Raport Tehnic: QuizzGame

Hurmuzache Maria-Ioana

Universitatea Alexandru Ioan Cuza Iași, Facultatea de Informatică

Abstract. Acest raport tehnic prezintă viziunea generală și obiectivele proiectului QuizzGame și cuprinde informații despre tehnologii utilizate, structura aplicației, implementare și concluziile obținute.

1 Introducere

În cadrul proiectului QuizzGame îmi doresc să realizez o aplicație interactivă și distractivă, concepută pentru a oferi utilizatorilor o experiență competitivă, captivantă. Jucătorii (clienții) se vor putea conecta la joc prin introducerea numelui lor. Jucatorii vor avea un timp limită de înscriere, iar condiția de începere a jocului este ca minim doi jucători sa fie conectați. Sarcina participanților va fi aceea de a răspunde unei serii de întrebări, într-un timp limită prestabilit. În funcție de răspuns, jucătorii vor acumula un anumit punctaj. La finalul jocului, fiecărui participant îi vor fi transmise punctajul pe care l-a obținut precum și clasamentul sesiunii de joc.

2 Tehnologii Aplicate

În scopul elaborării proiectului, voi crea un server TCP concurent. Alegerea acestui protocol este motivată de siguranța pe care o oferă la transmiterea datelor, principalul scop al programului fiind acela de a transmite informații de la client la server și vice versa. Concurența server-ului va fi asigurată prin multithreading, server-ul creând un număr de thread-uri introdus de administratorul jocului, acestea rezolvând sarcinile care apar. Fiecare astfel de thread va verifica dacă există o sarcină de rezolvat (sarcinile vor fi plasate într-o coadă în momentul apariției lor), iar în caz afirmativ va prelua prima sarcină din coadă. Pentru a gestiona accesul thread-urilor la zonele de memorie partajată (exemplu: vectorul care conține întrebările, coada de sarcini, etc.), voi folosi o variabilă mutex (pthread_mutex_t) pentru a asigura accesul exclusiv la „secțiunea critică”. Pentru sincronizărea thread-ului principal am creat variabile globale de tip condiție (pthread_cond_t), câte una pentru fiecare etapă a programului (acceptarea conexiunilor, actualizarea numelui fiecarui jucator, actualizarea punctajelor pe baza intrebarilor și trimiterea clasamentului), pentru a semnala trecerea la următoarea etapă. Nu în ultimul rând, am utilizat o bază de date sqlite pentru a reține întrebările și variantele de răspuns ale acestora.

3 Structura Aplicației

Aplicația este structurată în două entități principale, server-ul și clienții. La acestea se adaugă baza de date, menită să rețină informațiile privitoare la teste (întrebări și variante de răspuns), până când acestea vor fi accesate de server.

Clientul prezintă două funcționalități, de receptare și de trimitere a mesajelor dinspre și către server. Acesta are responsabilitatea de a afișa întrebările primite de la server și de a trimite răspunsurile introduse de jucător înapoi către server.

Server-ul are misiunea de a implementa cerințele administratorului jocului, de a stoca informațiile participantului (clientul) și de a prelucra răspunsurile primite de la acesta în scopul actualizării unora dintre informațiile clientului, cum ar fi numele și punctajul acestuia. Mai mult decât atât, după inițierea sesiunii, conectarea jucărorilor și începerea jocului, server-ul interacționează cu baza de date, extrăgând datele necesare susținerii chestionarului. Aceste date vor fi transmise succesiv fiecărui client de către server care va aștepta răspunsul clientului și va actualiza punctajele în funcție de situație. La final, server-ul va stabili punctajul clasamentul sesiunii de joc și va trimite fiecărui client o vopie a acestuia, care conține numele participanților și punctajele pe care le-au obținut.

3.1 Diagramă

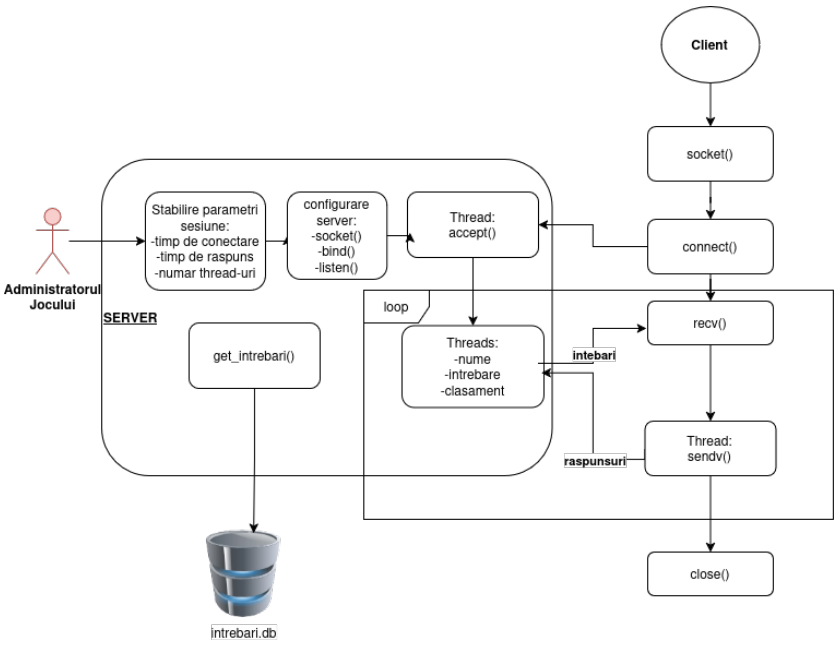

Fig. 1. Structura aplicației și interacțiuniea dintre entitățile acesteia

4 Aspecte de Implementare

4.1 Secvențe de cod

Listing 1.1. Clasa asociata entitatii client

```
class Client
{
    private:
    char* nume;
    int descriptor;
    int raspunsuri;
    int punctaj;
    bool active;
    public:
    Client(int descriptor);
    int GetDescriptor();
    void SetNume(const char*);
    const char* GetNume();
    int GetPunctaj();
    int GetRaspunsuri();
    void ActualizarePunctaj(int puncte);
    void ActualizareRaspunsuri();
    bool GetStatus();
    void Inactive();

};
```

Listing 1.2. Exemplu de utilizare a mutex-ului și a unei condiții. Codul este extras din funcția treat() și are următoarea funcționalitate: anunță thread-ul principal când clienții conectați și-au introdus numele prin pthread_cond_signal(&cond); de asemenea, pentru fiecare client conectat se adaugă în coada de sarcini pointer-ul către obiectul "Client" asociat clientului respectiv și sarcina de a i se trimite prima intrebare; mutex-ul asigură exclusivitatea thread-ului la variabila de condiție, la coada de sarcini, la vectorul de clienti și la restul variabilelor globale accesate.

```
pthread_mutex_lock(&mutex);
jucatori.push_back(c);
gata++;
if(gata==nrclienti)
{
    pthread_cond_signal(&cond);
    gata=0;
}
pair<Client*,string> s;
s.first=c;
s.second="intrebare";
sarcini.push(s);
pthread_mutex_unlock(&mutex);
```

Listing 1.3. Funcțiile de accesare și preluare a informațiilor din baza de date "intrebari".

```
void get_questions()
{
    char select[BUFFER_SIZE];
    sprintf(select, "select * from test;");
    char* errmsg=NULL;
    rc=sqlite3_exec(db, select, callback, NULL, &errmsg);
    if(rc!=SQLITE_OK)
    {
        fprintf(stderr, "SQL error: %s\n", errmsg);
        sqlite3_free(errmsg);
    }
}

static int callback(void *data, int nr_col, char** atribute,
    char** coloana)
{
    int id=atoi(atribute[0]);
    test[id-1].id=id;
    test[id-1].question=(char*)malloc(strlen(atribute[1]));
    strcpy(test[id-1].question,atribute[1]);
    test[id-1].A=(char*)malloc(strlen(atribute[2]));
    strcpy(test[id-1].A,atribute[2]);
    test[id-1].B=(char*)malloc(strlen(atribute[3]));
    strcpy(test[id-1].B,atribute[3]);
    test[id-1].R=(char*)malloc(strlen(atribute[4]));
    strcpy(test[id-1].R,atribute[4]);
    return 0;
}
```

4.2 Protocol la Nivelul Aplicației

Listing 1.4. Funcție de trimitere a unui mesaj de la client către server. Mesajul este precedat de lungimea sa, pentru a putea fi verificată integritatea datelor. O funcție similară se găsește și la nivelul server-ului.

```
void* trimite_mesaj(void* arg)
{
    int sd=*(int*)arg;
    char buffer[BUFFER_SIZE];
    while(1)
    {
        bzero(buffer, sizeof(buffer));
        fgets(buffer, sizeof(buffer), stdin);
        int lungime=strlen(buffer);
        if(-1==send(sd, &lungime, sizeof(int), 0))
        {
```

```
12              perror("[client]: Eroare la trimiterea lungimii
                    mesajului");
13              break;
14          }
15          if(-1==send(sd, buffer, lungime, 0))
16          {
17              perror("[client]: Eroare la trimiterea mesajului"
                    );
18              break;
19          }
20      }
21      return NULL;
22  }
```

Listing 1.5. Buclă asociată primirii mesajelor în client. Mesajul conține lungimea textului și textul propriu-zis. O funcție similară se regăsește în server.

```
1   while(1)
2   {
3       int length=0;
4       if(-1==recv(sd, &length, sizeof(int), 0))
5       {
6           perror("[client]: Eroare la recv() de lungimea
                mesajului");
7           break;
8       }
9       char* mesaj=malloc((length+1)*sizeof(char));
10      int bytes_received=0;
11      while(bytes_received<length)
12      {
13          int r=recv(sd, mesaj+bytes_received, length-
                bytes_received,0);
14          if(-1==r)
15          {
16              perror("[client]: Eroare la recv() de mesaj");
17              free(mesaj);
18              exit(1);
19          }
20          if(0==r)
21          {
22              perror("[client]: Server-ul a fost oprit");
23              free(mesaj);
24              break;
25          }
26          bytes_received+=r;
27      }
28      mesaj[length]='\0';
29      printf("%s\n", mesaj);
30      if(strcmp(mesaj,"\0")==0)
31      {
```

```
32          free(mesaj);
33          break;
34      }
35      free(mesaj);
36  }
```

Listing 1.6. Secvență din implementarea funcției asociată thread-urilor ce au ca scop rezolvarea sarcinilor. Se observă utilizarea mutex-urilor la accesarea datelor partajate.

```
1  void* treat(void* arg)
2  {
3      while(1)
4      {
5          char buffer[BUFFERSIZE];
6          string sarcina="";
7          Client* c=nullptr;
8
9          pthread_mutex_lock(&mutex);
10         if(sarcini.empty()==false)
11         {
12             sarcina=sarcini.front().second;
13             c=sarcini.front().first;
14             sarcini.pop();
15         }
16         pthread_mutex_unlock(&mutex);
17
18        /////.../////
19         else if(sarcina=="intrebare")
20         {
21             int i=c->GetRaspunsuri();
22             bzero(buffer, sizeof(buffer));
23             sprintf(buffer, "Intrebarea %d: %s\nVariante de
                   raspuns:\nA: %s\nB: %s\nIntroduceti majuscula
                    corespunzatoare raspunsului corect:", test[i
                   ].id, test[i].question, test[i].A, test[i].B)
                   ;
24             trimite_mesaj(c->GetDescriptor(), buffer);
25             int puncte=timed_response(c->GetDescriptor(), i);
26             c->ActualizarePunctaj(puncte);
27             c->ActualizareRaspunsuri();
28             if(c->GetRaspunsuri()<nrintrebari && puncte>=0)
29             {
30                 pthread_mutex_lock(&mutex);
31                 pair<Client*,string> s;
32                 s.first=c;
33                 s.second="intrebare";
34                 sarcini.push(s);
35                 pthread_cond_signal(&cond);
36                 pthread_mutex_unlock(&mutex);
37             }
```

```
38              else
39              {
40                  if(puncte<0)
41                  {
42                      c->Inactive();
43                  }
44                  pthread_mutex_lock(&mutex);
45                  gata++;
46                  if(gata==nrclienti)
47                  {
48                      pthread_cond_signal(&cond1);
49                      gata=0;
50                  }
51                  pthread_mutex_unlock(&mutex);
52              }
53          }
54         ///.../// 
55      }
56      return 0;
57  }
```

4.3 Scenarii reale de utilizare

Aplicația ar putea fi utilizată atât în scop recreativ, pentru testarea culturii generale individuale în raport cu prieteni sau membri ai familiei, dar și în scop educativ, pentru testarea cunoștințelor la o anumită disciplină. De exemplu, aplicația poate fi utilizată pentru susținerea unui unui test grilă la disciplina Biologie pentru a evalua ce au reținut elevii de clasa a 11-a în urma lecției despre Sistemul Nervos.

5 Concluzii

O primă îmbunătățire ar putea fi implementarea posibilității creării unui cont de utilizator, prin intermediul căruia un jucător ar putea să își creeze propriu test pe baza unui formular primit de la server, noul test urmând să fie memorat în baza de date. Având mai multe teste la dispoziție, un participant ar putea inița o sesiune de joc având posibilitatea de a alege un anumit test.

6 Referințe Bibliografice

References

1. Suporturile de curs, `https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php`
2. Suporturile de laborator, `https://profs.info.uaic.ro/andrei.scutelnicu/`
3. Pentru realizarea diagramei, `https://sequencediagram.org/`

4. Springer: LNCS Guidelines, https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines
5. SQLiteStudio, https://sqlitestudio.pl/
6. SQLite - C/C++, https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm
7. pthread_create(3) — Linux manual page, https://man7.org/linux/man-pages/man3/pthread_create.3.html
8. realloc(3p) — Linux manual page, https://man7.org/linux/man-pages/man3/realloc.3p.html
9. pthread_cond_broadcast(3p) — Linux manual page, https://man7.org/linux/man-pages/man3/pthread_cond_broadcast.3p.html
10. pthread_cond_wait(3p) — Linux manual page, https://man7.org/linux/man-pages/man3/pthread_cond_timedwait.3p.html
11. pthread_mutex_lock(3p) — Linux manual page, https://man7.org/linux/man-pages/man3/pthread_mutex_lock.3p.html
12. Vector in C++ STL — GeeksforGeeks, https://www.geeksforgeeks.org/vector-in-cpp-stl/
13. Queue in C++ Standard Template Library (STL) - GeeksforGeeks, https://www.geeksforgeeks.org/queue-cpp-stl/?ref=shm