Università degli Studi di Messina

Progetto MultiThreading LabReSiD23

Ciraolo Luca MAT: 527480

Christian Bombara MAT: 529463

Indice

Introduzione al Progetto

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.

Obiettivi del Progetto

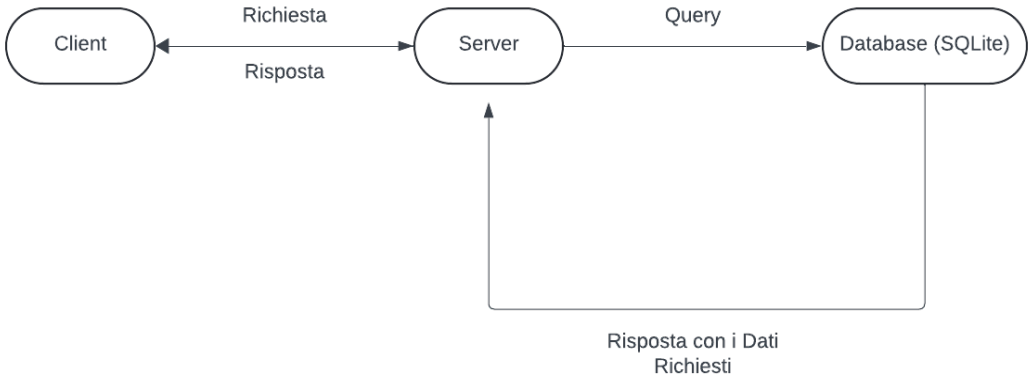
1. Implementare la registrazione e l'autenticazione degli utenti.
2. Realizzare la gestione dei libri disponibili per la prenotazione.
3. Consentire agli utenti di prenotare e cancellare le prenotazioni.
4. Utilizzare un database SQLite per garantire la persistenza dei dati.
5. Garantire l'utilizzo del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

Descrizione del Problema

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

Struttura del Progetto

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



Implementazione Lato-Client

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

Tecnologie Utilizzate

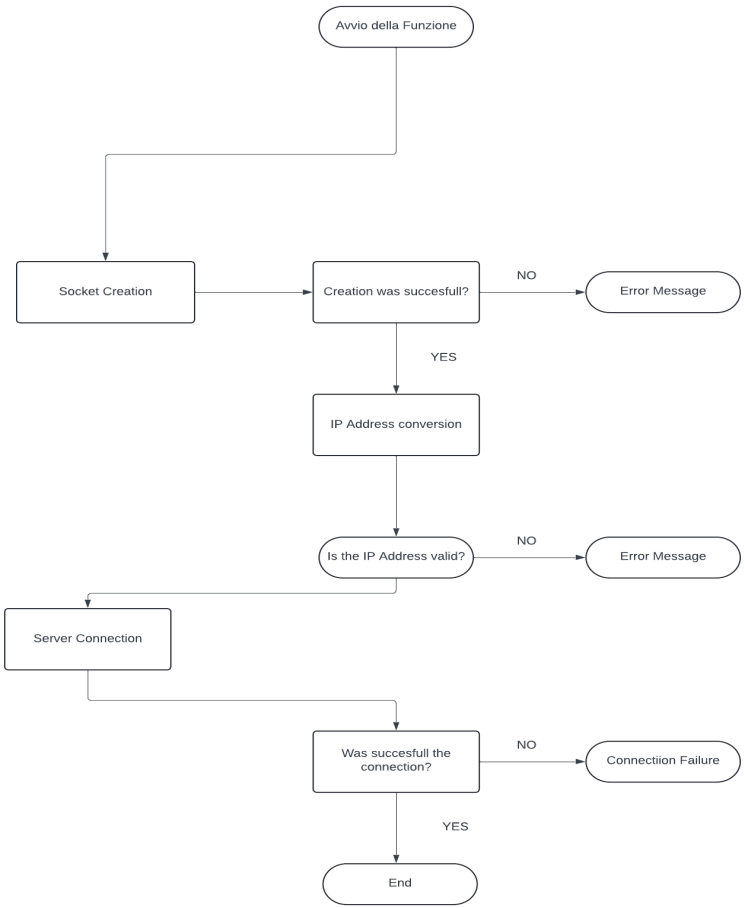
Il client è sviluppato in C, utilizzando le seguenti librerie standard:

- `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 andando a simulare il meccanismo utilizzato da Linux.

Connessione al Server

Per stabilire la connessione tra il client e il server nel sistema di prenotazione dei testi 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.

Connesione al server



Modalità di accesso e gestione della 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`.

Menù interattivo per l'utente

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

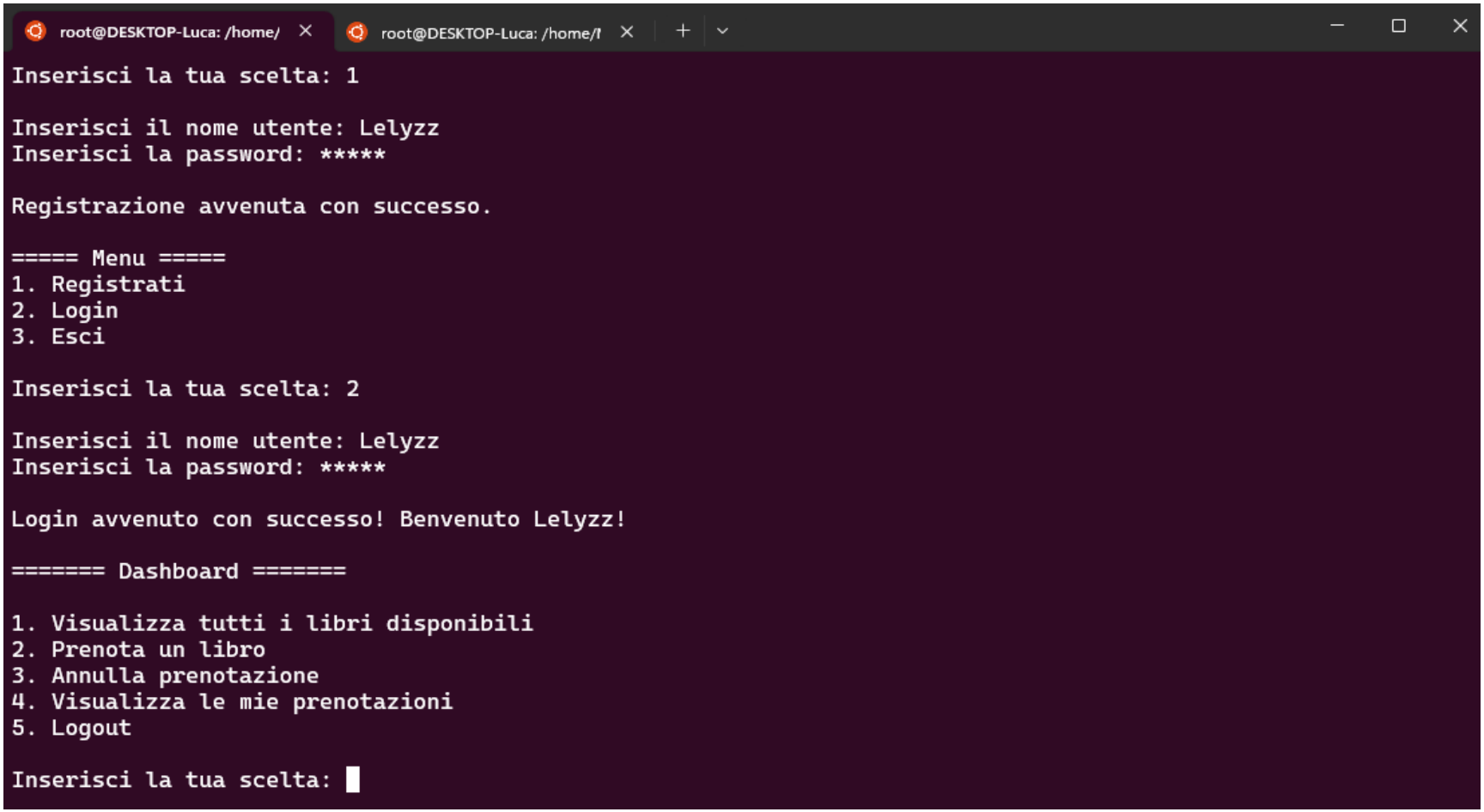
1. Registrati
2. Login
3. Esci

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.

Menù interattivo per l'utente

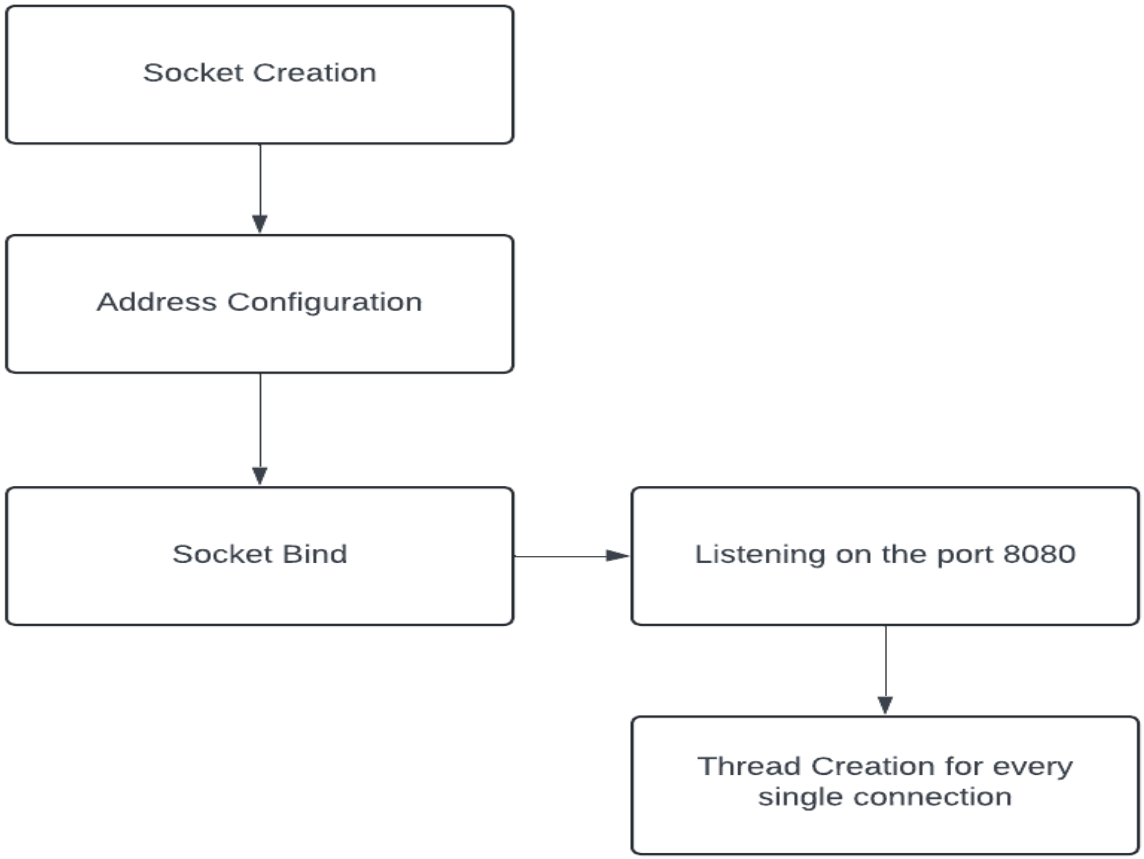


Implementazione Lato-Server

Librerie utilizzate:

- `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.

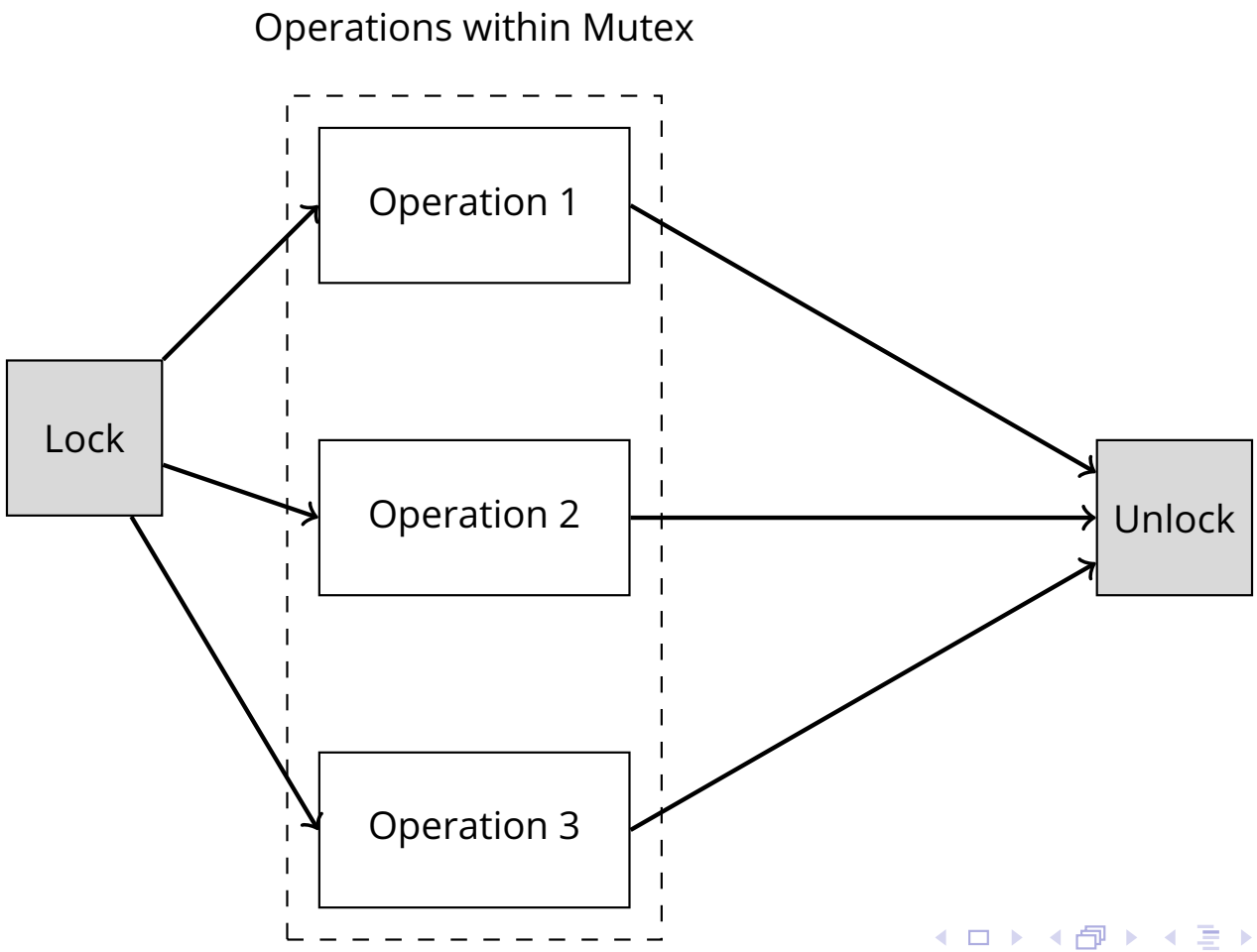
Implementazione Lato-Server



Sincronizzazione e mutua esclusione: Utilizzo del mutex

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.

Sincronizzazione e mutua esclusione: Utilizzo del mutex



Query SQL

Le query SQL sono eseguite nel server per recuperare e manipolare i dati nel database SQLite. Come evince dal grafico sopra presentato ogni operazione che richiede l'accesso al database viene protetta da un lock del mutex per garantire che l'accesso sia thread-safe.

All'interno del progetto vengono utilizzate diverse query tra cui:

1. Query di Inserimento: per andare a inserire gli utenti registrati all'interno del DB
2. Query di Selezione : per andare a selezionare i libri e gli utenti all'interno del DB
3. Query di Rimozione : per andare a rimuovere la quantità una volta fatta la prenotazione
4. Query di Aggiornamento: per andare a decrementare/ incrementare la disponibilità dei libri in base all'operazione di prenotazione / annullamento.

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.

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.

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

Utente	
Matricola Primary Key	integer
Nome	string
Username	string
Password	string

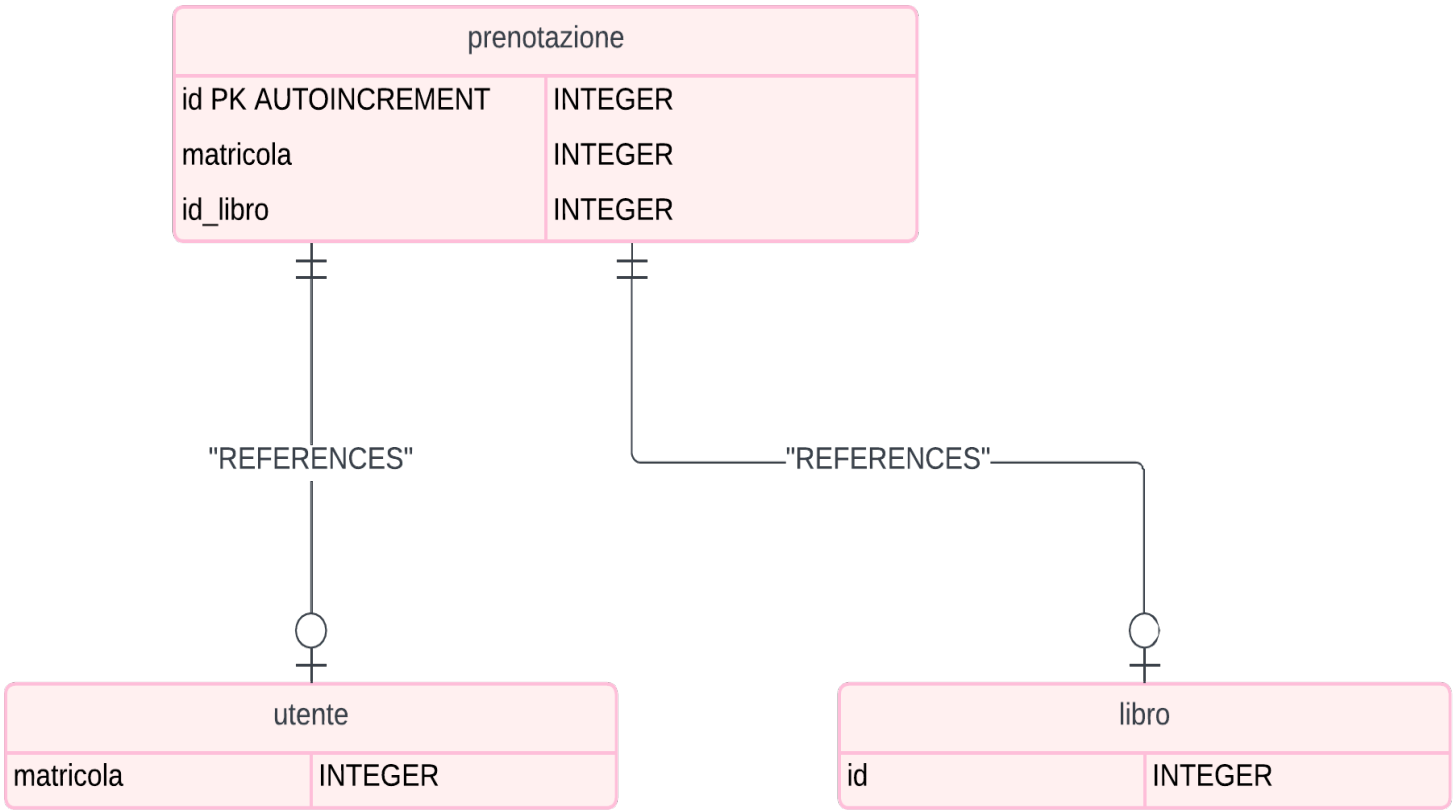
Struttura del Database

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

libro	
id PK "AUTOINCREMENT"	int
titolo "NOT NULL"	string
autore "NOT NULL"	string
categoria "NOT NULL"	string
anno_pubblicazione	int
disponibilita "NOT NULL"	int

Struttura del Database

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

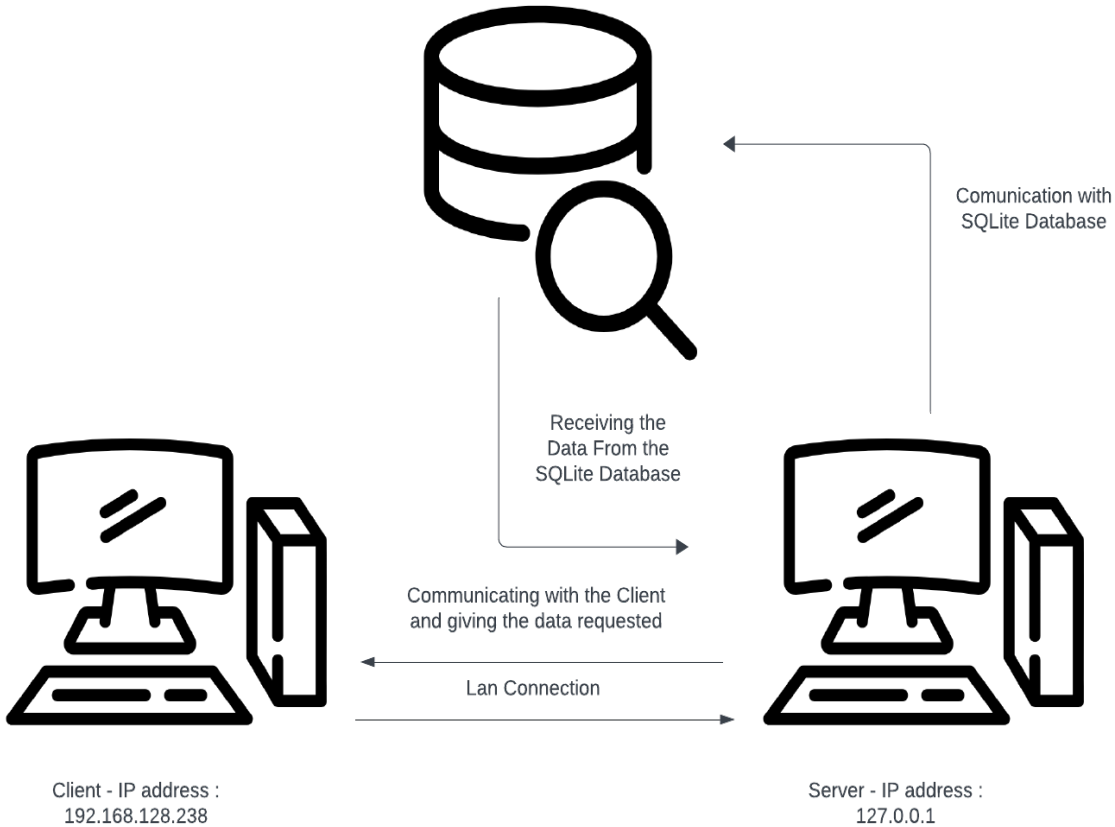


Risultati sperimentali

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 testi 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 testi 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.

Risultati sperimentali



Implementazioni futuri

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.

Conclusioni

Il progetto sviluppato ha dimostrato la validità dell'approccio multithreading per la gestione simultanea delle prenotazioni di testi 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.

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.

Grazie per l'Attenzione!!