato tramite il comando Linux **/proc/sys/kernel/random/uuid**; e verrà scritto un file chiamato **Guid.db** che conterrà una singola struttura.

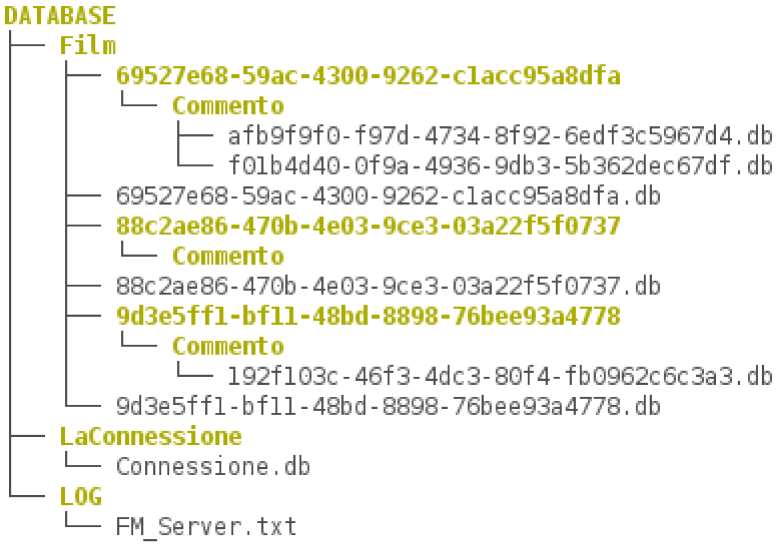

Illustrazione 1: Struttura del DB

2.3 Struttura logica del progetto

Il progetto FM è stato diviso in 3 sotto progetti: FM_Core, FM_Client, FM_Server.

2.3.1 FM_Server

Questo progetto console, utilizza la classe **FM_Core/Classi/NET/TCPServer.c** per mettersi in ascolto sulla porta selezionata e per ogni socket in arrivo genera tramite la funzione di sistema **fork()** un sotto processo che elabora la richiesta.

2.3.2 FM_Client

Questo progetto QT in C++, contiene tutti i file delle GUI per visualizzare e inserire i dati. Per ogni struttura dati utilizzata esistono due finestre: **NomeStruttura_Ricerca** e **NomeStruttura_Dettaglio**. All'avvio dell'applicazione viene creata una connessione TCP al server tramite la classe **FM_Core/Classi/NET/TCPClient.c**, e una volta stabilita, verrà utilizzata per inviare e ricevere i dati dal server.

2.3.3 FM_Core

Questo è il cuore del progetto, infatti contiene tutte le classi che gestiscono le strutture dati utilizzate, l' I/O delle strutture su disco e i trasferimenti TCP dal server al client e viceversa.

FM_Core è diviso, a sua volta, in quattro sotto sotto progetti: DB, FN, IO, NET.

- **DB**
  - La cartella DB contiene le 6 strutture dati utilizzate: **Film**, **Commenti**, **FilmValutazione**, **CommentoValutazione**, **Utente**, **Connessione**, più una classe di funzionalità per la scrittura e lettura su file: **DBf**.
- **FN**
  - La cartella FN contiene 2 classi: **GB** e **Util**, che si occupano di funzionalità generiche, come la copia degli array, la generazione dei guid, le formattazioni, e funzioni per la data e l'ora.
- **IO**
  - Per ogni struttura dati esiste una classe chiamata **IONomeStruttura** che si occupa dell'effettivo salvataggio e recupero della struttura dal disco.
- **NET**
  - La cartella NET è divisa, a sua volta, in 2 sotto cartelle: CLI e SER.
    - In CLI troviamo per ogni struttura una classe chiamata **TCPNomeStruttura** che si occupa dell'invio e della ricezione dei dati dal client al server.
    - In SER per ogni struttura esiste una classe chiamata **SERNomeStruttura** che si occupa di ricevere la richiesta dal client leggere e/o scrivere i dati sul disco tramite **IONomeStruttura** e inviarli al client.

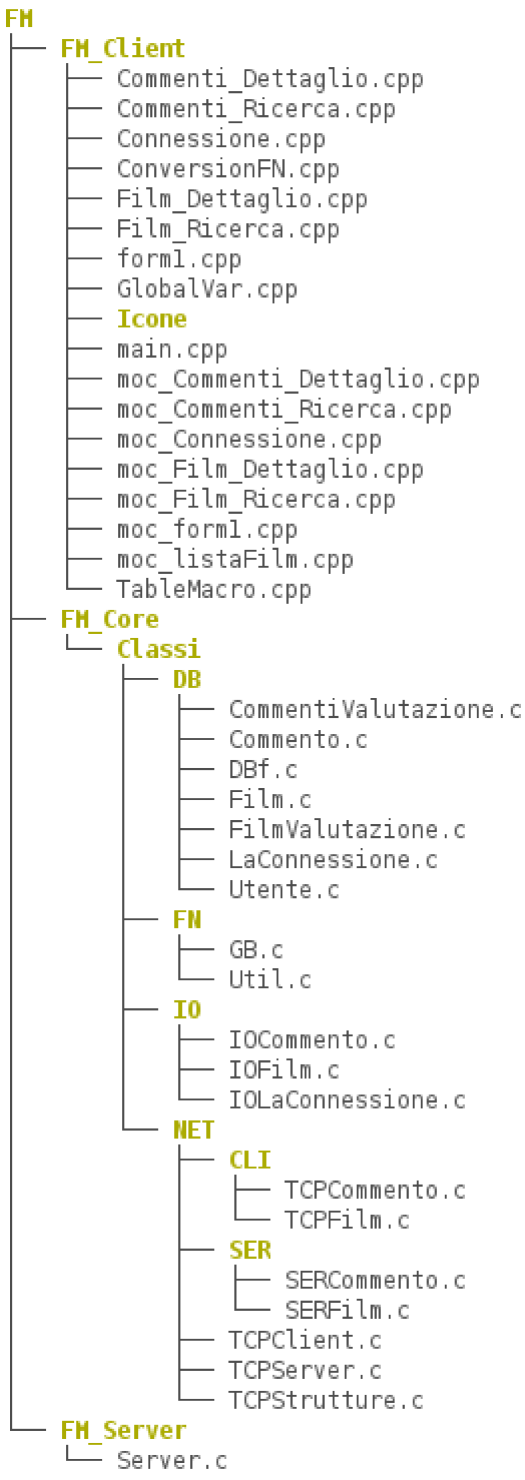

Illustrazione 2: Struttura del progetto

2.4 Specifiche generiche di implementazione delle classi

2.4.1 Generica GUI di ricerca

Una generica classe di ricerca dovrà implementare due funzioni:

```
void Cerca();
void Modifica(bool Nuovo);
```

Tramite la classe **TCPNomeStruttura.c**, faranno richiesta al server di tutti i dati e invieranno al server la struttura per il salvataggio.

2.4.2 Generica struttura dati

Una generica classe dati conterrà la struct e le funzioni di inizializzazione della stessa

```
typedef struct nomeStruttura
{
	int Setted;
	char IDNomeStruttura[LID];
	//Altri dati

	//Audit
	char AUDIT_INS_USER[LUser];
	char AUDIT_UPD_USER[LUser];
	char AUDIT_INS_DATA[LData];
	char AUDIT_UPD_DATA[LData];
} NomeStruttura;
NomeStruttura NomeStruttura_New(const Tipo dati, ...);
NomeStruttura NomeStruttura_Null();
```

2.4.3 Generica classe I/O

Una generica classe di I/O dovrà implementare almeno le seguenti 3 funzioni:

```
int NomeStruttura_Salva(NomeStruttura f);
int NomeStruttura_RicercaTutti(NomeStruttura elementi[LMAXArray]);
NomeStruttura NomeStruttura_ByID(const char id[LID]);
```

che sfruttando le funzioni contenute in **DBf.c** si occuperanno dell'effettivo salvataggio su disco della struttura tramite le funzioni **fopen**, **fclose**, **fwrite** e **fread**, contenute in **stdio.h**.

2.4.4 Generica classe TCP lato client

Mentre una generica classe TCP lato client dovrà implementare per ogni funzione contenuta in I/O la stessa funzione, che si occuperà, tramite le funzioni **send** e **recv**, contenute in **sys/socket.h**, di inviare al server le strutture, esempio:

```
int TCP_NomeStruttura_Salva(const int IDConnessione, NomeStruttura f);
NomeStruttura TCP_NomeStruttura_ByID(const int IDConnessione, const char id[LID]);
int TCP_NomeStruttura_RicercaTutti(const int IDConnessione, NomeStruttura elementi[LMAXArray]);
```

2.4.5 Generica classe TCP lato server

Una generica classe TCP lato server dovrà implementare per ogni funzione contenuta in I/O la stessa funzione, che si occuperà, tramite le funzioni **send** e **recv**, contenute in **sys/socket.h**, di inviare al client le strutture, esempio:

```
int SER_NomeStruttura_Salva(const int IDConnessione);
int SER_NomeStruttura_ByID(const int IDConnessione, const char id[LID]);
int SER_NomeStruttura_RicercaTutti(const int IDConnessione);
```

Bibliografia

- Linux, Guida di Riferimento - Siever, Spainhour, Figgins, ed Hekman - Apogeo
- Programmazione in C - Kim N. King - Apogeo