Quizzgame

Ungureanu Alexandra-Maria

grupa A2, anul II

1 Introducere

Proiectul "Quizzgame" implementeaza o aplicatie de tip client/server. Acesta are la baza ideea unui joc de cultura generala la care pot participa oricati jucatori. Dupa inregistrarea in aplicatie, acestia vor concura intre ei raspunzand la un anumit numar de intrebari. Fiecare utilizator va acumula un punctaj in functie de raspunsurile date, urmand ca pe baza acestuia sa se decida castigatorul. La finalul rundei de intrebari fiecare utilizator va fi notificat cu privire la identitatea invingatorului. Serverul administreaza o singura runda de joc la un moment dat, abia la sfarsitul acesteia incepand una noua.

2 Tehnologiile utilizate

Pentru a realiza proiectul "Quizzgame" am implementat un server TCP, acesta fiind un protocol orientat-conexiune, fara pierdere de informatii. Am facut aceasta alegere pentru a ma asigura ca toate intrebarile si raspunsurile ajung integral si in ordine, evitand astfel riscul de corupere a datelor si primirea unor informatii incomplete sau incorecte care ar fi afectat astfel experienta jucatorului, punctajul utilizatorului si calitatea jocului.

Deoarece jucatorii trebuie sa foloseasca aplicatia in acelasi timp, adica sa acceseze resursele serverului concomitent, este nevoie de un server concurent. Varianta de implementare aleasa a serverului concurent este cea pe baza de threaduri. Deoarece proiectul este realizat in linux, thread-urile vor fi implementate cu libraria pthread, care este o librarie C unix. Motivatia primara pentru folosirea thread-urilor este cea de a obtine performanta, deoarece comparand costul crearii si gestionarii unui proces, un thread poate fi creat cu o suprasarcina a sistemului de operare mai mica. Pentru a gestiona datele referitoare la intrebarile din aplicatie si variantele de raspuns ale acestora am folosit o baza de date SQL, implementata cu ajutorul librariei sqlite3. Am ales sa folosesc o baza de date sql, in defavoarea unui fisier XML, deoarece baza de date este mai rapida decat XML in ceea ce priveste procesarea datelor si necesita mai putine resurse pentru a rula decat XML.

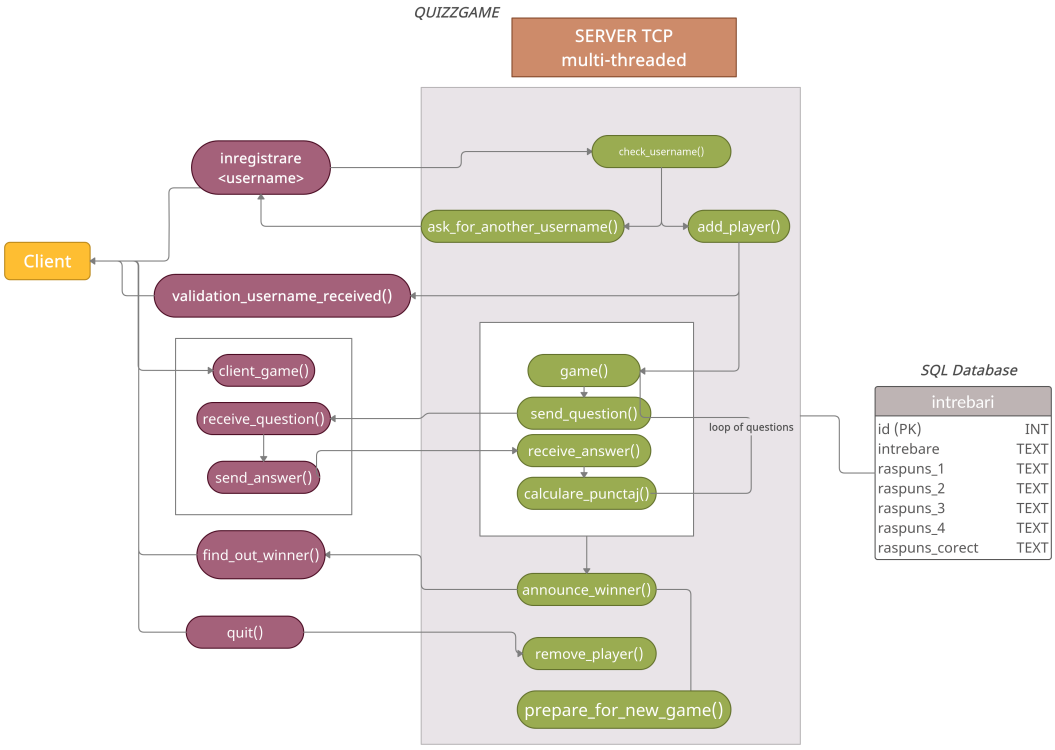
3 Arhitectura aplicatiei

3.1 Conceptele implicate

Conceptele implicate in arhitectura aplicatiei "Quizzgame" sunt reprezentate de urmatoarele elemente: server, client si baza de date. Un numar nelimitat

de clienti se poate conecta la server, fiecare cerere de conectare a acestora fiind procesata in mod paralel de catre server. Dupa conectarea la server, fiecare client se poate inregistra, fiecare inregistrare fiind verificata de server. Din baza de date vor fi citite intrebarile si raspunsurile stocate, urmand ca acestea sa fie trasmise catre client pe rand, iar raspunsurile clientului la interogarile mentionate anterior sa fie procesate in server, aceste actiuni fiind desfasurate, de asemenea, in paralel pentru toti clientii. Dupa terminarea interactiunii clientilor actuali cu serverul, aceasta procedura este reluata cu o noua serie de clienti.

3.2 Diagrama aplicatiei



4 Detalii de implementare

4.1 Cod relevant proiectului

In momentul de fata aplicatia este construita cu ajutorul a trei fisiere: *create_database.c*, *server_proiect.c*, *client_proiect*. In fisierul *create_database.c* am creat baza de date cu tabelul "intrebari" care stocheaza toate intrebarile si raspunsurile necesare jocului, folosind functii din libraria sqlite3(fig 1). In fisierul *server_proiect.c*, in functia main(), am creat socket-ul pentru comunicarea cu

clientii, am atasat socket-ul cu functia bind() si am pus serverul sa asculte clientii care vor sa se conecteze la server, cu listen(). Servirea clientilor se face in mod concurent, intr-o bucla infinita, prin crearea unui thread pentru fiecare client in parte. Toata operatiile pentru inregistrarea clientilor si planificarea jocului o sa fie realizate de threadul clientului respectiv.

Fig. 1. Figura 1: crearea bazei de date

```
sqlite3 *db;
char *err_msg = 0;

int rc=sqlite3_open("database.db",&db);

if(rc!=SQLITE_OK)
{
    fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
    sqlite3_close(db);

    return 1;
}

char * sql="DROP TABLE IF EXISTS intrebari;"
           "CREATE TABLE intrebari(id INT, text_intrebare TEXT, raspuns_1 TEXT, raspuns_2 Text, raspuns_3 TEXT, raspuns_4 TEXT,
           raspuns_corect TEXT);"
           "INSERT INTO intrebari VALUES(1,'Cine este Barack Obama?', 'a. fost presedinte','b. un cantaret celebru', 'c.
           magician', 'd. doctor', 'a');"
           "INSERT INTO intrebari VALUES(2,'Care este capitala Braziliei?', 'a. Rio','b. Brasil', 'c. Paris', 'd. New York', 'b');
           "
           "INSERT INTO intrebari VALUES(3,'Care a fost primul om pe luna?', 'a. Iuri Gagarin','b. Neil Armstrong', 'c. Pete
           Conrad', 'd. Buzz Aldrin', 'a');";
           "INSERT INTO intrebari VALUES(4,'Care este cel mai lung fluviu din Europa?', 'a. Dunarea','b. Sena', 'c. Tamisa', 'd.
           Volga', 'd');";
           "INSERT INTO intrebari VALUES(5,'Care este cel mai estic oras al Romaniei?', 'a. Arad','b. Timisoara', 'c. Sulina',
           'd. Sf. Gheorghe', 'c');";
           "INSERT INTO intrebari VALUES(6,'Cine a scris Traviata?', 'a. Giuseppe Verdi','b. Wolfgang Amadeus Mozart', 'c. Franz
           Lizt', 'd. Frederic Chopin', 'a');";
           "INSERT INTO intrebari VALUES(7,'Care este animalul desertului?', 'a. Ghepardul','b. zebra', 'c. rinocerul', 'd.
           camila', 'd');";

rc=sqlite3_exec(db, sql, 0,0,&err_msg);

if(rc!=SQLITE_OK)
{
    fprintf(stderr, "SQL error: %s\n", err_msg);

    sqlite3_free(err_msg);
    sqlite3_close(db);
```

Am ales sa utilizez pentru clientii participanti la runda de joc un vector global, alocat dinamic, de tip struct player($struct\, player\, \{char * username; int\, punctaj;\}$; care va contine username-ul introdus de client si punctajul acestuia. Fiecare client care vrea sa se conecteze la aplicatie trebuie sa introduca un username unic, care va fi verificat de functia $check\_username()$(fig2). In cazul in care exista deja un participant cu acelasi username, clientul este rugat sa introduca alt nume de utilizator. In caz contrar, va fi adaugat la lista de particpanti un nou jucator cu username-ul specificat si punctajul initializat cu 0. Dupa inscrierea participantilor, incepe jocul efectiv, aspect implementat in functia $game()$. Serverul intai trimite clientului numarul de intrebari la care trebuie sa raspunda, iar intr-o bucla for citeste cate un rand din tabelul "intrebari". Pentru a face asta am folosit interogari parametrizate, care maresc securitatea si performanta. Cum fiecare rand al tabelului contine o intrebare cu raspunsurile aferente acesteia, serverul trimite aceste informatii la client, asteapta raspunsul acestuia si ulterior trece la urmatoarea intrebare.

Clientul citeste pachetul trimis de server, il afiseaza pe ecran si asteapta timp de 20 de secunde ca utilizatorul sa raspunda cu una dintre variantele disponibile. Pentru a implementa cronometrul si pentru a putea citi raspunsul jucatorului in orice moment in perioada de timp alocata se creeaza doua thread-uri, unul care numara secundele si unul in care citeste inputul utilizatorului. In

cazul in care cele 10 de secunde trec fara a se da un raspuns, thread-ul pe post de timer inchide celalalt thread. In cazul in care se da un raspuns, al doilea thread il omoara pe primul. Rapunsul sau constatarea absentei lui sunt trimise la server si verificate. Daca acestea sunt corecte, se adauga un numar de puncte la scorul jucatorului. Dupa terminarea setului de intrebari jucatorii sunt sincronizati, iar castigatorul rundei respective este pus intr-o variabila globala "castigator", care va fi apoi transmisa tuturor jucatorilor. Conexiunile dintre server si clientii actuali sunt inchise, lista de jucatori este golita, la fel si variabila "castigator" pentru a permite ca o noua runda de joc sa inceapa. Sincronizarea thread-urilor va fi facuta cu primitivele mutexes, astfel incat de fiecare data cand apar schimbari ce trebuie facute la vectorul de jucatori sau la variabila "castigator", se va folosi variabila "pthread_mutex_t mutex_castigator" careia ii vom da lock si unlock pentru a ne asigura ca un singur thread are acces la anumite variabile pentru a evita stocarea gresita a datelor. De asemenea, dupa conectarea clientului la server, clientul trimite serverului pid-ul procesului sau pentru a verifica de-a lungul jocului daca clientul a incetat jocul. Jucatorul mai poate sa abndoneze jocul daca scrie "quit", ori cand trebuie sa raspunda la intrebari, ori chiar atunci cand isi alege username-ul.

In cadrul implementarii actuale, vectorul cu jucatori a fost implementat in mod dinamic, deci pot participa la joc oricati jucatori. Dupa anuntarea castigatorului, acest vector este golit, iar variabila "castigator" reinitializata.

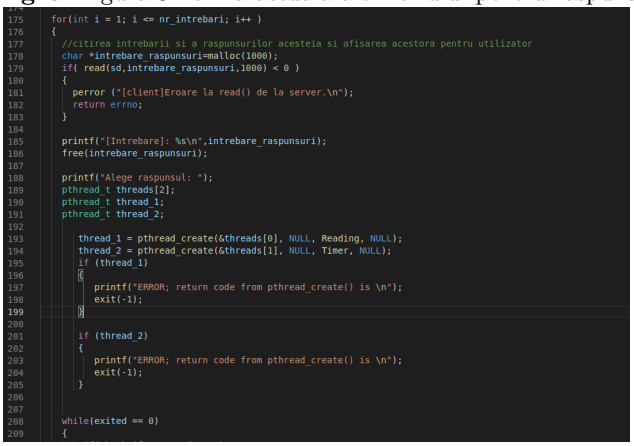
Fig. 2. Figura 2: forma actuala a functiei check_username()

```
int check_username(void *arg)
{
    int i=0;
    struct thData tdL;
    tdL= *((struct thData*)arg);


    char *buffer=malloc(100);
    if (read (tdL.cl, buffer,100) <= 0)
      {
        printf("[Thread %d]\n",tdL.idThread);
        perror ("Eroare la read() de la client.\n");

      }

  printf ("[Thread %d]Mesajul a fost receptionat...%s\n",tdL.idThread, buffer);

  while(username_unic(buffer)==0)
  {
    /*pregatim mesajul de raspuns */
    char *raspuns=malloc(100);
    raspuns[0]=0;
    strcat(raspuns,"Username already taken!");
    raspuns[strlen(raspuns)]=0;

    printf("[Thread %d]Trimitem mesajul inapoi...%s\n",tdL.idThread, raspuns);


    /* returnam mesajul clientului */
    if (write (tdL.cl, raspuns, strlen(raspuns)+1) <= 0)
      {
      printf("[Thread %d] ",tdL.idThread);
      perror ("[Thread]Eroare la write() catre client.\n");
      }
    else
      printf ("[Thread %d]Mesajul a fost trasmis cu succes.\n",tdL.idThread);
    free(raspuns);

    if (read (tdL.cl, buffer,100) <= 0)
    {
      printf("[Thread %d]\n",tdL.idThread);
      perror ("Eroare la read() de la client.\n");

    }

    printf ("[Thread %d]Mesajul a fost receptionat...%s\n",tdL.idThread, buffer);
}
vector[number_of_players].username=malloc(100);
strcpy(vector[number_of_players].username,buffer);
vector[number_of_players].punctaj=0;
number_of_players++;

for(int i=0; i<number_of_players; i++)
{
  printf("player:%d, username:%s",i,vector[i].username);
}

      /*pregatim mesajul de raspuns */
char *raspuns=malloc(100);
  raspuns[0]=0;
  strcat(raspuns,"V-ati inregistrat cu succes. ");
  strcat(raspuns,buffer);
  raspuns[strlen(raspuns)]=0;

printf("[Thread %d]Trimitem mesajul inapoi...%s\n",tdL.idThread, raspuns);

      /* returnam mesajul clientului */
if (write (tdL.cl, raspuns, strlen(raspuns)+1) <= 0)
  {
  printf("[Thread %d] ",tdL.idThread);
  perror ("[Thread]Eroare la write() catre client.\n");
  }
```

4.2 Scenarii de utilizare

Un numar de clienti se conecteaza la server intr-un anumit timp. Li se cere sa introduca un username care va fi valabil dor in cadrul rundei de joc actuale. Fiecare client introduce un nume de utilizator care va fi verificat pentru disponibilitate. Apoi primeste un set de reguli si instructiuni pentru joc. Dupa citirea acestora, el poate incepe jocul, primind cate o intrebare la care

Fig. 3. Figura 3: forma actuala a timer-ului pentru raspuns



```
for(int i = 1; i <= nr_intrebari; i++ )
{
  //citirea intrebarii si a raspunsurilor acesteia si afisarea acestora pentru utilizator
  char *intrebare_raspunsuri=malloc(1000);
  if( read(sd,intrebare_raspunsuri,1000) < 0 )
  {
    perror ("[client]Eroare la read() de la server.\n");
    return errno;
  }

  printf("[Intrebare]: %s\n",intrebare_raspunsuri);
  free(intrebare_raspunsuri);

  printf("Alege raspunsul: ");
  pthread_t threads[2];
  pthread_t thread_1;
  pthread_t thread_2;

    thread_1 = pthread_create(&threads[0], NULL, Reading, NULL);
    thread_2 = pthread_create(&threads[1], NULL, Timer, NULL);
    if (thread_1)
    {
       printf("ERROR; return code from pthread_create() is \n");
       exit(-1);
    }

    if (thread_2)
    {
       printf("ERROR; return code from pthread_create() is \n");
       exit(-1);
    }


  while(exited == 0)
  {
```

trebuie sa aleaga una dintre cele patru variante propuse. Poate raspunde oricand in intervalul de timp propus, dar daca niciun raspuns nu este inregistrat, la server va fi trimis mesajul ”no_answer”. Clientul va fi notificat cu privire la inceperea si incheierea timpului de raspuns. Pe tot parcursul jocului, utilizatorul are disponibila optiunea de ”quit” pentru a iesi din joc. Dupa terminarea intrebarilor, jucatorul este notificat cu privire la numele invingatorului, iar jocul se incheie.

5 Concluzii

Fata de solutia precedenta, in care citirea raspunsului jucatorului era facuta printr-un read neblocant in interiorul unui bloc while care masura timpul, solutia propusa de data aceasta, cu folosirea a doua thread-uri, este mai eficienta si nu foloseste la fel de multe resurse ca cea initiala. De asemenea, tot in comparatie cu prima solutie, acum intrebarile sunt alese random din baza de date pentru ca jocul sa fie diferit de fiecare data. La momentul de fata, sper sa imbunatatesc solutia prin crearea unei interete grafice atractive si prin vizualizarea la sfarsit a intrebarilor gresite.

6 Bibliografie

1. `https://profs.info.uaic.ro/~computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c`

2. `https://zetcode.com/db/sqlitec/`

3. `https://hpc-tutorials.llnl.gov/posix/?fbclid=IwAR1sLUrJvlLj1tnotZMowC_rO6fhJkMgnHyttbPZargjrnupJWDgIS9UHtg`