



Università
degli Studi di
Messina

Università degli Studi di Messina

Dipartimento MIFT

LABORATORIO DI RETI E SISTEMI DISTRIBUITI, Progetto Multithreading

Autori: Luca Ciraoło MAT: 527480
Christian Bombara MAT: 529463

MESSINA, 16/06/2024

Indice

1	stato dell'arte	4
1.1	Introduzione	4
1.2	Obiettivi	4
2	Descrizione del Problema	5
2.1	Struttura del Progetto	5
3	Implementazione	6
3.1	Implementazione client	6
3.1.1	Linguaggi e tecnologie utilizzate	6
3.1.2	Connessione al server	6
3.1.3	Modalità di accesso e gestione delle password	6
3.1.4	Menù e scelte dell'utente	7
4	Dashboard e funzionalità	8
4.1	Visualizza tutti i libri disponibili	8
4.1.1	Visualizza le mie prenotazioni	8
4.1.2	Prenota un libro	10
4.1.3	Annulla prenotazione	10
4.1.4	Logout	11
4.2	Connessione al Server	12
5	Implementazione del Server	14
5.1	Linguaggi e Strumenti Utilizzati	14
5.2	Gestione Sincronizzazione e Concorrenze (Mutua Esclusione)	16
5.3	Gestione delle Richieste da Parte del Client	17
5.4	Esecuzione delle Query - Comunicazione con il DB	17
5.5	Altre funzioni essenziali	19
6	Database SQLite	20
6.1	Struttura del Database	20
6.1.1	Tabella <code>utente</code>	20
6.1.2	Tabella <code>libro</code>	21
6.1.3	Tabella <code>prenotazione</code>	22
6.2	Gestione delle Query	22

6.2.1	Inserimento di un Nuovo Utente	22
6.2.2	Inserimento di un Nuovo Libro	23
6.2.3	Aggiornamento della Disponibilità di un Libro	23
6.2.4	Inserimento di una Nuova Prenotazione	23
7	Risultati sperimentali	24
8	Conclusioni e Implementazioni Future	25
8.1	Conclusioni	25
8.2	Implementazioni Future	25

1: STATO DELL'ARTE

Questa relazione presenta lo sviluppo di un sistema client-server per la gestione delle prenotazioni di libri. Il progetto include una registrazione e autenticazione degli utenti, la visualizzazione dei libri disponibili, la prenotazione di libri e la cancellazione delle prenotazioni. L'integrazione con un database SQLite garantisce la persistenza dei dati. Vengono illustrati gli obiettivi, la struttura del progetto, i dettagli dell'implementazione, i risultati ottenuti e le possibili estensioni future.

1.1 Introduzione

Nell'era digitale, la gestione efficiente delle risorse bibliotecarie è cruciale per garantire un accesso facile e rapido ai libri di testo. Questo progetto è stato sviluppato per creare un sistema che permette agli utenti di prenotare e gestire le proprie prenotazioni di libri universitari in modo efficiente. Il sistema utilizza un'architettura client-server e un database SQLite per memorizzare i dati degli utenti e delle prenotazioni.

1.2 Obiettivi

Gli obiettivi principali del progetto sono:

1. Implementare la registrazione e l'autenticazione degli utenti.
2. Realizzare la gestione dei libri disponibili per la prenotazione.
3. Consentire agli utenti di prenotare e cancellare le prenotazioni.
4. Utilizzare un database SQLite per garantire la persistenza dei dati.
5. Garantire l'utilizzo del software da parte di più utenti in contemporanea.

2: DESCRIZIONE DEL PROBLEMA

Realizzare in C un sistema client-server multithread per la prenotazione da remoto dei test universitari usando il database Sqlite.

2.1 Struttura del Progetto

Il progetto è suddiviso in due componenti principali:

- **Server:** Gestisce le richieste dei client, interagisce con il database SQLite e implementa la logica applicativa.
- **Client:** Fornisce un'interfaccia utente per registrare, autenticare e gestire le prenotazioni di libri.

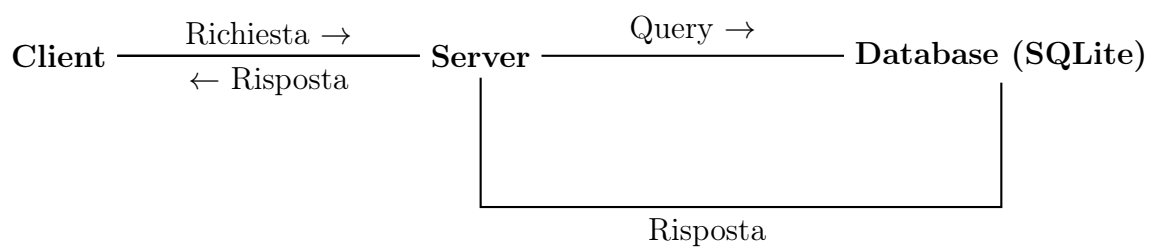


Figura 1: Flusso delle informazioni nel sistema client-server con database SQLite

3: IMPLEMENTAZIONE

Il sistema è stato sviluppato utilizzando il linguaggio di programmazione C per il server e il client, con l'uso del database SQLite per la persistenza dei dati. La sezione dettagliata include la struttura del database, le principali funzionalità implementate, il codice significativo e le procedure di test effettuate.

3.1 Implementazione client

Il client è implementato utilizzando il linguaggio di programmazione C. È progettato per comunicare con un server remoto tramite socket TCP/IP.

3.1.1 *Linguaggi e tecnologie utilizzate*

Il client è sviluppato in C, utilizzando le seguenti librerie standard:

- `stdio.h`, `stdlib.h`, `string.h` per operazioni di input/output e gestione delle stringhe.
- `unistd.h` per la gestione dei descrittori di file e operazioni di sistema.
- `arpa/inet.h` per la conversione di indirizzi IP.
- `termios.h` per la gestione dell'input della password senza eco.

3.1.2 *Connessione al server*

Il client si connette al server utilizzando socket TCP/IP. La connessione è stabilita tramite la funzione `socket()`, seguita da `connect()` per collegarsi all'indirizzo IP e alla porta specificati ("127.0.0.1").

3.1.3 *Modalità di accesso e gestione delle password*

Per garantire la sicurezza delle password, il client utilizza la funzione `get_password()` che disabilita l'eco dell'input durante l'immissione della password. Questo è realizzato modificando le impostazioni del terminale con `tcgetattr()` e `tcsetattr()` rese disponibili dalla libreria `termios.h`.

3.1.4 *Menù e scelte dell'utente*

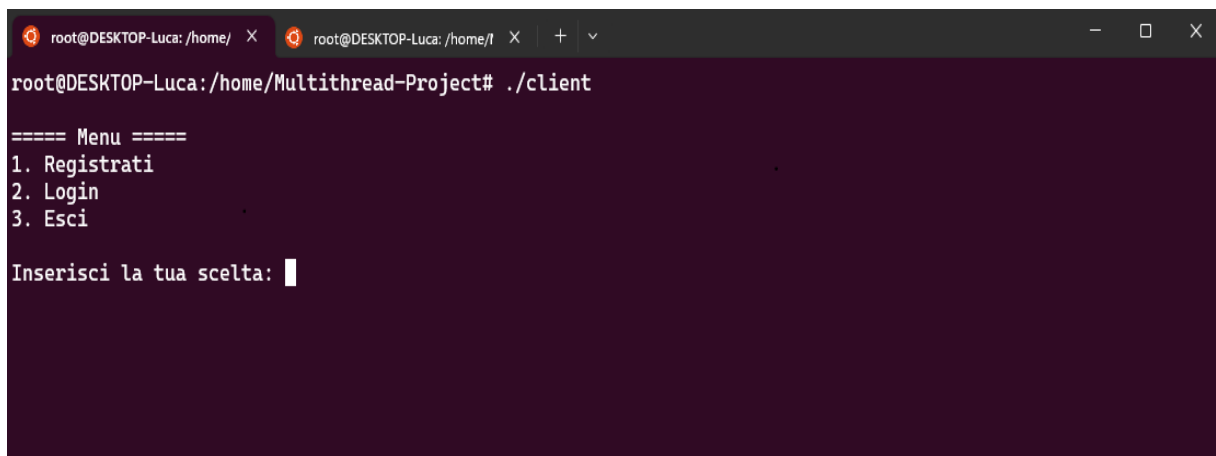
Il client offre un menu con le seguenti opzioni prima dell'accesso alla Dashboard:

1. Registrati
2. Login
3. Esci

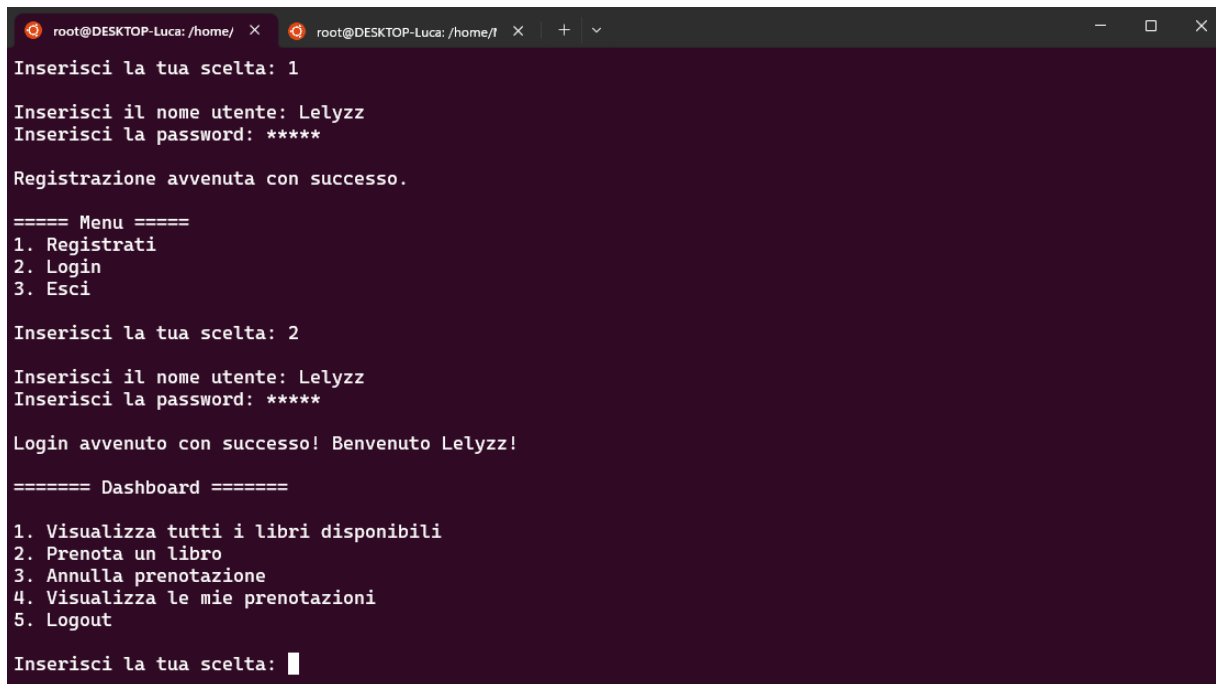
Le opzioni successive all'accesso , dove effettivamente l'utente può utilizzare il software, sono:

1. Visualizza tutti i libri disponibili
2. Prenota un libro
3. Annulla prenotazione
4. Visualizza le mie prenotazioni
5. Logout

Ogni opzione è gestita da una funzione specifica che invia le richieste appropriate al server e gestisce le risposte ricevute.

A screenshot of a terminal window with a dark background. The window title bar shows two tabs: 'root@DESKTOP-Luca: /home/' and 'root@DESKTOP-Luca: /home/l'. The terminal content shows the command './client' being executed in the directory '/home/Multithread-Project'. The output displays a menu with three options: '1. Registrati', '2. Login', and '3. Esci'. Below the menu, there is a prompt 'Inserisci la tua scelta: ' followed by a cursor. The terminal window has standard Linux window controls (minimize, maximize, close) in the top right corner.

```
root@DESKTOP-Luca: /home/ X root@DESKTOP-Luca: /home/l X + v
root@DESKTOP-Luca: /home/Multithread-Project# ./client
===== Menu =====
1. Registrati
2. Login
3. Esci
Inserisci la tua scelta: |
```



```
root@DESKTOP-Luca: /home/ X root@DESKTOP-Luca: /home/l X + v
Inserisci la tua scelta: 1
Inserisci il nome utente: Lelyzz
Inserisci la password: *****
Registrazione avvenuta con successo.
===== Menu =====
1. Registrati
2. Login
3. Esci
Inserisci la tua scelta: 2
Inserisci il nome utente: Lelyzz
Inserisci la password: *****
Login avvenuto con successo! Benvenuto Lelyzz!
===== Dashboard =====
1. Visualizza tutti i libri disponibili
2. Prenota un libro
3. Annulla prenotazione
4. Visualizza le mie prenotazioni
5. Logout
Inserisci la tua scelta: █
```

4: DASHBOARD E FUNZIONALITÀ

In questa sezione andremo a mostrare le funzionalità della dashboard dopo che l'utente si è loggato con successo, la dashboard è quella parte del software che permette l'interazione utente - database.

4.1 Visualizza tutti i libri disponibili

Questa sezione permette all'utente di visualizzare tutti i libri disponibili nel sistema con i relativi campi : ID, Titolo, Autore, Categoria, Anno-Pubblicazione, Disponibilità.

4.1.1 *Visualizza le mie prenotazioni*

Questa sezione permette all'utente di visualizzare le prenotazioni attualmente registrate a suo nome, quest'ultime sono fornite anche di un comodo ID prenotazione.

Di seguito i due codici:


```

1 void handle_view_books(int socket) {
2     send(socket, "VIEW_BOOKS\n", strlen("VIEW_BOOKS\n"),
3         ↪ 0);
4     char buffer[2048];
5     int total_bytes_received = 0, bytes_received = 0;
6     while ((bytes_received = recv(socket, buffer, sizeof(
7         ↪ buffer) - 1, 0)) > 0) {
8         buffer[bytes_received] = '\0';
9         printf("%s", buffer);
10        total_bytes_received += bytes_received;
11        if (bytes_received < sizeof(buffer) - 1) {
12            break;}}
13    if (total_bytes_received == 0) {
14        printf("\nErrore_nella_ricezione_dei_dati.\n");}}

```

Listing 1: Visualizzazione dei libri disponibili

```

1 void handle_view_reservations(int socket) {
2     send(socket, "VIEW_MY_RESERVATIONS\n", strlen("
3         ↪ VIEW_MY_RESERVATIONS\n"), 0);
4     char buffer[2048];
5     int total_bytes_received = 0, bytes_received;
6     while ((bytes_received = recv(socket, buffer, sizeof(
7         ↪ buffer) - 1, 0)) > 0) {
8         buffer[bytes_received] = '\0';
9         printf("%s", buffer);
10        total_bytes_received += bytes_received;
11        if (bytes_received < sizeof(buffer) - 1) {
12            break;
13        }}
14    if (total_bytes_received == 0) {
15        printf("\nNessuna_prenotazione_trovata.\n");}}

```

Listing 2: Visualizzazione delle prenotazioni dell'utente

4.1.2 Prenota un libro

Questa sezione consente all'utente di prenotare un libro specificando l'ID del libro desiderato.

4.1.3 Annulla prenotazione

Questa sezione consente all'utente di annullare una prenotazione specificando l'ID della prenotazione.

Entrambi le opzioni andranno ad influire sulla disponibilità dei libri andando a decrementare/incrementare la quantità nel Database... di seguito i codici.

```
1 void handle_reserve_book(int socket) {
2     char book_id[100];
3     printf("\nInserisci l'ID del libro da prenotare: ");
4     fgets(book_id, sizeof(book_id), stdin);
5     book_id[strcspn(book_id, "\n")] = '\0';
6     char message[256];
7     snprintf(message, sizeof(message), "RESERVE\n%s\n",
8         ↪ book_id);
9     send(socket, message, strlen(message), 0);
10    char response[256];
11    int valread = recv(socket, response, sizeof(response)
12        ↪ - 1, 0);
13    response[valread] = '\0';
14    if (strcmp(response, "RESERVATION_SUCCESS") == 0) {
15        printf("\nPrenotazione avvenuta con successo.\n")
16        ↪ ;
17    } else if (strcmp(response, "RESERVATION_FAILURE") ==
18        ↪ 0) {
19        printf("\nPrenotazione fallita.\n");
20    }
21 }
```

Listing 3: Prenotazione di un libro

```

1 void handle_cancel_reservation(int socket) {
2     char reservation_id[100];
3     printf("\nInserisci l'ID della prenotazione da
        ↳ annullare: ");
4     fgets(reservation_id, sizeof(reservation_id), stdin);
5     reservation_id[strcspn(reservation_id, "\n")] = '\0';
6     char message[256];
7     snprintf(message, sizeof(message), "
        ↳ CANCEL_RESERVATION\n%s\n", reservation_id);
8     send(socket, message, strlen(message), 0);
9     char response[256];
10    int valread = recv(socket, response, sizeof(response)
        ↳ - 1, 0);
11    response[valread] = '\0';
12    if (strcmp(response, "CANCEL_RESERVATION_SUCCESS") ==
        ↳ 0) {
13        printf("\nPrenotazione annullata con successo.\n"
        ↳ );
14    } else if (strcmp(response, "
        ↳ CANCEL_RESERVATION_FAILURE") == 0) {
15        printf("\nAnnullamento della prenotazione fallito
        ↳ .\n");
16    }
17 }

```

Listing 4: Annullamento di una prenotazione

4.1.4 Logout

Questa sezione consente all'utente di disconnettersi dal sistema, ciò non causerà lo spegnimento del client ma darà la possibilità all'utente di ritornare in fase di login se lo desidera.

```

1 void handle_logout(int socket) {
2     send(socket, "LOGOUT\n", strlen("LOGOUT\n"), 0);
3     char response[256];
4     int valread = recv(socket, response, sizeof(response)
5         ↪ - 1, 0);
6     response[valread] = '\0';
7     if (strcmp(response, "LOGOUT_SUCCESS") == 0) {
8         printf("\nDisconnessione avvenuta con successo.\n
9         ↪ ");
10    } else {
11        printf("\nErrore durante la disconnessione.\n");
12    }
13 }

```

Listing 5: Gestione del logout

4.2 Connessione al Server

Per stabilire la connessione tra il client e il server nel sistema di prenotazione dei test universitari il client crea un socket di tipo `SOCK_STREAM` per una connessione TCP utilizzando `socket(AF_INET, SOCK_STREAM, 0)`. Successivamente, imposta l'indirizzo IP del server e la porta a cui connettersi nella struttura `serv_addr`. Dopo aver convertito l'indirizzo IP in formato binario con `inet_pton`, il client utilizza `connect` per connettersi al server. Una volta stabilita la connessione con successo, il client può procedere con l'invio e la ricezione di dati dal server, come nel caso delle richieste di registrazione, login, e gestione delle prenotazioni descritte precedentemente.

Il codice seguente mostra come viene implementata questa connessione nel client:

```

1 #define PORT 8080
2 #define SERVER_IP "127.0.0.1" // Indirizzo IP del server
3
4 int main() {
5     int sock = 0;
6     struct sockaddr_in serv_addr;
7
8     // Creazione del socket
9     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
10         perror("Errore nella creazione del socket");
11         return -1;
12     }
13     // Impostazione degli indirizzi e della porta del
14     ➔ server
15     serv_addr.sin_family = AF_INET;
16     serv_addr.sin_port = htons(PORT);
17
18     // Conversione dell'indirizzo IP da stringa a formato
19     ➔ binario
20     if (inet_pton(AF_INET, SERVER_IP, &serv_addr.sin_addr
21     ➔ ) <= 0) {
22         perror("Indirizzo non valido / non supportato");
23         return -1;}
24
25     if (connect(sock, (struct sockaddr *)&serv_addr,
26     ➔ sizeof(serv_addr)) < 0) {
27         perror("Connessione fallita");
28         return -1;}
29
30     printf("Connessione al server riuscita!\n");
31     close(sock); // Chiudo il socket alla fine della
32     ➔ connessione
33     return 0;}

```

Listing 6: Connessione al server in C

5: IMPLEMENTAZIONE DEL SERVER

5.1 Linguaggi e Strumenti Utilizzati

Il server è stato implementato utilizzando il linguaggio di programmazione C e una combinazione di librerie per gestire le varie funzionalità richieste. Le librerie principali utilizzate sono:

- **pthread.h**: Libreria per la programmazione multithreading in C. Fornisce le strutture e le funzioni per la gestione dei thread. È utilizzata per gestire più client contemporaneamente tramite thread separati e per garantire la sincronizzazione tra di essi utilizzando mutex.
- **sqlite3.h**: Libreria per l'accesso al database SQLite in C. Fornisce le funzioni per eseguire query SQL e manipolare il database SQLite. È utilizzata per la gestione dei dati dei libri, degli utenti e delle prenotazioni nel server.
- **unistd.h**: Fornisce l'accesso alle funzioni di sistema standard, inclusi i meccanismi per la gestione dei processi e per la comunicazione con il sistema operativo. È utilizzata per le operazioni di sistema di base, come la chiusura dei socket e l'esecuzione di processi figlio.
- **arpa/inet.h**: Fornisce le funzioni e le strutture dati necessarie per la gestione degli indirizzi IP in formato binario e testo, oltre a funzioni per la conversione degli indirizzi tra rappresentazioni diverse. È utilizzata per la gestione delle connessioni di rete, inclusa l'implementazione del socket TCP/IP per la comunicazione client-server.

Queste librerie sono state selezionate per le loro capacità di fornire le funzionalità necessarie per il server, inclusa la gestione della comunicazione di rete, l'accesso al database, la manipolazione dei dati e la gestione dei thread per garantire la sicurezza e la concorrenza delle operazioni. L'utilizzo combinato di queste librerie ha permesso di realizzare un server robusto e scalabile per il sistema di prenotazione dei libri.

```

1  int main() {
2      init_db();
3
4      // Creazione del socket server
5      int server_fd, new_socket;
6      struct sockaddr_in address;
7      int addrlen = sizeof(address);
8      pthread_t thread_id[MAX_CLIENTS];
9      int client_count = 0;
10
11     if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) ==
12         ↪ 0) {
13         perror("socket_❏failed");
14         exit(EXIT_FAILURE);}
15
16     address.sin_family = AF_INET;
17     address.sin_addr.s_addr = INADDR_ANY;
18     address.sin_port = htons(PORT);
19
20     if (bind(server_fd, (struct sockaddr *)&address,
21         ↪ sizeof(address)) < 0) {
22         perror("bind_❏failed");
23         exit(EXIT_FAILURE);}
24
25     if (listen(server_fd, 3) < 0) {
26         perror("listen");
27         exit(EXIT_FAILURE);}
28
29     printf("Server_❏listening_❏on_❏port_❏%d...\n", PORT);
30
31     for (int i = 0; i < client_count; i++) {
32         pthread_join(thread_id[i], NULL);}
33     close_db();
34     return 0;}

```

Listing 7: Creazione del Server

5.2 Gestione Sincronizzazione e Concorrenze (Mutua Esclusione)

La gestione della mutua esclusione è cruciale per garantire l'integrità dei dati nel database condiviso tra i thread. È stato utilizzato un mutex ('db_mutex') per proteggere l'accesso al database SQLite da parte dei thread concorrenti. Prima di eseguire operazioni che modificano lo stato del database, come l'inserimento di un nuovo utente o l'aggiornamento della disponibilità di un libro, viene acquisito il lock del mutex per garantire che solo un thread alla volta possa eseguire tali operazioni critiche. Dopo aver completato l'operazione, il mutex viene rilasciato per consentire agli altri thread di accedere al database.

```
1 pthread_mutex_t db_mutex;
2 void init_db() {
3     pthread_mutex_init(&db_mutex, NULL);}
4
5 void close_db() {
6     sqlite3_close(db);
7     pthread_mutex_destroy(&db_mutex);}
8
9 void register_user(int client_socket, char *username,
10 ↪ char *password) {
11     pthread_mutex_lock(&db_mutex);
12     char *sql = sqlite3_mprintf("INSERT INTO utenti (
13 ↪ username, password) VALUES ('%s', '%s');",
14 ↪ username, password);
15     int rc = sqlite3_exec(db, sql, NULL, 0, NULL);
16     if (rc != SQLITE_OK) {
17         fprintf(stderr, "Errore durante l'inserimento
18 ↪ dell'utente: %s\n", sqlite3_errmsg(db));}
19     else {
20         printf("Utente %s registrato con successo.\n",
21 ↪ username);}
22     sqlite3_free(sql);
23     pthread_mutex_unlock(&db_mutex);}
```

Listing 8: Gestione della Mutua Esclusione

5.3 Gestione delle Richieste da Parte del Client

Le richieste inviate dai client al server sono gestite nel thread ‘handle_client’, che riceve i comandi e li processa in base al loro tipo. Prima di tutto, il server accetta una connessione dal client, creando un nuovo socket dedicato alla comunicazione con quel client specifico. Il server quindi avvia un thread separato (‘handle_client’) per gestire tutte le comunicazioni con quel client.

5.4 Esecuzione delle Query - Comunicazione con il DB

Le query SQL sono eseguite nel server per recuperare e manipolare i dati nel database SQLite. Ogni operazione che richiede l’accesso al database viene protetta da un lock del mutex per garantire che l’accesso sia thread-safe. Di seguito è riportato un esempio di come vengono eseguite le query per visualizzare i libri disponibili:

```

1 void view_books(int client_socket) {
2     pthread_mutex_lock(&db_mutex);
3
4     const char *sql = "SELECT id, titolo, autore,
        ↳ categoria, anno_pubblicazione, disponibilita
        ↳ FROM libri WHERE disponibilita > 0;";
5     sqlite3_stmt *res;
6     int rc = sqlite3_prepare_v2(db, sql, -1, &res, 0);
7     if (rc != SQLITE_OK) {
8         fprintf(stderr, "Errore durante la preparazione
        ↳ della query: %s\n", sqlite3_errmsg(db));
9     } else {
10         // Esecuzione della query e gestione dei
        ↳ risultati
11         while (sqlite3_step(res) == SQLITE_ROW) {
12             int id = sqlite3_column_int(res, 0);
13             const char *titolo = (const char *)
        ↳ sqlite3_column_text(res, 1);
14             const char *autore = (const char *)
        ↳ sqlite3_column_text(res, 2);
15             const char *categoria = (const char *)
        ↳ sqlite3_column_text(res, 3);
16             int anno = sqlite3_column_int(res, 4);
17             int disponibilita = sqlite3_column_int(res,
        ↳ 5);
18
19             // Invia i dati al client o li elabora nel
        ↳ server
20             // ... }
21             sqlite3_finalize(res);

```

Listing 9: Esecuzione delle Query

5.5 Altre funzioni essenziali

Ovviamente queste sono solo alcune funzioni che sono utili al corretto funzionamento del software, tra le più importanti per la manipolazione dati abbiamo:

1. **register-user**: gestisce lato server la registrazione dell'utente.
2. **login-user**: gestisce lato server il login dell'utente
3. **view-books**: gestisce lato server la visualizzazione del listato testi.
4. **send-response**: gestisce lato server l'invio di risposte al client.
5. **handle-reserve-book**: gestisce lato server la prenotazione dei testi universitari.
6. **view-my-reservations**: gestisce lato server il listato dei testi universitari prenotati dall'utente.
7. **cancel-reservation**: gestisce lato server la cancellazione della prenotazione.
8. **handle-client**: gestisce lato server la comunicazione con il client.

6: DATABASE SQLITE

Il server utilizza SQLite come sistema di gestione del database per memorizzare e gestire le informazioni relative agli utenti, ai libri e alle prenotazioni. SQLite è una libreria software che fornisce un motore di database SQL leggero, indipendente e senza necessità di un server separato.

6.1 Struttura del Database

Il database è composto da tre tabelle principali: **utente**, **libro**, e **prenotazione**. Di seguito viene descritta la struttura di ciascuna tabella e il significato dei suoi campi.

6.1.1 Tabella *utente*

La tabella **utente** contiene le informazioni degli utenti registrati nel sistema.

```
1 CREATE TABLE IF NOT EXISTS utente (  
2     matricola INTEGER PRIMARY KEY,  
3     nome TEXT NOT NULL,  
4     username TEXT UNIQUE NOT NULL,  
5     password TEXT NOT NULL  
6 );
```

I campi della tabella **utente** sono:

- **matricola**: Chiave primaria univoca per identificare ogni utente.
- **nome**: Nome completo dell'utente.
- **username**: Nome utente univoco utilizzato per il login.
- **password**: Password associata all'utente per l'autenticazione.

6.1.2 Tabella libro

La tabella `libro` contiene le informazioni relative ai libri disponibili per la prenotazione.

```
1 CREATE TABLE IF NOT EXISTS libro (  
2     id INTEGER PRIMARY KEY AUTOINCREMENT,  
3     titolo TEXT NOT NULL,  
4     autore TEXT NOT NULL,  
5     categoria TEXT NOT NULL,  
6     anno_pubblicazione INTEGER,  
7     disponibilita INTEGER NOT NULL  
8 );
```

I campi della tabella `libro` sono:

- `id`: Chiave primaria univoca per identificare ogni libro.
- `titolo`: Titolo del libro.
- `autore`: Autore del libro.
- `categoria`: Categoria o genere del libro.
- `anno-pubblicazione`: Anno di pubblicazione del libro.
- `disponibilita`: Numero di copie disponibili per la prenotazione.

6.1.3 Tabella prenotazione

La tabella `prenotazione` tiene traccia delle prenotazioni effettuate dagli utenti.

```
1 CREATE TABLE IF NOT EXISTS prenotazione (  
2     id INTEGER PRIMARY KEY AUTOINCREMENT,  
3     matricola INTEGER,  
4     id_libro INTEGER,  
5     FOREIGN KEY (matricola) REFERENCES utente(matricola),  
6     FOREIGN KEY (id_libro) REFERENCES libro(id)  
7 );
```

I campi della tabella `prenotazione` sono:

- `id`: Chiave primaria univoca per identificare ogni prenotazione.
- `matricola`: Matricola dell'utente che ha effettuato la prenotazione. Questo campo è una chiave esterna che fa riferimento alla chiave primaria della tabella `utente`.
- `id-libro`: ID del libro prenotato. Questo campo è una chiave esterna che fa riferimento alla chiave primaria della tabella `libro`.

6.2 Gestione delle Query

Le operazioni principali sul database includono l'inserimento di nuovi utenti, la registrazione di nuovi libri, l'aggiornamento della disponibilità dei libri, e la gestione delle prenotazioni. Di seguito sono illustrati alcuni esempi di query SQL utilizzate nel sistema:

6.2.1 Inserimento di un Nuovo Utente

```
1 INSERT INTO utente (matricola, nome, username, password)  
2 VALUES (?, ?, ?, ?);
```

6.2.2 Inserimento di un Nuovo Libro

```
1 INSERT INTO libro (titolo, autore, categoria,  
    ↪ anno_pubblicazione, disponibilita)  
2 VALUES (?, ?, ?, ?, ?);
```

6.2.3 Aggiornamento della Disponibilità di un Libro

```
1 UPDATE libro  
2 SET disponibilita = disponibilita - 1  
3 WHERE id = ? AND disponibilita > 0;
```

6.2.4 Inserimento di una Nuova Prenotazione

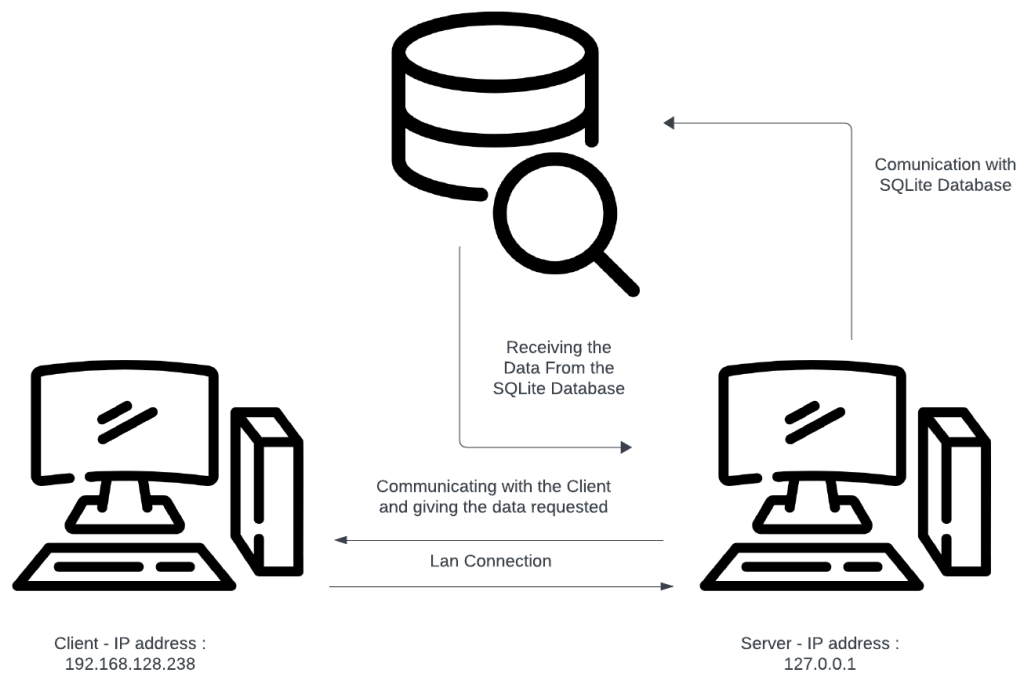
```
1 INSERT INTO prenotazione (matricola, id_libro)  
2 VALUES (?, ?);
```

Queste query vengono eseguite dal server per mantenere aggiornato lo stato del database in risposta alle richieste dei client.

7: RISULTATI SPERIMENTALI

I risultati ottenuti dalla realizzazione del progetto e dai test condotti confermano pienamente l'efficacia della soluzione proposta per il problema iniziale, ovvero garantire la prenotazione simultanea di test universitari a più utenti tramite un approccio multithreading. Al fine di verificare la robustezza del sistema, sono stati eseguiti numerosi test in cui diversi utenti accedevano contemporaneamente al sistema, effettuando operazioni di prenotazione e rimozione di test dal database.

L'implementazione ha dimostrato che, grazie all'utilizzo di meccanismi di mutua esclusione (mutex), ogni thread può accedere in modo sicuro alle sezioni critiche in cui vengono eseguite le modifiche sul database. Questo approccio ha assicurato l'integrità e la coerenza dei dati, evitando condizioni di corsa (race conditions) e garantendo che le operazioni di prenotazione e aggiornamento della disponibilità dei libri fossero gestite correttamente anche sotto carichi di accesso concorrenti.



8: CONCLUSIONI E IMPLEMENTAZIONI FUTURE

8.1 Conclusioni

Il progetto sviluppato ha dimostrato la validità dell'approccio multithreading per la gestione simultanea delle prenotazioni di test universitari. La combinazione di un'architettura server robusta e l'uso di meccanismi di mutua esclusione (mutex) ha permesso di garantire che le operazioni critiche sul database siano state eseguite in modo sicuro e senza conflitti. I test eseguiti hanno confermato che il sistema è in grado di gestire efficacemente accessi concorrenti, mantenendo l'integrità e la coerenza dei dati. Questo risultato è di fondamentale importanza per applicazioni reali, dove la gestione simultanea degli accessi è cruciale per garantire un servizio affidabile e continuo.

8.2 Implementazioni Future

Nonostante i risultati positivi ottenuti, ci sono diverse aree in cui il progetto può essere ulteriormente migliorato e ampliato. Di seguito vengono illustrate alcune delle possibili implementazioni future:

- **Ottimizzazione delle Prestazioni:** Sebbene il sistema attuale gestisca bene la concorrenza, ulteriori ottimizzazioni potrebbero migliorare le prestazioni sotto carichi molto elevati. Tecniche come il connection pooling per il database e l'uso di algoritmi di scheduling più avanzati per i thread potrebbero ridurre i tempi di latenza e aumentare l'efficienza.
- **Scalabilità Orizzontale:** Implementare un'architettura di server distribuito potrebbe aumentare la scalabilità del sistema. Utilizzando tecnologie come i bilanciatori di carico (load balancers) e i database distribuiti, il sistema potrebbe gestire un numero molto maggiore di richieste simultanee.
- **Miglioramento della Sicurezza:** L'integrazione di protocolli di sicurezza avanzati, come SSL/TLS per la crittografia delle comunicazioni e meccanismi di autenticazione più robusti, potrebbe proteggere meglio i dati degli utenti e garantire la riservatezza delle transazioni.
- **Implementazione di un'interfaccia:** L'implementazione di un'interfaccia potrebbe portare ad un utilizzo facilitato e più user-friendly.

In conclusione, il progetto ha posto solide basi per un sistema di gestione delle prenotazioni efficace e affidabile. Le possibili implementazioni future rappresentano un naturale passo successivo per migliorare ulteriormente il sistema e adattarlo alle crescenti esigenze degli utenti e delle istituzioni accademiche.