QuizzGame
Computer Networks

Popa Maria Eliza

January 8, 2024

1 Introducere

Proiectul QuizzGame presupune crearea unui server TCP Multithreading concurent la care se vor conecta un număr nelimitat de clienți. Clientii vor primi de la server un set de intrebari si vor avea un timp limitat de raspuns. Serverul se ocupa de transmiterea întrebărilor, de sincronizarea clienților, dar și de contorizarea punctajelor și anunțarea câștigătorului.

2 Tehnologii aplicate

În realizarea proiectului QuizzGame vom folosi tehnologia TCP(Transmission Control Protocol) pentru a servi n clienți simultan. Pentru fiecare client, serverul va crea câte un thread. Comunicarea dintre server și clienți va fi realizata cu ajutorul socket-urilor. Pentru realizarea proiectului am folosit limbajul C.

De ce am ales TCP? În cazul acestui proiect, fiind nevoie de gestionarea multiplilor clienți care trimit și primesc mesaje, putem spune ca TCP este mai sigur decât protocolul UDP, acesta asigurând fiabilitate a datelor, astfel având siguranța că datele transmise nu vor fi eronate. De altfel, TCP asigură o conexiune orientată, ceea ce înseamnă ca stabilirea conexiunii dintre client și server se realizează înaintea schimbului de date.

Pentru aplicația server vom utiliza următoarele biblioteci:

```
#include <stdio.h> //Pentru citire de la tastatura si afisarea textului in consola
#include <stdlib.h> //Libraria standard care permite operatii precum alocarea de memorie samd.
#include <string.h> //Pentru lucrul cu siruri de caractere
#include <sys/types.h> //Pentru definirea propriilor structuri de date
#include <sys/socket.h> //pentru utilizarea socket-urilor
#include <netinet/in.h> //Internet protocol family
#include <unistd.h> //Oferă acces la API-ul sistemului de operare POSIX
#include <pthread.h> //Pentru a  creea thread-uri
#include "cJSON/cJSON.h" //Librărie pentru utilizarea fișierelor JSON în C
#include "JSON_Functions.h" //Header file cu funcții pentru manipularea fișierelor de tip JSON
#include <sqlite3.h> //Librărie pentru utilizarea bazei de date SQLite
#include "SQLite_Functions.h" //Header file cu funcții pentru manipularea bazei de date SQLite
```

În aplicația client folosim în plus următoarele biblioteci:

```
#include <errno.h> //Pentru codul de eroare returnat de anumite apeluri
#include <netdb.h> //Conține definiții pentru operațiunile de bază de date în rețea
#include <signal.h> //Pentru a folosi semnale
```

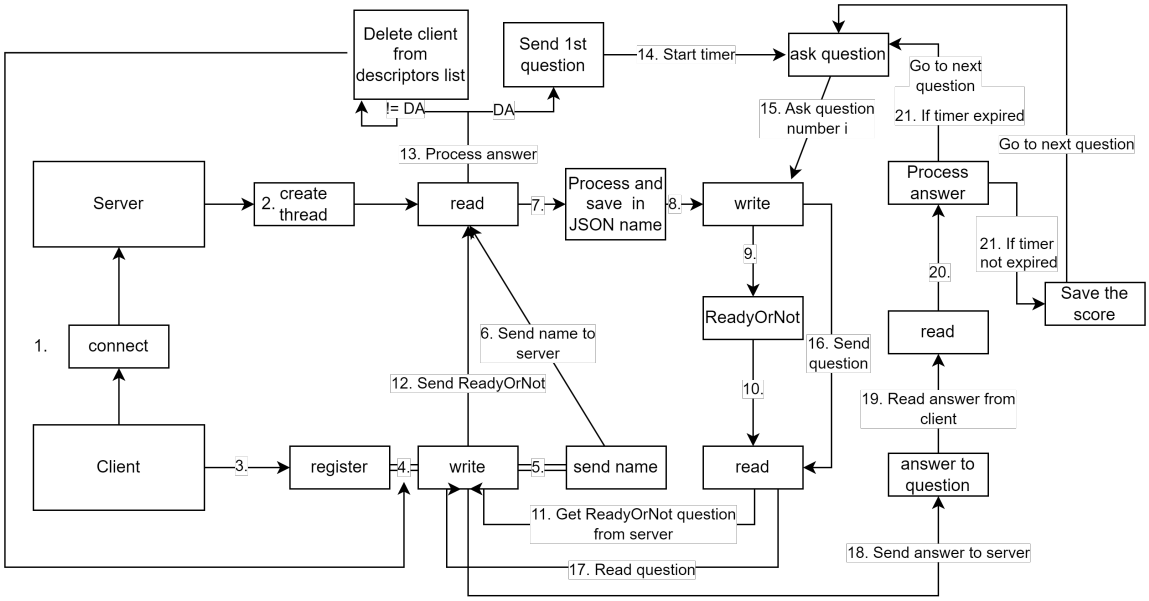
3 Arhitectura aplicatiei

3.1 Aspecte legate de conexiune

Pentru a realiza conexiunea dintre server și clienți, vom stabili familia de socket-uri și definim un port pentru conectare. Atașăm socket-ul folosind funcția bind(). Cu ajutorul funcției listen(), serverul va asculta dacă vin clienți să se conecteze. Vom folosi funcția accept() din librăria $< sys/socket.h >$ pentru a accepta conectarea clienților la server.

3.2 Structura aplicației

3.2.1 Diagrama detaliată



3.2.2 Server:

Socket: Se utilizează pentru a crea un canal de comunicare între server și clienți. Funcționează ca o interfață de rețea pentru transmiterea datelor.
Multithreading: Odată cu conectarea unui client la server, se va crea un nou fir de execuție specific clientului, pentru a asigura concurența serverului

3.2.3 Client:

Socket: La fel ca și în cazul serverului, clienții utilizează socket-uri pentru a comunica cu serverul.
Interfață cu Utilizatorul: Clienții comunică cu serverul prin trimiterea și primirea de mesaje în consolă.

3.2.4 Comunicare:

TCP: Asigură comunicarea server-client(și invers) cu avantajele livrării fiabile și ordonate a datelor.
Mesaje: Serverul și clienții schimbă mesaje pentru a realiza cerințele proiectului QuizzGame. Mesajele vor fi de tipul întrebări și răspunsuri sau alte informații necesare în cadrul aplicației, cum ar fi timp rămas pentru răspuns, anunțarea câștigătorului, afișarea scorului actual.

3.2.5 Sincronizare:

Mutex: Asigură sincronizarea accesului concurent la resurse partajate.
Barieră: Asigură sincronizarea clienților, astfel încât toți să ajungă la un punct specific înainte de a continua. Aceasta este utilă pentru a asigura că toți clienții au trimis un mesaj înainte ca serverul să răspundă.

Pentru implementarea codului am folosit codul din cadrul laboratorului 9 al cursului de Rețele de calculatoare, modificând serverul și clientul pentru a integra funcționalități precum înregistrarea clienților, verificarea raspunsurilor date, încărcarea întrebărilor dintr-o bază de date SQLite, etc. De asemenea, funcțiile folosite pentru fișierele JSON și baza de date SQLite sunt inspirate din diverse resurse online.

4 Detalii de implementare

4.1 Scenarii de utilizare

Comunicarea dintre client și server se realizează cu ajutorul funcției socket. Vom folosi structurile sockaddr_in server și sockaddr_in from.

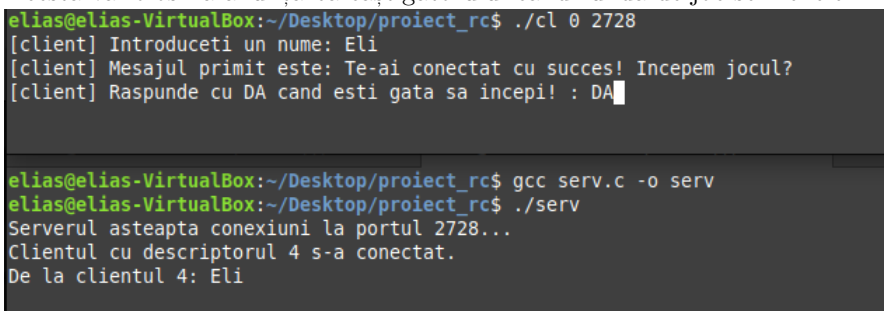
Pe partea de client vom folosi "struct sockaddr_in server;" pentru a ne conecta. Dacă conexiunea la server s-a realizat cu succes, se va crea câte un thread pentru fiecare client, cu ajutorul căruia serverul comunică cu clienții și invers, păstrându-se concurența.

```
connect (sd, (struct sockaddr *) &server,sizeof (struct sockaddr))
```

Odată conectat la server, user-ul va trebui să introducă un nume pentru înregistrare, care va fi salvat într-un fișier JSON de forma:

```
{   "clients": [{
            "name": "nume: John Doe\n", \\numele cu care s-a înregistrat
            "fd": 4,                    \\clientul cu descriptorul 4
            }, {
            "name": "nume: Marta Popescu\n",
            "fd": 5
            }]
}
```

Acesta va folosi la anunțarea câștigătorului când runda de joc se încheie.



După înregistrarea clienților urmează ca serverul să trimită la fiecare client înregistrat mesajul: "Te-ai conectat cu succes! Incepem jocul?"

```
[server]Clientul cu descriptorul 5 s-a conectat.
[client] Introduceti un nume: Marta Popescu
[server] De la clientul 5: nume: Marta Popescu
[client] Mesajul primit este: Te-ai conectat cu succes! Incepem jocul?
[client] Raspunde cu DA cand esti gata sa incepi! :
```

Atunci când primul client care se anunță ca fiind gata de joc (practic aici are loc înregistrarea în joc din enunț), clientul primește prima întrebare și va începe un timer global pentru toți clienții care se vor conecta ulterior. Astfel, dacă de exemplu clientul cu descriptorul 4 începe jocul primul, și întrebarea respectivă are un timp alocat de 15 secunde, dacă dupa ce au trecut 7 secunde se conectează la joc alt client, acesta va începe de la întrebarea la care se află descriptorul 4 și va avea alocat ca timp de răspuns doar 8 secunde, cele rămase și pentru clientul de la descriptorul 4. Astfel se asigură sincronizarea între clienți.

Fiecare client dintr-o sesiune de joc va avea același timp limitat pentru a răspunde la întrebări, și vor trimite către server litera corespunzătoare răspunsului considerat corect, după care vor aștepta expirarea timpului de răspuns pentru a aștepta restul clienților care poate răspund mai greu sau chiar deloc. Apoi serverul va procesa răspunsul primit de la clienți și le va trimite înapoi scorul actual. Astfel, serverul va continua să trimită întrebări la fiecare client până când setul de întrebări se termină. La final, fiecare client va primi un mesaj cu numele câștigătorului.

Dacă un client se deconectează, acesta va fi eliminat din fișierul JSON și nu va mai primi în continuare întrebări, și nici scorul nu va mai fi comparat pentru anunțarea câștigătorului.

4.2 Baza de date SQLite

Pentru a stoca și accesa întrebările puse de server, am creat o bază de date SQLite cu următoarea structură:

```
"CREATE TABLE IF NOT EXISTS questions ("
"id INTEGER PRIMARY KEY AUTOINCREMENT,"
"question_text TEXT NOT NULL,"
"answer_a TEXT NOT NULL,"
"answer_b TEXT NOT NULL,"
"answer_c TEXT NOT NULL,"
"correct_answer TEXT NOT NULL);";
```

ID	Question Text	Answer A	Answer B	Answer C	Correct answer
1	question 1	Option A	Option B	Option C	A
2	question 1	Option X	Option Y	Option Z	Z

Table 1: Structura bazei de date

Am folosit urmatoarele funcții pentru manipularea bazei de date:

```
\\Pentru afisarea bazei de date (doar pentru debug)
void displayAllQuestions(size_t bufferSize);

\\In cazul in care am nevoie sa sterg continutul bazei de date, resetez ID-urile intrebarilor
void resetAutoIncrement();

\\Pentru initializare baza de date
void initializeDatabase();
```

```
\\Pentru inserarea unor valori in baza de date
void insertQuestion(const char *questionText, const char *answerA, const char *answerB,
const char *answerC, const char *correctAnswer);

 \\Pentru a afisa raspunsul curent la intrebarea data ca parametru prin id
void printCorrectAnswer(int id) ;

\\Pentru a obtine raspunsul curent la intrebarea data ca parametru prin id, folosita pentru
verificarea raspunsului dat de client
char* getCorrectAnswer(int id) ;

 \\Pentru a afisa intrebarea cu id-ul dat ca parametru in server
char* displayQuestion(int id, size_t bufferSize) ;

\\Pentru a trimite intrebarea cu id-ul dat ca parametru din server la client
char* getQuestion(int id) ;

 \\Pentru a sterge continutul bazei de date (doar pentru a schimba intrebarile)
void deleteDatabaseContent() ;

\\Pentru a executa instructiunile SQL
int callback(void *NotUsed, int argc, char **argv, char **azColName)
```

Pentru a adauga mai multe întrebări deodată folosim:

```
void addQuestions(){
    insertQuestion("What is the capital of France?", "Paris", "Berlin", "London", "a");
    insertQuestion("Which planet is known as the Red Planet?", "Venus", "Mars", "Jupiter",
"b");
    insertQuestion("What's the national animal of Australia?", "Tasmanian Devil", "Red
Kangaroo", "Koala", "b");
    insertQuestion("In which year did the Titanic sink?", "1905", "1931", "1912", "c");
    insertQuestion("What is the largest mammal in the world?", "Blue Whale", "Elephant",
"Hippopotamus", "a");
    insertQuestion("What is the longest river in the world?", "Amazon River", "Yangtze
River", "Nile River", "c");
    insertQuestion("Which of the following empires had no written language?", "Roman",
"Egyptian", "Incan", "c");
    insertQuestion("What's the smallest country in the world?", "Monaco", "Vatican", "Nauru",
"b");
    insertQuestion("What city do The Beatles come from?", "Liverpool", "Munchen", "Birmingham",
"a");
    insertQuestion("What country has the highest life expectancy?", "Switzerland", "Canada",
"Japan", "c");
    insertQuestion("Who was the Ancient Greek God of the Sun?", "Zeus", "Apollo", "Hermes",
"b");
    insertQuestion("What country drinks the most coffee?", "Brazil", "Finland", "Italy",
"b");
    insertQuestion("What is Cynophobia?", "Fear of dogs", "Fear of heights", "Fear of
spiders", "a");
    insertQuestion("How many languages are written from right to left?", "10", "11", "12",
"c");
    insertQuestion("Who was the first woman to win a Nobel Prize (in 1903)?", "Rosalind
Franklin", "Maria Curie", "Jane Goodall", "b");
}
```

4.3 Manipularea fișierului JSON

Pentru a folosi fișierul JSON în program am declarat și implementat următoarele funcții:

```
\\Extrage numele unui jucător din fișierul JSON, după descriptorul dat ca parametru
char* extractNameByDescriptor(int descriptor, const char* filename) {

\\Atunci cand un client se deconecteaza, se apeleaza aceasta functie pentru a-l elimina
din JSON
void removeClientByFd(const char* filename, int client_fd)

\\ Inainte de initializarea serverului se sterge continutul fisierului pentru siguranta
void deleteFileContent(const char* filename)

\\ Atunci cand un client se inregistreaza, se apeleaza aceasta functie
void addClient(const char* filename, int client_fd, const char* client_name)
```

Este important de menționat că am folosit librăria cJSON pentru a implementa aceste funcții.

Structura fișierului JSON este dată de:

```
cJSON* clientObject = cJSON_CreateObject();
cJSON_AddStringToObject(clientObject, "name", client_name);
cJSON_AddNumberToObject(clientObject, "fd", client_fd);
```

4.4 Server

4.4.1 Inițializare server

```
void initialize_server() {
int server_fd, client_fd, i;
struct sockaddr_in server_addr, client_addr;
socklen_t client_len;

// Creare socket pentru server
server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd < 0) {
    perror("Eroare la crearea socket-ului");
    exit(EXIT_FAILURE);
}

// Inițializare structura server_addr
bzero((char*)&server_addr, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(PORT);

// Legare socket la adresa și port
if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
    perror("Eroare la legarea socket-ului");
    exit(EXIT_FAILURE);
}

// Ascultare pentru conexiuni
listen(server_fd, 5);

// Inițializare vector de descriptori de clienti
pthread_mutex_init(&mutex, NULL);
```

```
    pthread_barrier_init(&barrier, NULL, MAXTHREADS);
    for (i = 0; i < MAXTHREADS; ++i) {
        client_fds[i] = -1;
        client_scores[i] = 0;
    }

    printf("Serverul asteapta conexiuni la portul %d...\n", PORT);

    while (1) {
        // Accepta conexiunea de la un client
        client_len = sizeof(client_addr);
        client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &client_len);
        if (client_fd < 0) {
            perror("Eroare la acceptarea conexiunii");
            exit(EXIT_FAILURE);
        }

        // Adauga noul client la vectorul de descriptori
        pthread_mutex_lock(&mutex);
        for (i = 0; i < MAXTHREADS; ++i) {
            if (client_fds[i] == -1) {
                client_fds[i] = client_fd;
                pthread_create(&threads[i], NULL, handle_client, (void*)&client_fds[i]);
                printf("Clientul cu descriptorul %d s-a conectat.\n", client_fd);

                break;
            }
        }
        pthread_mutex_unlock(&mutex);
    }

    pthread_barrier_destroy(&barrier);
    pthread_mutex_destroy(&mutex);
    close(server_fd);
}
```

4.4.2 Comunicare cu clienții - Gestionarea Întrebărilor

```
    void* handle_client(void* arg) {
    int client_fd = *((int*)arg);
    int total_questions = 0;

    int i = currentQuestion;
    char buffer[1024];
    int bytes_received;


    while(total_questions<MAX_Questions){
        // Primeste date de la client
        bytes_received = recv(client_fd, buffer, sizeof(buffer), 0);
        if (bytes_received <= 0) {
            // Eroare sau clientul s-a deconectat
            printf("Clientul cu descriptorul %d s-a deconectat.\n", client_fd);
            shutdown(client_fd, SHUT_RDWR);
            // Excludere client din vectorul de descriptori
```

```
            pthread_mutex_lock(&mutex);
            close(client_fd);
            for (int i = 0; i < MAXTHREADS; ++i) {
                if (client_fds[i] == client_fd) {
                    client_fds[i] = -1;
                    client_scores[i] = 0;
                    break;
                }
            }
            pthread_mutex_unlock(&mutex);
            const char* filename = "clients.json";
            removeClientByFd(filename, client_fd);
            printf("Clientul cu descriptorul %d eliminat din fisierul JSON.\n", client_fd);
            break;
        } else
        {
            // Procesează datele de la client
            buffer[bytes_received] = '\0';
            printf("De la clientul %d: %s\n", client_fd, buffer);

            if (timer_expired && client_fd == current_client_fd) {
                // Trimite un mesaj către client că timpul a expirat
                // printf("Sending message to client about timer expiration.\n");
                // const char *timeout_message = "Timpul a expirat! Te rog raspunde la
intrebare.";
                // send(client_fd, timeout_message, strlen(timeout_message), 0);

                // Resetează variabilele pentru următorul ciclu
                timer_expired = 0;
                current_client_fd = -1;
            }

            const char *filename = "clients.json";
            if (strstr(buffer, "quit") != NULL){
                printf("Clientul cu descriptorul %d s-a deconectat.\n", client_fd);
                pthread_mutex_lock(&mutex);
                if(ready_clients!=0){
                    ready_clients--;
                    currentQuestion = 1;
                }
                for (int i = 0; i < MAXTHREADS; ++i) {
                    if (client_fds[i] == client_fd) {
                        client_fds[i] = -1;
                        break;
                    }
                }
                pthread_mutex_unlock(&mutex);
                break;
            }
            else
            if (strstr(buffer, "nume") != NULL) {
                addClient(filename, client_fd, buffer);
                pthread_mutex_lock(&mutex);
                const char* response = "Te-ai conectat cu succes! Incepem jocul?";
                send(client_fd, response, strlen(response), 0);
                pthread_mutex_unlock(&mutex);
```

```
            }
            else
            if (strstr(buffer, "ready") != NULL){
                ready_clients++;
                // Trimite un mesaj de la server către client
                char response[100];
                // Procesează răspunsul de la client
                buffer[bytes_received] = '\0';
                remove_newline(buffer);

                printf("Clientul %d a raspuns: %s\n", client_fd, buffer);
                if (strstr(buffer, "DA") !=NULL || strstr(buffer, "Da") !=NULL || strstr(buffer,
"da") !=NULL ) {
                    printf("Clientul %d a inceput jocul.\n", client_fd);

                    pthread_mutex_lock(&mutex);
                    current_client_fd = client_fd;
                    pthread_mutex_unlock(&mutex);

                    currentQuestion = i;
                    askQuestion(client_fd, i);


                    if (ready_clients == 1) {
                        pthread_mutex_lock(&timer_mutex);
                        if (!timer_started) {
                            timer_started = 1;
                            pthread_create(&timer_thread, NULL, timer_function, NULL);
                        }
                        pthread_mutex_unlock(&timer_mutex);
                        while (!timer_expired) {
                            sleep(1);
                            pthread_mutex_lock(&mutex);
                            int current_remaining_time = remaining_time;
                            pthread_mutex_unlock(&mutex);
                            printf("Timer thread: Remaining time: %d seconds\n", current_remaining_ti
                        }

                        // timer_expired = 0;
                    }
                    else {
                        int rem_time = remainingTime();
                        waitFor(rem_time);
                    }

                }
                else {
                    printf("Clientul %d nu a inceput jocul. Acesta va fi eliminat :(\n",
client_fd);
                    printf("Clientul cu descriptorul %d a fost deconectat.\n", client_fd);
                    pthread_mutex_lock(&mutex);
                    if(ready_clients!=0){
                        ready_clients--;
                        currentQuestion = 1;
                    }
                    for (int i = 0; i < MAXTHREADS; ++i) {
```

```
                        if (client_fds[i] == client_fd) {
                            client_fds[i] = -1;
                            break;
                        }
                    }
                    pthread_mutex_unlock(&mutex);
                    break;
                    }
            }
            else
            if (strstr(buffer, "raspuns") != NULL) {


                char response[100];
                const char *start = strstr(buffer, "raspuns: ");
                if (start != NULL) {
                    char answer = start[strlen("raspuns: ")];

                    printf("Clientul %d a raspuns cu litera '%c' la intrebarea %d. \n",
client_fd, answer, i);

                    char* right_answer = getCorrectAnswer(i);
                    int current_score = 0;
                    if (strncmp(&answer, right_answer, 1) != 0) {
                        current_score = updateScore(client_fd, 0); // Scorul rămâne neschimbat
la răspuns greșit
                        sprintf(response, "Raspuns gresit! Scorul tau actual: %d\n", current_score);
                        printf("Raspuns gresit! Scorul tau actual: %d\n\n", current_score);
                        printf("-------------------------------------------------\n\n");
                        send(client_fd, response, strlen(response), 0);
                    } else {
                        current_score = updateScore(client_fd, 1); // Actualizează scorul
cu 1 punct la răspuns corect
                        sprintf(response, "Raspuns corect! Scorul tau actual: %d\n", current_score);
                        printf("Raspuns corect!Scorul tau actual: %d\n\n", current_score);
                        send(client_fd, response, strlen(response), 0);
                    }
                } else {
                    printf("Nu s-a gasit inceputul răspunsului.\n");
                }
                timer_expired=0;
            }

            else
            if (strstr(buffer, "next") != NULL){
                i++;

                printf("Primita comanda 'next' de la client.\n");

                char response[100];
                if (i > MAX_Questions) {
                    int winnerIndex = findWinner(client_scores, MAXTHREADS);
                    char* winnerName= extractNameByDescriptor(client_fds[winnerIndex],
"clients.json");
                    if (winnerIndex != -1) {
                        printf("Câștigătorul este %s\n.", winnerName);
```

```
                        sprintf(response,"Câștigătorul este %s\n.", winnerName);
                    } else {
                        sprintf(response,"Niciun câștigător. Scoruri egale sau niciun
participant.\n");
                    }

                    currentQuestion = 1;
                    send(client_fd, response, strlen(response), 0);
//                    shutdown(client_fd, SHUT_RDWR);
//                    close(client_fd);

                    break;
                }
                currentQuestion = i;

                askQuestion(client_fd, i);
                if (ready_clients == 1) {
                        printf("Timer started?: %d\n", timer_started);
                        pthread_mutex_lock(&timer_mutex);
                        if (!timer_started) {
                            timer_started = 1;
                            pthread_create(&timer_thread, NULL, timer_function, NULL);
                        }
                        pthread_mutex_unlock(&timer_mutex);
                        while (!timer_expired) {
                            sleep(1);
                            pthread_mutex_lock(&mutex);
                            int current_remaining_time = remaining_time;
                            pthread_mutex_unlock(&mutex);
                            printf("Timer thread: Remaining time: %d seconds\n", current_remaining_ti
                        }
                    }
                    else {
                        int rem_time = remainingTime();
                        waitFor(rem_time);
                    }
            }
        }
    }
}
```

4.4.3 Gestionarea Timer-ului

Idea din spatele timer-ului este de a sincroniza toți clienții, folosindu-ne de faptul că ei au un timp maxim de răspuns alocat întrebărilor. Deci, am gândit timer-ul ca fiind un proces care lucrează în fundal și este pornit inițial doar când prima întrebare este trimisă la 1 client (sau mai mulți). Totuși, primul client conectat la joc pornește timer-ul:

```
     if (ready_clients == 1) {
        printf("Timer started?: %d\n", timer_started);
        pthread_mutex_lock(&timer_mutex);
        if (!timer_started) {
            timer_started = 1;  // Setează flag-ul pentru a indica că timer-ul a fost
pornit
            pthread_create(&timer_thread, NULL, timer_function, NULL);
        }
```

```
        pthread_mutex_unlock(&timer_mutex);
        while (!timer_expired) {
            sleep(1);
            pthread_mutex_lock(&mutex);
            int current_remaining_time = remaining_time;
            pthread_mutex_unlock(&mutex);
            printf("Timer thread: Remaining time: %d seconds\n", current_remaining_time);
        }
    }
```

Astfel, atunci când variabile ready_client, folosită pentru contorizarea clienților care au confirmat că sunt gata de joc, este egală cu 1, se pornește un thread separat pentru timer, și astfel se contorizează secundele și orice client conectat în mijlocul unei întrebări, va avea același timp de răspuns ca și ceilalți.

```
void *timer_function(void *arg) {
    while (1) {
        sleep(1);

        pthread_mutex_lock(&timer_mutex);
        pthread_mutex_lock(&mutex);
        remaining_time--;

        if (remaining_time <= 0) {
            printf("Timer expired!\n");
            timer_expired = 1;
            remaining_time = MAX_Time;

            current_client_fd = -1;
            for (int i = 0; i < MAXTHREADS; ++i) {
                if (client_fds[i] != -1) {
                    current_client_fd = client_fds[i];
                    break;
                }
            }

            pthread_mutex_unlock(&mutex);
            pthread_mutex_unlock(&timer_mutex);

            if (current_client_fd != -1) {

                printf("Sending message to client about timer expiration.\n");
                const char *timeout_message = "Timpul a expirat! Te rog raspunde la intrebare.";
                send(current_client_fd, timeout_message, strlen(timeout_message), 0);

            }
        } else {
            pthread_mutex_unlock(&mutex);
            pthread_mutex_unlock(&timer_mutex);
        }
    }

    return NULL;
}
```

Odată expirat timpul, serverul trimite un mesaj către toți clienții, pentru ai informa, iar apoi le trimite un mesaj în funcție de răspunsul lor la întrebare:

- Răspuns corect! Scorul tău actual: X
- Răspuns greșit! Scorul tău actual: X
- Timpul a expirat! Scorul tău actual: X

.

4.5 Client

4.5.1 Înregistrare client

```
    void registerClient(int sd) {
  char msg[100];

  /* citirea mesajului */
  bzero(msg, 100);
  printf("[client] Introduceti un nume: ");
  fflush(stdout);
  read(0, msg, 100);

  /* trimiterea mesajului la server */
  if (write(sd, msg, 100) <= 0)
  {
    perror("[client] Eroare la write() spre server.\n");
    exit(errno);
  }

  /* citirea raspunsului dat de server (apel blocant pana cand serverul raspunde) */
  if (read(sd, msg, 100) < 0)
  {
    perror("[client] Eroare la read() de la server.\n");
    exit(errno);
  }

  /* afisam mesajul primit */
  printf("[client] Mesajul primit este: %s\n", msg);
}
```

4.5.2 Condiție start Quizz

```
void readyOrNot(int sd) {
  char msg[100];

    /* citirea mesajului */
    bzero(msg, 100);
    printf("[client] Raspunde cu DA cand esti gata sa incepi! : ");
    fflush(stdout);
    read(0, msg, 100);

    /* trimiterea mesajului la server */
    if (write(sd, msg, 100) <= 0)
    {
      perror("[client] Eroare la write() spre server.\n");
      exit(errno);
    }
```

```
    /* citirea raspunsului dat de server (apel blocant pana cand serverul raspunde) */
    if (read(sd, msg, 100) < 0)
    {
      perror("[client] Eroare la read() de la server.\n");
      exit(errno);
    }


    /* afisam mesajul primit */
    printf("[client] Mesajul primit este: %s\n", msg);


}
```

4.5.3 Răspunde la întrebări

```
void answer(int sd) {
    char msg[1024];
    char fullMsg[1033];

    fd_set read_fds;
    FD_ZERO(&read_fds);

    while (1) {
        FD_SET(0, &read_fds); // Descriptorul pentru citire de la tastatură
        FD_SET(sd, &read_fds); // Descriptorul pentru citire de la server

        // Setăm timeout la 1 secundă
        struct timeval timeout;
        timeout.tv_sec = 1;
        timeout.tv_usec = 0;

        int ready_fds = select(sd + 1, &read_fds, NULL, NULL, &timeout);

        if (ready_fds < 0) {
            perror("[client] Eroare la select().\n");
            exit(errno);
        } else if (ready_fds > 0) {
            if (FD_ISSET(0, &read_fds)) {
                // Citire de la tastatură
                bzero(msg, sizeof(msg));
                fflush(stdout);
                fgets(msg, sizeof(msg), stdin);
                remove_newline(msg);

                snprintf(fullMsg, sizeof(fullMsg), "raspuns: %s", msg);

                // Trimitere mesaj la server
                if (write(sd, fullMsg, sizeof(fullMsg)) <= 0) {
                    perror("[client] Eroare la write() spre server.\n");
                    exit(errno);
                }
            }
```

```
            if (FD_ISSET(sd, &read_fds)) {
                // Citire de la server
                bzero(msg, sizeof(msg));
                if (read(sd, msg, sizeof(msg)) < 0) {
                    perror("[client] Eroare la read() de la server.\n");
                    exit(errno);
                }

                if (strstr(msg, "expirat") != NULL) {
                    timer_expired = 1;
                }

                if (timer_expired) {
                    printf("Handle timer expiration logic here.\n");

                    timer_expired = 0;

                }

                printf("[client] Mesajul primit este: %s\n", msg);

                // Trimitere comandă "next" la server
                if (write(sd, "next", strlen("next") + 1) <= 0) {
                    perror("[client] Eroare la write() spre server.\n");
                    exit(errno);
                }

                // Citire de la server pentru comanda "next"
                bzero(msg, sizeof(msg));
                if (read(sd, msg, sizeof(msg)) < 0) {
                    perror("[client] Eroare la read() de la server.\n");
                    exit(errno);
                }

                printf("--------------------------------------------------\n");
                printf("[client] Mesajul primit este: %s\n", msg);
            }
        }
    }
}
```

4.6 Protocol de comunicare

- Se conectează clientul la server
- Serverul creează un thread pentru acest client
- Clientul trimite la server un nume
- Serverul primește numele, îl înregistrează și trimite înapoi un mesaj, întrebând clientul dacă este pregătit pentru începerea jocului
- Clientul trimite la server ”DA” dacă este pregătit
- Serverul primește mesajul și verifică dacă clientul respectiv este sau nu pregătit
- Dacă răspunsul este ”DA”, ii trimite prima întrebare și pornește un timer definit global pe un thread nou

- Clientul poate raspunde la intrebare in timpul alocat (15s), altfel indiferent de raspuns, scorul său nu va fi contorizat
- Dacă un alt client se conectează în timpul unei întrebări, acesta are același timp de răspuns rămas ca ceilalți jucători, deci este dezavantajat
- Serverul preia răspunsul de tipul "a", "b", "c" de la toți clienții odată ce timpul pentru întrebare a expirat și verifică dacă este cel corect. Dacă da, punctajul acestuia crește cu 1 punct pentru acel client, altfel, punctajul va rămâne neschimbat. Dacă clientul nu a răspuns la timp, va fi atenționat iar răspunsul său nu va fi contorizat.
- Dacă un client s-a deconectat, acesta va fi eliminat din fișierul JSON, se va opri comunicarea cu serverul și nu v-a fi contorizat la anunțarea câștigătorului
- După ce s-a epuizat lista de întrebări, toți clienții din sesiunea curentă de joc vor primi rezultatul final care conține numele câștigătorului.
- Un client se poate deconecta folosind comanda "quit".

5 Concluzii

În prezent, există câteva probleme legate de timer, în sensul în care se pornește un thread doar dacă e un singur client conectat, și deci apar probleme la comunicare pe alocuri, astfel că clienții nu primesc notificarea cum că timpul a expirat, de fiecare dată, sau comunicarea se blochează deodată.

Altă problemă curentă ar fi că uneori, pentru a trece mai departe ca fiind primul client conectat, trebuie introdus un input, pentru a trimite serverului un semn că poți primi următoarea întrebare. Și aici este o problemă din cauza timer-ului care nu a fost foarte riguros implementat.

De asemenea, ar fi o îmbunătățire ca și clienții să vadă timpul rămas așa cum este afișat în server.

Proiectul ar putea fi îmbunătățit și cu ajutorul unei interfețe grafice pentru clienți, dar și cu un managment al sesiunilor de joc mai eficient. Pentru interfața grafică s-ar putea utiliza GDK+, o librărie grafică ce permite crearea unui GUI.

Managmentul sesiunilor de joc ar ajuta utilizatorii în alegerea adversarilor, de exemplu dacă un grup de prieteni vor să joace împreună în aceeași rundă, sau dacă vor să joace cu persoane random distribuite. Totuși, în acest ultim caz, ar trebui să fie suficiente conexiuni la server încât orice client conectat la server, la orice oră, să aibă parteneri de joc.

Un alt mod de a îmbunătăți acest proiect ar fi crearea unui set foarte mare de întrebări pentru a randomiza întrebările primite de către clienți, astfel încât de fiecare dată să primească întrebări diferite, cu probabilitate suficient de mică să primească aceleași set de întrebări, primit în altă sesiune de joc.

6 Bibliografie

Laboratoarele 7-9 din cadrul cursului Rețele de calculatoare
https://www.wikipedia.org/
https://man7.org/linux/man-pages
https://stackoverflow.com/
https://www.ibm.com/
https://www.geeksforgeeks.org/