Quizz Game

Hirtopanu Tudor-Alin (Grupa E1)

Facultatea de Informatica Iasi
1 Decembrie 2021

Abstract. Acest document are ca scop prezentarea tehnologiilor utilizate, a arhitecturii si a detaliilor de implementare a proiectului propus Quizz Game (B).

Cuprins

1 Introducere

Scopul acestui proiect este de a dezvolta un server multithreading care sa permita accesul unui numar cat mai mare de clienti, ce vor raspunde pe rand unor intrebari stocate in fisiere XML. Serverul va retine scorurile fiecarui client in parte si le vor incrementa dupa fiecare raspuns corect. Clientii se pot conecta sau deconecta de la server in timpul jocului, fara sa afecteze desfasurarea acestuia in vreun fel.

1.1 Motivatie

Subiectul abordat de acest proiect imi aduce de o simulare a bine cunoscutului joc "**Vrei sa fii milionar**" pe care o jucam pe primul meu calculator atunci cand eram mic. Inca din copilarie m-au pasionat jocurile pe calculator, iar acesta este in opinia mea un clasic pe care oricine ar trebui sa il incerce macar o data. Motivul pentru care am ales acest subiect este ca am vrut sa capat un nivel minim de intelegere al procesului prin care a fost creat jocul mentionat mai sus.

Fig. 1. Jocul "**Vrei sa fii milionar**"

2 Tehnologii utilizate

2.1 Protocolul TCP

Pentru implementarea acestei aplicatii am folosit protocolul TCP (Transmission Control Protocol). TCP este un protocol de transport orientat conexiune, fara pierdere de informatii, ce controleaza fuxul de date. In comparatie cu protocolul UDP (User Datagram Protocol), acesta asigura integritatea mesajelor, precum si faptul ca mesajele ajung la destinatie in ordine cronologica.

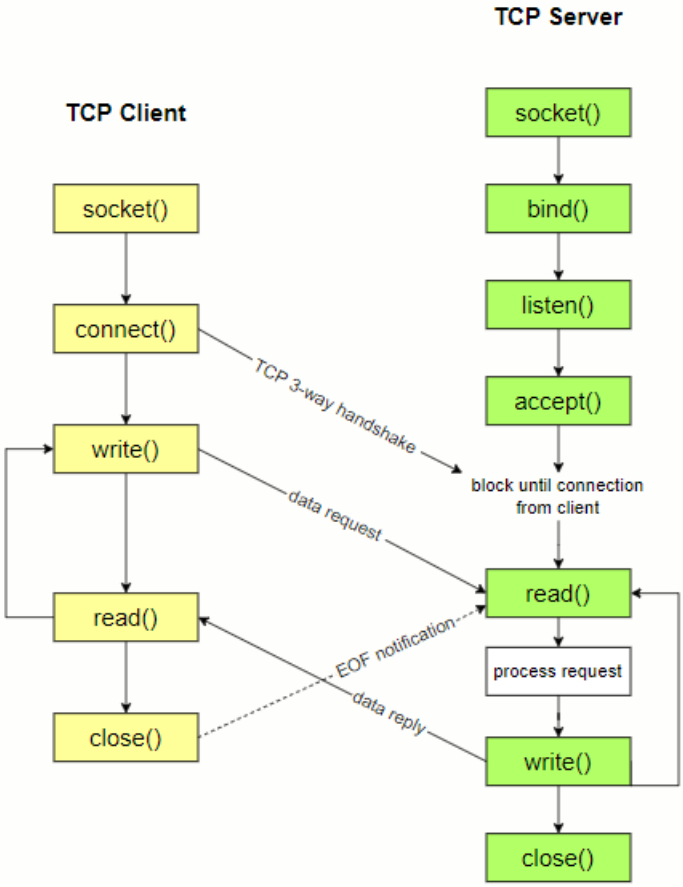

Fig. 2. Schema unei aplicatii de tip client-server

2.2 Multithreading

Varianta protocolului TCP utilizata este cea cu fire de executie multiple (implementata cu ajutorul librariei **pthread.h**). Fiecare proces client are propriul sau fir de executie care trateaza desfasurarea jocului pentru respectivul jucator. Acest aspect asigura viteza desfasurarii jocului, in lipsa sa jucatorii fiind nevoiti sa ii astepte pe cei din fata pentru a raspunde la randul lor.

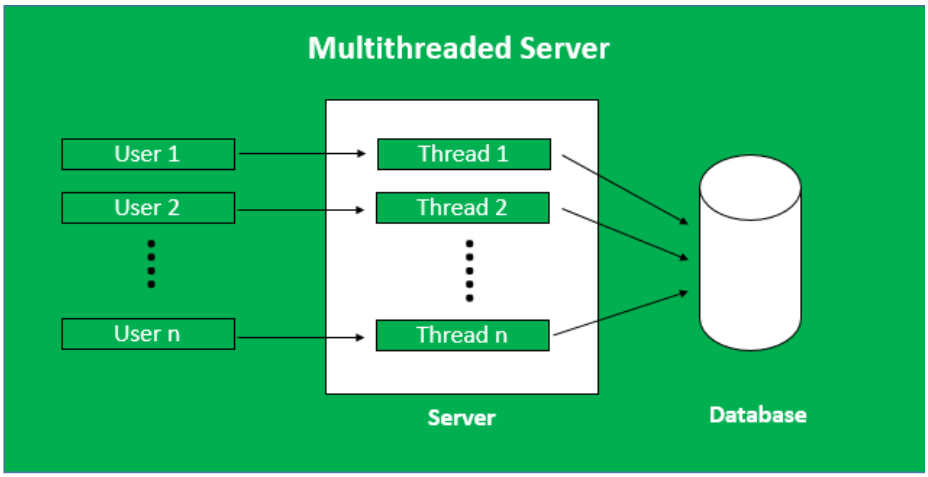

Fig. 3. Schema unui server cu fire de executie multiple

2.3 Fisiere XML

Pentru stocarea intrebarilor si a raspunsurilor corespunzatoare acestora, am ales folosirea fisierelor de tip XML (Extensible Markup Language). XML este un API ce defineste un set de reguli pentru codificarea documentelor intr-un format ce poate fi inteles atat de oameni cat si de calculator. Ca format, acestea sunt similare limbajului HTML, ceea ce le face extrem de usor de citit chiar si pentru utilizatorii neexperimentati. Alte avantaje ale acestor fisiere consta in compatibilitatea cu majoritatea limbajelor de programare actuale, precum si extensibilitatea lor, utilizatorii avand optiunea de a crea propriile lor taguri, utilizand un limbaj adecvat domeniului de utilizare, care sa fie inteles cu usurinta de cei ce le folosesc.

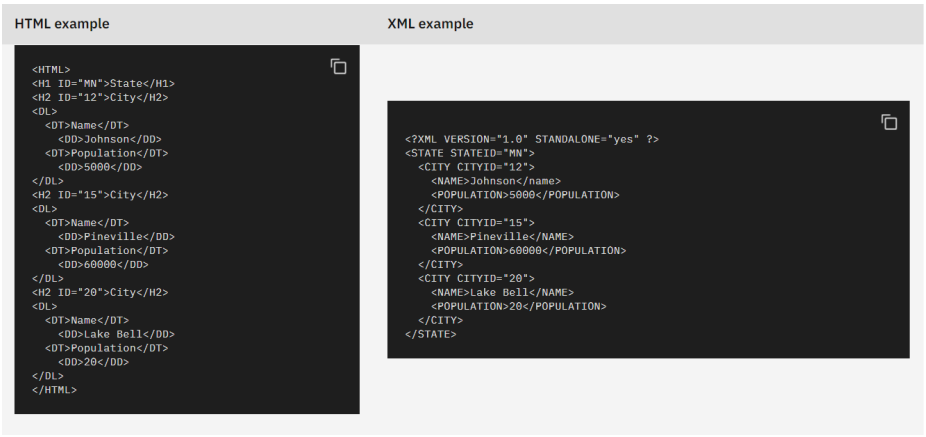

Fig. 4. Similaritatea dintre HTML si XML

3 Arhitectura aplicatiei

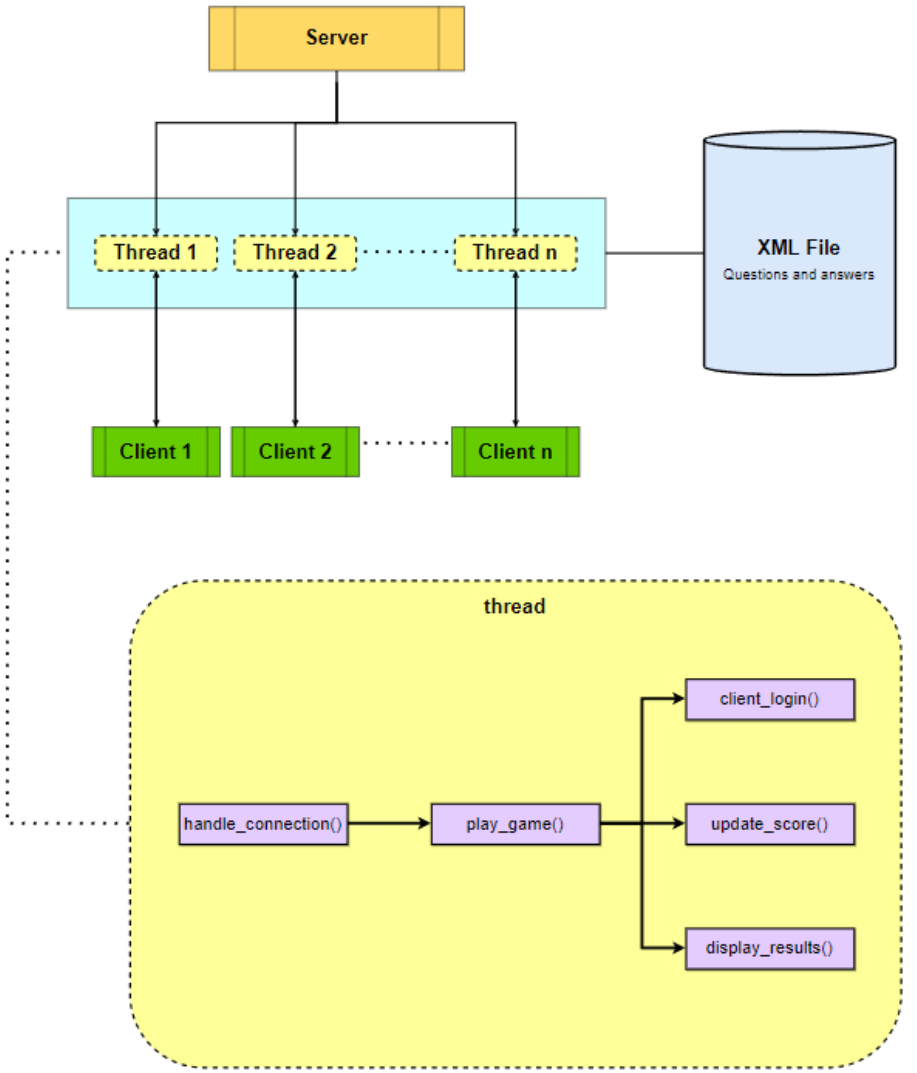

Fig. 5. Schema minimalista a arhitecturii aplicatiei (la momentul actual)

4 Detalii de implementare

4.1 Idee generala

Am stabilit ca protocolul folosit pentru conexiunea dintre server si client este de tip TCP cu fire de executie multiple. Odata ce serverul accepta o noua conexiune, clientului ii este atribuit un fir de executie care se ocupa de acesta. Pentru stocarea datelor fiecarui client am utilizat cateva variabile globale si am definit un nou tip de date sub forma unei structuri **thread_data**.

Fiecare fir de executie foloseste functia **handle_connection()** pentru a trata clientul. Aceasta apeleaza la randul ei functia **play_game()**, in care este implementata desfasurarea jocului. Serverul nu stocheaza utilizatorii intr-o baza de date. Ei se pot conecta cu orice nume doresc, iar atunci cand se deconecteaza jocul nu este afectat in niciun fel (decat daca nu a mai ramas niciun utilizator conectat).

4.2 Utilizare

Odata conectat la server, utilizatorul este nevoit sa se autentifice folosindu-se de comanda "**login :** [**username**]". Momentan, aplicatia este conceputa astfel incat sa accepte orice nume de utilizator. Alternativ, el poate iesi cu totul din aplicatie apeland comanda "**quit**". Dupa autentificare, clientul va incepe sa primeasca intrebarile, la care va trebui sa raspunda intr-un interval de maxim 60 de secunde. Cand toti clientii au raspuns la intrebare sau cand timpul s-a scurs, scorul fiecaruia este actualizat corespunzator si urmatoarea intrebare este adresata. Clientul are oricand posibilitatea de a parasi jocul folosind comanda "**logout**". La final, adica atunci cand toate intrebarile au fost utilizate, jucatorii primesc un mesaj care ii anunta cine este castigatorul.

4.3 Exemple de cod sursa

```
19
20    int player_counter, max_score;
21
22    typedef struct thread_data
23    {
24        int client_id, client_score;
25        int client_socket;
26        char* client_username;
27    }thread_data;
28
```

Fig. 6. Structura pentru memorarea datelor despre clienti

```
5     int server_socket;
6     struct sockaddr_in server_addr;
7
8     if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == -1)
9     {
10      perror("[client] Erorr occured while creating socket.\n");
11      return errno;
12    }
13
14    server_addr.sin_family = AF_INET;
15    server_addr.sin_port = htons(PORT);
16    server_addr.sin_addr.s_addr = INADDR_ANY;
17
18    if (connect(server_socket, (struct sockaddr*)&server_addr, sizeof(struct sockaddr)) == -1)
19    {
20      perror("[client] Erorr occured while connecting to server.\n");
21      return errno;
22    }
23
```

Fig. 7. Implementarea socketului in procesul client

```
2
3   static void* handle_connection(void* arg)
4   {
5       struct thread_data td;
6       td = *((struct thread_data*)arg);
7       pthread_detach(pthread_self());
8       play_game(td);
9       close ((intptr_t)arg);
10      return NULL;
11  };
12
```

Fig. 8. Functia **handle_connection()** prin care firul de executie trateaza un client

```
38
39  void client_login(struct thread_data td)
40  {
41      char client_request[BUFFER_SIZE];
42      char server_response[BUFFER_SIZE] = "Welcome to QuizzGame! To login, please use the command login : [username].\n";
43
44      _Bool login_status = 0;
45
46      while(!login_status)
47      {
48          write(td.client_socket, server_response, BUFFER_SIZE);
49          read(td.client_socket, client_request, BUFFER_SIZE );
50
51          if (strncmp(client_request, "login : ", strlen("login : ")) == 0)
52          {
53              td.client_username = (char*)(client_request + strlen("login : "));
54              if (td.client_username != NULL && td.client_username[0] != '\0')
55              {
56                  login_status = 1;
57              }
58          }
59      }
60  };
61
```

Fig. 9. Functia **client_login()** prin care se cere autentificarea utilizatorului

5 Concluzii

Proiectul nu se afla in forma sa finala. Majoritatea functionalitatilor vor fi implementate ulterior predarii acestei documentatii. La randul ei, si aceasta documentatie va fi actualizata conform schimbarilor ce vor avea loc la nivelul proiectului.

5.1 Idei de dezvoltare

Cateva puncte tinta pentru starea finala a aplicatiei ar fi:

- o modalitate functionala de logare a clientilor la server
- implementarea fisierelor XML si stocarea datelor in acestea
- schimbul de date de tip intrebare-raspuns dintre client si server (prin introducerea unui username)
- afisarea scorului pentru fiecare client
- implementarea unui timp de raspuns adecvat pentru intrebari

6 Bibliografie

- https://profs.info.uaic.ro/~computernetworks/
- https://en.wikipedia.org/wiki/XML
- https://www.ibm.com/docs/en/i/7.3?topic=programming-xml-toolkit
- https://www.geeksforgeeks.org/handling-multiple-clients-on-server-with-multithreading-using-socket-programming-in-c-cpp/