Facultatea de Informatica IASI

TopMusic(B)

Nume student: Tanasiu
Prenume:Florentina
Grupa:B6
An de studiu:II
An universitar:2017-2018

Cuprins:

1.Introducere

Acest proiect presupune implementarea unei aplicații de tipul client/server pentru managmentul unui top muzical. Topul conține piese ce apartin mai multor genuri muzicale, iar clientii vor avea acces atât la topul general, cât și la cel pentru un anumit gen. Dealtfel, aceștia vor putea vota piesele favorite sau vor putea adauga altele noi și vor putea sa comenteze sau sa vadă descrierile melodiilor.

Fiecare piesa conține numele, o scurta descriere, genul, un link către vidioclipul de pe YouTube și comentariile adaugate de utilizatori. Clientii serverului vor trebui sa își creeze un cont sau să se logheze cu cel deja existent; în caz contrar ei nu vor putea accesa meniul. Utilizatorii sunt de doua tipuri: obișnuiti(inregistarti cu nume de forma <nume1.nume2>) și administartorul topului(înregistrat cu un usernane de forma <admin.nume1.nume2>). Clientii care își creaza un nou cont nu vor putea introduce cuvântul „admin” în numele de cont.

Meniul topului conține 8 comenzi, dintre care 2 sunt exclusiv destinate adrministratorului: **rvot**(pentru restrictionarea votului unui utilizator) și **sterge** (pentru a sterge o piesa din top). Celelate comezi, valabile pentru toate tipurile de utilizator sunt: **vot** [piesa], **vezi** [piesa], **top** [general/anumit gen], **adauga** [piesa], **com** [piesa], **ieși** [aplicație].

2.Tehnologii utilizate

Modelul standard pentru aplicatii in retea este client-server. Serverul este un proces care asteapta sa il contacteze clienții. Rolul acestuia este de a prelua și procesa cererile acestora, precum și de a furniza raspunsurile cerute. Modul de functionare pentru un astfel de sistem este: procesul server este pornit, iar acesta intra intr-o stare de asteptare (primitiva listen()) pana la conectarea unui posibil client.

Procesul client are rolul de a adresa cereri serverului si de a procesa raspunsurile primite. El se va conecta la server prin primitiva connect() utilizand portul și adresa folosite de server si va face cereri pentru serviciile pe care le ofera acesta. Clientul poate fi pornit pe acelasi sistem ca si serverul sau pe un altul. Urmeaza o conversatie intre client si server care se termina in momentul in care clientul a obtinut rezultatul dorit de la server. In acel moment serverul revine la starea de asteptare dupa potentiali clienți.

Realizarea aplicației se va face folosind un protocol TCP-IP concurent, întrucât dorim sa servim clientii în ordinea în care aceștia acceseaza topul, dar dorim și monitorizarea actiunilor făcute de client/server(avem nevoie de confirmari pentru a cunoaște în orice moment starile în care se afla cele doua).

„Transmission Control Protocol (sau TCP, în traducere liberă din engleză Protocolul de Control al Transmisiei) este un protocol folosit de obicei de aplicații care au nevoie de confirmare de primire a datelor. Acesta efectuează o conectare virtuală full duplex între două puncte terminale, fiecare punct fiind definit de către o adresă IP și de către un port TCP. Acesta oferă încredere, asigura livrarea ordonata a unui flux de octeți de la un program de pe un computer la alt program de pe un alt computer aflat în rețea. Pe lângă sarcinile sale de gestionare a traficului, TCP controlează mărimea segmentului de date, debitul de informație, rata la care se face schimbul de date, precum și evitarea congestionării traficului de rețea. ”
(sursa Wikipedia)

3.Arhitectura aplicației

Comunicarea dintre server și client se face prin socket-uri. Un socket poate fi privit ca un punct de conectare (soclu) pentru "legarea" procesului curent la o sesiune de comunicatie.

Interfaţa Socket API este o interfaţă între un program de aplicaţie şi serviciul de transport, fiind furnizat de o bibliotecă socket sau de sistemul de operare. Se foloseşte conceptul de descriptor, fiecare socket fiind tratat asemănător cu un fişier local. Acest descriptor este transmis aplicaţiei la crearea socket-ului şi apoi este utilizat ca argument în apelurile următoare.

Serverul va „servi” utilizatorii folosind thread-uri. Prin crearea de threaduri, procesul poate executa diferite secvente de cod in paralel pe fiecare din firele de executie. Spre deosebire de procesele fii lansate in cadrul unui proces parinte, threadurile create vor fi parte componenta a procesului care le-a creat, deci partajeaza spatiul de executie al acestuia (variabilele declarate in procesul "parinte" sunt comune tuturor threadurilor). Aplicatia trebuie sa stocheze si sa pastreze toate modificarile facute de fiecare client in parte. Dealtfel, toti clientii trebuie sa poata vedea schimbarile, chair si cei care sunt conectati in acelasi timp. Unul din motivele pentru care am ales threaduri este tocmai acet lucru.

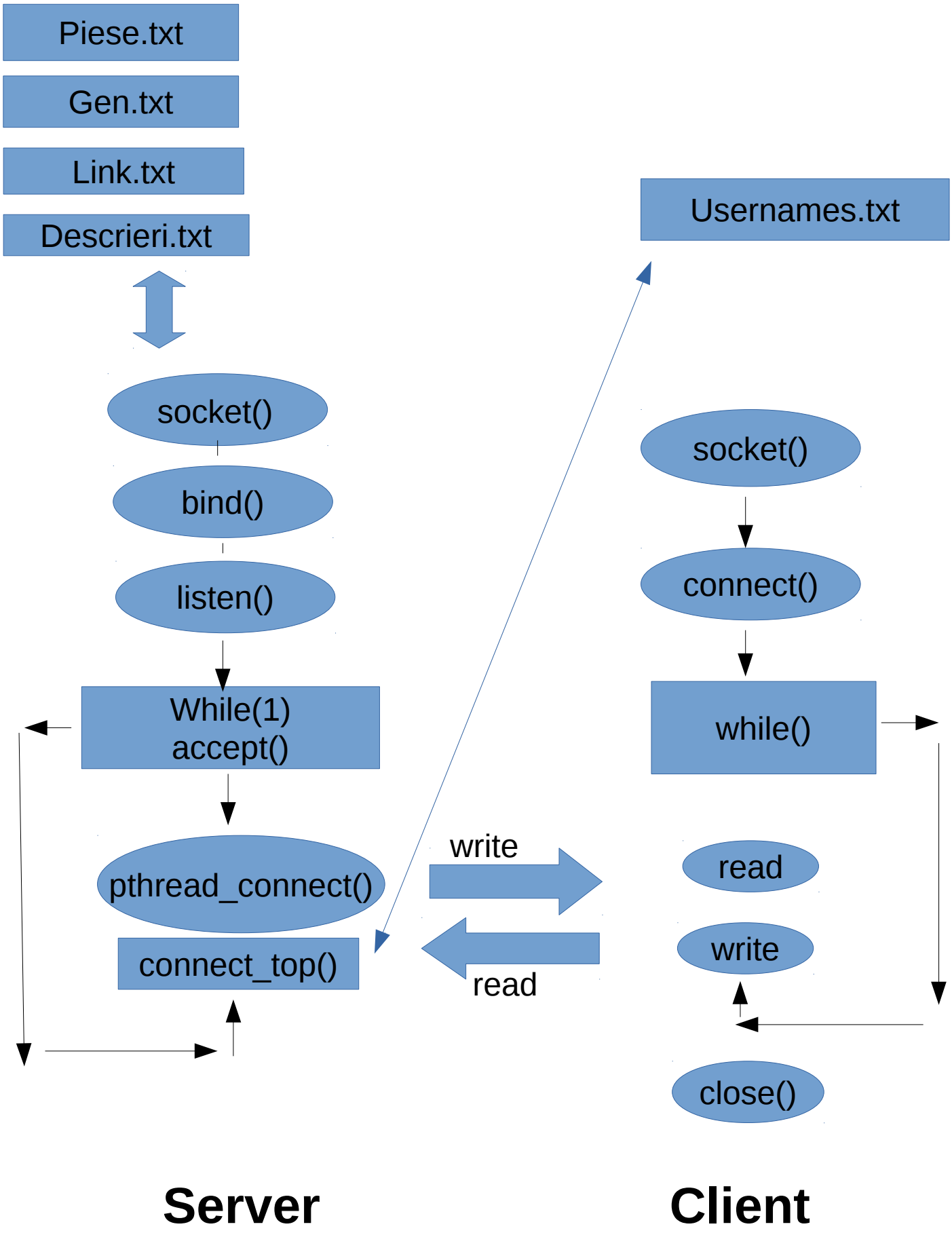
„Conceptul de thread (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către sistemul de operare. Este folosit în programare pentru a eficientiza execuția programelor, executând porțiuni distincte de cod în paralel în interiorul aceluiași proces. Câteodata însă, aceste portiuni de cod care constituie corpul threadurilor, nu sunt complet independente și în anumite momente ale execuției, se poate întampla ca un thread să trebuiască să

aștepte execuţia unor instructiuni din alt thread, pentru a putea continua execuţia propriilor instrucţiuni. Această tehnică prin care un thread asteaptă execuţia altor threaduri înainte de a continua propria execuţie, se numește sincronizarea threadurilor.”
(sursa: Wikipedia)

Alte motive pentru care am ales Threaduri în locul proceselor sunt urmatoarele:

1) Crearea threadurilor este mult mai rapidă
2) Schimbarea de context între Thread-uri este mult mai rapidă
3) Pot fi terminate cu ușurință
4) Comunicarea dintre ele este mai rapidă.

Diagrama aplicației:



4.Detalii implementare

Functia connect_top() este folosită de fiecare thread sa ofere clientilor acces la aplicație:

```
 void *connect_top(void *socket_desc)
{

    int sock = *(int*)socket_desc;
    int read_size;
    char message[2000] , client_message[2000],activ_usr[100]; // mesajul trimis către/primit de la client și numele utiliztorului curent(vom avea nevoie de el la vot și pt comezile disponibile doar administratorului)
    char detalii_piesa[2000];
    int acc=0;  // pentru login sau creare cont nou

//login și creare cont nou
    bzero(message,2000);

    strcat(message,"Momentan nu esti logat.\n Nu poti accesa informatii daca nu esti logat. \n");
    strcat(message,"Aveti deja un cont? da/nu \n");
    write(sock , message , strlen(message));

    bzero(message,2000);
    bzero(client_message,2000);

    read_size = read(sock , client_message ,2000);

    if(strcmp(client_message,"da")==0)
    {
            strcat(message,"Introduceti numele de utilizator:\n");
            write(sock , message , strlen(message));
            bzero(message,2000);
            read_size = read(sock , client_message ,2000);
            acc=login(client_message);
            while(acc==0)
            {
                strcat(message,"Cont inexistent..Va rugam introduceti alt nume de utilizator:");
                write(sock , message , strlen(message));
                bzero(client_message,2000);
                read_size = read(sock , client_message ,2000);
                acc=login(client_message); //functia este descrisa în fișierul top.c astfel:
    /*
```

```
 int login(char ans[100])
{
char check[100];
FILE * user;
int access=0;

user = fopen("usernames.txt", "r");

if(user == NULL)
{
  perror("can't open the usernames.txt");
  exit(EXIT_FAILURE);
}
printf("\n one more client try to login..procesing..\n");
sleep(1);
printf("\n searching in usernames.txt.. \n");

while(!feof(user))
{
      if( fscanf(user,"%s",check)==0)
      {
        perror("can't read from usernames.txt");
        exit(EXIT_FAILURE);
      }

      if(strcmp(ans,check)==0)
      {

        access=1;

      }
}

fclose(user);

if(access==1)
printf("\n...success..connected..\n");

return access;
}
*/

        }

        bzero(message,2000);
```

```
            strcat(message,"\n conexiune reusita..\n  Salut " );
            strcat(message,client_message);
            write(sock , message , strlen(message));
            strcat(activ_usr,client_message);


    }

   if(strcmp(client_message,"nu")==0)
    {
            bzero(message,2000);
            strcat(message,"Daca nu iti creezi un cont nu poti vedea continutul.Te rog sa
alegi un nume de cont:");
            write(sock , message , strlen(message));
            read_size = read(sock , client_message ,2000);

            if(strstr(client_message,"admin")!= NULL)
            {
               bzero(message,2000);
               strcat(message,"Nu poti crea un cot folosind numele 'admin'..alege alt
nume de cont:");
               write(sock , message , strlen(message));
               bzero(client_message,2000);
               read_size = read(sock , client_message ,2000);
            }
            acc=create_newCont(client_message); //definita în top.c astfel:
/*
int create_newCont( char ans[100])
{
    FILE * user;
    user = fopen("usernames.txt", "r+");
    char check[100];
    int repeat=0;

    if(user == NULL)
    {
        perror("can't open the usernames.txt");
        exit(EXIT_FAILURE);
    }
    while(!feof(user))
    {
         if( fscanf(user,"%s",check)== 0)
         {
           perror("can't read from usernames.txt");
           exit(EXIT_FAILURE);
```

```
        }

        if(strcmp(ans,check)==0)
        {
          repeat=1;
        }
    }

    if(repeat==0)
    {
        fprintf(user, "%s\n", ans);
        fclose(user);
        printf("\n new cont created.\n");

    }

    return repeat;
}
*/
            while(acc==1)
            {
               bzero(message,2000);
               strcat(message,"Numele deja exista..te rog sa alegi altul " );
               write(sock , message , strlen(message));
               bzero(client_message,2000);
               read_size = read(sock , client_message ,2000);
               if(strstr(client_message,"admin")!= NULL)
               {
                   bzero(message,2000);
                   strcat(message,"Nu poti crea un cot folosind numele 'admin'..alege alt
nume de cont:");
                   write(sock , message , strlen(message));
                   bzero(client_message,2000);
                   read_size = read(sock , client_message ,2000);
               }
               acc=create_newCont(client_message);
            }
            if(acc==0)
            {

               bzero(message,2000);
               strcat(message,"Felicitari..ti-ai creat un nou cont!\n Acum esti logat cu
numele ");
               strcat(message,client_message);
               write(sock , message , strlen(message));
```

```
            strcpy(users[nr_usr].user,message);// users este un vector în care sunt înregistraţi toţi utilizatorii şi va fi folosit pentru a vedea dacă un client are sau nu drept de vot
            users[nr_usr].acc=1; //initial, toţi clientii au drept de vot(1)
            nr_usr++;
            strcat(activ_usr,client_message);

        }
    }

    bzero(message,2000);
    strcat(message,"\n Alege una din comenzile: \n  vot -> voteaza o piesa \n  vezi -> vezi detaliile unei piese \n  top -> consulta unul din topuri(general sau pentru un anumit gen) \n  adauga -> adauga o piesa \n  com -> adauga comentariu \n  rvot -> restrictioneaza vot (doar pentru admin) \n  sterge -> sterge o piesa (doar pentru admin) \n  iesi -> iesi din aplicatie");
    write(sock , message , strlen(message));
    bzero(client_message,2000);

// cât timp un client este activ pe server, poate trimite comenzi către acesta:
    while( (read_size = read(sock , client_message ,2000)) > 0 )
    {
        if(strcmp(client_message,"vezi")==0) // pentru a vedea detaliile unei piese, clientul va introduce numele partial (doar titlul) sau complet( numele unei piese cuprinde atât titlul, cat şi artistul <ex. Pink – What about us>)
        {

            bzero(message,2000);
            strcat(message,"Introdu numele piesei: " );
            write(sock , message , strlen(message));
            bzero(client_message,2000);
            read_size = read(sock , client_message ,2000);

            for (int i=0;i<nr_piese;i++)
            {
                if( strstr(p[i].nume,client_message)!=NULL)// piesele sunt stocat într-un vector p de tipul struct ce se găseşte în fişierul top.c
                {
                  bzero(detalii_piesa,2000);
                  strcat(detalii_piesa,"\n Titlu: ");
                  strcat(detalii_piesa,p[i].nume);
                  strcat(detalii_piesa,"\n Gen: ");
                  strcat(detalii_piesa,p[i].gen);
                  strcat(detalii_piesa,"\n Descriere:\n ");
                  strcat(detalii_piesa,p[i].descriere);
```

```
                strcat(detalii_piesa,"\n Urmareste link-ul urmator pentru a asculta piesa:
");
                strcat(detalii_piesa,p[i].link);
                strcat(detalii_piesa,"\n Comenarii: \n");
                struct nod* new_nod2 = (struct nod*)malloc(sizeof(struct nod));
                new_nod2=p[i].comentarii;
                while(new_nod2!= NULL)
                {
                   strcat(detalii_piesa,"user:");
                   strcat(detalii_piesa,&new_nod2->user);
                   strcat(detalii_piesa,"\n");
                   strcat(detalii_piesa,&new_nod2->content);
                   new_nod2=new_nod2->urm;
                 }
                write(sock , detalii_piesa , strlen(detalii_piesa));
                break;

              }

           }

           bzero(client_message,2000);
        }

        if(strcmp(client_message,"vot")==0)// functia vot funcţionează în felul
următor: dacă administratorul nu a restrictionat votul clientului activ, acesta poate
introduce numele piesei şi severul va inregistra votul sau, altfel el va fi înştiinţat ca
nu mai are drept de vot
        {
           for (int i=0;i<nr_usr;i++)
           {
             if( strcmp(activ_usr,users[i].user)==0)
             {
                if(users[i].acc==0)
                {
                   bzero(message,2000);
                   strcat(message,"Ne pare rau dar nu mai ai dreptul sa votezi o piesa:("
);
                   write(sock , message , strlen(message));
                }

                else
                {
                   bzero(message,2000);
                   strcat(message,"Introdu numele piesei: " );
```

```
                write(sock , message , strlen(message));
                bzero(client_message,2000);
                read_size = read(sock , client_message ,2000);

                for (int i=0;i<nr_piese;i++)
                {
                  if( strstr(p[i].nume,client_message)!=NULL)
                  {
                      p[i].vot++;
                      bzero(message,2000);
                      strcat(message,"felicitari! Tocmai ati votat piesa aleasa ^_^" );
                      write(sock , message , strlen(message));
                      break;
                   }

                }//for

              }//else

            }//if_user

          }//for_user

          bzero(client_message,2000);

        }//if_vot

        if(strcmp(client_message,"top")==0) //comanda va afisa topul ales de client (general sau pt un gen anume);
        {
            quickSort(p,0,nr_piese-1); //inainte de a afisa topul, piesele trebuiesc sortate după nr. voturi
            bzero(message,2000);
            strcat(message,"\n Va rog alegeti topul pe care vreti sa il consultati: general/(anumit_gen) \n" );
            write(sock , message , strlen(message));
            bzero(client_message,2000);
            read_size = read(sock , client_message ,2000);
            bzero(message,2000);
            strcat(message,"\n Acesta este topul ");
            strcat(message,client_message);
            strcat(message," ^_^:\n\n");

            char c[2];
            int index;
```

```
index=1;
if(strcmp(client_message,"general")==0)
{

    for(int i=nr_piese-1;i>=0;i--)
    {
        if(index<10)
        {
           c[0]=index+'0';
           c[1]='\0';
        }
        else
        {
           c[0]=index/10+'0';
           c[1]=index%10+'0';
           c[2]='\0';
        }
        strcat(message,c);
        strcat(message,". ");
        index++;
        strcat(message,p[i].nume);
        strcat(message,"\n");
    }

    write(sock , message , strlen(message));
 }
 else
 {
    char c[2];
    int index;
    index=1;
    for(int i=nr_piese-1;i>=0;i--)
    {
        if((strstr(p[i].gen,client_message)!=NULL))
        {
           if(index<10)
           {
              c[0]=index+'0';
              c[1]='\0';
           }
           else
           {
              c[0]=index/10+'0';
              c[1]=index%10+'0';
              c[2]='\0';
```

```
                }
                strcat(message,c);
                strcat(message,". ");
                index++;
                strcat(message,p[i].nume);
                strcat(message,"\n");
            }
        }

        write(sock , message , strlen(message));
    }

    bzero(client_message,2000);

}//if_top

    if(strcmp(client_message,"adauga")==0)// pentru a adauga o piesa noua, clientul trebuie să trimită serverului, pe rând, toate caracteristicile unei piese: nume, descriere, link către YouTube și genul; nr. voturi va fi setat implicit la 0 și pentruinceput piesa nu va cotine nici un comentariu
    {
        bzero(message,2000);
        strcat(message,"Introduceti numele piesei: " );
        write(sock , message , strlen(message));
        bzero(client_message,2000);
        read_size = read(sock , client_message ,2000);

        strcpy(p[nr_piese].nume,client_message);

        bzero(message,2000);
        strcat(message,"Introduceti genul de care apartine piesa: " );
        write(sock , message , strlen(message));
        bzero(client_message,2000);
        read_size = read(sock , client_message ,2000);

        strcpy(p[nr_piese].gen,client_message);

        bzero(message,2000);
        strcat(message,"Introduceti o descriere pentru piesa: " );
        write(sock , message , strlen(message));
        bzero(client_message,2000);
        read_size = read(sock , client_message ,2000);

        strcpy(p[nr_piese].descriere,client_message);
```

```
            bzero(message,2000);
            strcat(message,"Introduceti link-ul catre piesa: " );
            write(sock , message , strlen(message));
            bzero(client_message,2000);
            read_size = read(sock , client_message ,2000);

            strcpy(p[nr_piese].link,client_message);

            p[nr_piese].vot=0;
            nr_piese ++;

            bzero(client_message,2000);
        }

        if(strcmp(client_message,"com")==0) // un comentariu introdus de un utilizator este stocat într-o lista simpla, în care fiecare nod conţine atât numele utilizatorului cât şi comentariul; noile comentarii sunt adaugate în vârful listei, aşa ca cele mai noi vor fi afisate primele
        {
            bzero(message,2000);
            strcat(message,"Introduceti numele piesei: " );
            write(sock , message , strlen(message));
            bzero(client_message,2000);
            read_size = read(sock , client_message ,2000);

            for (int i=0;i<nr_piese;i++)
            {
               if( strstr(p[i].nume,client_message)!=NULL)
               {

                  struct nod* new_nod = (struct nod*)malloc(sizeof(struct nod));
                  strcpy(new_nod->user,activ_usr);

                  bzero(message,2000);
                  strcat(message,"Introduceti comentariul: " );
                  write(sock , message , strlen(message));
                  bzero(client_message,2000);
                  read_size = read(sock , client_message ,2000);

                  strcpy(new_nod->content,client_message);
                  new_nod->urm = p[i].comentarii;
                  p[i].comentarii = new_nod;

                  bzero(message,2000);
                  strcat(message,"Comentariu adaugat cu succes! ^_^ " );
```

```
                write(sock , message , strlen(message));

              }

             }

            bzero(client_message,2000);
          }

          if(strcmp(client_message,"sterge")==0)// doar administratorul poate sterge o piesa din  top, iar ceilalti utilizatori care încearcă sa excute comanda vor fi reintiintati printr-un mesaj( în meniu este specificat restrictia privind utilizarea acestei comenzi)
          {
            int de_sters;
            if(strstr(activ_usr,"admin")==NULL)
            {
                bzero(message,2000);
                strcat(message,"Doar administaratorul poate sterge o piesa ^_^ " );
                write(sock , message , strlen(message));
             }

             else
             {
                bzero(message,2000);
                strcat(message,"Introduceti titlul piesei: " );
                write(sock , message , strlen(message));
                bzero(client_message,2000);
                read_size = read(sock , client_message ,2000);

                for (int i=0;i<nr_piese;i++)
                {
                    if( strstr(p[i].nume,client_message)!=NULL)
                    {
                       de_sters=i;
                       break;
                    }
                }

                for (int i=de_sters;i<nr_piese;i++)
                {
                    p[i].nume=p[i+1].nume;
                    p[i].vot=p[i+1].vot;
                    p[i].descriere=p[i+1].descriere;
                    p[i].link=p[i+1].link;
                    p[i].gen=p[i+1].gen;
```

```
                    p[i].comentarii=p[i+1].comentarii;
                }
                nr_piese--;
                bzero(message,2000);
                strcat(message,"Tocmai ati sters piesa " );
                strcat(message,client_message);
                write(sock , message , strlen(message));

            }
            bzero(client_message,2000);
        }

        if(strcmp(client_message,"rvot")==0) // analog sterge();
        {
            int de_sters;
            if(strstr(activ_usr,"admin")==NULL)
            {
                bzero(message,2000);
                strcat(message,"Doar administaratorul poate restrictiona dreptul de
vot al unui utilizator ^_^ " );
                write(sock , message , strlen(message));
            }

            else
            {
                bzero(message,2000);
                strcat(message,"Introduceti numele utilizatorului: " );
                write(sock , message , strlen(message));
                bzero(client_message,2000);
                read_size = read(sock , client_message ,2000);

                for (int i=0;i<nr_usr;i++)
                {
                    if( strcmp(users[i].user,client_message)==0)
                    {
                        users[i].acc=0;
                        break;
                    }
                }
                bzero(message,2000);
                strcat(message,"Tocmai ati restrictionat dreptul de vot al
utilizatorului " );
                strcat(message,client_message);
                write(sock , message , strlen(message));
```

```
                }//else

               bzero(client_message,2000);
            }

            if(strcmp(client_message,"iesi")==0)// severul deconecteaza clientul
            {
                close(sock);
            }

            if( strstr("top vezi vot com rvot sterge iesi ",client_message)==NULL)
            {
                bzero(message,2000);
                strcat(message,"Comanda inexistenta ^_^" );
                write(sock , message , strlen(message));
            }

 }//while


   if(read_size == 0) /// clientul s-a deconectat
   {
      puts("Client disconnected");
      fflush(stdout);
   }
   else if(read_size == -1)   /// eroare la citirea din socket a mesajului venit de la
client
   {
      perror("read() failed");
   }


   return 0;
}
```

Toate functiile necesare excutarii comenzilor din main, precum și variabilele globale care stocheaza informațiile din server (piese, conturile utilizatorului) sunt definite în fișierul top.c:

```
struct nod {
        struct nod* urm;
        char user[100];
        char content[200];
};
```

```
struct piesa {
            int vot;
            char *nume;
            char *descriere;
            char *gen;
            char *link;
            struct nod* comentarii;
};

struct dr_vot {
             char user[100];
             int acc;
};


struct dr_vot users[1000];
struct piesa p[300];
int nr_piese, nr_usr;


int vot_right(struct dr_vot users[1000])
{
   FILE * f1;
   size_t len=0;
   ssize_t read;

   f1 = fopen("usernames.txt", "r");

   if(f1 == NULL)
   {
     perror("can't open the usernames.txt");
     exit(EXIT_FAILURE);
   }

   nr_usr=0;
   while(!feof(f1))
   {
      if( fscanf(f1,"%s",users[nr_usr].user)==0)
      {
         perror("can't read from usernames.txt");
         exit(EXIT_FAILURE);
      }
      users[nr_usr].acc=1;
      nr_usr++;
```

```
    }
    fclose(f1);

    return nr_usr;
}

int populate_songList(struct piesa p[300])
{
    FILE * f1;
    FILE * f2;
    FILE * f3;
    FILE * f4;
    int nr_piese=0;
    size_t len = 0;
    ssize_t read;

    f1 = fopen("piese.txt", "r");
    f2 = fopen("descrieri.txt", "r");
    f3 = fopen("gen.txt", "r");
    f4 = fopen("link.txt","r");


    if(f1 == NULL)
    {
      perror("can't open the piese.txt");
      exit(EXIT_FAILURE);
    }

    if(f2 == NULL)
    {
      perror("can't open the descrieri.txt");
      exit(EXIT_FAILURE);
    }

    if(f3 == NULL)
    {
      perror("can't open the gen.txt");
      exit(EXIT_FAILURE);
    }

    if(f4 == NULL)
    {
      perror("can't open the link.txt");
      exit(EXIT_FAILURE);
    }
```

```
    printf("\n populate the song list ..\n");
     while ((read = getline(&p[nr_piese].nume, &len, f1)) != -1)
    {
           nr_piese++;
    }

    int i=0;
    while ((read = getline(&p[i].descriere, &len, f2)) != -1)
   {
          i++;
    }
    i=0;
    while ((read = getline(&p[i].gen, &len, f3)) != -1)
    {
          i++;
    }

    i=0;
    while ((read = getline(&p[i].link, &len, f4)) != -1)
    {
          i++;
    }

   for(int i=0;i<nr_piese;i++)
   {
      p[i].vot=0;
      p[i].comentarii= NULL;
   }

   return nr_piese;

}

int partition (struct piesa p[300], int start, int stop) ///pentru quick sort
{
    int pivot = p[stop].vot;
    int i = (start - 1);

    for (int j = start; j <= stop- 1; j++)
    {
        if (p[j].vot <= pivot)
        {
            i++;
```

```
            struct piesa x;
            x.nume=p[i].nume;
            x.vot=p[i].vot;
            x.descriere=p[i].descriere;
            x.link=p[i].link;
            x.gen=p[i].gen;

            p[i].nume=p[j].nume;
            p[i].vot=p[j].vot;
            p[i].descriere=p[j].descriere;
            p[i].link=p[j].link;
            p[i].gen=p[j].gen;

            p[j].nume=x.nume;
            p[j].vot=x.vot;
            p[j].descriere=x.descriere;
            p[j].link=x.link;
            p[j].gen=x.gen;

        }
    }
    struct piesa x;
    x.nume=p[i+1].nume;
    x.vot=p[i+1].vot;
    x.descriere=p[i+1].descriere;
    x.link=p[i+1].link;
    x.gen=p[i+1].gen;

    p[i+1].nume=p[stop].nume;
    p[i+1].vot=p[stop].vot;
    p[i+1].descriere=p[stop].descriere;
    p[i+1].link=p[stop].link;
    p[i+1].gen=p[stop].gen;

    p[stop].nume=x.nume;
    p[stop].vot=x.vot;
    p[stop].descriere=x.descriere;
    p[stop].link=x.link;
    p[stop].gen=x.gen;
    return (i + 1);
}

void quickSort(struct piesa p[300], int start, int stop) /// functia este folosită pentru
afisarea topului
{
```

```
    if (start < stop)
    {
        int nr = partition(p, start, stop);

        quickSort(p, start, nr - 1);
        quickSort(p, nr + 1, stop);
    }
}
```

5.Concluzii

Proiectul poate fi imbunatatit prin utilizarea unei baze de date în locul fisierelor, pentru o mai buna gestionare a datelor și pentru a stoca un numar mare de informatii. De altfel, acesta poate fi imbunatatit și prin crearea unei interfete grafice atractive.

Bibliografie:

- http://en.wikipedia.org/wiki/Transmission_Control_Protocol
- https://www.google.ro
- http://stackoverflow.com/
- Cursurile disciplinei “Retele de calculatoare ” - Lenuta Alboaie, Facultatea de Informatica Iasi.