QuizzGame (B)

Jîtcă Diana

Universitatea Alexandru Ioan Cuza

1 Introducere

QuizzGame este o aplicație de tipul server-client multithreading implementat în C/C++ ce suportă mai mulți clienți și îi servește în ordinea în care s-au înregistrat.
O sesiune de joc reprezintă conectarea a n clienți în care aceștia trebuie să raspundă la o serie de întrebări într-un interval de timp. Dacă răspunsul este corect, se adună un anumit punctaj. La final, clientul cu cele mai multe puncte câștigă. Atât întrebările, răspunsurile corecte cât și clasamentul clienților sunt stocate într-o bază de date SQLite. Dacă unul din jucători dorește să părăsească jocul, acesta va fi descalificat, iar jocul va continua fără acesta.
Am ales acest proiect deoarece îl consider perfect pentru a aprofunda noțiunile învățate la materia *Rețele de Calculatoare* cu privire la implementarea unui server concurent dar și la *Baze de Date* pentru a stoca întrebările și punctajele clienților.
Întrebările sunt în limba engleză pentru ca jocul să fie accesat de o varietate mai largă de clienți, iar aria din care acestea fac parte este variată și distractivă, pentru testarea cunoștințelor de cultură generală dar și destinderea jucătorilor.

2 Tehnologii utilizate

2.1 Protocol

Pentru acest proiect am ales folosirea protocolului **TCP** deoarece asigură o transmisie sigură, orientată conexiune ce garantează livrarea pachetelor în ordine, iar dacă acestea sunt pierdute, vor fi retrimise.
Efectuează o conectare virtuală full duplex între două puncte terminale, fiecare punct fiind definit de către o adresă IP și de către un port TCP.
TCP oferă încredere, asigură livrarea ordonată a unui flux de octeți de la un program de pe un computer la alt program de pe un alt computer aflat în retea.

În cadrul proiectului *QuizzGame* este importantă transmiterea datelor în ordine, o amestecare a acestora ar face întrebarea indescifrabilă și astfel clientul va fi dus în eroare. Mai mult decât atât, în cazul în care un pachet se pierde, protocolul îl va retrimite, iar întrebarea nu va fi deteriorată, în acest mod se asigură corectitudinea datelor transmise.

2.2 Baza de date

Întrebările, răspunsurile corecte și clasamentul jucătorilor sunt stocate într-o bază de date. Aceasta este o colecție organizată de informații sau de date structurate, stocate electronic într-un computer.
Pentru acest proiect baza de date pe care am utilizat-o este *SQLite*, o mică bibliotecă C care implementează un motor de baze de date SQL încapsulat. O caracteristică importantă este fiabilitatea, tranzacțiile sunt atomice, consistente, izolate și durabile (ACID) chiar după căderi de sistem și pene de current.

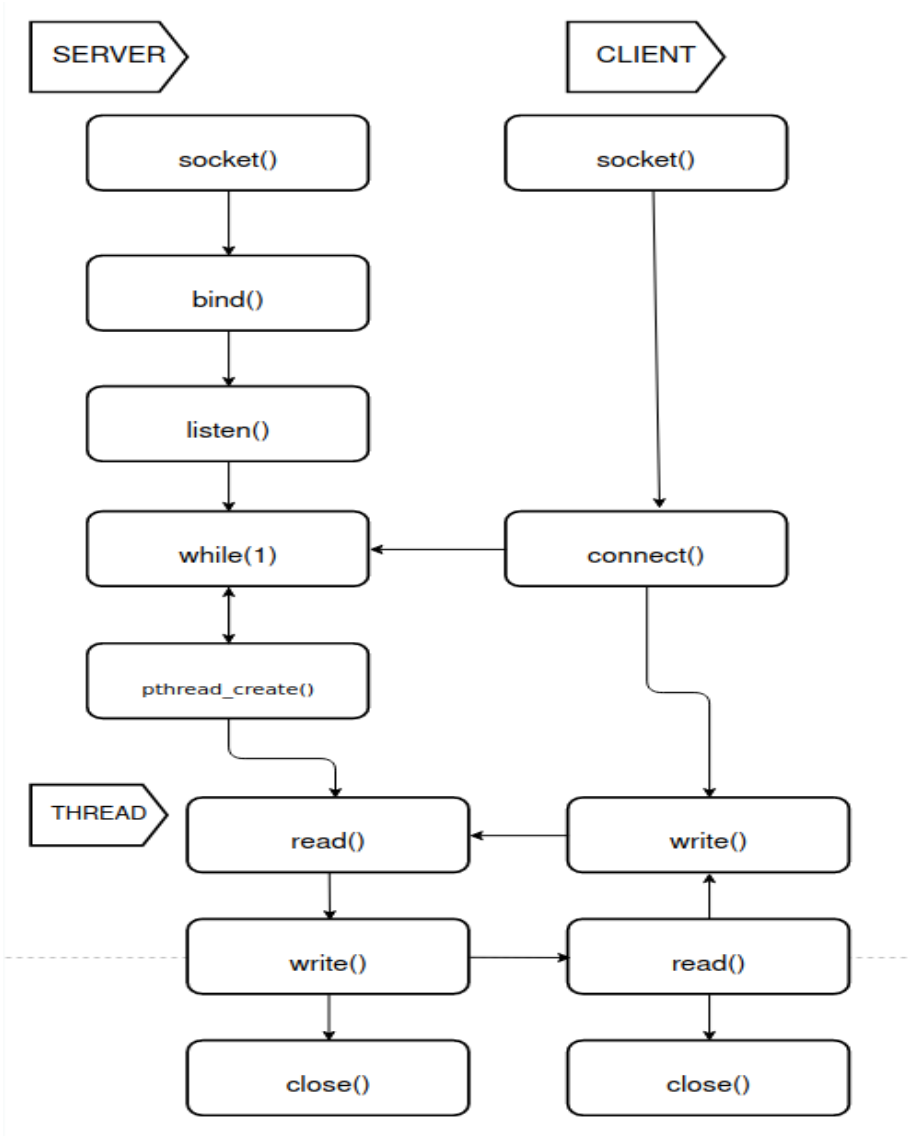
3 Arhitectura aplicației

Serverul TCP concurent este implementat prin intermediul *socketului* și a *firelor de execuție*. Primul pas este creearea socketului pentru domeniul Internet, transmiterea datelor fiind prin $SOCK\_STREAM$, specific TCP. Apoi, socketului i se asociaza o adresă IP și portul la care se va conecta prin primitiva *bind*(). După ce socketul are cel puțin un port atașat poate intra într-o stare de așteptare pasivă a conexiunilor de la potențialii clienți, prin intermediul unui apel *listen*().
Pentru transferul efectiv de date va fi creat un nou socket prin apelul *accept*(). Acesta elimină o conexiune din coada de așteptare (o limită de conexiune primite la socket făcute de listen()). Apoi se va realiza transferul de date prin primitivele *read*() și *write*(), iar la final se termină conexiunea prin apelul *close*().
TCP fiind un protocol orientat conexiune implică conectarea socketului serverul cu socketul clientului, în prealabil creat, realizându-se prin *connect*() la adresa IP a serverului și portul aferent.
Pentru a asigura concurența, fiecărui client conectat i se va atribui un thread creat prin primitiva *pthread_create*().

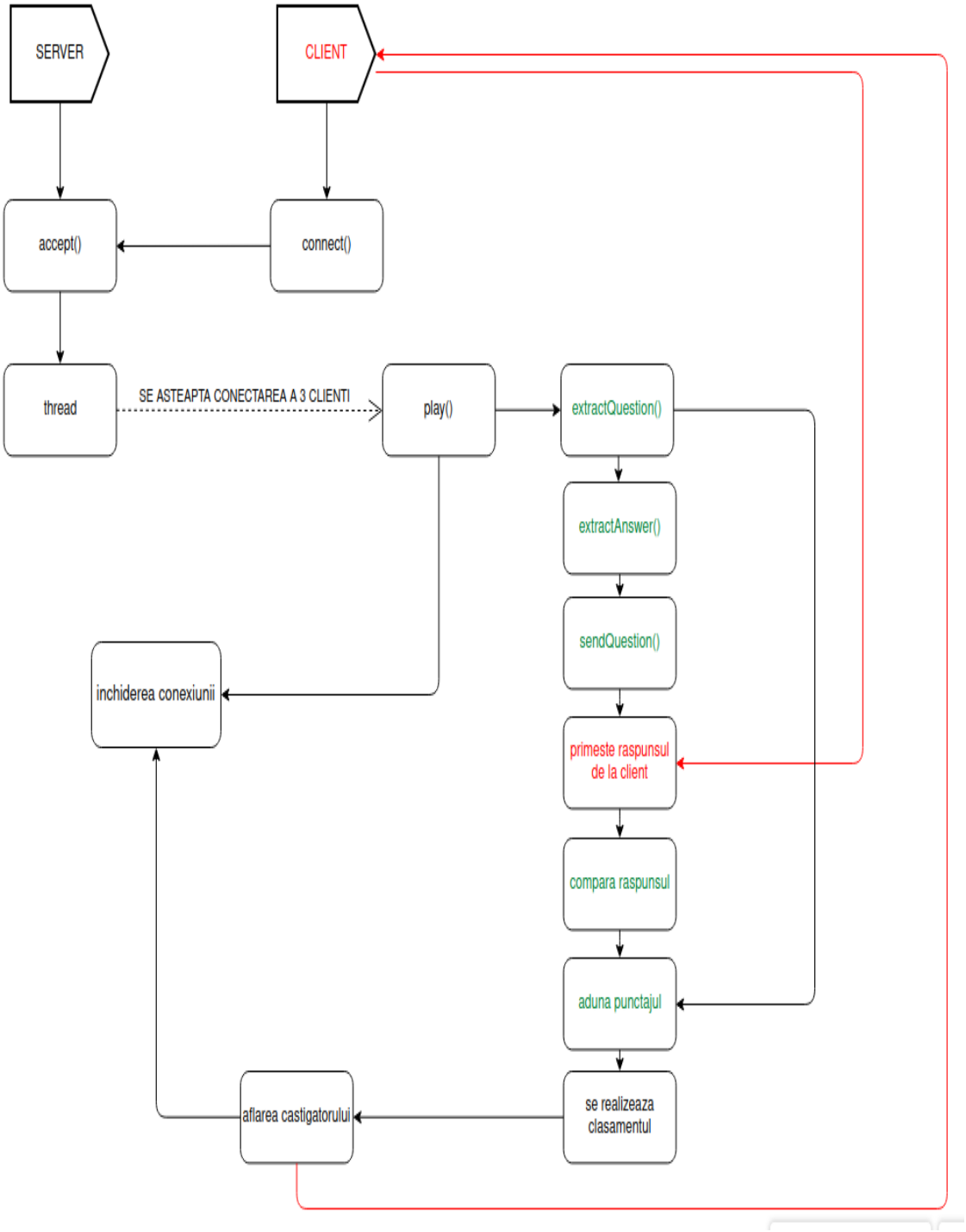
Când threadurile sunt create, se cere clientului un username ce va fi reținut într-o structură aferentă fiecăruia alături de puncte și descriptorul socketului.

Când numărul de jucători este atins, sesiunea de joc poate începe:

1. Sunt extrase pe rând întrebările din baza de date, scrise prin socket de server pentru a fi citie de client.
2. Serverul va extage și raspunsul corect, dar și punctele aferente fiecărei întrebări
3. Clienții citesc din socket și vor trimite înapoi răspunsul serverului într-un interval de timp
4. În cele din urmă, serverul va compara răspunsurile primite cu cel corect și va modifica punctele fiecărui jucător
5. La final, un clasament al punctajelor va fi calculat și afișat de server, eventual stocat în baza de date



Datorită faptului că mai multe threaduri ce aparțin aceluiași proces pot accesa în comun date declarate în cadrul procesului, apare necesitatea sincronizării acestui acces. Aceasta se va realiza prin folosirea unor mecanisme denumite *mutex*. Threadul ce deține *mutexul* va accesa anumite bucăți din cod, celălalte threaduri fiind blocate până la eliberarea *mutexului*. Când *mutexul* este eliberat, unul din celălalte threaduri îl va bloca și procesul se continuă.


SERVER
CLIENT
accept()
connect()
thread
SE ASTEAPTA CONECTAREA A 3 CLIENTI
play()
extractQuestion()
extractAnswer()
sendQuestion()
primeste raspunsul de la client
compara raspunsul
aduna punctajul
se realizeaza clasamentul
aflarea castigatorului
inchiderea conexiunii

4 Detalii de implementare

Serverul TCP concurent a fost implementat prin creearea unui *socket*() care va face *bind*(), *listen*() și *accept*() cu viitorii clienții ce se vor conecta prin primitiva *connect*(). Fiecăruia i se va atribui un thread. Comunicarea dintre server și client se realizează prin intermediul socketului.

```
while (1)
    {

        pthread_t tid; // id thread
        clientThread *thread_arg;

        printf("[server]Asteptam la portul %d...\n", PORT);
        fflush(stdout);

        thread_arg = (clientThread *)malloc(sizeof(
    clientThread));
        thread_arg->idThread = id;
        id++;

        thread_arg->fdClient = accept(fdserver, &thread_arg->
    address, &thread_arg->addr_len);
        if (thread_arg->fdClient == -1)
        {
            perror("[server]Eroare la accept().\n");
            continue;
        }

        clients[thread_arg->idThread].socketDescriptor =
    thread_arg->fdClient;
        game.numberPlayers = 0;
        game.numberPlayers++;
        game.noQuestion = 3;

        tid = pthread_create(&tid, NULL, &threadFunction,
    thread_arg);
        if (tid < 0)
        {
            perror("[server] eroare la pthread_create()");
            // free(thread_arg);
            return errno;
        }
    }
```

Fiecărui client i se atribuie o structură în care se reține informații despre thread, descriptorul socketului, username-ul, punctele rezultate din urma jocului:

```
    typedef struct thread
{
    int fdClient; // socketul pt conectarea cu clientul
```

```
    int idThread; // al catelea thread este
    bool exited;
    struct sockaddr address;
    struct sockaddr_in client; // adresa clientului conectat
    char username[256];
    int addr_len; // lungimea lui adress

} clientThread;

struct client
{
    char username[256];
    int points;
    int socketDescriptor;
    bool exited;

} clients[6];
```

Funcția *createDataBase*() creează o bază de date SQLite în care vor fi stocate numărul întrebării, pentru o manevrare ușoară atunci când va fi nevoie de aflarea răspunsului corect și a punctajului, textul întrebării, răspunsul corect, respectiv numărul de puncte.

```
    void createDataBase()
{
    // sqlite3 *db;
    char *zErrMsg = 0;
    int rc = sqlite3_open("test.db", &db);

    if (rc)
    {
        fprintf(stderr, "Can't open database: %s\n",
    sqlite3_errmsg(db));
        fflush(stdout);
    }
    else
    {
        fprintf(stderr, "Opened database successfully\n");
        fflush(stdout);
    }

    char *sql;
    sql = "CREATE TABLE QUIZ("
          "ID          INT              NOT NULL, "
          "QUESTION    VARCHAR(1000)    NOT NULL, "
          "ANSWER      VARCHAR(1000)    NOT NULL, "
          "POINTS      INT              NOT NULL);";

    rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);

```

```
    if (rc != SQLITE_OK)
    {
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
        fflush(stdout);
        sqlite3_free(zErrMsg);
    }
    else
    {
        fprintf(stdout, "Table created successfully\n");
        fflush(stdout);
    }

    populateTable(); //populez tabelul
    sqlite3_close(db);
}
```

Această funcție execută o interogare în baza de date pentru a extrage întrebarea cu numărul i ce urmează a fi trimisă prin socket clienților pentru ca aceștia să poată raspunde.

```
    void sendQuestion( int i, char* intrebare)
{
    int rc = sqlite3_open("test.db", &db);

    if (rc)
    {
        fprintf(stderr, "Can't open database: %s\n",
    sqlite3_errmsg(db));
    }
    else
    {
        fprintf(stderr, "Opened database successfully\n");
    }

    // casting int la char pierde din informatii si nu pot
    folosit (char)
    // folosesc sprintf
    char inregistrare[100];
    sprintf(inregistrare, "SELECT QUESTION FROM QUIZ WHERE ID
    =%d", i);

    sqlite3_stmt *stmt;

    rc = sqlite3_prepare_v2(db, inregistrare, -1, &stmt, NULL)
    ;

    if (rc < 0)
    {
        printf("Error executing sql statement\n");
        fflush(stdout);
    }
```

```
28
29      rc = sqlite3_step(stmt);
30
31      strcpy(intrebare, sqlite3_column_text(stmt, 0));
32      sqlite3_finalize(stmt);
33      sqlite3_close(db);
34  }
```

Urmează a fi implementată o funcție de parsare a mesajelor primite de server din partea clientului:

1. "answer : răspuns", unde se va testa dacă răspunsul primit e cel corect
2. "exit", în care jucătorul va cere să părăsească sesiunea de joc

O altă funcție importantă este extragerea răspunsului corect și a punctajelor din baza de date.

5 Concluzii

O îmbunătățire a soluției propuse este alegerea limbii în care să se trimită întrebările clienților, divizarea pe domenii diferite a întrebărilor și, de exemplu, posibilitatea de a se conecta clientul pe baza unui username și parole, creându-și astfel un cont.
Având contul creat se pot reține punctajele obținute și astfel se pot oferi recompense în schimbul punctelor.

Bibliografie

1. TCP `https://ro.wikipedia.org/wiki/Transmission_Control_Protocol`
2. SQLite `https://ro.wikipedia.org/wiki/SQLite`
3. C în Latex `https://tex.stackexchange.com/questions/348651/c-code-to-add-in-the-document`
4. Diagrame UML `https://online.visual-paradigm.com/app/diagrams/#diagram:proj=0&type=ParametricDiagram&width=11&height=8.5&unit=inch`
5. Informații despre funcțiile din SQLite `https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.html`
6. Informații despre multithreading `http://www.mario-konrad.ch/blog/programming/multithread/tutorial-04.html`