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. Odată cu înscrierea primului jucător, ceilalți vor avea o limită de timp pentru a se conecta. Sarcina participanților va fi aceea de a răspunde unei serii de întrebări, într-un timp limită prestabilit. În funcție de răspuns, dar și de viteza de reacție, jucătorii vor acumula un anumit punctaj. La finalul jocului, fiecărui participant îi vor fi transmise cel mai mare punctaj obținut precum și numele câștigătorului/câștigătorilor jocului (cei care obțin scorul maxim).

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 thread separat pentru fiecare client conectat. Pentru a gestiona accesul thread-urilor la zonele de memorie partajată (exemplu: vectorul care conține întrebările), voi folosi o variabilă mutex (pthread_mutex_t) pentru a asigura accesul exclusiv la „secțiunea critică". Pentru sincronizărea thread-urilor am creat două variabile globale de tip condiție (pthread_cond_t), una pentru acțiuni în cadrul thread-urilor corespunzătoare clienților care încep la semnalul thread-ului principal si una pentru cazul când în thread-ul princpal este semnalizată terminarea unei anumite sarcini în thread-urile corespunzătoare clienților. 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 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 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 maxim acumulat și va trimite fiecărui client o listă a câștigătorilor, care conține numele participanților care au obținut punctajul maxim.

3.1 Diagramă

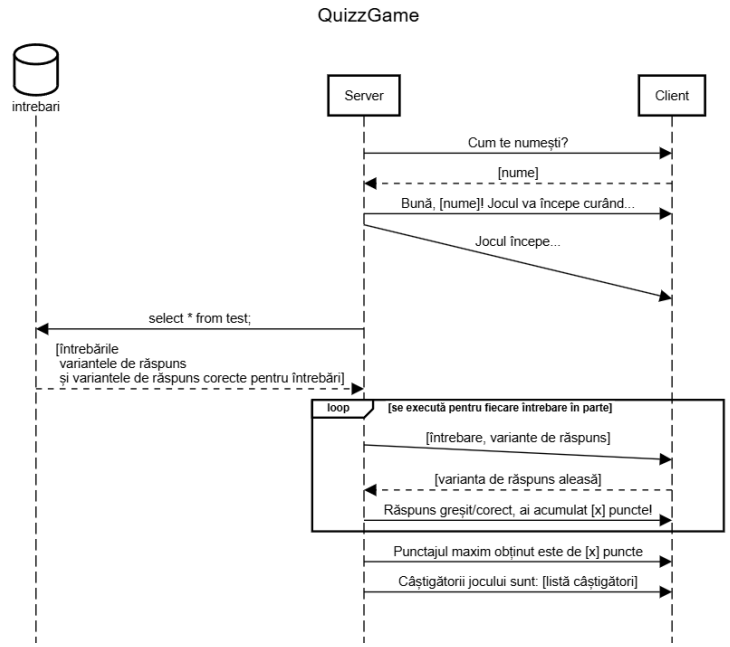

Fig. 1. Structura aplicației și interacțiuniea dintre entitățile acesteia

4 Aspecte de Implementare

4.1 Secvențe de cod

Listing 1.1. Crearea unui nou thread pentru clientul abia conectat

```
p=(struct player*)malloc(sizeof(struct player));
p->nume=NULL;
p->id=nrPlayers++;
p->client_descriptor=client;
p->punctaj=0;

if(pthread_create(&threads[nrPlayers-1], NULL, handle_client,
     p)<0)
{
     perror("[server]: Eroare la crearea thread-ului");
     free(p);
     if(nrPlayers!=1)
     {
          nrPlayers--;
     }
     else
     {
          free(threads);
          nrPlayers--;
     }
     continue;
}
```

Listing 1.2. Exemplu de utilizare a mutex-ului și a celor două condiții. Codul este extras din funcția handle_client() și are următoarea funcționalitate: anunță thread-ul principal când timpul de conectare a expirat și toți clienții deja conectați și-au introdus numele prin pthread_cond_broadcast(&condp), apoi așteaptă ca thread-ul principal să trimită semnalul de continuare a sarcinilor prin pthread_cond_wait(&cond, &mutex); mutex-ul asigură exclusivitatea thread-ului la variabilele de condiție accesate.

```
pthread_mutex_lock(&mutex);
if(difftime(time(NULL), start)>TIME_LIMIT && p.id==(nrPlayers
    -1))
{
     pthread_cond_broadcast(&condp);
}
pthread_mutex_unlock(&mutex);

pthread_mutex_lock(&mutex);
pthread_cond_wait(&cond, &mutex);
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;");
```

```
5       char* errmsg=NULL;
6       rc=sqlite3_exec(db, select, callback, NULL, &errmsg);
7       if(rc!=SQLITE_OK)
8       {
9           fprintf(stderr, "SQL error: %s\n", errmsg);
10          sqlite3_free(errmsg);
11      }
12  }
13
14  static int callback(void *data, int nr_col, char** atribute,
        char** coloana)
15  {
16      int id=atoi(atribute[0]);
17      test[id-1].id=id;
18      test[id-1].question=(char*)malloc(strlen(atribute[1]));
19      strcpy(test[id-1].question,atribute[1]);
20      test[id-1].A=(char*)malloc(strlen(atribute[2]));
21      strcpy(test[id-1].A,atribute[2]);
22      test[id-1].B=(char*)malloc(strlen(atribute[3]));
23      strcpy(test[id-1].B,atribute[3]);
24      test[id-1].R=(char*)malloc(strlen(atribute[4]));
25      strcpy(test[id-1].R,atribute[4]);
26      return 0;
27  }
```

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.

```
1   void trimite_mesaj(int sd, char* msg)
2   {
3       int lungime=strlen(msg);
4       if(-1==send(sd, &lungime, sizeof(int), 0))
5       {
6           perror("[client]: Eroare la trimiterea lungimii
                mesajului");
7           exit(1);
8       }
9       if(-1==send(sd, msg, lungime, 0))
10      {
11          perror("[client]: Eroare la trimiterea mesajului");
12          exit(1);
13      }
14  }
```

Listing 1.5. Funcție de primire a unui mesaj în client. Mesajul conține lungimea textului și textul propriu-zis. O funcție similară se regăsește în server.

```
char* primeste_mesaj(int sd)
{
    int length;
    if(-1==recv(sd, &length, sizeof(int), 0))
    {
        perror("[client]: Eroare la recv() de lungimea
            mesajului");
        exit(1);
    }
    char* mesaj=malloc((length+1)*sizeof(char));
    int bytes_received=0;
    while(bytes_received<length)
    {
        int r=recv(sd, mesaj+bytes_received, length-
            bytes_received,0);
        if(-1==r)
        {
            perror("[client]: Eroare la recv() de mesaj");
            exit(1);
        }
        if(0==r)
        {
            perror("[client]: Server-ul a fost oprit");
            exit(1);
        }
        bytes_received+=r;
    }
    mesaj[length-1]='\0';

    return mesaj;
}
```

Listing 1.6. Secvențe din implementarea funcției de gestionare a jucătorului(clientul); Se observă trimiterea mesajelor către client și recunoașterea răspunsurilor acestuia în concordanță cu cererea anterioară adresată de server

```
static void* handle_client(void* arg)
{
    struct player p=*((struct player*) arg);
    printf("Thread %d:\n", p.id);
    fflush(stdout);
    trimite_mesaj(p.client_descriptor, "Bun venit in
        QUIZZGAME! Cum te numesti?");
    char* aux;
    aux=primeste_mesaj(p.client_descriptor);
    p.nume=(char*)malloc(strlen(aux)+1);
    strcpy(p.nume, aux);
    free(aux);
```

```
12      char buffer[BUFFER_SIZE];
13      sprintf(buffer, "Buna, %s! Id-ul tau este %d. Jocul va
            incepe...", p.nume, p.id);
14      trimite_mesaj(p.client_descriptor, buffer);
15      //////...//////
16      trimite_mesaj(p.client_descriptor, "Jocul incepe...");
17      //////...//////
18      for(int i=0; i<3; i++)
19      {
20          bzero(buffer, sizeof(buffer));
21          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);
22          trimite_mesaj(p.client_descriptor, buffer);
23          char* raspuns;
24          raspuns=primeste_mesaj(p.client_descriptor);
25          if(strcmp(raspuns, test[i].R)==0)
26          {
27              p.punctaj+=5;
28              printf("pl %d: %d\n", p.id, p.punctaj);
29              trimite_mesaj(p.client_descriptor,"Raspuns corect
                    ! Ai acumulat 5 puncte in plus!");
30          }
31          else
32          {
33              trimite_mesaj(p.client_descriptor,"Raspuns gresit
                    !");
34          }
35      }
36      //////...//////
37      bzero(buffer, sizeof(buffer));
38      sprintf(buffer, "Punctajul maxim a fost de %d puncte\n",
            maxim);
39      trimite_mesaj(p.client_descriptor, buffer);
40      trimite_mesaj(p.client_descriptor, castigatori);
41      //////.../////
42  }
```

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 testa 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 fiind memorat în baza de date. Având mai multe teste la dispoziție, un participant ar putea iniția o sesiune de joc având posibilitatea de a alege un anumit test. O altă îmbunătățire posibilă ar fi crearea unei sesiuni de joc private, la care înscrierea s-ar face în baza unui cod cunoscut de către primul jucător și transmis verbal restul jucătorilor.

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