**PROJECT REPORT**

**Spring 2024****:**

**CSE-204L Operating Systems Lab**

Submitted by:

**MUHAMMAD MUSAB (22PWCSE2201)  
 HASSAN ZAIB JADOON (22PWCSE2144)  
 SHAHBAZ UD DIN (22PWCSE2147)**

Class Section: **A**

“On our honor, as students of University of Engineering and Technology, We have neither given nor received unauthorized assistance on this academic work.”

Submitted to:

**Engr. ABDULLAH HAMID**

**Department of Computer Systems Engineering**

University of Engineering and Technology, Peshawar

**Abstract**

This project report details the development and implementation of a Hotel Management System designed to handle multiple client connections simultaneously using threads and reliable TCP socket communication. The primary objective of this system is to facilitate efficient hotel operations, such as room booking, check-in/check-out processes, and availability management, through a robust and scalable solution.

The system's architecture employs a client-server model, where the server manages the hotel's operations and clients interact with the server to perform various tasks. The server is designed to handle multiple clients concurrently using POSIX threads (pthread) and TCP sockets, ensuring reliable communication and thread-safe operations. Each client connection is managed by a dedicated thread, allowing simultaneous operations without data inconsistencies or conflicts.

The methodology involves developing the server and client applications in C programming language, chosen for its efficiency and control over system resources. The system's functionalities include booking rooms, checking in and out, displaying available rooms, and proper session termination. The server maintains the status of rooms efficiently and ensures that all operations are thread-safe using mutexes.

Comprehensive testing was conducted to evaluate the system's performance, including unit testing, integration testing, and user acceptance testing. The results indicate that the system successfully handles multiple client connections, performs all hotel management operations correctly, and maintains data consistency without crashes.

In conclusion, the Hotel Management System meets all project objectives, providing a reliable and efficient solution for managing hotel operations. Future enhancements could include a graphical user interface, online payment integration, and advanced booking options. This system can significantly improve the efficiency and customer service of hotel operations, making it a valuable tool for the hospitality industry.

## 1. Introduction

Hotel management systems are essential for the efficient operation of hotels, providing functionalities like room booking, check-in/check-out, and availability management. This project aims to develop a robust and efficient hotel management system using TCP socket communication and threading to handle multiple clients concurrently. This report outlines the objectives, methodologies, implementation, and testing of the system.

## 2. Literature Review

Existing hotel management systems vary in complexity and functionality, ranging from simple reservation systems to comprehensive management solutions integrating with other business operations. Many systems face challenges such as scalability, concurrent user management, and ensuring data consistency. Our project addresses these issues by using a multithreaded approach and reliable TCP communication.

**3. Objectives**

* Use threads to handle multiple client connections simultaneously.
* Establish reliable TCP socket communication between clients and the server.
* Perform hotel management operations:
  + Check-In: Book a room.
  + Check-Out: Check out of a room.
  + Display Available Rooms: List available rooms.
  + Exit: Disconnect from the server.
* Maintain room status efficiently and ensure thread-safe operations.
* Allow proper termination of client sessions and clean server shutdown.
* Ensure the client and server programs run on an Ubuntu terminal.

**4. Methodology**

The project was developed using the C programming language due to its efficiency and control over system resources. The following tools and libraries were used:

* **POSIX Threads (pthread)** for handling multiple threads.
* **Berkeley sockets** for TCP/IP communication.

Design choices included a multithreaded server to handle multiple clients concurrently and ensuring thread-safe operations using mutexes. The server listens for incoming client connections and spawns a new thread to handle each connection.

## 5. System Architecture

### **Overview**

The system consists of a client-server architecture where the server manages hotel operations, and the clients interact with the server to perform various tasks.

**Diagram**

**Server**

**(Hotel Management System)**

**Client 1:**

**Client 1:**

**Client 1:**

**Client 1:**

### **Components**

* **Server**: Handles client requests, manages room status, and ensures thread-safe operations.
* **Client**: Connects to the server and allows users to perform hotel management operations.

## 6. Implementation

### **Pseudo Code for Server Code:**

// Constants

PORT = 8080

MAX\_ROOMS = 10

// Global Variables

availableRooms[MAX\_ROOMS] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

// Main Function

function main():

    server\_fd = create\_tcp\_socket()

    bind\_to\_port(server\_fd, PORT)

    start\_listening(server\_fd)

    while True:

        client\_socket = accept\_connection(server\_fd)

        create\_thread(handle\_client, client\_socket)

    close(server\_fd)

// Function to handle each client

function handle\_client(client\_socket):

    send\_menu\_options(client\_socket)

    while True:

        choice = read\_client\_choice(client\_socket)

        switch choice:

            case 1: book\_room(client\_socket)

            case 2: check\_out\_room(client\_socket)

            case 3: display\_available\_rooms(client\_socket)

            case 4: close(client\_socket); exit\_thread()

            default: handle\_invalid\_choice(client\_socket)

// Function to book a room

function book\_room(client\_socket):

    room\_number = read\_room\_number(client\_socket)

    if room\_number < 1 or room\_number > MAX\_ROOMS:

        send\_invalid\_room\_message(client\_socket)

        return

    if availableRooms[room\_number - 1] == 1:

        availableRooms[room\_number - 1] = 0

        send\_booking\_success\_message(client\_socket, room\_number)

    else:

        send\_room\_already\_booked\_message(client\_socket, room\_number)

// Function to check out from a room

function check\_out\_room(client\_socket):

    room\_number = read\_room\_number(client\_socket)

    if room\_number < 1 or room\_number > MAX\_ROOMS:

        send\_invalid\_room\_message(client\_socket)

        return

    availableRooms[room\_number - 1] = 1

    send\_checkout\_success\_message(client\_socket, room\_number)

// Function to display available rooms

function display\_available\_rooms(client\_socket):

    send\_available\_rooms\_list(client\_socket)

// Additional Helper Functions:

// - create\_tcp\_socket(), bind\_to\_port(), start\_listening(), accept\_connection()

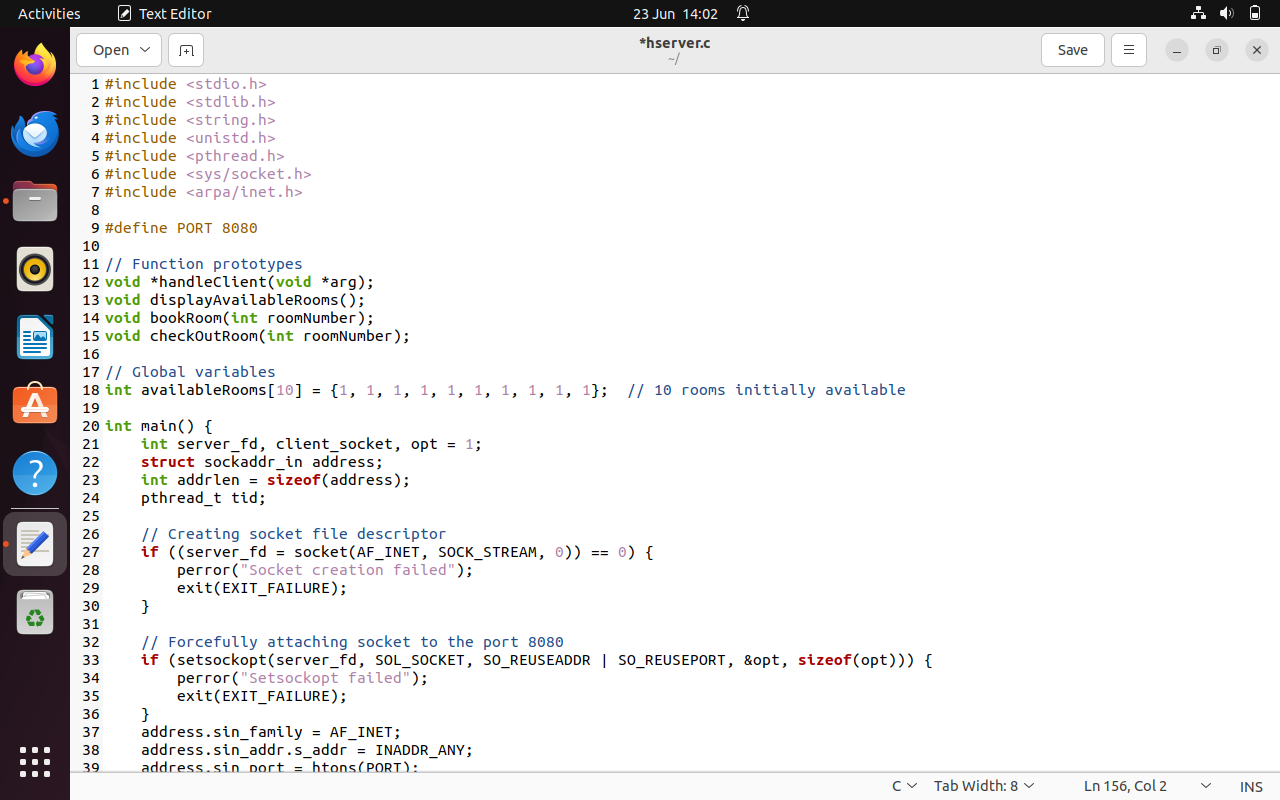
// - send\_menu\_options(), read\_client\_choice(), read\_room\_number()

// - send\_booking\_success\_message(), send\_room\_already\_booked\_message()

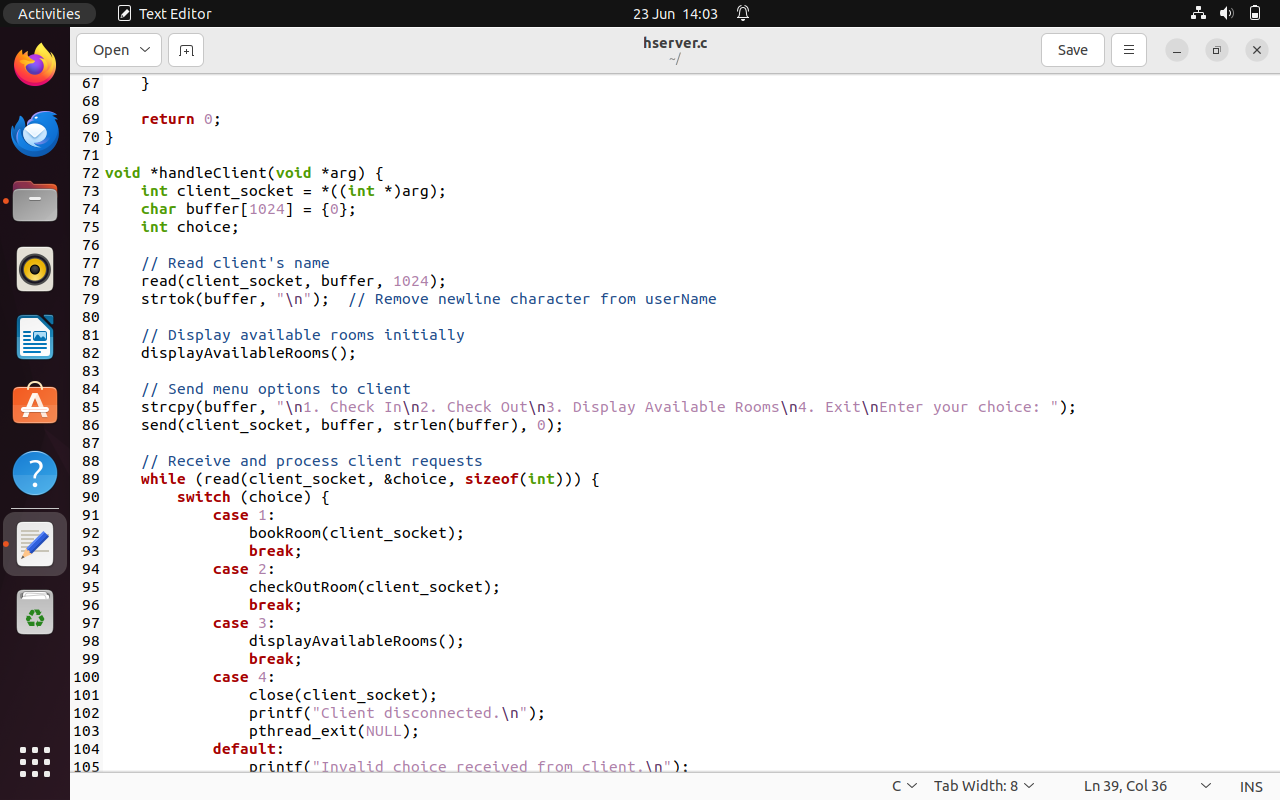
// - send\_checkout\_success\_message(), send\_invalid\_room\_message()

// - send\_available\_rooms\_list(), handle\_invalid\_choice()

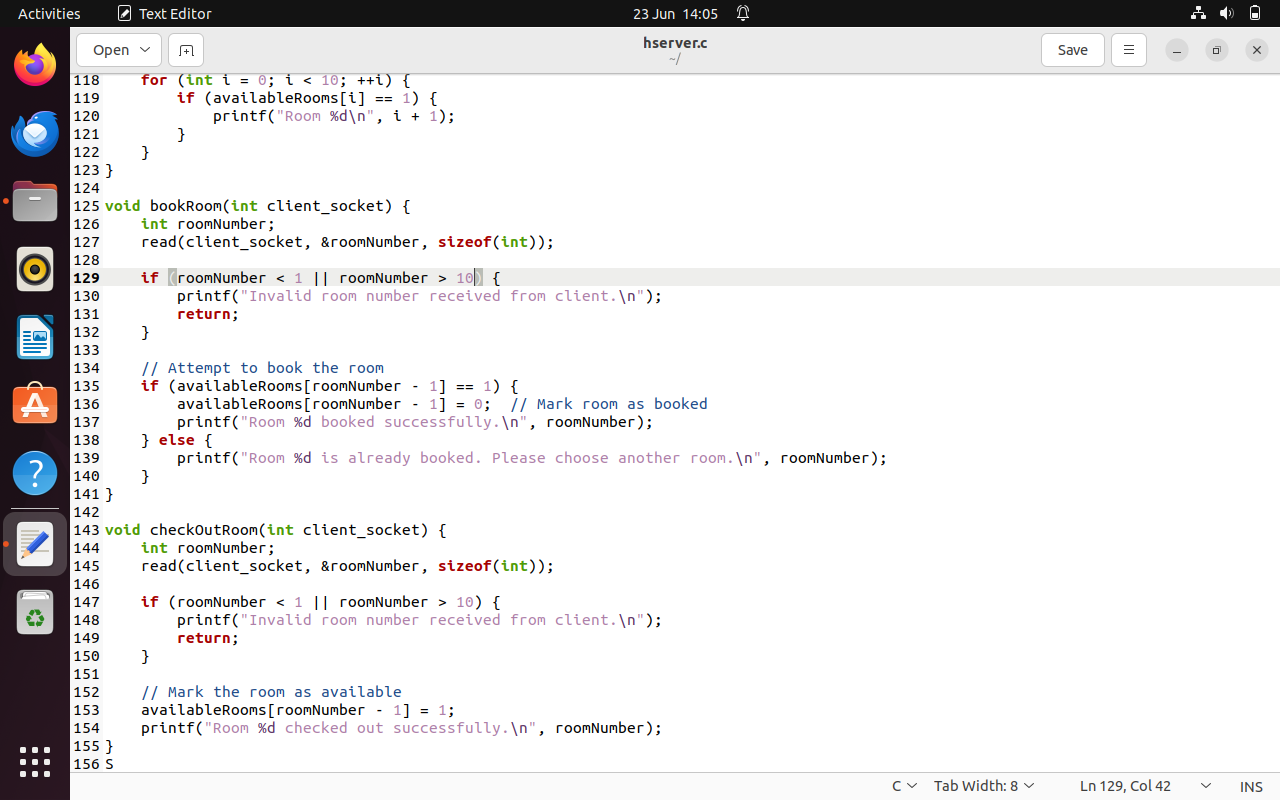
### **Code Screenshots for Server:**











### **Pseudo Code for Client:**

// Constants

PORT = 8080

// Main Function

function main():

    sock = create\_tcp\_socket()  // Create TCP socket

    serv\_addr = initialize\_server\_address()  // Initialize server address

    // Connect to server

    if connect\_to\_server(sock, serv\_addr) < 0:

        print("Connection Failed")

        return -1

    // Get user's name

    userName = get\_user\_name()

    // Send user name to server

    send\_user\_name(sock, userName)

    // Receive and display initial menu options from server

    display\_initial\_menu\_options(sock)

    // Loop to interact with server

    while True:

        // Get user's choice

        choice = get\_user\_choice()

        // Send user's choice to server

        send\_user\_choice(sock, choice)

        // Process server responses based on user's choice

        switch choice:

            case 1:

            case 2:

                // Get room number from user

                roomNumber = get\_room\_number()

                // Send room number to server

                send\_room\_number(sock, roomNumber)

            case 4:

                // Close connection to server

                close\_connection(sock)

                print("Disconnected from server.")

                return 0

            default:

                // Invalid choice, do nothing and continue

                break

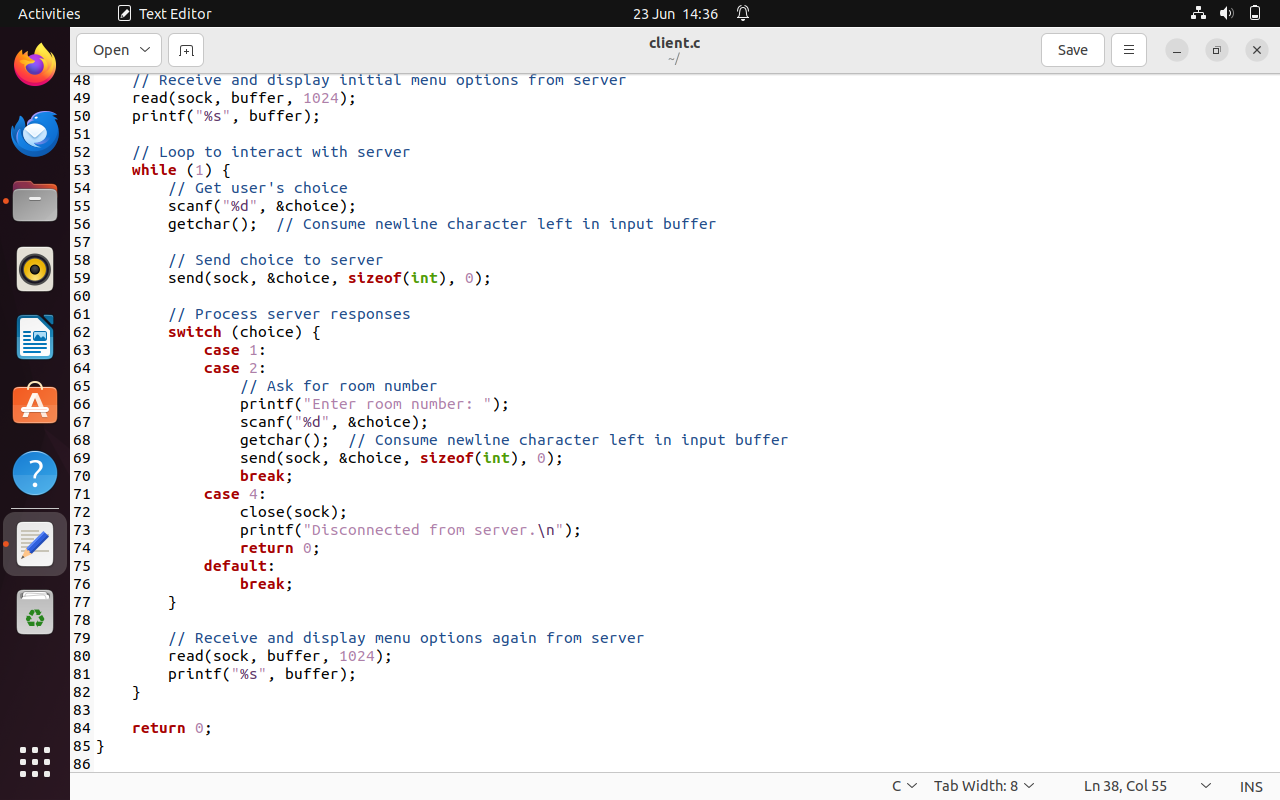
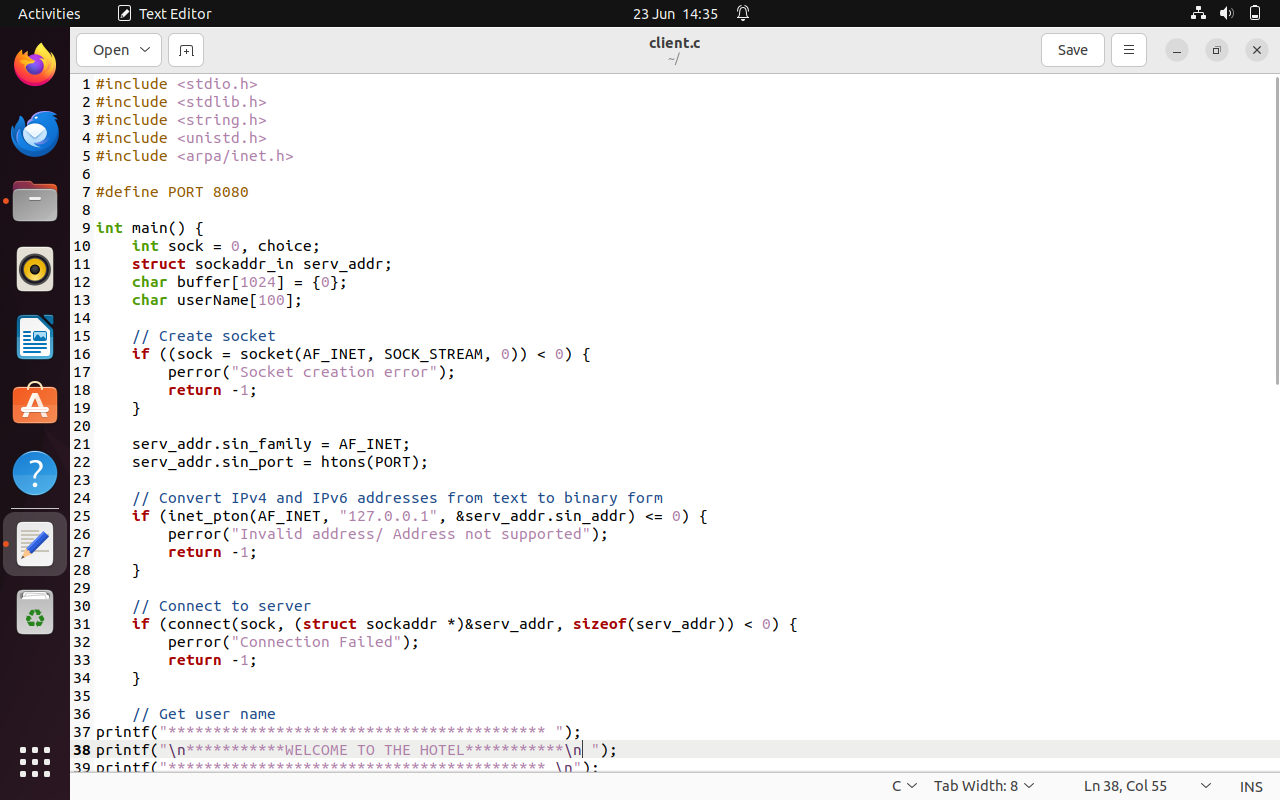
        // Receive and display menu options again from server

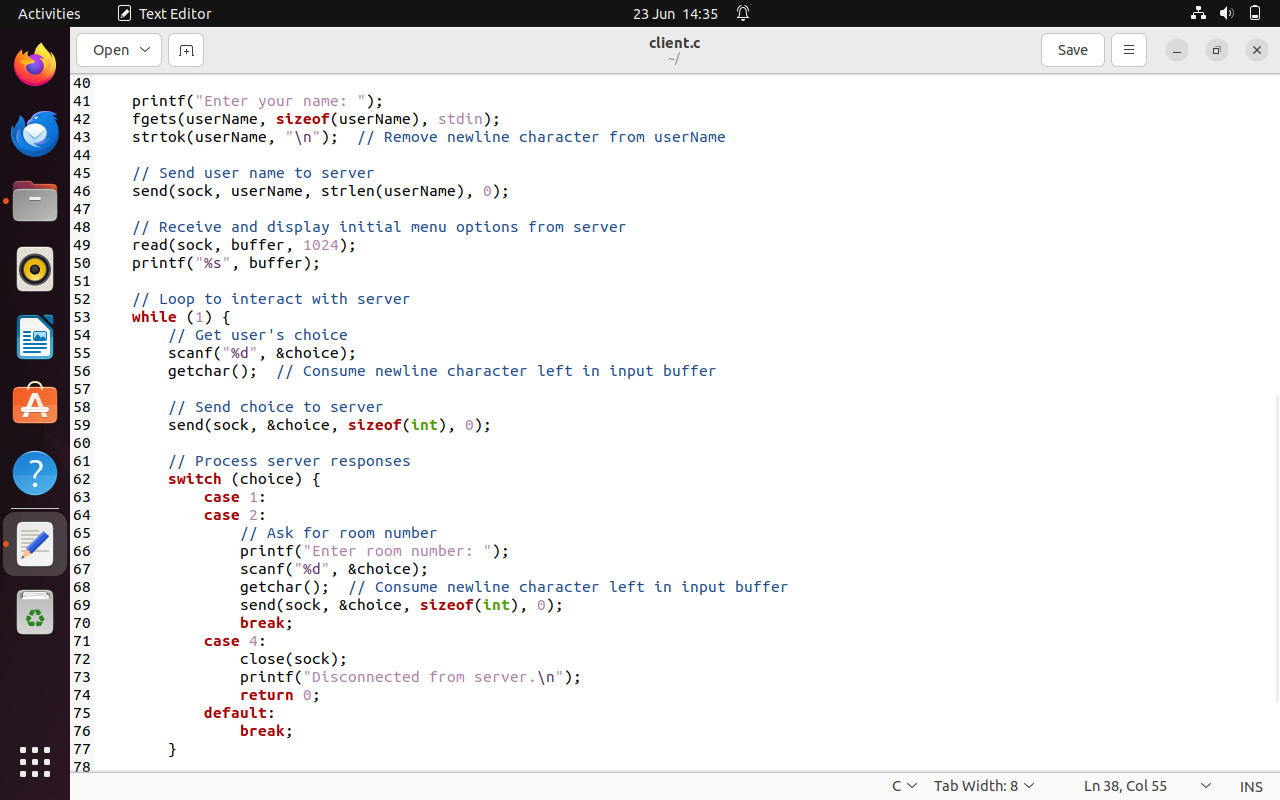
        display\_menu\_options(sock)

    // End of main function

    return 0

### **Client Code Screen Shots:**





### **User Interface**

The user interacts with the system through a terminal interface. Clients send commands to the server, which processes them and responds accordingly. The interface is text-based and simple to use.

## 7. Output

## 

Figure 1: Client

## 

Figure 2: Server

## 8. Testing and Evaluation

### **Testing Procedures**

* **Unit Testing**: Each function was tested independently to ensure correctness.
* **Integration Testing**: The interaction between the server and multiple clients was tested to ensure seamless communication.
* **User Acceptance Testing**: End-users tested the system to validate its functionality and usability.

### **Test Results**

* The system successfully handled multiple client connections concurrently.
* All hotel management operations were performed correctly.
* No data inconsistencies or crashes were observed during testing

## 9. Conclusion

The Hotel Management System successfully meets the project objectives, providing a reliable and efficient solution for managing hotel operations. The use of threads and TCP socket communication ensures scalability and robustness. Future enhancements could include a graphical user interface and additional features like online payments and advanced booking options.

## 10. Appendices

### **GitHub Repository**

The complete source code for this project can be found on GitHub at the following link:

<https://github.com/hzjadoon/OS-Project>

### **System Calls**

This section provides an overview of the system calls used in the implementation of the Hotel Management System:

* **socket**: Creates a new socket of the specified domain type and protocol. In this case, it creates a TCP/IP socket.
* **setsockopt**: Sets options on the socket. Here it is used to allow reuse of local addresses and ports.
* **bind**: Binds the socket to an address and port number specified in the address structure.
* **listen**: Marks the socket as a passive socket that will be used to accept incoming connection requests.
* **accept**: Accepts an incoming connection on a listening socket. It creates a new socket for the connection.
* **pthread\_create**: Creates a new thread of execution. Here it creates a thread to handle each new client connection.
* **read**: Reads data from a file descriptor (in this case a socket). Used to read data sent by the client.
* **send**: Sends data on a connected socket.
* **close**: Closes a file descriptor (socket) and terminates the connection.

Understanding these system calls is crucial for comprehending how the server and client communicate, handle connections, and manage data flow. Detailed knowledge of these calls ensures efficient and effective implementation of network applications.