QuizzGame

Bondor Laura-Genoveva 2A2

Universitatea Alexandru Ioan Cuza, Facultatea de Informatica, 700259 Iași, România
laurabondor05@gmail.com

1 Introducere

QuizzGame este o aplicatie sub forma unui joc, in care serverul este de tipul multithreading si accepta un numar nelimitat de jucatori(clienti). Serverul preia dintr-o baza de date intrebarile, pe care le adreseaza fiecarui jucator care s-a inregistrat. Acesta are la dispozitie un numar de n secunde pentru a raspunde la intrebare. Intrebarile vor fi de tip grila, iar raspunsurile jucatorilor ar putea fi de forma {a, b, c, d}. Dupa trecerea celor n secunde, serverul va verifica corectitudinea raspunsurilor si va retine/actualiza punctajul pentru fiecare jucator. In cazul in care, un jucator paraseste jocul, acesta va fi eliminat din rundele ulterioare de intrebari, jocul continuand sa se desfasoare fara acesta. La final, jucatorul care a acumulat cel mai mare punctaj va fi declarant castigator.

2 Tehnologiile utilizate

Pentru implementarea acestei aplicatii am folosit protocolul TCP pentru a ma asigura ca datele ajung la destinatia potrivita exact in ordinea in care au fost trimise. Acest lucru este absolut necesar in cazul acestei aplicatii deoarece trebuie sa asiguram faptul ca intrebarile transmise din server ajung la client si invers, ca raspunsurile acestora nu se vor pierde pe parcurs.

De asemenea, este important ca serverul sa fie implementat in mod concurent pentru a trata toti clientii in acelasi timp. Fiind vorba despre un joc, vom vrea sa participe simultan cat mai multi jucatori(clienti), astfel incat implementarea in mod concuret va face posibil acest lucru.

Mai mult decat atat, vom folosi un server multithreading pentru a raspunde mai rapid si eficient tuturor jucatorilor, nefiind necesar ca fiecare jucator sa astepte pana la finalizarea procesului de rulare. Acest lucru este posibil prin generarea unui fir de executie pentru fiecare jucator(client), folosind thread-uri, pentru a putea comunica in mod direct cu serverul. Deoarece firele de executie sunt independente unele de altele, inregistarea unui nou jucator sau parasirea jocului al unui membru din joc, nu va afecta celelalte fire de executie, astfel incat jocul va continua sa ruleze normal.

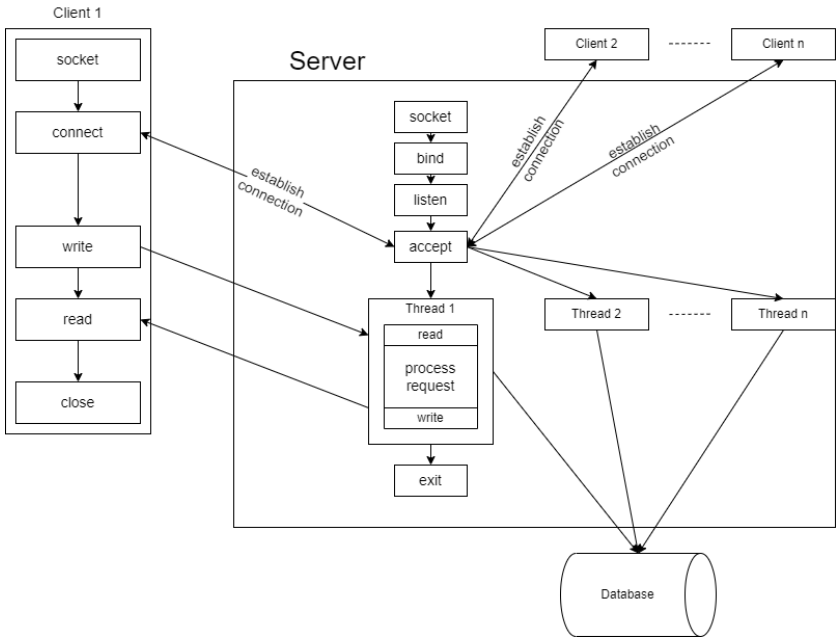
Pentru comunicarea intre client si server vom folosi socket-uri. Aceasta modalitate ne ajuta sa facem posibila comunicarea bidirectionala, un socket fiind posibil de utilizat atat pentru transmiterea cat si primirea datelor.

3 Arhitectura aplicatiei

Pentru crearea conexiunii intre client si server, vom trimite cererea pentru conexiune de la client catre server, iar serverul va fi cel care accepta cererea pentru conexiune din partea clientului/clientilor si va crea un fir de executie(thread) pentru fiecare client. In acest fel, serverul va putea transmite intrebarile catre jucatori si va putea primi raspunsurile acestora in mod simultan pentru toti jucatorii.

Vom folosi si o baza de date SQLite care va fi gestionata de server. Acesta va pune intrebari jucatorilor, apoi va astepta raspunsul acestora intr-un timp de n secunde, dupa care va verifica corectitudinea raspunsurilor pentru fiecare jucator si va actualiza punctajul unde este cazul. La final vom afisa castigatorul(clientul cu punctajul maxim).

In diagrama de mai jos am reprezentat conexiunea intre server si clienti:



4 Detalii de implementare

– Pentru inceput vom crea socketul atat in client, cat si in server:

```c
int sd;              // descriptorul de socket
/* cream socketul */
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
  perror("Eroare la socket().\n");
  return errno;
}
```

– Vom lega serverul la port si il vom pune sa asculte cererile de conexiune de la clienti:

```c
  /* umplem structura folosita de server */
  /* stabilirea familiei de socket-uri */
  server.sin_family = AF_INET;
  /* acceptam orice adresa */
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  /* utilizam un port utilizator */
  server.sin_port = htons(PORT);

  /* atasam socketul */
  if (bind(sd, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1)
  {
      perror("[server]Eroare la bind().\n");
      return errno;
  }

  /* punem serverul sa asculte daca vin clienti sa se conecteze */
  if (listen(sd, 1) == -1)
  {
      perror("[server]Eroare la listen().\n");
      return errno;
  }
```

– Vom lega clientul la portul IP si il vom conecta la server prin transmiterea unei cereri de conexiune:

```c
/* umplem structura folosita pentru realizarea conexiunii cu serverul */
/* familia socket-ului */
server.sin_family = AF_INET;
/* adresa IP a serverului */
server.sin_addr.s_addr = inet_addr(argv[1]);
/* portul de conectare */
server.sin_port = htons(port);

/* ne conectam la server */
if (connect(sd, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1)
{
  perror("[client]Eroare la connect().\n");
  return errno;
}
```

– Servim in mod concurent clientii si acceptam cererile acestora:

```
/* servim in mod concurent clientii...folosind thread-uri */
while (1)
  {
    int client;
    thData * td; //parametru functia executata de thread
    int length = sizeof (from);

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

    // client= malloc(sizeof(int));
    /* acceptam un client (stare blocanta pina la realizarea conexiunii) */
    if ( (client = accept (sd, (struct sockaddr *) &from, &length)) < 0)
{
  perror ("[server]Eroare la accept().\n");
  continue;
}
```

– Vom crea thread-uri(fire de executie) pentru fiecare client conectat la server:

```
// int idThread; //id-ul threadului
// int cl; //descriptorul intors de accept

td=(struct thData*)malloc(sizeof(struct thData));
td->idThread=i++;
td->cl=client;
td->score=0;

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

In continuare vom prelua intrebarile din baza de date pe care le vom pune clientilor, asteptand raspunsul lor intr-un timp de n secunde. Vom verifica raspunsurile, actualizand punctajele unde este cazul. . . s.a.m.d.

5 Concluzii

Cateva idei pentru care soulutia propusa ar putea fi imbunatatita, sunt:

– Afisarea scorurile tuturor jucatorilor participanti, inclusiv a unui podium cu primii 3
– După verificarea răspunsului primit de la client, serverul sa ii transmită acestuia mesajul ”Ai răspuns corect!”, in cazul in care jucătorul a raspuns corect si mesajul ”Raspunsul tau este gresit! Raspunsul corect era: ” si i se va transmite răspunsul care era corect.
– Afisarea unui timer a celor n secunde in care jucatorul va trebui sa raspunda
– Implementarea unei interfete grafice pentru acest joc

References

1. Autor: Lenuta Alboaie: **C**ursul 4,6,7
2. https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
3. https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c
4. https://profs.info.uaic.ro/ eonica/rc/lab04.html
5. https://profs.info.uaic.ro/ eonica/rc/lab06.html
6. https://profs.info.uaic.ro/ eonica/rc/lab07.html
7. https://profs.info.uaic.ro/ eonica/rc/lab11.html
8. https://www.geeksforgeeks.org/socket-programming-cc/
9. https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/
10. https://www.geeksforgeeks.org/multithreaded-servers-in-java/
11. https://www.geeksforgeeks.org/sql-using-c-c-and-sqlite/
12. https://app.diagrams.net/