ONLINE LIBRARY MANAGEMENT SYSTEM USING CLIENT-SERVER ARCHITECTURE

Project Overview:

Introduction:

This project is an online library management system implemented using a server-client architecture in C++. The server handles multiple clients concurrently through multithreading. Each client can perform actions such as borrowing or returning books. The system ensures that only available books can be borrowed, and it handles book availability and client details efficiently.

Objective:

The objective of the Multithreaded Online Library Management System is to efficiently manage book borrowing and returning processes by enabling multiple clients to interact with the system simultaneously, ensuring real-time updates on book availability, and maintaining data integrity through secure and scalable multithreaded server architecture.

Motivation

In today’s fast-paced digital world, the need for efficient and automated library management systems is crucial. Traditional library systems often require manual operations, which can be time-consuming and prone to errors. The motivation behind this project is to develop a robust and scalable library management system that allows multiple clients to interact with the system simultaneously, ensuring a seamless borrowing and returning process. The multithreaded architecture ensures that the system can handle concurrent client requests without performance degradation, making it suitable for modern libraries with heavy traffic.

Project Scope

This project aims to build a server-client library management system with the following features:-

Client Registration: Clients can register their details (name, address, and phone number) when they connect to the system.

Book Borrowing: Clients can borrow books by entering the book title. The system checks the availability and allows borrowing for up to 7 days.

Book Returning: Clients can return borrowed books by entering the book ID, making the book available for others.

Multi-client Support: The system can handle up to 10 clients concurrently, allowing multiple clients to borrow and return books simultaneously.

Real-time Updates: The server updates the availability of books in real-time and informs clients about the status of their requests.

Security: The system ensures that each client can only borrow or return their books, preventing unauthorized actions.

System Requirements

Hardware:

1. Server:

   Processor: Multi-core (e.g., Intel Core i5)

   RAM: 4 GB (8 GB recommended)

   Storage: 20 GB available

   Network: Ethernet or Wi-Fi

2. Client:

Processor: Modern (e.g., Intel Core i3)

RAM: 2 GB

Storage: 5 GB available

Network: Ethernet or Wi-Fi

Software:

1. Server:

OS: Linux or Windows Server

Compiler: GCC or Visual Studio

Libraries: POSIX Threads, Socket libraries

2. Client:

OS: Linux, Windows, or macOS

Compiler: GCC (GNU Compiler Collection): For Linux-based system

Libraries: Standard C++ libraries

System Design

Architecture

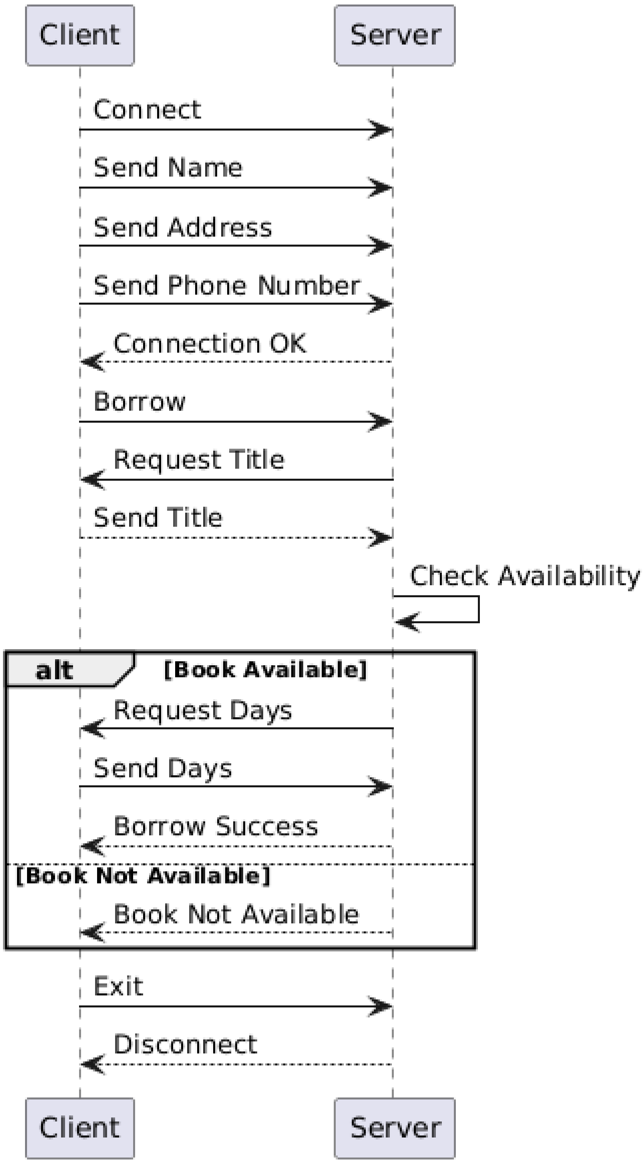
The system is built on a client-server architecture:

- Server: Manages client connections, processes book borrowing and returning requests, tracks overdue books, and handles client information.
- Client: Allows users to query book availability, borrow books, return books, and view their borrowed books list.

Components

- Server (server.cpp): Manages client connections, book queries, and tracks the borrowing and returning of books using threads.
- Client (client.cpp): Provides an interface for clients to interact with the library, including querying book availability and managing their borrowings.

System Flow



Application Tools Used

Programming Language: C++ Language

Development Tools:

- GCC(GNU Compiler collection),
- g++(C++ compiler)

Version Control: Git for source code management.

Text Editor/IDE: Vim/VS Code for code development.

Modules

Server Module

- Connection Handling: Manages client connections, including accepting new connections and spawning threads for each client.
- Book Management: Handles requests for querying, borrowing, and returning books.
- Client Management: Tracks client information, including personal details and borrowed books.

Client Module

- User Interface: Provides a text-based interface for interacting with the server.
- Book Operations: Allows clients to query book availability, borrow, and return books.
- Client Information Handling: Collects and sends client details to the server, and displays borrowed books.

Compilation and Execution

Compilation

Server:

```
g++ server.cpp -o server
```

Client:

```
g++ client.cpp -o client
```

Execution

1. Start the Server:

    ```
    ./server
    ```

2. Run the Client:

    ```
    ./client
    ```

Testing

Test Scenarios:

- Multiple Clients Borrowing Different Books: Verify that multiple clients can borrow different books simultaneously and that book availability updates correctly.
- Borrowing Unavailable Books: Ensure the server correctly informs clients when a book is not available.
- Returning Books:Test that returning a book updates the availability and allows other clients to borrow it.

Expected Outcomes:

- Books should not be borrowable if they are already borrowed.
- Once a book is returned, it should be available for other clients to borrow.

Source code:

server.cpp

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>

#define PORT 8080
#define MAX_CLIENTS 10
#define MAX_BOOKS 15

struct Book {
    char title[50];
    char author[50];
    int available;  // 1: Available, 0: Not Available
    int book_id;
};

struct Client {
    char name[50];
    char address[100];
    char phone[15];
    int client_socket;
    int borrowed_book_id;
```

```
};

Book books[MAX_BOOKS] = {
    {"The Great Gatsby", "F. Scott Fitzgerald", 1, 1001},
    {"1984", "George Orwell", 1, 2002},
    {"To Kill a Mockingbird", "Harper Lee", 1, 3003},
    {"Moby-Dick", "Herman Melville", 0, 4004},
    {"War and Peace", "Leo Tolstoy", 1, 5005},
    {"Pride and Prejudice", "Jane Austen", 1, 6006},
    {"The Catcher in the Rye", "J.D. Salinger", 1, 7007},
    {"The Hobbit", "J.R.R. Tolkien", 1, 8008},
    {"Ulysses", "James Joyce", 0, 9009},
    {"The Odyssey", "Homer", 1, 1010},
    {"Brave New World", "Aldous Huxley", 1, 1100},
    {"The Lord of the Rings", "J.R.R. Tolkien", 1, 1200},
    {"Fahrenheit 451", "Ray Bradbury", 1, 1300},
    {"The Picture of Dorian Gray", "Oscar Wilde", 1, 1400},
    {"Crime and Punishment", "Fyodor Dostoevsky", 1, 1500}
};

Client clients[MAX_CLIENTS];
int client_count = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void *handle_client(void *arg) {
    Client *client = (Client *)arg;
    char buffer[1024];
    int n;

    std::cout << "Client connected: " << client->name << ", " << client->address << ", " << client->phone << std::endl;

    while ((n = recv(client->client_socket, buffer, sizeof(buffer), 0)) > 0) {
        buffer[n] = '\0';
        std::cout << "Message from " << client->name << ": " << buffer << std::endl;

        if (strcmp(buffer, "borrow") == 0) {
            send(client->client_socket, "Enter book title: ", 18, 0);
            n = recv(client->client_socket, buffer, sizeof(buffer), 0);
            buffer[n] = '\0';

            int book_found = 0;
            pthread_mutex_lock(&lock);
            for (int i = 0; i < MAX_BOOKS; i++) {
                if (strcmp(books[i].title, buffer) == 0) {
                    book_found = 1;
                    if (books[i].available) {
                        books[i].available = 0;
                        client->borrowed_book_id = books[i].book_id;
                        send(client->client_socket, "How many days? (Max 7): ", 25, 0);
                        n = recv(client->client_socket, buffer, sizeof(buffer), 0);
                        buffer[n] = '\0';
                        int days = atoi(buffer);
                        snprintf(buffer, sizeof(buffer), "Book borrowed successfully. Book ID: %d. Please return within %d days.", client->borrowed_book_id, days);
                        send(client->client_socket, buffer, strlen(buffer), 0);
                    } else {
                        send(client->client_socket, "Book is not available. It will be available soon.", 50, 0);
                    }
                    break;
                }
            }
            pthread_mutex_unlock(&lock);

            if (!book_found) {
                send(client->client_socket, "Book not found.", 15, 0);
            }
        } else if (strcmp(buffer, "return") == 0) {
            send(client->client_socket, "Enter book ID to return: ", 25, 0);
            n = recv(client->client_socket, buffer, sizeof(buffer), 0);
            buffer[n] = '\0';
            int return_book_id = atoi(buffer);

            pthread_mutex_lock(&lock);
            for (int i = 0; i < MAX_BOOKS; i++) {
                if (books[i].book_id == return_book_id && return_book_id
== client->borrowed_book_id) {
                    books[i].available = 1;
                    client->borrowed_book_id = 0;
                    send(client->client_socket, "Book returned successfully.",
27, 0);
                    break;
                }
            }
            pthread_mutex_unlock(&lock);
        } else if (strcmp(buffer, "exit") == 0) {
            break;
        }
    }

    std::cout << "Client disconnected: " << client->name << std::endl;

    // Add a space after client disconnects
    std::cout << std::endl;

    close(client->client_socket);
    pthread_mutex_lock(&lock);
    client_count--;
    pthread_mutex_unlock(&lock);
    return NULL;
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;
    pthread_t tid;

    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    if (bind(server_socket, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Bind failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    if (listen(server_socket, MAX_CLIENTS) < 0) {
        perror("Listen failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    std::cout << "Server listening on port " << PORT << std::endl;

    while (1) {
        addr_size = sizeof(client_addr);
        client_socket = accept(server_socket, (struct sockaddr
*)&client_addr, &addr_size);
        if (client_socket < 0) {
            perror("Accept failed");
            continue;
        }

        pthread_mutex_lock(&lock);
        Client *client = &clients[client_count++];
        client->client_socket = client_socket;
        pthread_mutex_unlock(&lock);
        recv(client_socket, client->name, sizeof(client->name), 0);
        recv(client_socket, client->address, sizeof(client->address), 0);
        recv(client_socket, client->phone, sizeof(client->phone), 0);

        pthread_create(&tid, NULL, handle_client, (void *)client);
    }

    close(server_socket);
    return 0;
}
```

client.cpp

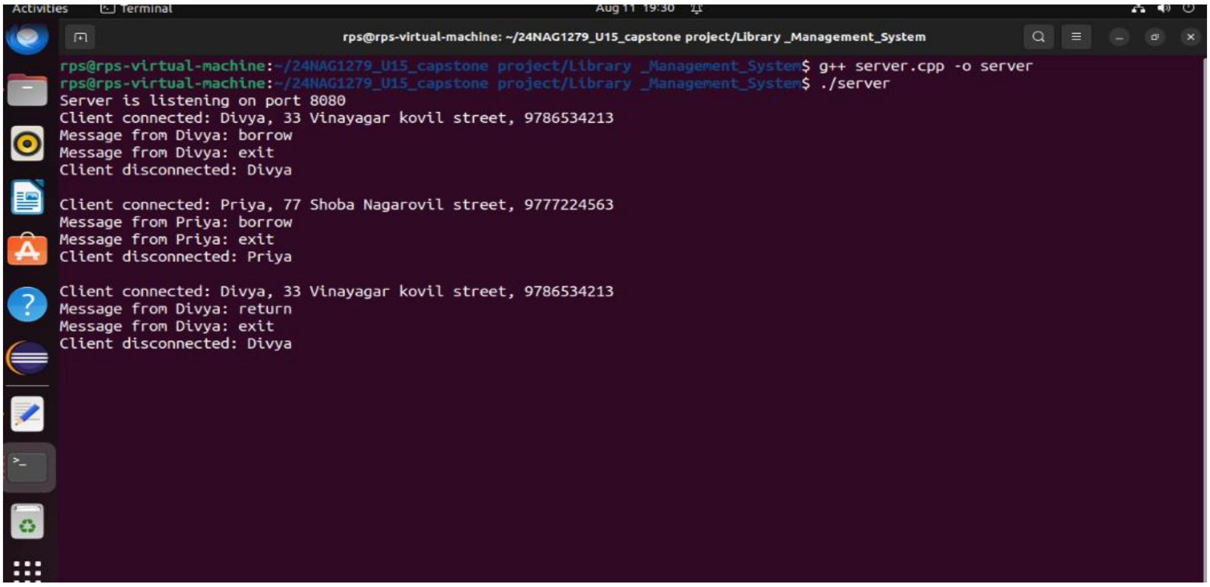
```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080

int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    char buffer[1024];
    int n;

    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if (connect(client_socket, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Connection to the server failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    // Send client details
    std::cout << "Enter your name: ";
    std::cin.getline(buffer, sizeof(buffer));
    send(client_socket, buffer, strlen(buffer), 0);

    std::cout << "Enter your address: ";
```

```
std::cin.getline(buffer, sizeof(buffer));
send(client_socket, buffer, strlen(buffer), 0);

std::cout << "Enter your phone number: ";
std::cin.getline(buffer, sizeof(buffer));
send(client_socket, buffer, strlen(buffer), 0);

while (true) {
    std::cout << "\nChoose an action (borrow/return/exit): ";
    std::cin.getline(buffer, sizeof(buffer));
    send(client_socket, buffer, strlen(buffer), 0);

    if (strcmp(buffer, "borrow") == 0) {
        recv(client_socket, buffer, sizeof(buffer), 0);
        std::cout << buffer; // "Enter book title: "

        std::cin.getline(buffer, sizeof(buffer));
        send(client_socket, buffer, strlen(buffer), 0);

        recv(client_socket, buffer, sizeof(buffer), 0);
        std::cout << buffer << std::endl; // Availability message

        if (strstr(buffer, "How many days?") != NULL) {
            std::cin.getline(buffer, sizeof(buffer));
            send(client_socket, buffer, strlen(buffer), 0);
```

```
                recv(client_socket, buffer, sizeof(buffer), 0);
                std::cout << buffer << std::endl; // Borrow confirmation
message
            }
        } else if (strcmp(buffer, "return") == 0) {
            recv(client_socket, buffer, sizeof(buffer), 0);
            std::cout << buffer; // "Enter book ID to return: "

            std::cin.getline(buffer, sizeof(buffer));
            send(client_socket, buffer, strlen(buffer), 0);

            recv(client_socket, buffer, sizeof(buffer), 0);
            std::cout << buffer << std::endl; // Return confirmation
message
        } else if (strcmp(buffer, "exit") == 0) {
            break;
        }
    }

    close(client_socket);
    return 0;
}
```

Output:

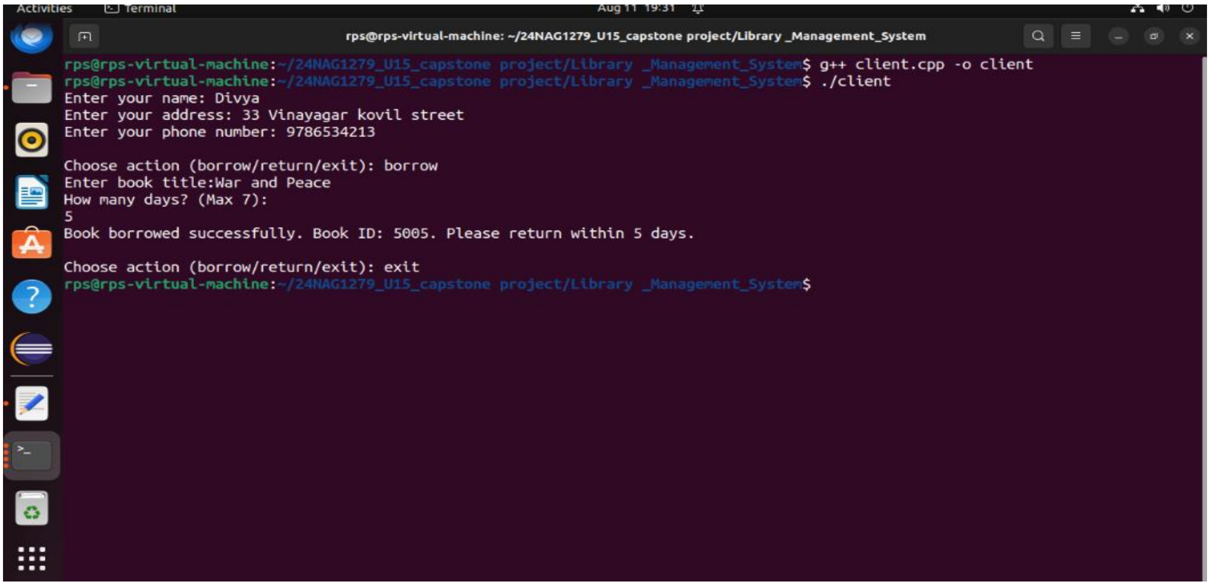
Server:


Activities
Terminal
Aug 11 19:30
rps@rps-virtual-machine: ~/24NAG1279_U15_capstone project/Library _Management_System

```
rps@rps-virtual-machine:~/24NAG1279_U15_capstone project/Library _Management_System$ g++ server.cpp -o server
rps@rps-virtual-machine:~/24NAG1279_U15_capstone project/Library _Management_System$ ./server
Server is listening on port 8080
Client connected: Divya, 33 Vinayagar kovil street, 9786534213
Message from Divya: borrow
Message from Divya: exit
Client disconnected: Divya

Client connected: Priya, 77 Shoba Nagarovil street, 9777224563
Message from Priya: borrow
Message from Priya: exit
Client disconnected: Priya

Client connected: Divya, 33 Vinayagar kovil street, 9786534213
Message from Divya: return
Message from Divya: exit
Client disconnected: Divya
```

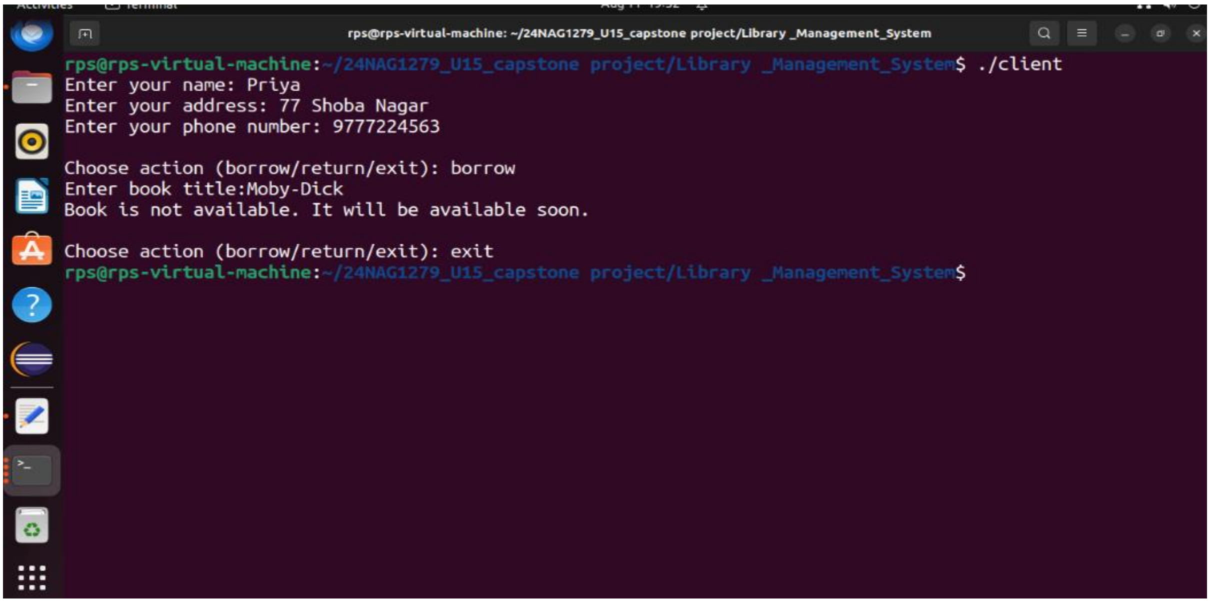
Client:

Client 1:


Activities
Terminal
Aug 11 19:31
rps@rps-virtual-machine: ~/24NAG1279_U15_capstone project/Library _Management_System

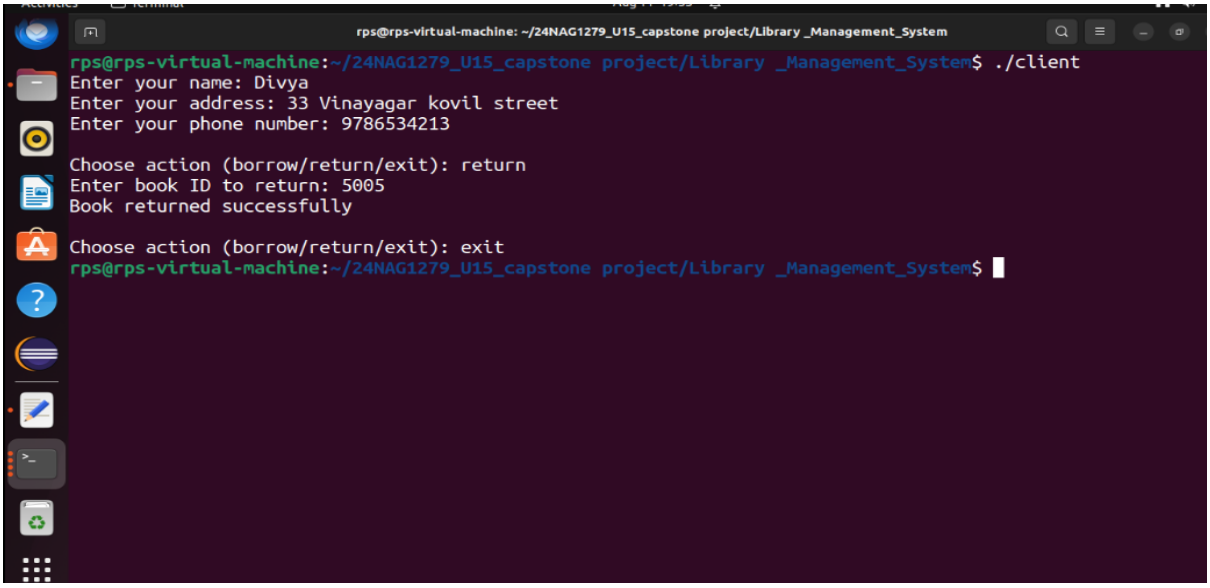
```
rps@rps-virtual-machine:~/24NAG1279_U15_capstone project/Library _Management_System$ g++ client.cpp -o client
rps@rps-virtual-machine:~/24NAG1279_U15_capstone project/Library _Management_System$ ./client
Enter your name: Divya
Enter your address: 33 Vinayagar kovil street
Enter your phone number: 9786534213

Choose action (borrow/return/exit): borrow
Enter book title:War and Peace
How many days? (Max 7):
5
Book borrowed successfully. Book ID: 5005. Please return within 5 days.

Choose action (borrow/return/exit): exit
rps@rps-virtual-machine:~/24NAG1279_U15_capstone project/Library _Management_System$
```

Client 2:


rps@rps-virtual-machine: ~/24NAG1279_U15_capstone project/Library _Management_System

```
rps@rps-virtual-machine:~/24NAG1279_U15_capstone project/Library _Management_System$ ./client
Enter your name: Priya
Enter your address: 77 Shoba Nagar
Enter your phone number: 9777224563

Choose action (borrow/return/exit): borrow
Enter book title:Moby-Dick
Book is not available. It will be available soon.

Choose action (borrow/return/exit): exit
rps@rps-virtual-machine:~/24NAG1279_U15_capstone project/Library _Management_System$
```

Client 3:


rps@rps-virtual-machine: ~/24NAG1279_U15_capstone project/Library _Management_System

```
rps@rps-virtual-machine:~/24NAG1279_U15_capstone project/Library _Management_System$ ./client
Enter your name: Divya
Enter your address: 33 Vinayagar kovil street
Enter your phone number: 9786534213

Choose action (borrow/return/exit): return
Enter book ID to return: 5005
Book returned successfully

Choose action (borrow/return/exit): exit
rps@rps-virtual-machine:~/24NAG1279_U15_capstone project/Library _Management_System$ 
```

Advantages

- Concurrent Handling:Supports multiple clients simultaneously for efficient operations.
- Real-Time Updates: Instant book availability updates prevent errors.
- Scalability: Easily adapts to growing library needs.
- User-Friendly: Simple interface for all users.
- Automation: Reduces manual work and human errors.
- Security: Protects against unauthorized access.
- Customizable Borrowing: Flexible borrowing periods within limits.

Future Enhancement

- Client Authentication: Implement user login functionality to keep track of user borrowing history.
- Graphical User Interface: Develop a GUI for the client application to make the system more user-friendly.
- Notification System: Implement a system to notify clients when borrowed books are due.

Conclusion

The Multithreaded Online Library Management System provides a robust implementation of a client-server architecture for managing book borrowings and returns. It efficiently handles multiple clients using multithreading, ensuring smooth and concurrent interactions. The system serves as a solid foundation for further development, including database integration, enhanced user interfaces, and improved security measures.