Raport Proiect Retele de Calculatoare
Championship
Tema 2

Ungureanu C. Ana-Maria
Anul II, Grupa B4

January 12, 2024

1 Introducere

Acest raport este conceput pentru a demonstra implementarea unui client si a unui server concurent pentru proiectul Championship, un proiect de tipul B.

Acest proiect a fost creat cu scopul de avea o aplicatie de tipul client-server prin care se pot inregistra anumite campionate de jocuri electronice apartinand mai multor categorii de jocuri si la care pot participa mau multe echipe.

Implementarea clientului si al serverului concurent va prelucra cererile clientilor si va oferi posibilitatea de inregistrare a unui campionat, specificare a jocului, al numarului de jucatori, a diferitelor reguli sau structuri de campionat(single-elimination, double-elimination), al modului de extragere a partidelor, inregistrare a unui utilizator(atat administratorilor, cat si celor obisnuiti), de a vizualiza informatii despre ultimele campionate inregistrate, avand posibiliatea de a se inscrie la ele, urmand sa fie informati ulterior via e-mail daca au fost acceptati in campionatul respectiv si vor primi informatii sumplimentare despre partidele lor(ora, adeversarul, etc). De asemenea utilizatorul are posibilitatea de a reprograma o sesiune de joc, iar administratorii vor detine un istoric de evidenta al scorurilor partidelor.

2 Tehnologii utilizate

2.1 TCP - Protocol de control al transmisiei

Pentru realizarea conexiunii dintre client si server va fi utilizat un protocol de comunicare de tipul TCP/IP concurent, astfel exista posibilitatea de conectare a mai multor utilizatori simultan.

TCP a fost folosit intrucat fiind un protocol de transport orientat conexiune, fara riscul de pierdere a informatiei, ofera o calitate superioara a serviciilor printre care se numara viteza mai buna la nivelul schimbului de informatii, mecanisme de stabilire si eliberare a conexiunii. Stabilirea conexiunii se realizeaza printr-un mecanism numit "Three-way handshake" in care crearea conexiunii dintre client si server are loc inainte ca datele sa fie trimise si care se ocupa la final de inchiderea corecta a sesiunii de lucru. De asemenea, este capabil sa detecteze posibile erori, sa secventieze datele si sa le retransmita in cazul in care se pierd in timpul trimiterii.

Avand in vedere faptul ca aplicatia trebuie sa serveasca clientii in mod concurent, tehnica optima din punct de vedere al memoriei este utilizarea thread-urilor, astfel vom crea un thread cu un id unic pentru fiecare client(utilizator).

2.2 Baza de date SQLite

Pentru stocarea informatiilor voi crea in SQLite o baza de date intrucat citirea/scrierea/cautarea/actualizarea datelor intr-o astfel de baza este mult mai favorabila in detrimentul unui fisier text, nefiind astfel nevoie de a accesa date stocate pe disk. Baza va fi formata din mai multe tabele, cum ar fi:

1. o tabela va contine numele utilizatorului, tipul(obisnuit sau administrator), email-ul, parola, punctul slab si forte al jucatorului;

2. o tabela va contine numele, structura, numarul de jucatori, jocul asociat, regulile, modul de extragere al partidelor ale fiecarui campionat, istoricul, castigatorii si scorurile partidelor anterior jucate, numarul participantilor, participantii, data si ora urmatoarei partide;

Astfel, in functie de fiecare comanda introdusa de clienti(utilizatori), se va returna raspunsul care corespunde interogarii specifice in limbajul SQL asociate comenzii respective.

Un alt avantaj este organizarea pe linii si coloane, fiind simplu sa filtram anumite informatii. Spre exemplu: un jucator doreste sa se logheze, aplicatia va cauta in baza de date, in tabela cu toti utilizatorii inregistrati numele acestuia, daca il gaseste atunci va primi acces si va putea sa utilizeze ulterior alte comenzi. In cazul in care nu il gaseste va returna un mesaj de eroare.

Pentru crearea bazei se vor folosi functiile din biblioteca "sqlite3.h".

3 Arhitectura aplicatiei

3.1 Arhitectura clientului

Clientul creaza un socket prin intermediul functiei socket(), si salveaza descriptorul rezultat intr-o variabila urmand ca, mai apoi, sa incerce conectarea sa la server prin intermediul apelului connect().

Dupa conectare, se citeste comanda din terminal care va fi transmisa serverului. Dupa ce primeste un raspuns de la server, il va citi si il va afisa in terminal.

3.2 Arhitectura serverului

Serverul creaza un socket prin intermediul functiei socket(), si salveaza descriptorul rezultat intr-o variabila, populeaza baza de date si ataseaza socketului informatiile respective prin apelul bind().

Acest socket este atasat la un port prestabilit pentru a putea oferi servicii la acel port si intra intr-o stare de asteptare pasiva a conexiunilor clientilor, prin intermediul primitivei listen().

Intra intr-o bucla infinita, in cadrul careia accepta conexiunea cu clientul prin intermediul apelului accept(), salvand , de asemenea, si descriptorul returnat, urmand sa creeze un fir de executie(thread) dedicat clientului respectiv prin apelul pthread_create().

In handler-ul thread-ului, treat(), va citi comanda transmisa de client si o va procesa. Ulterior, dupa procesare, serverul va trimite catre client un raspuns.

In cazul in care clientul paraseste aplicatia(printr-un semnal SIGINT, SIGQUIT, SIGTSTP sau prin intermediul unei comenzi "quit"), acesta va iesi din bucla, altfel va relua executia buclei.

Diagrama de mai jos ilustreaza arhitectura aplicatiei client/server:

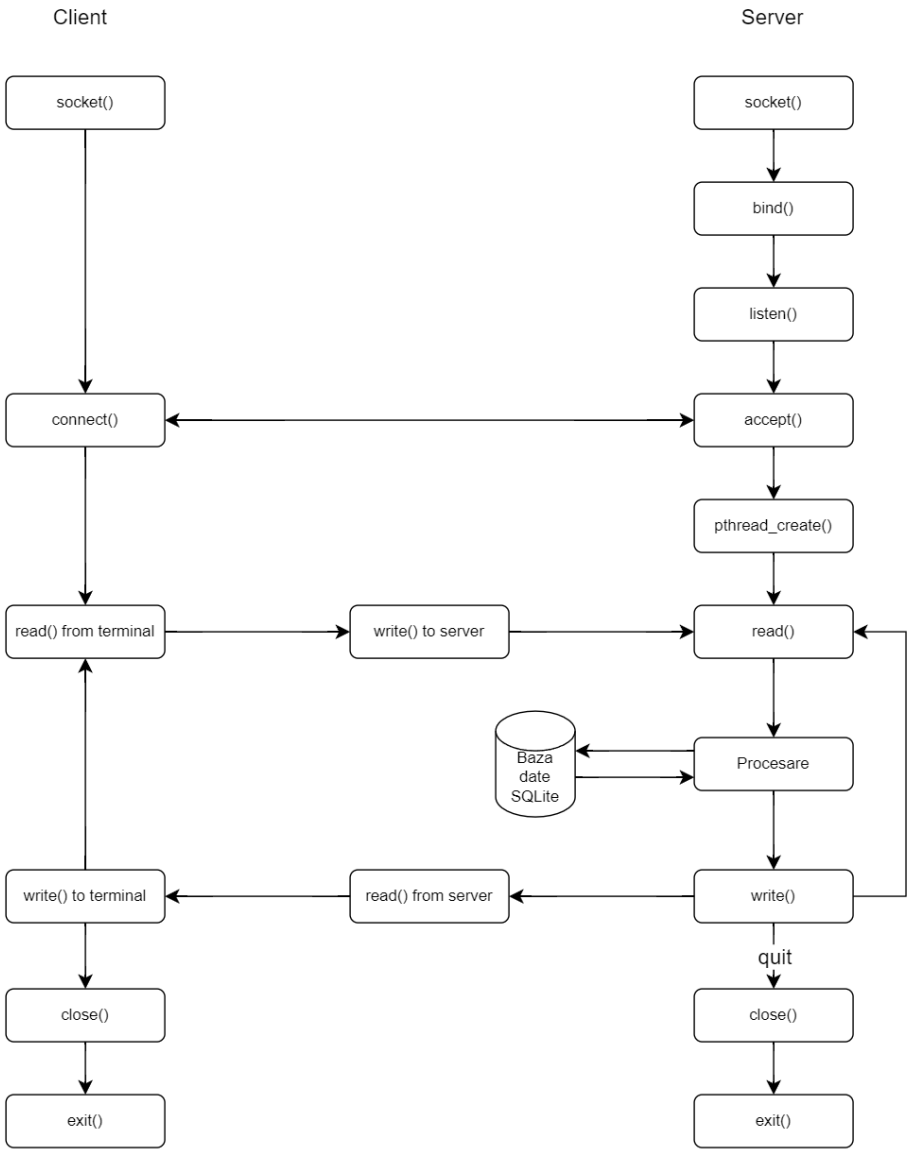

Figura 1: Structura programului

4 Detalii de implementare

In cadrul aplicatiei, comunicarea efectiva intre client si server se realizeaza cu ajutorul unor comenzi specifice:

1. inregistrare: utilizatorul are optiunea de a crea un cont daca nu are deja unul. Datele necesare sunt: numele, tipul utilizatorului(obisnuit sau administrator), parola, email-ul, punct slab si punct forte. Ulterior, aceste informatii vor fi stocate in baza de date. In cazul in care numele utilizatorului si/sau email-ul se regasesc in aceasta baza, va fi returnat un mesaj de eroare, iar utilizatorul are doua optiuni, mai exact, de a incerca inregistrarea cu alte date sau sa paraseasca aplicatia.

2. conectare: permite unui utilizator deja inregistrat in baza de date sa se conecteze la aplicatie. In cazul in care numele de utilizator si/sau parola sunt gresite, va fi returnat un mesaj de eroare, iar utilizatorul are mai multe optiuni: de a reintroduce datele, de a schimba parola(in cazul in care aceasta este problema), de a se conecta la alt cont sau de a parasi aplicatia.

3. inregistrare campionat: se va crea un nou campionat la care pot participa utilizatori obisnuiti. Aceasta comanda poate fi utilizata doar de catre administratori, ei avand responsabilitatea de a oferi mai multe specificatii referitoare la acest campionat: numele jocului, numarul de participanti, tipul, diferite reguli, etc.

4. inregistrare utilizator intr-un campionat: presupune inregistrarea unui utilizator obisnuit intr-un campionat din baza de date. Aceasta comanda poate fi utilizata doar de catre utilizatori obisnuiti. Dupa inregistrare, acesta va fi informat via email daca cererea sa de inscriere a fost acceptata si in acest caz va primi informatii aditionale despre campionat.In cazul in care numele campionatului este neconform sau utilizatorul participa deja la campionat va fi returnat un mesaj de eroare, iar utilizatorul va avea posibilitatea de a reintroduce datele.

5. reprogramare sesiune de joc: in cazul in care un utilizator nu poate participa la o partida de joc, acesta are posibilitatea de a o reprograma. Aceasta comanda poate fi utilizata doar de catre utilizatori obisnuiti.In cazul in care numele campionatului este neconform sau in care utilizatorul nu participa in sesiunea de joc va fi returnat un mesaj de eroare, iar utilizatorul va avea posibilitatea de a reintroduce datele.

6. istoric partide: va afisa un istoric al partidelor deja jucate de utilizatori obisnuiti. Aceasta comanda poate fi utilizata doar de administratori.

7. informatii campionate: vor fi afisate informatii despre anumite campionate(numele campionatului, diferite reguli, tipul jocului, etc). Aceasta comanda poate fi utilizata atat de catre utilizatori obisnuiti, cat si de administratori.

8. editare parola: utilizatorul are optiunea de a modifica parola. In cazul in care nu exista numele utilizatorului in baza de date se afiseaza un mesaj de eroare. Aceasta comanda poate fi utilizata atat de catre utilizatori obisnuiti, cat si de administratori.

9. delogare: delogeaza utilizatorul, iar conexiunea dintre server si thread-ul unic este incetata.

10. quit: permite utilizatorului sa paraseasca serverul.

11. help: va afisa o lista cu comenzile pe care le poate da utilizatorul cu parametrii corespunzatori.

In urmatoarele sectiuni este explificata implementarea clientului si al serverului concurent. Pentru fiecare client acceptat se creeaza un nou thread, care ii va fi atribuit, iar pentru fiecare comanda vom primi de la server confirmari.

4.1 Client

```c
int port = 2024;

int main()
{
    int sd;
    struct sockaddr_in server;
    char comanda[100];
    char raspuns[100];

    if((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
      perror("Eroare la socket().\n");
      exit(EXIT_FAILURE);
    }

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(port);

    if((connect(sd, (struct sockaddr *) &server, sizeof(struct sockaddr))) == -1)
    {
       perror("[CLIENT]Eroare la connect().\n");
       exit(EXIT_FAILURE);
    }

    while(1)
    {
       bzero(comanda, sizeof(comanda));
       printf("[CLIENT]Introduceti comanda:");
       fflush(stdout);
       read(0, comanda, sizeof(comanda));
       comanda[strlen(comanda)]='\0';

       if((write(sd, &comanda, sizeof(comanda))) <=0)
       {
          perror("[CLIENT]Eroare la write() spre server.\n");
          exit(EXIT_FAILURE);
       }

       if((read(sd, &raspuns, sizeof(raspuns))) <=0)
       {
          perror("[CLIENT]Eroare la read() de la server.\n");
          exit(EXIT_FAILURE);
       }

       raspuns[strlen(raspuns)-1] = '\0';
       printf("[CLIENT]Rezultatul primit este: %s\n",raspuns);
    }
```

4.2 Server

```c
int main()
{
   struct sockaddr_in server;
   struct sockaddr_in client;
   int sd;
   pthread_t th[100];
   int on = 1;
   int i = 0;

   if((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
   {
      perror("[SERVER]Eroare la socket().\n");
      exit(EXIT_FAILURE);
   }

   setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

   bzero(&server, sizeof(server));
   bzero(&client, sizeof(client));

   server.sin_family = AF_INET;
   server.sin_addr.s_addr = htonl(INADDR_ANY);
   server.sin_port = htons(PORT);

   if((bind(sd, (struct sockaddr *) &server, sizeof(struct sockaddr))) == -1)
   {
      perror("[SERVER]Eroare la bind().\n");
      exit(EXIT_FAILURE);
   }

   if(listen(sd,2) == -1)
   {
     perror("[SERVER]Eroare la listen().\n");
     exit(EXIT_FAILURE);
   }

   while(1)
   {
     int cd;
     socklen_t length = sizeof(client);
     thData *td;

     printf("[SERVER]Asteptam la portul %d...\n", PORT);
     fflush(stdout);

     cd = accept(sd, (struct sockaddr *) &client, &length);
     if(cd < 0)
     {
       perror("[SERVER]Eroare la accept().\n");
       continue;
     }

     td = (struct thData*)malloc(sizeof(struct thData));
     td->idThread = i++;
     td->cl = cd;

     pthread_create(&th[i], NULL, &treat, td);
   }
}
```

Functia *treat(void*): este executata de fiecare thread pentru a gestiona comunicarea cu clientul aferent thread-ului. Cand clientul isi termina activitatea, se incheie si executia thread-ului.

```c
static void *treat(void *arg)
{
   struct thData tdL;
   tdL = *((struct thData*)arg);

   printf("[thread] - %d - Asteptam mesajul...\n", tdL.idThread);
   fflush(stdout);
   pthread_detach(pthread_self());
   verificare_comanda((struct thData*)arg);

   close((intptr_t)arg);
   return(NULL);
}
```

Functia pentru verificarea comnzilor(void*): apelata in handler-ul mai sus mentionat, va realiza comunicarea efectiva cu clientul prin tratarea comezilor si trimiterea confirmarilor.

```c
int verificare_comanda(void *arg)
{
   char comanda[100];
   char raspuns[100]="";
   struct thData tdL;
   tdL= *((struct thData*)arg);

   while(1)
   {
      if((read(tdL.cl, &comanda, sizeof(comanda))) == -1)
      {
         printf("[thread %d]\n", tdL.idThread);
         printf("Eroare la read() de la client.\n");
      }

      comanda[strlen(comanda)-1]='\0';

      printf("[thread %d] Am primit comanda: %s\n",tdL.idThread, comanda);

      if(strstr(comanda, "conectare") != 0)
      {
         if((write(tdL.cl, &raspuns, strlen(raspuns))) == -1)
         {
           perror("Eroare la write() catre client.\n");
           exit(EXIT_FAILURE);
         }
      }
   }
}
```

5 Concluzii

Proiectul "Championship" este bazat pe modelul client/server folosind comunicarea TCP concurent. Aplicatia poate fi imbunatatita prin urmatoarele implementari:

1. In cazul reprogramarii unei partide, toti participantii trebuie sa fie de acord cu aceasta actiune, nu doar cel care o realizeaza.
2. Inregistrarea administratorilor sa se faca de un alt administrator deja conectat in aplicatie.
3. O alta imbunatatire ar consta in schimbarea protocolului. Un avantaj semnificativ al UDP/IP l-ar reprezenta viteza de transmisie superioara TCP/IP. Insa in acest scenariu retransmiterea datelor pierdute nu mai este posibila.

6 Bibliografie

1. https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
2. https://sites.google.com/view/fii-lab-retele-de-calculatoare/laboratoare
3. https://www.geeksforgeeks.org/multithreading-in-c/
4. https://www.geeksforgeeks.org/tcp-ip-model/
5. https://www.geeksforgeeks.org/introduction-to-sqlite/
6. https://app.diagrams.net/