**MULTITHTREDED ONLINE TICKET BOOKING SYSTEM**

# Introduction

The Online Ticket Booking System is a client-server application designed to facilitate online ticket booking for travelling to any location. ticket booking system is a platform enabling users to purchase and manage tickets for various events, travel, or services through the web. it is essential to develop a secure, efficient, and reliable platform that can handle a large number of users login and provide accurate ticket booking system. Allows users to book tickets from anywhere, at any time, without the need to visit a physical location or make phone calls.

# Objective

* The objective of an online ticket booking system is to efficiently facilitate the purchase, management, and distribution of tickets for various events, services, or travel options. The key objectives include:
* Simplify the steps required to find, select, and purchase tickets, reducing the time and effort involved for users.
* Offer up-to-date details on ticket availability, pricing, and event schedules, ensuring users have accurate information for their decisions.
* Protect user data and payment information through secure transactions and encryption, fostering trust and safeguarding against fraud.To provide real-time updates of election results
* Deliver instant booking confirmations and tickets, either digitally or through email/SMS, allowing users to receive immediate proof of purchase.
* Provide tools for users to view, modify, or cancel their bookings easily, and for organizers to manage events, track sales, and handle customer inquiries.
* Collect data on booking trends, user behaviour, and sales performance, helping organizers and providers make informed decisions and optimize their offerings
* To handle multiple client connections concurrently using multi-threading

# System Requirements

## Hardware Requirements

* Operating System: Linux/Unix-based
* Processor: Multi-core processor
* Memory: 4 GB RAM or more
* Storage: 10 GB or more

## Software Requirements

* Programming Language: C,C++
* Libraries: sys/socket.h, netinet/in.h, arpa/inet.h, unistd.h
* Compiler: GCC

# Functionality

* **User Interface:**
* **Homepage:** Displays search options, featured events, and promotions.
* **Search and Filters:** Allows users to find tickets based on various criteria like date, location, event type, or price range.
* **Account Management:**
* **Registration and Login:** Users can create and access personal accounts for managing bookings and preferences.
* **Profile Management:** Update personal information, payment details, and preferences.
* **Booking Process:**
* Search and Selection: Users search for and select events, dates, and ticket types or quantities.

Seat Selection (if applicable): Interactive seating charts for events with assigned seating, allowing users to choose specific seats.

* Review and Confirmation: Users review their booking details before proceeding to payment.
* Payment Processing: Secure integration with payment gateways to handle transactions through various methods (credit/debit cards, digital wallets, etc.)
* Ticket Management: E-tickets are generated and can be downloaded or accessed through a mobile app.
* **Security Features**:
* **Data Encryption:** Secure handling of personal and payment information.

# Modules (Components)

## Server Components

1. Socket Programming: The server creates a socket and binds it to a specific address and port. It listens for incoming client connections and accepts them.
2. Multi-Threading: The server creates a new thread for each incoming client connection. Each thread handles a single client connection and executes the handle\_client() function.
3. Mutexes and Condition Variables: The server uses a mutex (mtx) to protect access to the booking results map (booking\_results). It also uses a condition variable (cv) to notify waiting threads when a new booking is process.
4. Ticket Booking Map: The server maintains a map (ticket\_avail) to store the user details. The ticket is updated whenever a new update done by user.

## Client Components

1. Socket Programming: The client creates a socket and connects to the server's socket. It sends requests to cast votes or retrieve election results.
2. Ticket Booking : The client sends a request to book a ticket for a specific user.

# Implementation

Here's a high-level overview of how these mechanisms are implemented in your project:

1. **Socket programming**:
   * The server creates a socket and binds it to a specific address and port.
   * Clients create sockets and connect to the server's socket.
   * Both the server and clients use **send()** and **recv()** to exchange data.

1. **Multi-threading**:
   * The server creates a new thread for each incoming client connection using **std::thread**.
   * Each thread handles a single client connection and executes the **handle\_client()** function.
   * pthread\_create()
   * Prototype: int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine) (void \*), void \*arg);
   * Usage: This function is used to create a new thread. In the program, it spawns NUM\_THREADS threads, each executing the bookTickets function
   * pthread\_join()
   * Prototype: int pthread\_join(pthread\_t thread, void \*\*retval);
   * Usage: This function waits for the specified thread to terminate. It ensures that the main program waits for all threads to complete before proceeding.
   * pthread\_mutex\_init()
   * Prototype: int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*attr);
   * Usage: This function initializes a mutex for synchronization. The mutex is used to control access to the availableTickets shared resource to avoid race conditions.

**Testing**

The Multithreded ticket booking system has been tested using the following scenarios:

1. Single User: A single USER book ticket and get confirmation.
2. Multiple Users: Multiple users book ticket and get confirmation concurrently.

**SOURCE CODE:**

**MULTITHREDED TICKET BOOKING SYSTEM**

// TicketBooking.c

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

// Define initial number of tickets and number of threads

#define INITIAL\_TICKETS 100

#define NUM\_THREADS 10

#define TICKETS\_PER\_REQUEST 15

// Shared resource

typedef struct {

int availableTickets;

pthread\_mutex\_t mutex;

} TicketBookingSystem;

// Function to book tickets

void\* bookTickets(void\* arg) {

TicketBookingSystem\* system = (TicketBookingSystem\*) arg;

int numTickets = TICKETS\_PER\_REQUEST;

pthread\_mutex\_lock(&system->mutex);

if (numTickets > system->availableTickets) {

printf("Not enough tickets available.\n");

} else {

// Simulate booking time

usleep(100000); // 100 milliseconds

system->availableTickets -= numTickets;

printf("Successfully booked %d tickets. Remaining tickets: %d\n", numTickets, system->availableTickets);

}

pthread\_mutex\_unlock(&system->mutex);

return NULL;

}

int main() {

pthread\_t threads[NUM\_THREADS];

TicketBookingSystem system;

// Initialize the system

system.availableTickets = INITIAL\_TICKETS;

pthread\_mutex\_init(&system.mutex, NULL);

// Create threads

for (int i = 0; i < NUM\_THREADS; i++) {

if (pthread\_create(&threads[i], NULL, bookTickets, (void\*)&system) != 0) {

perror("Failed to create thread");

return 1;

}

}

// Join threads

for (int i = 0; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

// Destroy mutex

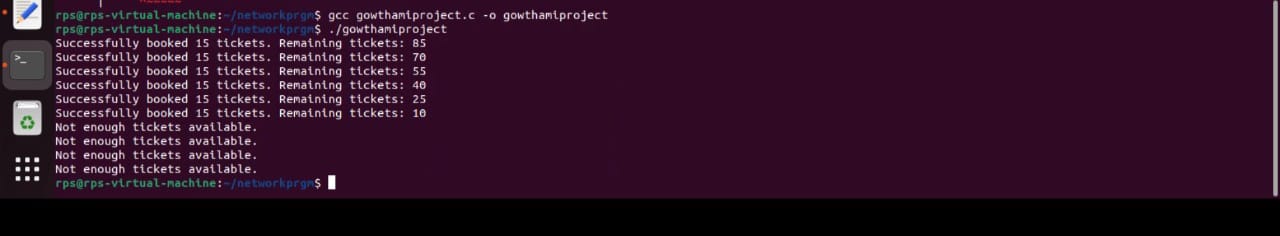
pthread\_mutex\_destroy(&system.mutex);

printf("All threads completed. Final remaining tickets: %d\n", system.availableTickets);

return 0;

}

**OUTPUT SCREENSHOT SEVER OUTPUT:**



**Conclusion**

The conclusion of an online ticket booking system is that it provides a modern, efficient, and user-friendly platform for purchasing and managing tickets for events, travel, or services. Users can access the system from anywhere at any time, simplifying the ticket purchasing process and eliminating the need for physical visits or phone calls. The system automates and streamlines the booking process, reducing manual effort, minimizing errors, and speeding up transactions. Provides users with up-to-date availability, pricing, and event details, ensuring they have accurate information to make informed decisions. Protects user data and payment information through encryption and secure payment gateways, fostering trust and reducing the risk of fraud. Offers instant confirmation of bookings and tickets, either digitally or through email/SMS, ensuring users receive prompt proof of purchase.