Quizz Game

Autorul : Simioniuc Ionut

[1] Instituția: Faculty of Computer Science ,Iasi,Romania
[2] Email : simioniucionut1@gmail.com

Abstract. Quizz Game este un proiect care implementează un server multithreading care coordonează în mod sincronizat și concurent accesul mai multor clienți la server. Acest document prezintă detaliile de implementare și structura proiectului, inclusiv aspecte legate de gestionarea clienților,baze de date și sincronizarea clienților.

Keywords: Multithreading · Socket · SQLLite · Mutex · TCP .

1 Introducere

Viziunea generală: Scopul proiectului este de a implementa un server care să permită conectarea a n clienți într-un mod sigur, rapid și sincronizat. După apelarea funcției "START", nu se vor mai putea conecta alți clienți la Quizz-ul respectiv. Întrebările vor fi preluate din baza de date SQLite si trimise de către server în mod concurat către toți clienții, aceștia având un timp de n secunde să răspundă pentru fiecare întrebare.Daca nu exista o baza de date,serverul va crea una standard.La finalul jocului se va prezenta un LeaderBoard cu câștigătorii Jocului.

Obiectivele proiectului:
-Conectarea a n clienți la server, într-un mod sigur și eficient.
-Crearea unui server multithreading concurent.
-Implementarea tabelei de întrebări prin baza de date SQLite.
-Trimiterea unui LeaderBoard catre client care sa includa mai multe detalii despre
fiecar jucator inregistrat.
-Crearea unui cronometru grafic in client.
-Serverul va fi capabil sa ruleze iar sesiunea de Quizz.

2 Tehnologii Aplicate

Pentru implementarea proiectului, se folosește tehnologia de comunicare TCP, deoarece aceasta este mai sigură, păstrează integritatea datelor și consistența acestora, trimiterea datelor către client fiind sigură și sincronizată.
O altă tehnologie folosită este baza de date SQLite care stochează întrebările și variantele de răspuns ale jocului.
Folosirea bazei de date SQLite îmi oferă o performanță și o siguranță mult mai bună.

3 Structura Aplicației

Concepte folosite:
-TCP: Realizarea unui server si a mai multor clienti prin care se comunica.
-Socket: Serverul va avea un socket pentru a realiza conexiunea si comunicarea cu alti clienți.
-SQLite: O baza de date pentru stocarea intrebarilor si a raspunsurilor.
-MultiThreading: Utilizarea thread-urilor pentru a gestiona conexiunile multiple în mod concurent.
-Mutexuri: Utilizarea mutex-urilor pentru a sincroniza accesul la resurse partajate între thread-uri si pentru a evita Race Conditions.

A se accesa diagrama aplicației la linkul indicat : Diagrama

4 Aspecte de Implementare

Protocolul la nivel de aplicație: Când se vor conecta clienții la server, se vor crea threaduri pentru fiecare client, apoi clientii se vor înregistra, urmând să aștepte un semnal de start trimis de către server.Semnalul de start va fi creat de un alt thread.După începere, serverul trimite in mod concurent întrebări ,sub forma unor structuri specifice, către toți clienții; Clientii trimit răspunsurile înapoi către server intr un timp de n secunde, urmând ca serverul să trimită următoarea întrebare. La final, serverul trimite către toți clienții Leaderboard-ul cu scorurile obținute de către toți participantii ,putand apoi sa ruleze alta sesiune de Quizz.
Scenarii de utilizare:

I Un scenariu de utilizare ar fi la facultate, atunci când la finalul cursului profesorul dorește să vadă cât de bine au înțeles studenții cursul. Astfel, profesorul verifică înțelegerea fiecărui student prin intermediul unui quiz, punând studenții să răspundă la un set de întrebări referitoare la cursul respectiv. Profesorul va putea, la final, să vadă, prin punctajul obținut, cât de bine s-a înțeles cursul respectiv.

II Un alt scenariu ar fi la evaluarea unui test. Studenții pot vedea punctajul doar la final, după ce toți ceilalți colegi au terminat de completat quiz-ul. Datorită timpului de n secunde per întrebare, studenții nu vor putea frauda examenul, deoarece, dacă aceștia nu răspund la timp la o întrebare, vor primi 0 puncte pentru întrebarea respectivă.
Sectiuni de cod Specifice:

Poza cu implementarea tabelului de întrebări. :

Implementarea secțiunii de START:
În Fig.1 → zona de creare a thread-urilor și de acceptare a conexiunilor atunci

când Quizz-ul nu a început.
În Fig.2 → un mecanism prin care Serverul respinge toți ceilalți clienți care încearcă să se conecteze după începerea jocului Quizz.
În Fig.3 → funcția care citește din terminal-ul serverului comanda pentru începere.
Se folosesc lacăte pentru a evita scenariul când un thread se creează în momentul în care deja am dat Start, astfel ca acel thread să nu se mai creeze, folosesc mutex.

```
172         while (1)
173         {
174           int client;
175           int length = sizeof (from);
176           printf("start%d\n",Start);
177
178           if ( (client = accept (sd, (struct sockaddr *) &from, &length)) < 0)
179           {
180             perror ("[server]Eroare la accept().\n");
181             continue;
182           }
183
184           if(Start!=1)
185           {
186             threadCreate(client);
187           }
188           else
189           {
190
191             Count DejaInceput;
192             DejaInceput.questions=-1;
193             DejaInceput.threads=-1;
194            if (write (client, &DejaInceput, sizeof(Count)) <= 0)
195            {
196             perror ("Clientul a intampinat o eroare imediat dupa ce a inceract sa se conecteze la server.\n");
197            }
198
199             close(client);
200           }
201
202         }
203
```

Fig. 1: main()

```
436 void threadCreate(int client){
437     void *treat(void *);
438     void modificariClient(int ,int , int ,char *,long,long,long);
439     void *StartFunction(void *);
440     void leaderboard();
441       //zona de creare de thread
442     threadData * td;
443     pthread_mutex_lock(&mutex_lock);
444     if(Start != 1)
445     {
446
447       td=(threadData*)malloc(sizeof(threadData));
448       td->idThread=nrthreads++; //nrthreads trebuie actualizata intre lacate
449       td->clientSocket=client;
450
451        if (td == NULL) {
452         // Eroare la alocare
453           printf("ERPARE la alocare threadului \n");
454           exit(-1);
455         }
456
457      //crestem numarul de clienti din quizz
458      nrClientsFinished++;
459     }
460     else
461     {
462       pthread_mutex_unlock(&mutex_lock); //deblocam lacatul inainte de a iesi
463        exit(-1); // daca se creaza un thread in acelasi timp cand s-a dat deja
464     }
465     pthread_mutex_unlock(&mutex_lock);
466
467     pthread_create(&th[nrthreads], NULL, &treat, td);
468 }
```

Fig. 2: threadCreate()

```
void *StartFunction(void *arg) {
    int input;
    printf("Introduceți 1 pentru a începe Quizz ul: \n");
    scanf("%d", &input);
    if (input == 1) {
        printf("Quizz ul  a început!\n");

     pthread_mutex_lock(&mutex_lock);

      if(nrthreads==0){

         printf("Nu puteti incepe fara jucatori\n");

         pthread_create(&StartThread, NULL, StartFunction, NULL); //cream iar t
      }
      else{
       Start = 1;}
     pthread_mutex_unlock(&mutex_lock);
    }else{
        printf("Opțiune nevalidă. Serverul rămâne in asteptare de clienti.\n");
    }
    pthread_exit(NULL);
}
```

Fig. 3: startFunction()

5 Concluzii

Eventuale îmbunătățiri:
-Se va dezvolta un site web prin intermediul căruia clienții pot să se înregistreze și să răspundă la chestionare. Site-ul va fi găzduit prin intermediul unui server.
-Creșterea securității și scalabilității sistemului.

6 Referințe Bibliografice

Referințele pentru tehnologiile folosite:
documentația limbajului C : C Documentation.com
biblioteca de rețea : GNU Networking.com
alte surse : Computer Networks.com