QuizGame

Hulubei Tudor

Facultatea de Informatică - Universitatea Alexandru Ioan Cuza Iași

Abstract. Documentul conține informații referitoare la QuizGame creat atât despre modul de operare al serverului, cât și despre client. Documentația este formată dintr-o introducere descriptivă, o scurtă prezentare a serverului implementat, secțiuni din cod și explicația protocolului implementat, modul de structurare a aplicației împreună cu o analiză a posibilelor îmbunătățiri.

1 Introducere

Proiectul QuizGame reprezintă o aplicație client/server care coordonează și sincronizează clienții în timp real într-un joc de întrebări. Utilizatorii vor avea acces la următoarele acțiuni:

Să răspundă la întrebări în ordinea în care s-au înregistrat. Fiecare client primește o întrebare și are un număr n de secunde pentru a răspunde la întrebare. Să primească feedback de la server despre corectitudinea răspunsului lor și să vizualizeze punctajul lor curent.

Toată logica aplicației va fi realizată în partea de server, astfel încât clientul doar va răspunde la întrebări, după ce s-au înregistrat. Întrebările cu variantele de răspuns vor fi stocate într-o bază de date SQLite. Serverul va gestiona situațiile în care unul dintre participanți părăsește jocul astfel încât jocul să continue fără probleme. Comunicarea între server și client se va realiza folosind thread-uri, thread-urile sunt utilizate pentru a gestiona cererile de la clienți.

2 Tehnologii Aplicate

În realizarea proiectului, am ales să utilizez un server TCP concurent. TCP (Transmission Control Protocol) este un protocol de transport orientat conexiune, fără pierdere de informații, în care acestea sunt primite în ordinea în care au fost transmise de către server. Am ales acest protocol datorită preciziei și integralității datelor transmise. Întrucât este vorba despre primirea clientului de la server a întrebărilor în ordinea conectării, informația transmisă trebuie să fie cât mai exactă și să nu se piardă, o abordare UDP nefiind convenabilă în acest context. Pentru asigurarea concurenței, am folosit thread-uri, întrucât reprezintă o metodă rapidă și eficientă de a diviza procesele pentru clienți. Deoarece thread-urile sunt independente unul față de celălalt, clientul poate face request-uri fără a fi afectat de acțiunile altui client.

3 Arhitectura Aplicației

Comunicarea cu baza de date se face prin intermediul librăriei sqlite3.h. Am ales să creez o astfel de bază de date, deoarece pot gestiona și modifica foarte ușor detaliile despre întrebări (variantele de răspuns, întrebarea, corectitudinea). Toate informațiile despre întrebări sunt stocate într-o tabelă Info, care conține următoarele coloane: Question, Answer a, Answer b, Answer c, CorrectAnswer.

Logica aplicației este următoarea: -se pornește timp de 10 secunde faza de înregistrare, timp în care clienții se conectează la server. După aceea, toți ceilalți clienți sunt respinși.

-se conectează și le este cerut username-ul

-serverul comunică cu baza de date SQLite și preia întrebarile

-serverul transmite clienților la daca s-au conectat in registration phase și după ce și au pus username-ul

-clienții un răspuns, într-un timp limitat de 10 secunde și îl trimit către server. Dacă nu este oferit un răspuns, întrebarea va primi 0 puncte și clientul o va primi pe următoarea

-serverul verifică răspunsul și îl validează, acordând un punctaj fiecărui jucător și calculând high score-ul de asemenea

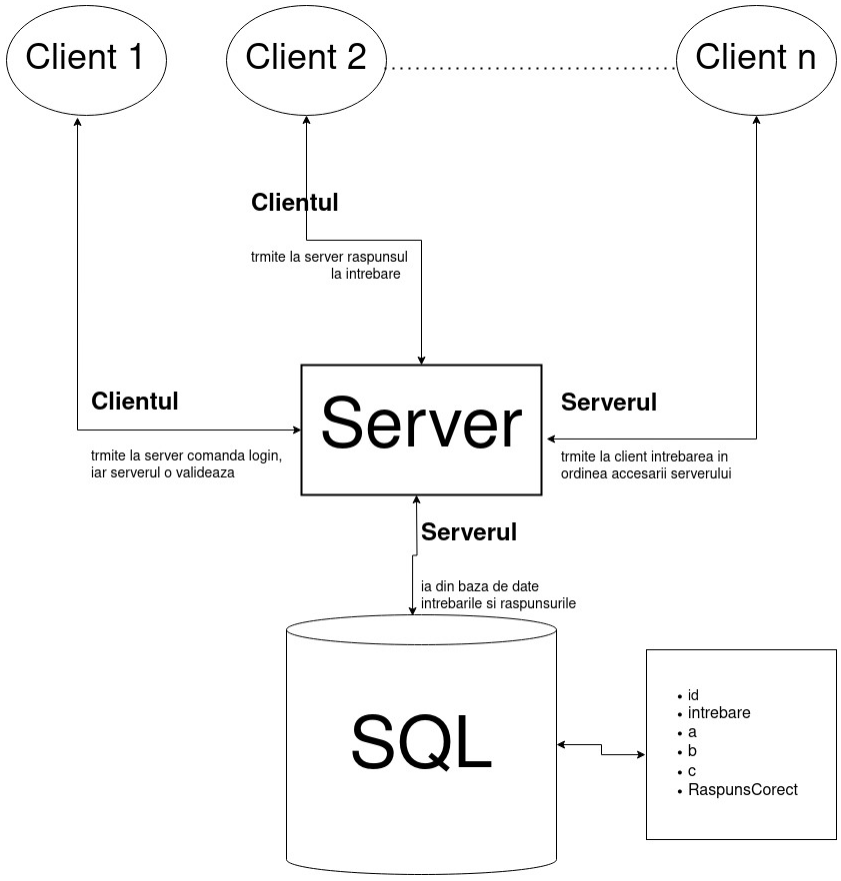

Fig. 1. Diagrama pentru Arhitectura Aplicației

4 Aspecte De Implementare

4.1 Registration Phase

Serverul, atunci cand se deschide, timp de 10 secunde este in registration phase. In aces stadiu, utilizatorul este acceptat sa se conecteze si se înregistrează pe platforma cu un username, ce va fi anunțat către toți clienții în cazul câștigătorului, impreuna cu scorul acestuia.

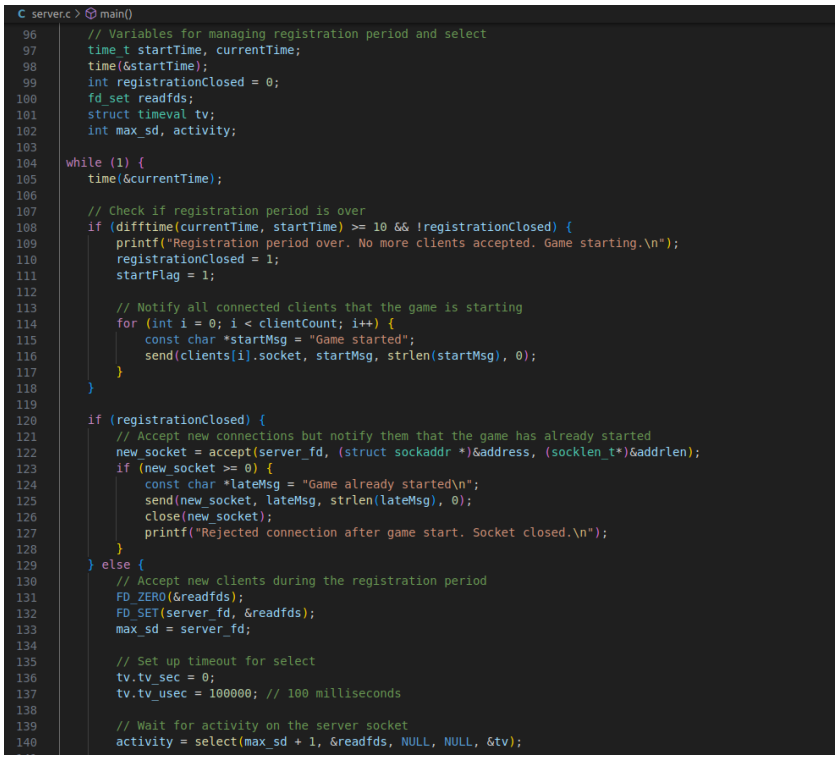

Fig. 2. Secvență cu cod din server

4.2 Protocolul TCP – creare si conectare

Crearea unui socket pe server (până la trecerea în starea pasivă de ascultare) și conectarea clientului la acest socket. Pentru fiecare client se creeaza un thread la server pentru comunicare. Se foloseste un semafor pentru a putea adauga fara probleme in array fiecare client. De asemenea, se foloseste un mutex pentru a evita data racing, in privinta variabilelor globale ce se ocupa de scorul maxim. Astfel, se creeaza un socket TCP astfel: Configurarea Socket-ului: Serverul creează

un socket TCP și îl leagă de un port specific. Acesta ascultă conexiunile care vin pe acest port. Acceptarea Clienților: În perioada de înregistrare, serverul acceptă conexiunile care vin și creează un nou fir de execuție, clientHandler, pentru fiecare client folosind pthreadcreate. Utilizează un semafor, clientSem, pentru a asigura că modificările aduse matricei de clienți sunt sigure în ceea ce privește execuția în firurile de execuție. Gestionarea Clienților: Fiecare fir de execuție al clientului interacționează cu clientul, trimițând întrebări din quiz și primind răspunsuri. Sincronizare: Serverul folosește un mutex adica scoreMutex și o variabilă de condiție, alldonecond, pentru a sincroniza accesul la resursele comune,scorul maxim, și pentru a gestiona starea quiz-ului, cum ar fi verificarea dacă toți clienții au terminat.

```
81    // creez socket
82    int server_sock = socket(AF_INET, SOCK_STREAM, 0);
83    if (server_sock == -1) {
84        perror("Socket creation failed");
85        exit(EXIT_FAILURE);
86    }
87
88    //adresa serverului
89    struct sockaddr_in server_addr;
90    server_addr.sin_family = AF_INET;
91    server_addr.sin_port = htons(PORT);
92    server_addr.sin_addr.s_addr = INADDR_ANY;
93
94    //conectez socketul la adresa
95    if (bind(server_sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
96        perror("Bind failed");
97        exit(EXIT_FAILURE);
98    }
99
100   //ascult clienții
101   if (listen(server_sock, MAX_CLIENTS) == -1) {
102       perror("Listen failed");
103       exit(EXIT_FAILURE);
104   }
```

```
11    // creez socket-ul
12    int client_sock = socket(AF_INET, SOCK_STREAM, 0);
13    if (client_sock == -1) {
14        perror("Socket creation failed");
15        exit(EXIT_FAILURE);
16    }
17
18    // server configuration
19    struct sockaddr_in server_addr;
20    server_addr.sin_family = AF_INET;
21    server_addr.sin_port = htons(PORT);
22    server_addr.sin_addr.s_addr = INADDR_ANY;
23
24    // ma conectez la server
25    if (connect(client_sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
26        perror("Connection failed");
27        close(client_sock);
28        exit(EXIT_FAILURE);
29    }
```

Fig. 4. Secvență cu cod din client

4.3 SQLite

În codul serverului prezentat, conexiunea cu baza de date este realizată folosind SQLite, o bibliotecă în limbajul C care oferă un sistem de gestiune a bazelor de date relaționale. Funcția loadQuestionsFromDB() este responsabilă pentru încărcarea întrebărilor din baza de date. Folosește sqlite3open("questions.db", db)

pentru a deschide o conexiune cu baza de date questions.db. Dacă nu reușește, afișează un mesaj de eroare și iese din funcție.

După deschiderea cu succes a bazei de date, pregătește o interogare SQL folosind sqlite3preparev2. Interogarea SQL este: "SELECT Question, AnswerA, AnswerB, AnswerC, CorrectAnswer FROM Questions;". Aceasta interogare este destinată extragerii întrebărilor și a răspunsurilor din tabelul Questions.

Utilizează sqlite3step(stmt) într-o buclă pentru a itera prin rândurile returnate de interogare. Pentru fiecare rând, extrage informațiile despre întrebare și răspunsuri și le stochează în array-ul questions

La finalul extragerii datelor, se închide interogarea (sqlite3finalize(stmt)) și conexiunea cu baza de date (sqlite3close(db)).

4.4 Analiza gestionării erorilor

Tratarea erorilor se concentrează în principal pe gestionarea situațiilor în care o funcție internă eșuează. Prin urmare, monitorizarea codurilor returnate de acestea și transmiterea unui mesaj de eroare sugestiv ajută la gestionarea eficientă a erorilor. Cateva gestionări relevante:

1. Deschiderea Socket-ului și Setările Acestuia

Erorile din procesul de creare și configurare a socket-ului sunt tratate folosind perror și exit(EXITFAILURE).

Acest lucru asigură că programul se încheie imediat dacă socket-ul nu poate fi creat, prevenind astfel orice acțiuni ulterioare pe un socket invalid.

2. Bind și Listen

Similar, bind și listen sunt urmate de verificări ale erorilor. Dacă aceste apeluri eșuează, programul afișează un mesaj de eroare și se încheie.

3. Acceptarea Conexiunilor Client

În bucla principală, serverul acceptă conexiuni de la clienți. Dacă accept eșuează, afișează un mesaj de eroare, dar nu încheie programul. Acest lucru permite serverului să continue să funcționeze chiar și dacă o singură tentativă de conectare eșuează.

4. Crearea Thread-urilor

La crearea unui nou thread, este verificată întoarcerea funcției pthreadcreate. În cazul unei erori, afișează un mesaj de eroare.

5. Interacțiunea cu Baza de Date

Deschiderea bazei de date, pregătirea interogărilor și execuția acestora sunt însoțite de verificări ale erorilor. În caz de eșec, se afișează mesaje de eroare și se închid resursele corespunzătoare.

6. Comunicarea cu Clienții

În clientHandler, citirea de la și scrierea către socket-ul clientului sunt urmate de verificări ale erorilor. Dacă o citire sau scriere eșuează, se închide socket-ul clientului.

5 Concluzii și îmbunătățiri

Ar trebui să se dezvolte o interfață grafică pentru comenzile pe care clientul le poate executa, pentru a face aplicația mai atrăgătoare, intuitivă și ușor de utilizat.

Posibilitatea alegerii unor moduri diferite de joc și vizualizarea în timp real a numărului de răspunsuri date.

6 Bibliografie

1.Model de server TCP concurent implementat cu thread-uri

$https://www.geeksforgeeks.org/handling-multiple-clients-on-server-with-multithreading-using-socket-programming-in-c-cpp/$

2. Database Connection in C++ with SQLite

$https://www.youtube.com/watch?v=cSZvq7Kv6_0$

3.Mutex usage

$https://en.cppreference.com/w/cpp/thread/mutex$