Raport TopMusic

Retele de calculatoare

Cioata Matei-Alexandru
Grupa A1

January 8, 2019

1 Introducere

Acest raport va urmari prezentarea conceptelor si ideilor utilizate de mine in aplicatia **TopMusic**. Am ales acest proiect deoarece din ce in ce mai multa lume isi formeaza hobby-uri legate de muzica, iar o astfel de aplicatie ar putea avea succes daca este dezvoltata cum trebuie. Avand o idee asemanatoare cu **Spotify**, **TopMusic** ar avea potentialul de a forma o comunitate organizata in care utilizatorii socializeaza despre melodiile lor preferate si despre muzica in general.

Aplicatia TopMusic va avea mai multi clienti conectati la un server, iar actiunile lor vor avea legatura cu un top de melodii. Ei vor putea vota, comenta, citi descrieri despre muzica lor preferata. De asemenea, vor putea accesa melodiile prin intermediul unor link-uri pe care le pot primi de la server.

2 Tehnologiile utilizate

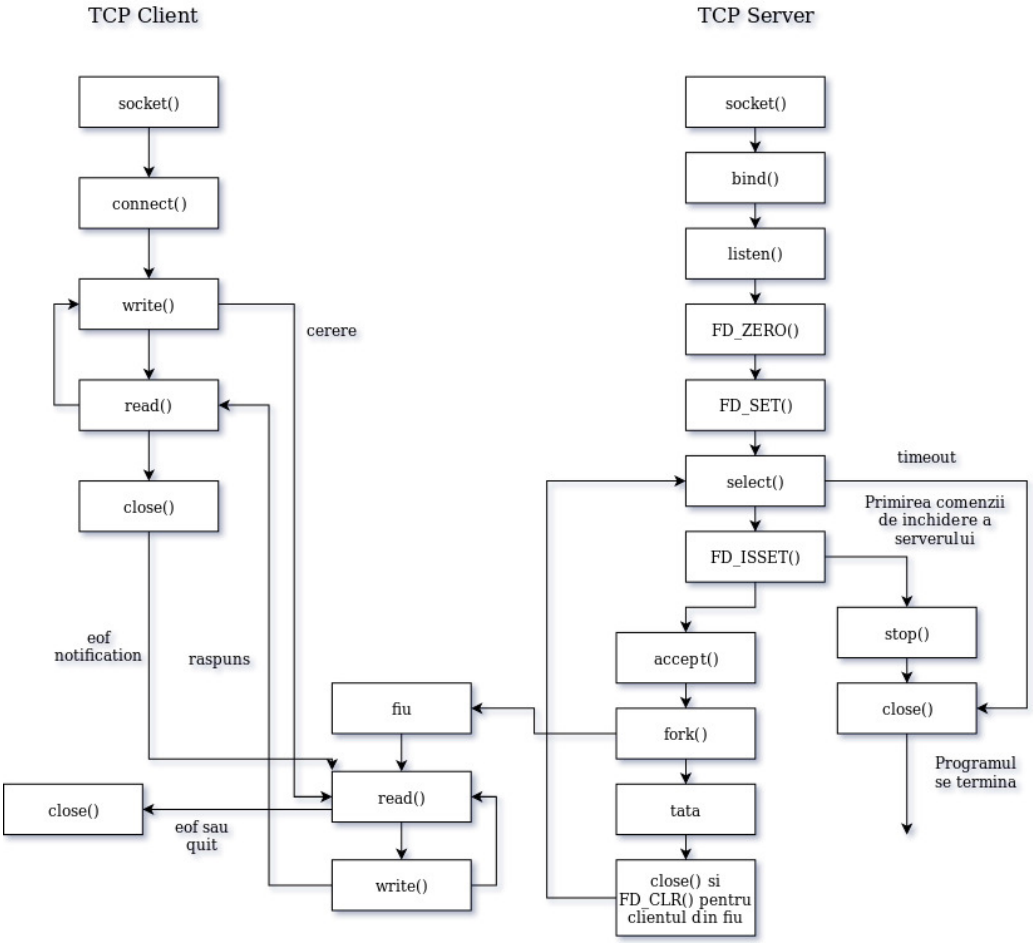
In realizarea proiectului a fost utilizat protocolul TCP/IP. Acesta are ca si scop principal transmiterea cat mai exacta a informatiilor. Din acest motiv, exista posibilitatea ca acestea sa ajunga cu anumite intarzieri fata de protocolul UDP, care prefera transmiterea rapida inaintea sigurantei sosirii informatiilor. TCP transmite astfel date fara pierderi si in aceeasi ordine in care au fost trimise. Cu schimbul de blocuri de informatii se ocupa IP. Acestea sunt numite pachete si reprezinta de fapt siruri de octeti.

In acest proiect a fost nevoie de TCP/IP din mai multe motive. In primul rand, clientii pot transmite mai multe comenzi catre server. Daca acestea nu ajung in aceeasi ordine, serverul nu va respecta cerintele clientului in totalitate. De asemenea, daca sunt pierderi de informatii, e posibil ca serverul sa nu recunoasca comenzile clientului cand de fapt ele sunt corecte.

In al doilea rand, se doreste ca datele cerute de client sa fie transmise integral si corect de catre server.

3 Arhitectura aplicatiei

3.1 Diagrama aplicatiei



3.2 Descrierea arhitecturii si a conceptelor utilizate

Proiectul TopMusic necesita un server TCP concurent. Mai multi clienti vor fi conectati la server in acelasi timp si vor da comenzi. Concurenta a fost realizata astfel: Primitiva **select()** va urmari daca trebuie inchis serverul(daca am primit o comanda de la consola serverului) sau daca un nou client trebuie acceptat. In cazul din urma, va fi apelata primitiva **fork()**, iar noul client va fi tratat in fiu. Tatal va continua sa accepte alti clienti.

Clientul poate da oricate comenzi doreste, avand si el o bucla infinita in care poate scrie mai multe mesaje. El are optiunea de a se deconecta de la server folosind comanda **quit**, caz in care descriptorul ii este inchis. Serverul va detecta si deconectarea fortata a clientilor datorate unei pene de curent, de internet, a comenzii **CTRL+C**, etc., si le va inchide si acestora descriptorii:

```
else if(pid_fiu==0)
{
    close(sd);
    FD_CLR(sd,&actfds);
    while(1)
        if (getFunc(client)==-1) {
            if(utilizatori.loggedIn())
                printf("[server] S-a deconectat clientul cu userul %s.\n", utilizatori.getName().c_str());
            else printf("[server] S-a deconectat un client nelogat!\n");
            fflush(stdout);
            utilizatori.logIn=false;
            close(client);
            FD_CLR (client, &actfds);
            break;
        }
    exit(0);
} else
{
    close(client);
    FD_CLR(client,&actfds);
    while(waitpid(-1,NULL,WNOHANG));
    continue;
}
```

Valoarea -1 este returnata de functia **getFunc()** cand trebuie inchis clientul, indiferent de caz.

Dupa primirea unei comenzi de la un client de catre server, acesta va efectua anumite operatii in functie de cererea primita, si va trimite un raspuns inapoi.

4 Detalii de implementare

4.1 Login si Register

O aplicatie ca TopMusic are nevoie de o modalitate de a recunoaste clientii care se conecteaza, alta decat descriptorii. Pentru acest lucru, sunt necesare functii de **login**, respectiv **register**. Utilizatorii nu au dreptul la nicio alta comanda inafara de **login** si **register** pana cand nu se logheaza, pentru a preveni astfel haosul in datele salvate. Conturile clientilor vor fi salvate intr-un fisier de tip **json** sub urmatoarea forma:

Pentru lucrul cu fisierele de tip **json** am folosit biblioteca **nlohmann/json.hpp**. Cu ajutorul acesteia putem sa stocam informatia intr-un obiect de tip nlohmann::json:

ifstream userfile(”../users.json”);
json users = nlohmann::json::parse(userfile);

Astfel, in obiectul **users** vom avea salvate toate conturile create vreodata in aplicatie. De fiecare daca cand un client creaza un cont nou, va fi creata si in fisier, de asemenea, o noua inregistrare:

```
if (users.find(user) != users.end())
{
    strcat(msgrasp,"User deja existent!");
}
else{
    users[user]["password"]=pass;
    users[user]["tipUser"]="regular";
    strcat(msgrasp,"Utilizatorul a fost inregistrat cu succes!");
    ofstream user_filewr("../users.json");
    user_filewr<<users.dump(2);
    user_filewr.close();
}
```

In codul de mai sus, **user** si **pass** sunt siruri de caractere transmise de client catre server. Parola, la citirea de la tastatura, nu va fi vizibila din motive de securitate.

Asemanator pentru **login**, existenta datelor transmise de client este verificata cu ajutorul fisierului **json** care contine pentru fiecare utilizator numele, parola si tipul(admin sau regular user).

Utilizatorii care nu sunt logati nu au dreptul sa execute alte comenzi. Asadar, este nevoie de o modalitate de a tine minte care clienti sunt logati si care nu.

Acum, exista urmatoarele 2 intrebari: "ce username are clientul?", respectiv "este clientul admin sau restrictionat?". Aceste informatii sunt retinute intr-un obiect de tipul **thisUser**, o clasa definita astfel:

```
class thisUser {
    string name;
    bool admin;
    bool restricted;
public:
    bool logIn;
    thisUser();
    string getName();
    bool isAdmin();
    void setProp(char user[], bool adm, bool res);
    bool loggedIn();
    bool isRestricted();
    void setRestrict(bool res);
    void setAdmin(bool adm);
    ~thisUser();
};
```

La fel ca la **register**, **user** si **pass** sunt transmise de la client si sunt preluate de server.

4.2 Cum sunt retinute melodiile

Melodiile, impreuna cu toate informatiile lor vor fi retinute intr-un alt fisier de tip **json**. Cand vom dori sa aducem in memorie informatiile, vom folosi instante ale unei clase, numite **Song**.

In map-ul **myVote**, votul utilizatorilor astfel: daca myVote[user] = 1, atunci votul este pozitiv, iar daca myVote[user]=-1, atunci votul este negativ pentru user-ul respectiv.

Dupa citirea din fisierul **json**, melodiile vor fi salvate intr-un vector de forma:

vector<Song*> top;

Acest vector va putea fi sortat cu usurinta dupa numarul de voturi, iar utilizatorii vor putea trimite comenzi dand ca argument doar numarul de ordine al melodiei in top (si eventual genul). Astfel, ele vor putea fi sortate foarte usor si dupa genuri.

```
class Song
{
    string name;
    string description;
    string artist;
    vector<string> genres;
    string link;
    string proposedBy;
    int votes;
    map<string,int> myVote;
    multimap<string,string> comments;
public:
    Song();
    Song(string nume, string descriere, string singer, vector<string> genuri, string url, int voturi, multimap<string,string> comentarii, map<string,int> votulMeu
    void upVote(string user);
    void downVote(string user);
    void addComment(string user,string comm);
    int howManyVotes();
    string getName();
    string getArtist();
    vector<string> &getGenres();
    void showInfo(string &msgrasp);
    string getDescription();
    string getUrl();
    multimap<string,string> &getComments();
    map<string,int> &getVotes();
    void changeVote(string user);
    string getUser();
    ~Song();
};
```

Vectorul de melodii va apartine unei alte clase, Songs, care se va ocupa si de sortarea acestora dupa numarul de voturi, respectiv de organizarea genurilor:

```
class Songs {
    vector<Song*> topGenre;
public:
    vector<Song*> top;
    Songs();
    void addSong(Song* s);
    void sortVoturi();
    void afiseaza(string &msgrasp);
    void selectGenre(string genres);
    void showTop(string &msgrasp);
    void showGenreTop(string &msgrasp);
    void showInfo(int s, string &msgrasp);
    void showGenreInfo(int s, string &msgrasp);
    unsigned int getSize();
    void addComment(int s,string user,string comm, string &msgrasp);
    void addGenreComment(int s,string user,string comm, string &msgrasp);
    void upVote(int s, string user, string &msgrasp);
    void upVoteGenre(int s, string user, string &msgrasp);
    void downVote(int s, string user, string &msgrasp);
    void downVoteGenre(int s, string user, string &msgrasp);
    void changeVote(int s, string user, string &msgrasp);
    void changeVoteGenre(int s, string user, string &msgrasp);
    void deleteGenreSong(int s, string &msgrasp);
};
```

Inainte de executia unei comenzi, vom citi din fisierul json informatiile despre melodii si le vom aduce in memorie. Dupa executia comenzii, vom scrie modificarile inapoi in fisier.

A mai ramas de rezolvat o singura problema: Daca doi sau mai multi clienti comenteaza, adauga o melodie sau voteaza in acelasi timp(foarte mic, de ordinul milisecundelor), se vor pierde modificari la scrierea in fisierul json. Asadar, aceasta problema poate fi rezolvata cu un lacat:

```
int descriptor=open("../songs.json",O_RDWR);
struct flock lacat;
lacat.l_type=F_RDLCK;
lacat.l_whence=SEEK_SET;
lacat.l_start=0;
lacat.l_len=0;
struct flock deblocare;
deblocare.l_type=F_UNLCK;
deblocare.l_whence=SEEK_SET;
deblocare.l_start=0;
deblocare.l_len=0;
if(-1==(fcntl(descriptor,F_SETLKW,&lacat)))
{
    perror("A aparut o eroare la punerea lacatului.\n");
    return errno;
}
makeTop();
```

Lacatul se va inlatura dupa scrierea in fisierul json a tuturor modificarilor rezultate in urma executiei unei comenzi.

Toate comenzile sunt explicate chiar in aplicatie, cu ajutorul comenzii "help".

5 Concluzii

Ca solutii pentru imbunatatirea proiectului, as avea urmatoarele idei:

- un client poate propune o melodie cu o informatie gresita, poate sa faca glume proaste sau sa foloseasca un limbaj neadecvat. Din acest motiv, toate melodiile propuse vor fi verificate de catre un administrator, care decide daca va permite intrarea in top a melodiilor sau nu;

- o interfata grafica pentru clientii aplicatiei;

- adaugarea campului **varsta** la inregistrare, pentru a nu permite celor cu o varsta mai mica de 14 ani ascultarea melodiilor cu un limbaj neadecvat.

6 Bibliografie

- www.wikipedia.org
- www.stackoverflow.com

- www.cplusplus.com
- https://github.com/nlohmann/json
- sites.google.com/view/fii-rc