Documentație Quizz Game

Haivas Daniel-Andrei, grupa 2B4

Facultatea de Informatică, Universitatea "Alexandru Ioan Cuza", Iași

Abstract. Acest raport are scopul de a prezenta proiectul "Quizz Game" în cele mai mici detalii. Aici vor fi prezentate lucruri precum motivația alegerii acestui proiect dar și tehnologiile folosite în procesul de dezvoltare al acestuia. Ulterior, vor fi ilustrate detalii despre modul în care a fost implementata fiecare funcționalitate specifica proiectului realizat, ca spre final să argumentăm modul în care acesta ar putea fi îmbunătățit.

1 Introducere

Pentru aceasta tema am ales să iau spre analiza proiectul "Quizz Game". Acesta simuleza un joc de tip competiție cu întrebări de cultura generala intre mai multe persoane. Fiecare utilizator este întâmpinat de un meniu în care îi este ceruta logarea sau autentificarea (în cazul în care nu are un cont deja creat). În momentul în care acesta este logat poate alege să își vadă statisticile sau să înceapă un joc.

După ce hotăraște să participe la un joc, utilizatoru trebuie să decidă daca vrea să intre într-o camera de joc cunoscuta de el sau să creeze una noua. Intrarea în orice sala de joc se face pe baza unui cod secret (minim 4 caractere, doar litere și cifte), cod stabilit de administratorul jocului în momentul în care se creează camera. Evident, programul poate suporta mai multe camere de joc. În momentul în care toți utilizatorii dintr-o camera sunt pregătiți, începe jocul propriu-zis.

Vor fi adresate întrebări jucătorilor, pe rand, în ordinea în care aceștia au intrat în camera de joc (ceilalți participanți fiind nevoiți să își aștepte concurenții să răspundă la întrebări). Fiecare utilizator are un timp limita de răspuns și vă fi eliminat în cazul în care nu îl respecta, lucru care nu vă afecta desfășurarea jocului. Toate întrebările au 3 variante de răspuns, dintre care doar una corecta. După ce se vor adresa întrebările tuturor jucătorilor, se vă afișă câștigătorul și punctajul fiecăruia, calculat pe baza numărului de răspunsuri corecte.

Am ales acest proiect deoarece mereu am fost atras de astfel de concursuri de cultura generala, fiind o persoana foarte competitiva. Am privit realizarea acestuia că pe o provocare, deoarece implica stăpânirea a foarte multe cunoștinte, nu doar din Rețele de Calculatoare cât și din Baze de Date (pentru stocarea persistenta a întrebărilor și conturilor de utilizator ale jucătorilor).

2 Tehnologii utilizate

Pentru a realiza comunicarea client-server, în acest proiect, se folosește protocolul TCP concurent prin threaduri. Este absolut necesara folosirea unui protocol concurent pentru a putea fi serviți mai mulți clienți simultan. Acest protocol presupune crearea unui thread nou pentru fiecare client nou venit. Un proces este format din mai multe thread-uri, lucru care face protocolul folosit aici mai eficient decât protocoalele TCP bazate pe fork, care creează procese copil pentru noii clienți [2].

TCP concurent cu thread-uri se mulează cel mai bine pe problema ceruta, deoarece este mai rapid decât celelalte modele de implementări (cu fork sau cu select cu multiplexare I/O), oferind garanția comunicării rapide intre server și clienți [2]. Desi protocolul UDP este mai eficient din punct de vedere al vitezei, acesta are un dezavantaj: nu exista garanția că se transmit toate datele intre client și sever, fapt ce se dorește a fi evitat în dezvoltarea acestui proiect.

Pentru stocarea persistenta a tuturor datelor (conturile utilizatorilor și întrebările propuse) va fi folosită o bază de date SQLite. Acest model de baze de date este utilizat datorită simplității de a fi gestionat într-un cod $C++$. Totodată, volumul de date ce vor fi stocate nu este mare și va putea fi reprezentat cu ușurință într-o bază de date SQLite [6].

3 Arhitectura aplicației

Proiectul are la baza concepte precum: client, server, protocol de comunicare, socket, fir de execuție, bază de date, etc. Procesul server este cel care accepta cereri de la procesele clienți. Totodată, el trebuie să evalueze cererea și să trimită un răspuns corespunzător înapoi clientului. Procesul client initializeaza comunicarea cu serverul și îi solicita acestuia un anumit serviciu [1].

Pentru a putea realiza comunicarea intre client și server este nevoie de folosirea unui protocol de comunicare. Conform celor menționate mai sus, în aceasta aplicație este folosit protocolul concurent TCP mulți-threading. Acesta lucrează cu noțiunea de thread (fir de execuție), fiecare client nou fiind preluat de către un thread.

Un fir de execuție poate fi văzut că cea mai mica parte de cod gestionata de sistemul de operare. În momentul în care un nou thread se creează, acestuia nu i se aloca memorie, firul de execuție și procesul principal împărtășind aceeași memorie [3]. Pentru comunicarea efectiva intre aceștia se folosește un socket. Acesta este independent de arhitectura hardware și prin intermediul sau se realizează schimbul de mesaje [1].

Un alt concept implicat în dezvoltarea acestei aplicații este bază de date. O bază de date reprezintă o modalitate de stocare a datelor în scopul găsirii rapide a informațiilor căutate [4]. Totodată, o bază de date trebuie să ne dea garanția că datele introduse sunt în siguranță.

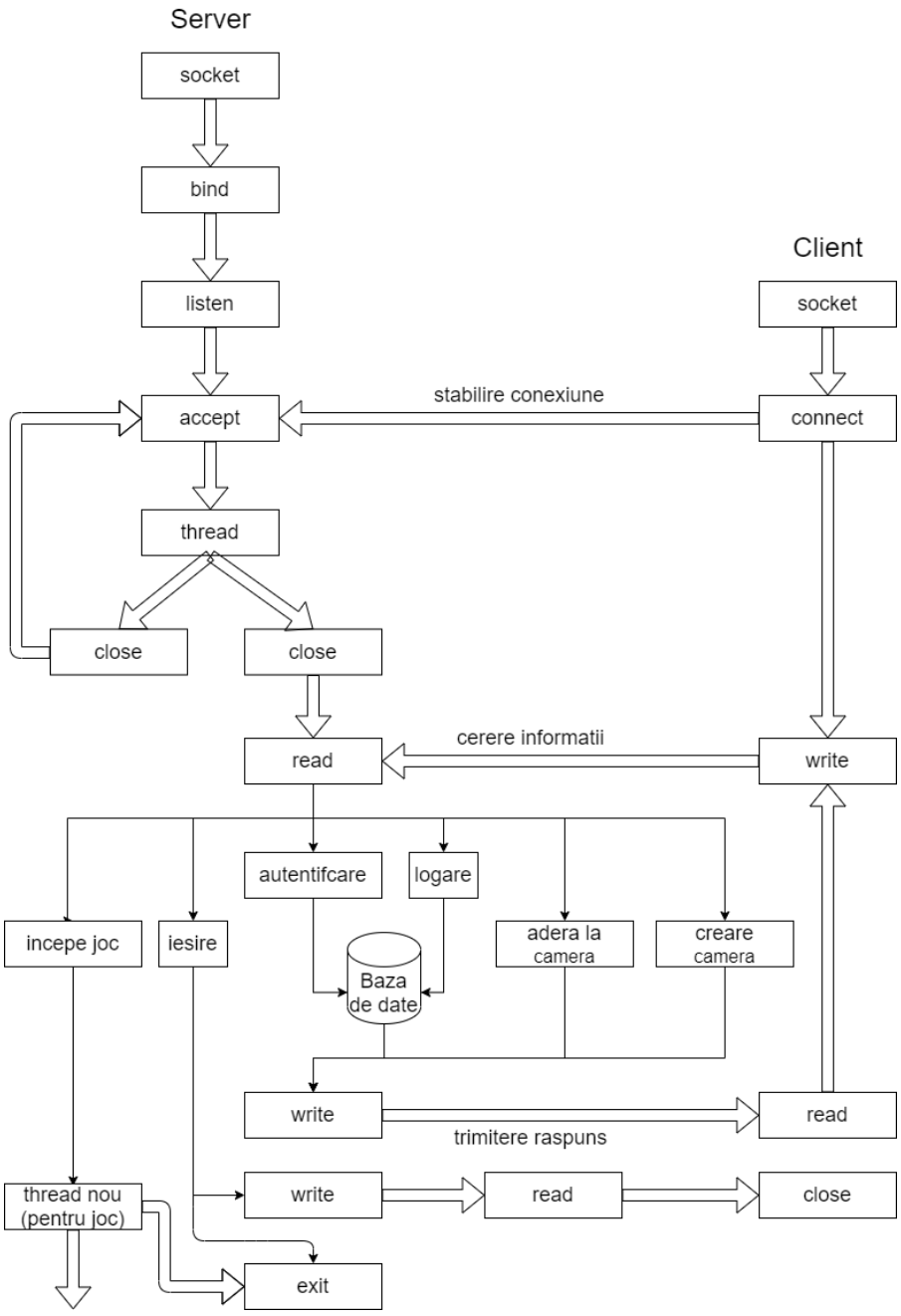

Fig. 1. Protocol concurent TCP multithreading [5] cu funcționalitățile aplicației

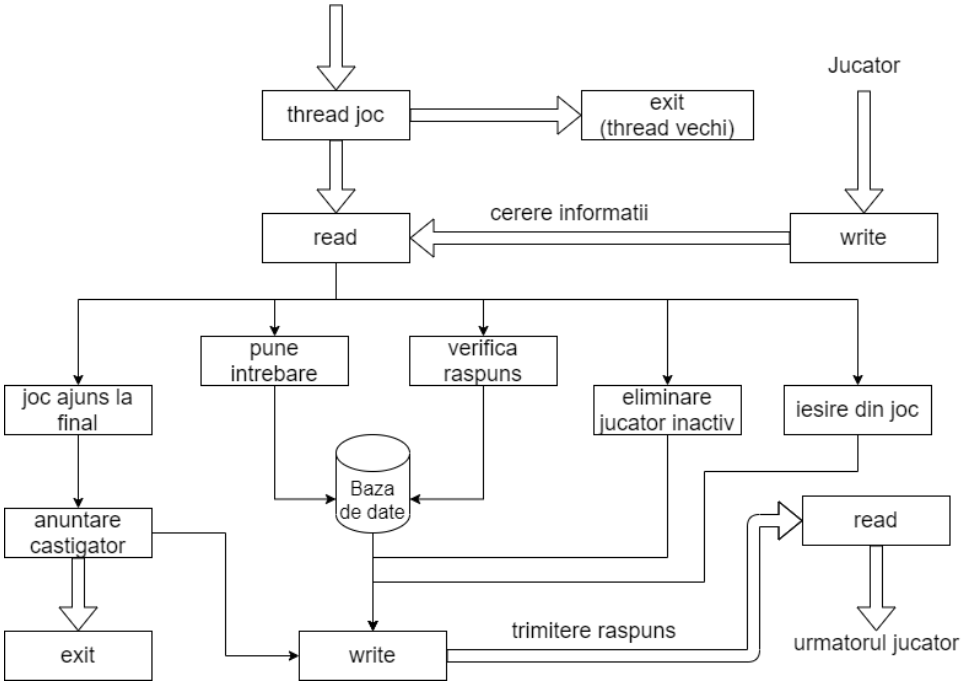

Fig. 2. Fir de execuție pentru un joc

4 Detalii de implementare

Pentru a afișă modul în care un utilizator este gestionat de către program se vă folosi o diagrama Ușe Case. Aceasta are rolul de a prezenta functionalitatile pe care le vă putea utiliza un utilizator, și raspusurile pe care le vă primi. Totodată, aceasta diagrama este foarte utila pentru testarea aplicației. Pentru a arata că implementarea este corecta vom luat prezenta doua scenarii de succes și doua scenarii de eșec pentru cererile unui utilizator.

Un prim scenariu de succes îl reprezintă creearea unei camere. Pentru a fi validata camera, codul stabilit de jucătorul-administrator trebuie să nu fie folosit de alta camera și să fie de cel puțin lungime 4 (doar litere și cifre). Un cod corect introdus de utilizator duce la creeare camerei de joc. Un alt scenariu de succes este cererea unei întrebări. Acest procedeu presupune că un utilizator să își aștepte rândul la coada. În momentul în care i-a venit rândul, serverul interoghează baza de date, oferindu-i o întrebare pentru a raspunde.

Un scenariu de eșec pentru acest proiect poate fi considerat aderarea unui utilizator la o camera. Aici avem doua cazuri: primul caz presupune să nu exista nicio camera creata până acum, lucru ce îl face pe acel jucător să creeze una. În al doilea caz exista deja camere create. Introducerea unui cod care nu aparține niciunei camere nu îl vă plasa pe utilizator nicăieri, el trebuind să încerce alt cod.

Un alt scenariu de eșec e reprezentat de părăsirea unei camere de către un jucător, înainte de începerea meciului. Daca utilizatorul este unul obișnuit, camera nu vă avea nicio problema, aceasta pastrandu-și comportamentul. Dar, daca utilizatorul este administratorul camerei, aceasta se vă distruge, ceilalți participanți fiind eliminați din ea.

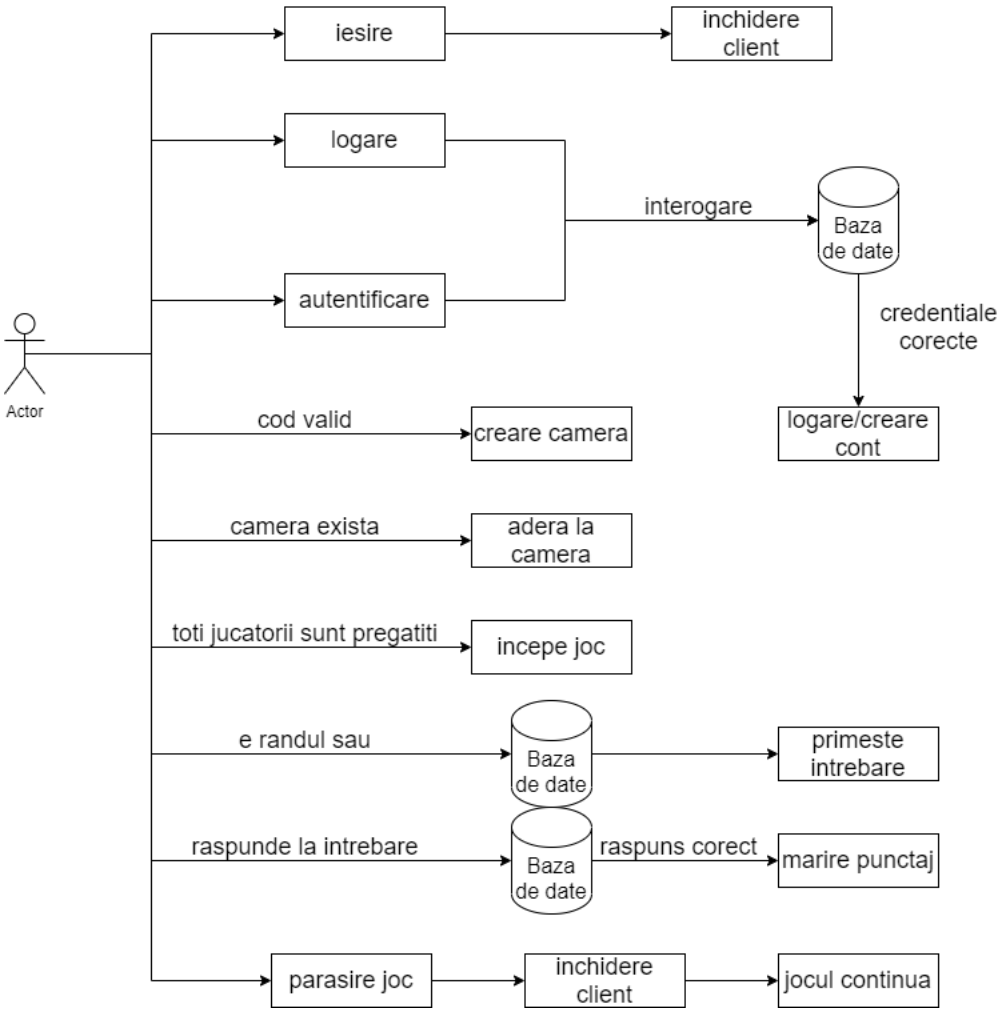

Fig. 3. Diagrama Use Case

Pentru a demonstra corectitudinea implementării, mai jos vor fi prezentate porțiuni din procesul de dezvoltare al proiectului:

```
typedef struct thData{
  int idThread; //id thread
  int client; //descriptorul client
}thData;

struct newClient{ // un thread pentru fiecare jucator
  thData* td;
  int typeUser = -1; // 1 -> obisnuit, 2-> owner;
  //int idGame = -1;
  bool inRoom = false;
  bool ready = false;
};
```

În cadrul programului vom avea un vector de structuri newClient care va retine detalii despre fiecare thread creat pentru fiecare client nou (până când se va începe un joc).

```
class Room{
public:
    vector <int> dClient; // int -> descriptor
    int idRoom;
    string cod;
    int idQuestion = 0;
    vector <vector <int>> questions;

    Room(int d, int idRoom_, string s) : cod(s), idRoom(idRoom_) {
      dClient.push_back(d);
    }
    void addClient(int k){
      dClient.push_back(k);
    }
    /* ... */
};
```

Camerele de joc vor fi memorate într-un vector de clase Room și va păstra informații precum: descriptorul fiecărui client participant, ID-ul camerei, codul secret de acces, dar și o matrice unde vor fi reținute întrebările ce au fost puse fiecărui jucător (pentru a nu se repeta).

```
while (true)
{
  int client;
  unsigned int length = sizeof (from);
  fflush (stdout);

  /* acceptam un client (stare blocanta pina la realizarea conexiunii) */
  if ( (client = accept (sd, (struct sockaddr *) &from, &length)) < 0)
  {
    perror ("[server]Eroare la accept().\n");
    continue;
  }
  newClient t;
  t.td = (struct thData*)malloc(sizeof(struct thData));
  t.td->idThread = ++idCl;
  t.td->client = client;

  clients_NIG.push_back(t); // Not In Game
  pthread_create(&clientsTh[idCl], NULL, &treat, &clients_NIG[clients_NIG.size()- 1]);

}//while
```

În aceasta secvență de cod se așteaptă mereu clienți prin primitiva accept. Se memorează descriptorul clientului și se creează un fir de execuție care vă avea grija de client prin intermediul funcției treat.

```
do{
  fflush (stdout);
  msj.clear();
  if(th.typeUser == -1)
    th.typeUser = chooseType(*th.td, first);
  else
    if(th.inRoom == false)
    {
      if(th.typeUser == 1)
        th.inRoom = joinRoom  (th, first);
      if(th.typeUser == 2)
        th.inRoom = createRoom(*th.td, first);
    }
    else
    {
      bool start = waitForGameToStart(th.td[0]);
      if(start == true)
        break;
    }
}while(true);
/* creare thread pt jocul care incepe, inchidere threaduri vechi*/
```

Semnificativ de ilustrat în funcția treat este aceasta secvență care prezinta modul în care este gestionat ușerul. Mai întăi este pus să aleagă daca vrea să creeze sau să intre într-o camera. După ce ajunge să fie într-o camera, așteaptă să înceapă jocul. Ulterior, se creează un thread pentru jocul care începe, iar thread-urile vechi se închid.

```
string s = comunicate(*th.td, msj);
for(int i = 0; i < rooms.size(); ++i)
  if(rooms[i].cod == s)
  {
    rooms[i].addClient(th.td->client);
    return true;
  }
first = false;
return false;
```

Această secvență este din funcția joinRoom. Aici serverul primește de la client codul introdus de acesta (prin funcția comunicate) și va verifica daca exista o camera cu acel cod, caz în care se returnează true.

```
string s = comunicate(th, msj);
first = false;

if(s.length() < 4)
  return false;
for(char c : s)
  if(!((c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')))
    return false;
for(int i = 0; i < rooms.size(); ++i)
  if(rooms[i].cod == s) // daca mai exista alta camera cu acelasi cod
    return false;
//first = true;
rooms.push_back(Room(th.idThread, ++idRoom, s));

return true;
```

Aceasta secțiune de cod este din funcția createRoom și se verifica daca codul introdus de ownerul camerei este unul valid (minim 4 caractere, litere sau cifre) și daca nu e folosit de alta camera deja. În cazul în care se îndeplinesc ambele condiții, se creează o noua camera în vectorul rooms.

```
string comunicate(struct thData& th, string s)
{
  strcpy(mesaj, s.c_str());
  int lung = strlen(mesaj) + 1;
  cout << "[Thread " << th.idThread << "] Trimitem mesaj... \n";
  write(th.client, &lung, sizeof(int));
  write(th.client, mesaj, lung);

  memset(mesaj, '\0', 1023);
  read(th.client, &lung, sizeof(int));
  read(th.client, mesaj, lung);
  cout << "[Thread " << th.idThread << "] Am primit raspuns... ";
  cout << (s = mesaj) << "\n";
  return s;
}
```

Funcția comunicate este cea care face trasmiterea de mesaje de la server la client și înapoi, prin intermediul funcțiilor read și write, cunoscandu-se descriptorul clientului. Aceasta returnează mesajul transmis de client.

5 Concluzii

O îmbunătățire ce poate fi adusă proiectului, pentru a oferi clientului o mai bună experiență a jocului, este adăugarea unei interfețe grafice. Utilizatorul va avea la dispoziție butoane cu text prin intermediul cărora să își aleagă comenzile, în loc să le introducă în terminal. Astfel, îi va fi mult mai ușor să ceară ceea ce are nevoie de la aplicație.

O altă modalitate de îmbunătățire a acestei aplicații ar fi mărirea rețelei pentru a se putea conecta și clienți de pe alte mașini la acest server. În acest mod, experiența ar fi mult mai plăcută pentru ca ar da șansa unor utilizatori (aflați la distanță mare unul de celălalt) să joace acest joc împreună, doar prin intermediul internetului.

În cazul în care proiectului prezentat i se adaugă și cele două funcționalități menționate mai sus, atunci aplicația ar fi utilă în viața reală. Ea va putea fi utilizată pentru simularea de concursuri de cultură generală, oferind o interfață plăcută utilizatorilor care vor putea participa de la ei de acasă.

References

1. Curs 5 Rețele de Calculatoare
   https://profs.info.uaic.ro/∼computernetworks/files/5rc$_{P}rogramareaInReteaI_{r}o.pdf$
2. Curs 7 Rețele de Calculatoare
   https://profs.info.uaic.ro/∼computernetworks/files/7rc$_{P}rogramareaInReteaIII_{E}n.pdf$
3. Laborator 12 Rețele de Calculatoare
   https://profs.info.uaic.ro/∼gcalancea/Laboratorul$_{1}$2*.pdf*
4. Baze de Date
   https://www.oracle.com/ro/database/what-is-database/
5. Model diagramă
   https://profs.info.uaic.ro/∼gcalancea/Laboratorul$_{7}$*.pdf*
6. SQLite
   https://www.sqlite.org/index.html