ReadsProfiler

Diana Martișcă

Facultatea de Informatică, "Universitatea Alexandru Ioan Cuza", Iași
dianamariamartisca@gmail.com

1 Introducere

1.1 Viziune

Proiectul ReadsProfiler constă în implementarea unui server concurent și a unui client, pentru accesarea unei librării online. Clienții pot căuta, descărca, evalua și cere recomandări de cărți, în funcție de preferințele lor. Pe măsură ce activitatea clienților crește, aceștia vor primi recomandări din ce în ce mai bune, pe baza căutărilor, descărcărilor și rating-ului oferit.

1.2 Obiective

1. **Crearea Bazei de Date**
   Pentru a memora informațiile despre cărți, autori, genuri, căutări și descărcări, am utilizat o bază de date, salvată într-un fișier denumit "database.db". Am creat tabelele *clients, books, authors, genres, books_genres, authors_genres, search_history, rating_history.*
2. **Infrastructura Client-Server**
   Am creat un server capabil să accepte conexiuni din partea mai multor clienți, ale căror cereri le tratează în paralel, prin thread-uri. Comunicarea dintre client și server se face prin socket-uri.
3. **Căutarea, Descărcarea și Evaluarea Cărților**
   Pentru căutarea și descărcarea anumitor cărți, autori, genuri am utilizat interogări ale bazei de date și inserări în tabela *search_history.* Descărcarea se face prin trimiterea conținutului cărții clientului, care creează un fișier **txt** în memorie. Evaluarea cărților se realizează prin inserări în tabela *rating_history.*
4. **Oferirea de Recomandări**
   Recomandările sunt făcute prin interogări ale tabelei *search_history*, pentru a determina cărțile pe care le-a descărcat clientul, informații pe baza cărora caut câteva cărți asemănătoare.

2 Tehnologii Aplicate

1. **TCP**
   Protocolul de Comunicație în Timp Real (TCP) a fost utilizat pentru a asigura o conexiune stabilă și fiabilă între server și clienți. TCP este un protocol de transport orientat conexiune, fără pierderi de informații, care oferă o calitate maximă a serviciilor. Acesta integrează mecanisme de stabilire și de eliberare a conexiunii și controlează fluxul de date. Totodată, creează conexiunea dintre client și server înainte ca datele să fie trimise și se ocupă de închiderea corectă a conexiunii la finalul sesiunii de lucru. Este capabil să detecteze eventuale erori, să secvențieze datele și să le retransmită, în cazul în care se pierd.
2. **Sockets**
   Socket-urile permit crearea unui canal de comunicație bidirecțional între server și clienți, facilitând schimbul de date în timp real. Acestea asigură livrarea fiabilă a datelor, ordonarea lor și detecția erorilor, dar și o interfață flexibilă pentru gestionarea conexiunilor, fiind indispensabile în lucrul cu TCP.
3. **Threads**
   Firele de execuție (threads) sunt unități de execuție mai mici ale unui program care pot rula concurent, împărtășind aceleași resurse ale procesului în care sunt create, cum ar fi spațiul de memorie și variabilele globale. Ele permit execuția simultană a mai multor sarcini, asigurând concurența serverului și îmbunătățind astfel performanța și timpul de răspuns al aplicației.
4. **SQLite3**
   Pentru stocarea și gestionarea datelor am utilizat SQLite3. Este un sistem de gestionare a bazelor de date încorporat, ceea ce înseamnă că este integrat direct în aplicație, fiind ușor de administrat și de interogat.

3 Structura Aplicației

3.1 Structura Serverului

Serverul apelează funcția **socket()**, salvează descriptorul returnat într-o variabilă și populează structura de date utilizată de server. Apoi, atașează socket-ului informațiile prin apelul funcției **bind()**.

Ascultă dacă se conectează clienți, folosind funcția **listen()**.

Intră într-o buclă infinită, în cadrul căreia acceptă conexiunea cu un client prin apelul funcției **accept()**.

Creează un thread folosind descriptorul întors de **accept()**, prin apelul funcției **pthread_create()**.

În handler-ul thread-ului, **treat()**, serverul intră într-o buclă infinită, în cadrul căreia citește comanda trimisă de client și o procesează. În timpul procesării comenzii, serverul accesează informațiile conținute de baza de date "database.db" și, eventual, le modifică, în funcție de comanda clientului. În urma procesării unei comenzi, serverul trimite clientului un răspuns conform cu cererea trimisă.

Dacă clientul părăsește aplicația, prin trimiterea unui semnal precum **SIGINT** sau prin comanda **"quit"**, acesta va ieși din bucla infinită și va închide conexiunea cu clientul.

3.2 Structura Clientului

Clientul apelează funcția **socket()**, salvează descriptorul returnat într-o variabilă și populează structura de date utilizată pentru conexiunea cu serverul, cu IP-ul și portul oferite la lansarea în execuție a clientului.

Se conectează la server prin apelul primitivei **connect()**.

După conectarea la server, intră într-o buclă infinită, în care se citesc comenzi date de utilizator și se trimite serverului mesajul. După ce serverul va trimite un răspuns adecvat, îl citește și îl afișează.

La întâlnirea comenzii **"quit"**, clientul va ieși din bucla infinită, întrerupând programul.

Dacă se va trimite serverului o comandă care presupune descărcarea unei cărți, atunci serverul va trimite înapoi clientului un text, acestuia revenindu-i sarcina de a crea un document **txt** în memoria locală.

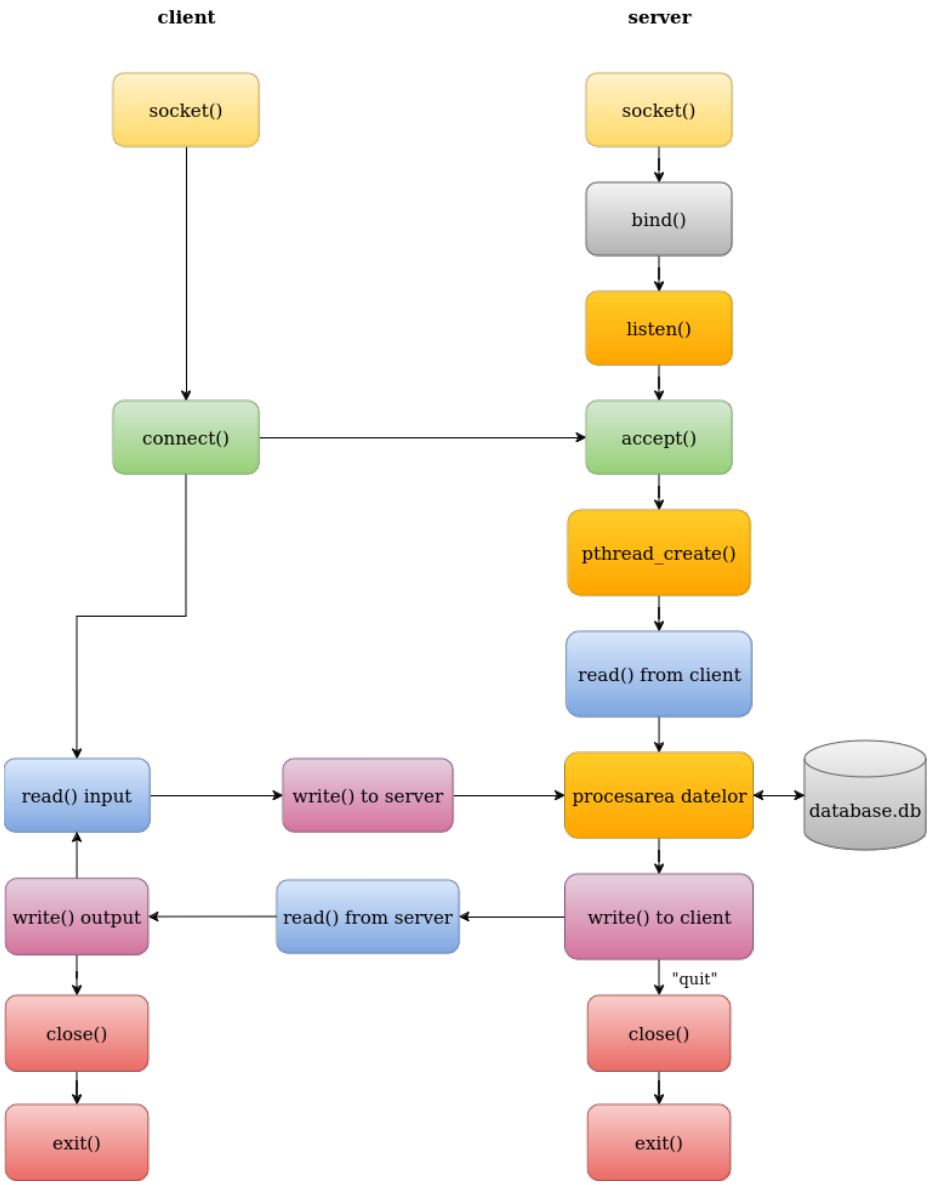


Fig. 1. Diagramă a aplicației

4 Aspecte de Implementare

4.1 Implementarea Serverului

```c
// Funcție pentru a deschide baza de date
int open_database(const char *filename)
{
    if (sqlite3_open(filename, &db))
    {
        fprintf(stderr, "Eroare la deschiderea bazei de date: %s\n", sqlite3_errmsg(db));
        return 0;
    }
    return 1;
}
```

Fig. 2. Deschiderea bazei de date

```c
/* crearea unui socket */
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
    perror("[server] Eroare la socket().\n");
    return errno;
}

/* utilizarea optiunii SO_REUSEADDR */
int on = 1;
setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
```

Fig. 3. Crearea socket-ului

```c
/* pregatirea structurilor de date */
bzero(&server, sizeof(server));

server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(PORT);
```

Fig. 4. Pregătirea structurii utilizate de server

```c
/* atasam socketul */
if (bind(sd, (struct sockaddr *) &server, sizeof(struct sockaddr)) == -1)
{
    perror("[server] Eroare la bind().\n");
    return errno;
}
```

Fig. 5. Atașarea socket-ului cu bind()

```c
/* punem serverul sa asculte daca vin clienti sa se conecteze */
if (listen(sd, 2) == -1)
{
    perror("[server] Eroare la listen().\n");
    return errno;
}
```

Fig. 6. Așteptarea apariției clienților cu listen()

```c
while(1)
{
    struct sockaddr_in from;
    socklen_t length = sizeof(from);
    int client;

    /* accepta o conexiune de la client */
    client = accept(sd, (struct sockaddr *) &from, &length);
    if (client < 0)
    {
        perror("[server] Eroare la accept().\n");
        continue;
    }
```

Fig. 7. Buclă infinită în care acceptăm clienți

```c
/* crearea unui nou thread pentru a gestiona clientul */
pthread_t threadId;
if (pthread_create(&threadId, NULL, &treat, (void *)(intptr_t)client) != 0)
{
    perror("[server] Eroare la crearea thread-ului.\n");
    close(client);
}
else
{
    /* detasam thread-ul pentru a nu necesita join */
    pthread_detach(threadId);
}
```

Fig. 8. Crearea unui thread pentru fiecare client

```c
/* functia care raspunde clientului */
void *treat(void *arg)
{
    int cl = (int)(intptr_t)arg;
    char buffer[256]; // buffer pentru comenzi
    char response[256];
    int ok=0; //utilizatorul nu e logat
    int id_user;
    while (1)
    {
        bzero(buffer, sizeof(buffer));

        // Citim mesajul de la client
        int bytesRead = read(cl, buffer, sizeof(buffer) - 1);
        if (bytesRead <= 0)
        {
            printf("[Thread] Conexiunea cu clientul cu descriptorul %d s-a inchis.\n",cl);
            break;
        }

        printf("[Thread] Mesajul primit: %s\n", buffer);
```

Fig. 9. Buclă infinită în funcția thread-ului

```
// Verificam daca clientul a trimis comanda "quit"
if (strncmp(buffer, "quit", 4) == 0)
{
    printf("[Thread] Clientul a trimis comanda 'quit'. Inchidem conexiunea.\n");
    break;
}
if (strncmp(buffer, "login", 5) == 0)
{
    printf("[Thread] Clientul a trimis comanda 'login'.\n");
    if(ok==1)
    {
        strcpy(response, "Utilizatorul este deja logat.");
    }
    else
    {
        ok=1;
        strcpy(response, "Utilizator logat cu succes.");
        char username[256];
        strcpy(username, buffer+8);
        id_user=get_id_user(username);
    }
}
```

Fig. 10. Verificare quit și login

```
else
{
    if(ok==0)
        strcpy(response, "Niciun utilizator logat. Utilizeaza login : <username>");
    else
    {
        if(strncmp(buffer, "search-title", 12)==0)
        {
            char title[256];
            strcpy(title, buffer+15);
            strcpy(response, search_title(title, id_user));
            snprintf(response, sizeof(response), "%s", search_title(title, id_user));
        }
        else
        if(strncmp(buffer, "search-author", 13)==0)
        {
            char author[256];
            strcpy(author, buffer+16);
            snprintf(response, sizeof(response), "%s", search_author(author, id_user));
        }
```

Fig. 11. Identificarea comenzii trimise de utilizator

```c
int get_id_user(const char *username)
{
    sqlite3_stmt *stmt;
    int user_id = -1;

    // caută utilizatorul în baza de date
    const char *query_select = "select client_id from clients where name = ?";
    if (sqlite3_prepare_v2(db, query_select, -1, &stmt, NULL) != SQLITE_OK)
    {
        fprintf(stderr, "Eroare la pregătirea select.\n");
        return -1;
    }

    sqlite3_bind_text(stmt, 1, username, -1, SQLITE_STATIC);

    if (sqlite3_step(stmt) == SQLITE_ROW)
    {
        user_id = sqlite3_column_int(stmt, 0);
    }
    sqlite3_finalize(stmt);

    // dacă utilizatorul nu există, inserează-l
    if (user_id == -1)
    {
        const char *query_insert = "insert into clients (name) values (?)";
        if (sqlite3_prepare_v2(db, query_insert, -1, &stmt, NULL) != SQLITE_OK)
        {
            fprintf(stderr, "Eroare la pregătirea insert.\n");
            return -1;
        }

        sqlite3_bind_text(stmt, 1, username, -1, SQLITE_STATIC);
```

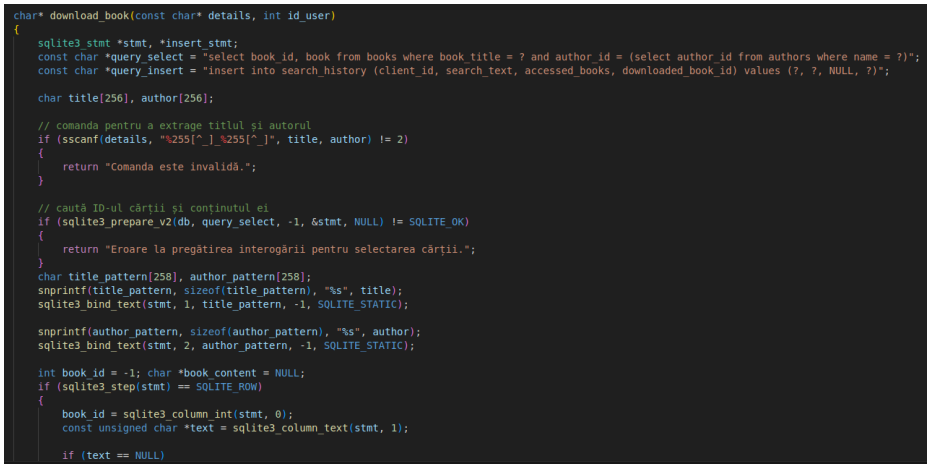
Fig. 12. Funcție de identificare a ID-ului unui client

```c
char* search_title(const char* title, int id_user)
{
    sqlite3_stmt *stmt;
    sqlite3_stmt *insert_stmt;
    const char *query = "select books.book_title, authors.name from books join authors on books.author_id = authors.author_id  where upper(books.book_title) like upper(?)";
    const char *insert_query = "insert into search_history (client_id, search_text, accessed_books, downloaded_book_id) values (?, ?, ?, NULL)";
    char *response = malloc(1024);
    char *accessed_books = malloc(1024);
    if (!response || !accessed_books)
    {
        fprintf(stderr, "Eroare la alocarea memoriei.\n");
        free(response);
        free(accessed_books);
        return NULL;
    }
    strcpy(response, "Cărțile cu titlul căutat sunt:\n");
    strcpy(accessed_books, "");

    if (sqlite3_prepare_v2(db, query, -1, &stmt, NULL) != SQLITE_OK)
    {
        fprintf(stderr, "Eroare la pregătirea interogării.\n");
        free(response);
        free(accessed_books);
        return NULL;
    }
    char search_pattern[256];
    snprintf(search_pattern, sizeof(search_pattern), "%%%s%%", title);
    sqlite3_bind_text(stmt, 1, search_pattern, -1, SQLITE_STATIC);

    while (sqlite3_step(stmt) == SQLITE_ROW)
    {
        const char *book_title = (const char *)sqlite3_column_text(stmt, 0);
        const char *author_name = (const char *)sqlite3_column_text(stmt, 1);
        char book_info[256];
        snprintf(book_info, sizeof(book_info), "%s, de %s\n", book_title, author_name);
        if (strlen(response) + strlen(book_info) + 1 > 1024)
        {
            response = realloc(response, strlen(response) + strlen(book_info) + 1);
            if (!response)
            {
                fprintf(stderr, "Eroare la realocarea memoriei.\n");
                sqlite3_finalize(stmt);
```

Fig. 13. Funcție de căutare a unei cărți după titlu

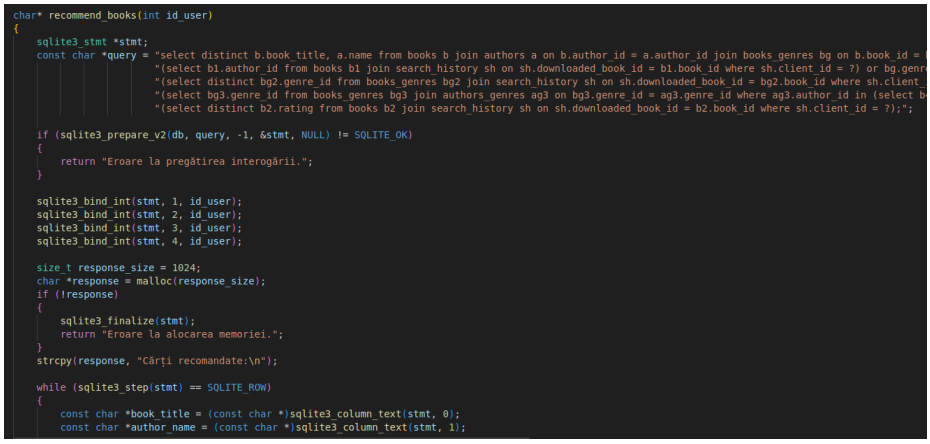
```c
char* rate_book(const char* details, int id_user)
{
    sqlite3_stmt *stmt, *insert_stmt;
    const char *query_select = "select book_id, rating from books where book_title = ? and author_id = (select author_id from authors where name = ?)";
    const char *query_update = "update books set rating = ? where book_id = ?";
    const char *query_insert = "insert into rating_history (client_id, book_id, rating) values (?, ?, ?)";

    char title[256], author[256];
    float new_rating, old_rating, avg_rating;

    // comanda pentru a extrage titlul, autorul și rating-ul
    if (sscanf(details, "%255[^_]_%255[^_]_%f", title, author, &new_rating) != 3)
    {
        return "Comanda este invalidă.";
    }

    // caută ID-ul cărții și rating-ul vechi
    if (sqlite3_prepare_v2(db, query_select, -1, &stmt, NULL) != SQLITE_OK)
    {
        return "Eroare la pregătirea interogării pentru selectarea cărții.";
    }
    char title_pattern[258], author_pattern[258];
    snprintf(title_pattern, sizeof(title_pattern), "%s", title);
    sqlite3_bind_text(stmt, 1, title_pattern, -1, SQLITE_STATIC);

    snprintf(author_pattern, sizeof(author_pattern), "%s", author);
    sqlite3_bind_text(stmt, 2, author_pattern, -1, SQLITE_STATIC);

    int book_id = -1;
    if (sqlite3_step(stmt) == SQLITE_ROW)
    {
        book_id = sqlite3_column_int(stmt, 0);
```

Fig. 14. Funcție de evaluare a unei cărți



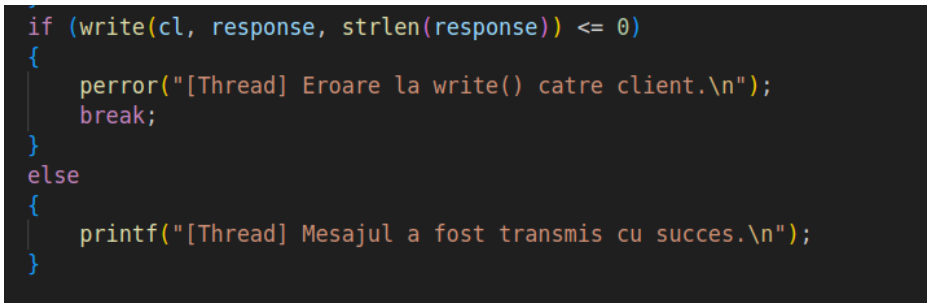
```c
char* download_book(const char* details, int id_user)
{
    sqlite3_stmt *stmt, *insert_stmt;
    const char *query_select = "select book_id, book from books where book_title = ? and author_id = (select author_id from authors where name = ?)";
    const char *query_insert = "insert into search_history (client_id, search_text, accessed_books, downloaded_book_id) values (?, ?, NULL, ?)";

    char title[256], author[256];

    // comanda pentru a extrage titlul și autorul
    if (sscanf(details, "%255[^_]_%255[^_]", title, author) != 2)
    {
        return "Comanda este invalidă.";
    }

    // caută ID-ul cărții și conținutul ei
    if (sqlite3_prepare_v2(db, query_select, -1, &stmt, NULL) != SQLITE_OK)
    {
        return "Eroare la pregătirea interogării pentru selectarea cărții.";
    }
    char title_pattern[258], author_pattern[258];
    snprintf(title_pattern, sizeof(title_pattern), "%s", title);
    sqlite3_bind_text(stmt, 1, title_pattern, -1, SQLITE_STATIC);

    snprintf(author_pattern, sizeof(author_pattern), "%s", author);
    sqlite3_bind_text(stmt, 2, author_pattern, -1, SQLITE_STATIC);

    int book_id = -1; char *book_content = NULL;
    if (sqlite3_step(stmt) == SQLITE_ROW)
    {
        book_id = sqlite3_column_int(stmt, 0);
        const unsigned char *text = sqlite3_column_text(stmt, 1);

        if (text == NULL)
```

Fig. 15. Funcție de descărcare a unei cărți



```c
char* recommend_books(int id_user)
{
    sqlite3_stmt *stmt;
    const char *query = "select distinct b.book_title, a.name from books b join authors a on b.author_id = a.author_id join books_genres bg on b.book_id =
                        "(select b1.author_id from books b1 join search_history sh on sh.downloaded_book_id = b1.book_id where sh.client_id = ?) or bg.genr
                        "(select distinct bg2.genre_id from books_genres bg2 join search_history sh on sh.downloaded_book_id = bg2.book_id where sh.client_
                        "(select bg3.genre_id from books_genres bg3 join authors_genres ag3 on bg3.genre_id = ag3.genre_id where ag3.author_id in (select b
                        "(select distinct b2.rating from books b2 join search_history sh on sh.downloaded_book_id = b2.book_id where sh.client_id = ?);";

    if (sqlite3_prepare_v2(db, query, -1, &stmt, NULL) != SQLITE_OK)
    {
        return "Eroare la pregătirea interogării.";
    }

    sqlite3_bind_int(stmt, 1, id_user);
    sqlite3_bind_int(stmt, 2, id_user);
    sqlite3_bind_int(stmt, 3, id_user);
    sqlite3_bind_int(stmt, 4, id_user);

    size_t response_size = 1024;
    char *response = malloc(response_size);
    if (!response)
    {
        sqlite3_finalize(stmt);
        return "Eroare la alocarea memoriei.";
    }
    strcpy(response, "Cărți recomandate:\n");

    while (sqlite3_step(stmt) == SQLITE_ROW)
    {
        const char *book_title = (const char *)sqlite3_column_text(stmt, 0);
        const char *author_name = (const char *)sqlite3_column_text(stmt, 1);
```

Fig. 16. Funcție pentru recomandare de cărți



```c
if (write(cl, response, strlen(response)) <= 0)
{
    perror("[Thread] Eroare la write() catre client.\n");
    break;
}
else
{
    printf("[Thread] Mesajul a fost transmis cu succes.\n");
}
```

Fig. 17. Trimiterea răspunsului clientului

4.2 Implementarea Clientului

```
  /* cream socketul */
  if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
  {
      perror("[client] Eroare la socket().\n");
      return errno;
  }
```

Fig. 18. Crearea socket-ului

```
/* umplem structura pentru realizarea conexiunii cu serverul */
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr(argv[1]);
server.sin_port = htons(port);
```

Fig. 19. Pregătirea structurii pentru conexiunea cu serverul

```
/* ne conectam la server */
if (connect(sd, (struct sockaddr *) &server, sizeof(struct sockaddr)) == -1)
{
    perror("[client] Eroare la connect().\n");
    return errno;
}
```

Fig. 20. Conectarea la server

```c
/* bucla pentru trimiterea mesajelor */
while (1)
{
    /* citirea mesajului de la utilizator */
    printf("[client] Introduceti un mesaj (sau 'quit' pentru a iesi): ");
    fflush(stdout);
    bzero(buffer, sizeof(buffer)); // resetam buffer-ul
    fgets(buffer, sizeof(buffer) - 1, stdin);
```

Fig. 21. Buclă infinită pentru citirea comenzilor date de utilizator

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

Fig. 22. Trimiterea mesajului la server

```c
// Daca utilizatorul introduce "quit", iesim din bucla
if (strcmp(buffer, "quit") == 0)
{
    printf("[client] Se inchide conexiunea.\n");
    break;
}
```

Fig. 23. Verificare quit

```c
void download_book(const char *buffer)
{
    const char *title_start = buffer + 10;
    const char *content_start = strchr(title_start, '\n');

    // calculăm lungimea titlului și alocăm memorie pentru numele fișierului
    size_t title_length = content_start - title_start;
    char *filename = malloc(title_length + 5);  // +5 pentru ".txt" și null-terminator

    if (!filename)
    {
        fprintf(stderr, "Eroare la alocarea memoriei pentru numele fișierului.\n");
        return;
    }

    strncpy(filename, title_start, title_length);
    filename[title_length] = '\0';
    strcat(filename, ".txt");

    // deschidem fișierul pentru scriere
    FILE *file = fopen(filename, "w");
    if (!file)
    {
        fprintf(stderr, "Eroare la crearea fișierului '%s'.\n", filename);
        free(filename);
        return;
    }

    fprintf(file, "%s", content_start + 1);
    fclose(file);

    printf("Carte salvată cu succes în fișierul '%s'.\n", filename);
    free(filename);
}
```

Fig. 24. Descărcarea unei cărți în memorie

```c
/* citirea răspunsului de la server */
bzero(buffer, sizeof(buffer));
if (read(sd, buffer, sizeof(buffer) - 1) < 0)
{
    perror("[client] Eroare la read() de la server.\n");
    return errno;
}

if(strncmp(buffer, "Download: ", 10)==0)
{
    // descărcăm cartea
    download_book(buffer);
}
else
{
    /* afișăm mesajul primit */
    printf("[client] Mesajul primit de la server: %s\n", buffer);
}
```

Fig. 25. Citirea și afișarea răspunsului de la server

5 Concluzii

Proiectul oferă acces rapid și eficient clienților la o librărie online, aceștia putând să caute, descarce, evalueze și solicite recomandări de cărți.

Librăria online ar putea fi dezvoltată prin adăugarea unor funcționalități noi, cum ar fi o "listă de dorințe" (wishlist) pentru utilizatori, sau posibilitatea de a adăuga comentarii și recenzii la cărți.

O posibilă îmbunătățire a soluției ar fi crearea unui model de tip rețea neuronală, care să învețe preferințele clienților, pentru a le oferi recomandări mai bune.

6 Referințe Bibliografice

1. https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm
2. https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php
3. https://sites.google.com/view/sbranisteanu/laboratorul-6?authuser=0
4. https://app.diagrams.net