Raport tehnic - QuizzGame

Golache Denisa-Ioana -2B4

Universitatea Alexandru Ioan Cuza, Facultatea de Informatică, Iași, România

1 Introducere

Proiectul propus presupune crearea unui server multithreading dedicat unui joc de întrebări și răspunsuri, conceput pentru a deservi un număr nelimitat de clienți simultan. **Scopul** proiectului constă în furnizarea unei experiențe interactive și sincronizate, unde fiecare client primește întrebări, răspunde într-un interval prestabilit, iar serverul se ocupă de gestionarea punctajelor și sincronizarea între participanți. **Obiectivele** principale ale proiectului includ înregistrarea clienților, încărcarea întrebărilor dintr-o bază de date specifică (**SQLite**), distribuirea întrebărilor către clienți, administrarea situațiilor în care un participant părăsește jocul și comunicarea câștigătorului tuturor clienților.

2 Tehnologii aplicate

În cadrul acestui proiect, am ales să utilizez limbajul de programare C pentru implementare, combinându-l cu tehnologii specifice pentru a dezvolta o aplicație bazată pe comunicarea prin **TCP și gestionarea concurentă a clienților**. Limbajul C oferă control detaliat asupra resurselor sistemului și este potrivit pentru dezvoltarea aplicațiilor de rețea eficiente.

Prin integrarea comunicării prin TCP, asigurăm o transmitere fiabilă și ordonată a datelor între server și clienți. Acest protocol este esențial pentru un joc interactiv, unde sincronizarea și coerența informațiilor sunt vitale.

Gestionarea concurentă a clienților este implementată pentru a permite unui număr nelimitat de participanți să interacționeze simultan cu serverul. Tehnologiile specifice includ utilizarea firelor de execuție (thread-urilor) pentru a gestiona eficient multiplele conexiuni în mod concurent. Această abordare permite serverului să comunice cu mai mulți clienți simultan, fără a afecta performanța generală a aplicației.

Prin combinarea limbajului de programare C cu tehnologii precum socketuri și fire de execuție, proiectul va beneficia de o implementare eficientă, ușor de gestionat și potrivită pentru contextul specific al unui joc de întrebări și răspunsuri cu participanți concurenți.
Proiectul va utiliza socket-uri pentru a facilita o comunicare fiabilă și ordonată între server și clienți, implementând TCP (**Transmission Control Protocol**) pentru aceast scop. TCP este un protocol orientat conexiune, ceea ce înseamnă că stabilește și menține o conexiune între server și clienți.

Protocolul TCP folosește un proces numit "3-way-handshaking" pentru a iniția și confirma o conexiune între server și clienți. Această procedură constă în etapele de sincronizare, sincronizare-acknowledgment și acknowledgment, asigurându-se astfel că ambele părți sunt pregătite să înceapă schimbul de date.

Conexiunile TCP sunt caracterizate de capacitatea lor full-duplex, ceea ce înseamnă că permit transmiterea și primirea datelor simultan și independent. Această funcționalitate este esențială pentru a asigura o interacțiune fluentă între server și clienți în cadrul jocului de întrebări și răspunsuri propus.

Prin implementarea TCP, proiectul beneficiază de o comunicare sigură, în care datele sunt transferate într-un mod secvențial și fără pierderi. Protocolul oferă, de asemenea, mecanisme incorporate pentru a gestiona congestiile și a asigura livrarea corectă a datelor, ceea ce este crucial în contextul unui joc interactiv unde sincronizarea și consistența sunt prioritare.

3 Structura Aplicației

Aplicația este divizată în două componente principale: server și clienți, conectate prin intermediul socket-urilor. Programarea cu socket-uri este o modalitate de a conecta două noduri pe o rețea pentru a comunica între ele. Un socket (nod) ascultă pe un port specific la o adresă IP, în timp ce celălalt socket inițiază conexiunea pentru a se conecta la celălalt. Serverul formează socket-ul ascultător, în timp ce clientul se conectează la server.

Serverul are responsabilitatea de a efectua operațiuni de citire și scriere către baza de date. Aici, sunt gestionate operațiuni precum înregistrarea utilizatorilor noi, stocarea scorurilor și obținerea întrebărilor și răspunsurilor. Serverul conține, de asemenea, logica pentru distribuirea întrebărilor către clienți în ordinea înregistrării, sincronizarea răspunsurilor și a punctajelor, gestionarea situațiilor în care un participant părăsește jocul și anunțarea câștigătorului către toți clienții la finalul jocului.

Implementarea serverului multithreading se realizează cu ajutorul bibliotecii POSIX Threads(pthread). Un fir de execuție este o singură secvență de instrucțiuni în cadrul unui proces. Firele de execuție sunt numite și procese ușoare, deoarece posedă unele dintre proprietățile proceselor. Fiecare fir de execuție aparține exact unui singur proces. Threadurile sunt diferite față de clasicele procese gestionte de sistemele de operare ce suportă multitasking, în principal prin faptul că, spre deosebire de procese, toate threadurile asociate unui proces folosesc același spațiu de adresare. Procesele sunt în general independente, în timp ce mai multe threaduri pot fi asociate unui unic proces. Procesele stochează un număr semnificativ de informații de stare, în timp ce threadurile dintr-un proces impart aceeași stare, memorie sau alte resurse. Procesele pot interacționa numai prin mecanisme de comunicare interproces speciale oferite de sistemul de operare (semnale, semafoare, cozi de mesaje și altele asemenea). Cum împart același spațiu de adresare, threadurile pot comunica prin modificarea unor variabile asociate procesului și se pot sincroniza prin mecanismele proprii. În general este mult mai simplu și rapid schimbul de informații intre

threaduri decât între procese. Va exista un fir de execuție (thread) pentru fiecare client care se conectează la server.

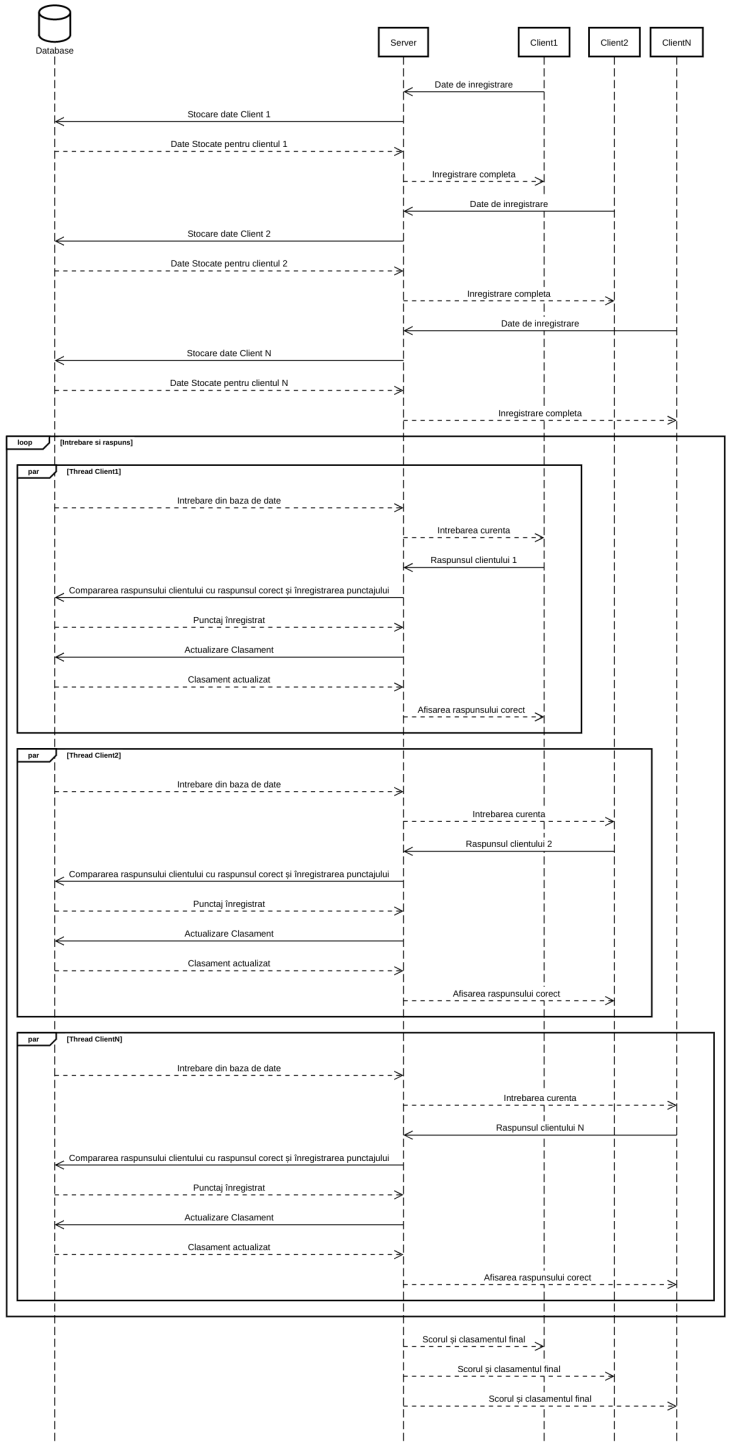
Clienții, în schimb, au un rol simplu și interacționează cu întrebările primite de la server. Aceștia primesc întrebările, răspund în intervalul de timp alocat și trimit răspunsurile înapoi către server pentru a fi procesate. Interfața simplă a clienților le permite să se concentreze pe participarea la joc fără a gestiona aspecte complexe ale logicilor de joc sau stocării datelor.
Prin această structură, serverul acționează ca o entitate centrală, coordonând desfășurarea jocului, iar clienții interacționează activ cu întrebările primite. Comunicarea eficientă între aceste două componente permite o experiență de joc fluidă și sincronizată pentru toți participanții.

Modul de gestionare a înregistrării clienților
Clientii se conectează la server pentru a putea participa la joc iar serverul acceptă conexiunea și îi atribuie fiecărui client un identificator unic pentru a putea fi identificat pe parcursul jocului. Informațiile despre utilizatori (nume, puntaj) sunt înregistrate în baza de date a aplicație pentru o evidență mai bună. În situația în care clienții folosesc același nume și parolă, serverul acceptă primul client care a folosit acele informații iar pe restul îi va refuza.

Modul de Încărcare a întrebărilor
Pentru gestionarea eficientă întrebărilor voi utiliza SQLite, o bază de date incorporată ușor în aplicație. Întrebarile vor avea variante de răspuns care vor fi transmise clienților dar va fi stoca și varianta de răspuns corectă pentru fiecare întrebare care ulterior va fi comparată cu răspunsul dat de client.

Gestionarea situațiilor de părăsire a jocului și anunțarea câștigătorului
Dacă un client dorește să părăsească jocul înainte de terminarea întrebărilor atunci lui îi va fi furnizat punctajul dar nu va fi luat în considerare în clasamentul final. Serverul va anunța rezultatele finale către toți clienții care încă sunt conectați, furnizând informații despre căstigator, dar și punctajul individual.

Diagrama de mai jos include componentele de bază, fluxurile de date dintre ele și modul în care datele sunt gestionate în timpul jocului.


Database
Server
Client1
Client2
ClientN
Date de inregistrare
Stocare date Client 1
Date Stocate pentru clientul 1
Inregistrare completa
Date de inregistrare
Stocare date Client 2
Date Stocate pentru clientul 2
Inregistrare completa
Date de inregistrare
Stocare date Client N
Date Stocate pentru clientul N
Inregistrare completa
loop
[Intrebare si raspuns]
par
[Thread Client1]
Intrebare din baza de date
Intrebarea curenta
Raspunsul clientului 1
Compararea raspunsului clientului cu raspunsul corect și înregistrarea punctajului
Punctaj înregistrat
Actualizare Clasament
Clasament actualizat
Afisarea raspunsului corect
par
[Thread Client2]
Intrebare din baza de date
Intrebarea curenta
Raspunsul clientului 2
Compararea raspunsului clientului cu raspunsul corect și înregistrarea punctajului
Punctaj înregistrat
Actualizare Clasament
Clasament actualizat
Afisarea raspunsului corect
par
[Thread ClientN]
Intrebare din baza de date
Intrebarea curenta
Raspunsul clientului N
Compararea raspunsului clientului cu raspunsul corect și înregistrarea punctajului
Punctaj înregistrat
Actualizare Clasament
Clasament actualizat
Afisarea raspunsului corect
Scorul și clasamentul final
Scorul și clasamentul final
Scorul și clasamentul final

4 Aspecte de Implementare

```
if (sqlite3_open("quizzgame_database.db", &db) != SQLITE_OK)
{
  perror("Eroare la deschiderea bazei de date\n");
  return errno;
}

if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
  perror("[server]Eroare la socket().\n");
  return errno;
}
```

În cadrul acestei porțiuni de cod deschidem baza de date prin intermediul funcției **sqlite3_open** pentru ca serverul să aibă acces la datele stocate în baza de date, iar după deschiderea ei se crează socket-ul, specificând tipul de protocol și familia de adrese. În cazul proiectului, se poate observa că se creează un socket TCP în familia de adrese **IPv4**, prin flag-urile specifice **AF_INET** și **SOCK_STREAM**

```
for (;;)
{
  int client;
  thData *td;
  int length = sizeof(from);
  printf("[server]Asteptam la portul %d...\n", PORT);
  fflush(stdout);
  if ((client = accept(sd, (struct sockaddr *)&from, &length)) < 0)
  {
    perror("[server]Eroare la accept().\n");
    continue;
  }
  td = (struct thData *)malloc(sizeof(struct thData));
  td->idThread = i++;
  td->cl = client;

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

În cadrul acestei porțiuni de cod, deservim în mod concurent clienții prin intrmediul thread-urilor. Serverul rulează într-o buclă infinită ceea ce înseamna că va aștepta și deservi clienți în mod continuu.

Funcția **accept** este utilizată pentru a accepta o conexiune de la un client. Serverul va sta blocat în această linie de cod până când un client se conectează. Funcția **pthread_create** este folosită pentru a crea un nou thread care va executa funcția **treat**. Astfel, fiecare client are propriul său thread dedicat pentru a gestiona comunicația cu serverul.

```
}
void *treat(void *arg)
{
  struct thData tdL;
  tdL = *((struct thData *)arg);
  int logged = 0;
  printf("[Thread %d] - Asteptam mesajul...\n", tdL.idThread);
  fflush(stdout);
  printf("[Thread %d] - Mesajul a fost receptionat.\n", tdL.idThread);

  pthread_detach(pthread_self());

  char question[MAX_SIZE];
  question[0]='\0';
  while (1)
  {
    const char *interogare = "SELECT intrebare, raspuns1, raspuns2, raspuns3 FROM intrebari ORDER BY RANDOM()";
    sqlite3_stmt *stmt;

    if (sqlite3_prepare_v2(db, interogare, -1, &stmt, NULL) == SQLITE_OK)
    {
      if (sqlite3_step(stmt) == SQLITE_ROW)
        snprintf(question, MAX_SIZE, "%s\nA. %s\nB. %s\nC. %s\n", sqlite3_column_text(stmt, 0), sqlite3_column_text(stmt, 1), sqlite3_column_text(stmt, 2), sqlite3_column_text(stmt, 3));
      else
        snprintf(question, MAX_SIZE, "Eroare la preluarea întrebării din baza de date");
    }
    else
      snprintf(question, MAX_SIZE, "Eroare la pregătirea interogării");

    send(tdL.cl, question, sizeof(question), 0);
    char user_response[MAX_SIZE];
    user_response[0]='\0';
    if (recv(tdL.cl, user_response, sizeof(user_response), 0) < 0)
    {
      perror("Eroare la primirea răspunsului de la client");
      break;
    }

    const char *correct_answer = "C";
    if (strstr(user_response, correct_answer) != NULL)
      send(tdL.cl, "Răspunsul este corect!\n", sizeof("Răspunsul este corect!\n"), 0);
    else
      send(tdL.cl, "Răspunsul este greșit.\n", sizeof("Răspunsul este greșit.\n"), 0);
  }
  close(tdL.cl);
  pthread_mutex_lock(&mlock);
  pthread_mutex_unlock(&mlock);
  pthread_exit(NULL);
}
```

Funcția treat este funcția executată de fiecare thread ce realizeaza comunicarea cu clienții. Funcția sqlite3_prepare_v2 este o funcție din biblioteca sqlite3.h, care este utilizată pentru a compila o interogare SQL. În cazul proiectului, vreau sa accesez tabelul ce conține întrebările si variantele de răspuns pentru a putea fi transmise clienților. Folosirea mutex-urilor asigură accesul sincronizat la resursele partajate în cod. Atunci când un fir de execuție intră într-o secțiune critică (o parte a codului care accesează resursa partajată), acesta poate activa mutex-ul, permițându-i doar lui să acceseze acea secțiune. Alte fire de execuție care încearcă să intre în aceeași secțiune critică vor aștepta până când mutex-ul este dezactivat de către firul care l-a activat.

În proiect, utilizarea mutex-urilor este benefică deoarece lucrez cu resurse partajate între diferite fire de execuție, adică baza de date. Aceasta asigură că operațiile sunt efectuate în mod sincronizat și previne conflictele care pot apărea atunci când mai multe fire de execuție încearcă să acceseze aceleași resurse simultan. Mutex-urile sunt esențiale pentru a evita problemele de concurență și pentru a asigura consistența și corectitudinea într-un mediu cu mai multe fire de execuție.

```
server.sin_family = AF_INET;

server.sin_addr.s_addr = inet_addr(argv[1]);
server.sin_port = htons(port);

if (connect(sd, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1)
{
  perror("[client]Eroare la connect().\n");
  return errno;
}
while (1)
{
  char question[MAX_SIZE];
  recv(sd, question, sizeof(question), 0);

  printf("%s", question);

  char user_response[MAX_SIZE];
  printf("Raspunsul clientului: ");
  fgets(user_response, sizeof(user_response), stdin);

  if (send(sd, user_response, sizeof(user_response), 0) < 0)
  {
    perror("[client]Eroare la send() spre server.\n");
    return errno;
  }

  char correct_response[MAX_SIZE];
  if (recv(sd, correct_response, sizeof(correct_response), 0) < 0)
  {
    perror("[client]Eroare la recv() de la server.\n");
    return errno;
  }
  printf("[client] Am citit raspunsul de la server. %s\n", correct_response);
}
close(sd);
}
```

În cadrul programului client are loc conexiunea cu serverul, prin intermediul funcției **connect**. În cadrul buclei, clientul primește întrebările și variantele de răspuns de la server, care au fost înregistrate în baza de date, prin intermediul functiei **recv** iar clientul trimite răspunsul înapoi la server pentru verificare prin intermediul functiei **send**.

Proiectul poate fi folosit în cadrul universităților și școlilor, pentru susținerea examenelor de tip grilă, pentru o evaluare rapidă a studenților și elevilor.

5 Concluzii

Proiectul oferă o experiență interactivă și captivantă pentru utilizatori, dar pot exista îmbunătățiri: optiuni personalizate pentru fiecare utilizator(alegerea gradului de dificultate, al domeniului), dar și realizarea unei interfețe grafice pentru aplicație.

References

https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
https://en.wikipedia.org/wiki/Transmission_Control_Protocol
https://profs.info.uaic.ro/ computernetworks/files/7rc_ProgramareaInReteaIII_Ro.pdf

https://www.geeksforgeeks.org/thread-in-operating-system/
https://www.geeksforgeeks.org/socket-programming-cc/
https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/
https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/
https://sqlite.org/docs.html