JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

B.TECH V SEMESTER

OPERATING SYSTEM & SYSTEM PROGRAMMING LAB PBL REPORT

Restaurant Order Scheduling & Threading

Supervision of:

Dr. Alka Singhal &

Dr. Parmeet Kaur

Department Of Computer Science

JIIT, Sector 62, Noida

Submitted By:

Enrolment No.	Name
22103217	Yash Mittal
22103232	Aviral

ABSTRACT

The **Restaurant Order Management System** is a comprehensive project that integrates a multithreaded backend server and an interactive web-based frontend interface to simulate and manage customer orders in a restaurant environment. The backend, developed in C, demonstrates the effective use of **multithreading**, **semaphores**, and **mutex locks** for synchronizing waiter availability and efficiently handling concurrent client requests. The frontend, created using HTML, CSS, and JavaScript, provides an intuitive interface for customers to place orders and receive real-time feedback on their status.

This system addresses the common challenges in restaurant order management, such as ensuring that customer requests are processed promptly, resources (waiters) are allocated efficiently, and customer satisfaction is maintained. The backend handles order queuing, dynamic waiter-thread allocation, and simulates real-world serving times, making it both a practical solution for small-scale operations and a foundation for scaling up to enterprise-level systems.

The project showcases the synergy between **system-level programming** and **web development**, emphasizing scalability, reliability, and user experience. It is an excellent learning example of applying core computer science concepts such as threading, synchronization, and client-server communication to solve real-world problems.

INTRODUCTION

Problem Statement

Restaurants face challenges in managing dynamic customer requests efficiently. The unavailability of waiters or uncoordinated order handling can lead to delays and customer dissatisfaction. A robust system is required to handle multiple orders concurrently, allocate resources dynamically, and provide real-time updates to customers.

Project Objectives

The primary objectives of this project are:

1. Design a multithreaded server to manage client requests and waiter availability.
2. Develop a frontend interface to allow customers to submit orders easily.
3. Implement mechanisms for synchronization, ensuring thread-safe operations.
4. Provide meaningful feedback to customers about their order status.

Scope of the Project

The **Restaurant Order Management System** has a wide scope that extends beyond the basic functionalities of managing customer orders and waiter allocation. This system's design and modularity make it applicable to various industries and adaptable to future advancements. Below is a detailed exploration of its scope:

1. Practical Applications

The system addresses common challenges in the hospitality industry and has direct practical applications:

- **Restaurants and Cafes:** Efficiently handle high volumes of customer orders, especially during peak hours, while minimizing delays and errors.
- **Catering Services:** Manage simultaneous orders and resources in off-site catering operations.
- **Customer Support Centers:** Adapt the system for ticket handling in support or complaint management scenarios.
- **Healthcare Industry:** Use the framework for patient scheduling and resource allocation in hospitals.

SYSTEM REQUIREMENTS

Hardware Requirements

- **Processor**: Intel i5 or equivalent for multitasking capabilities**.**
- **Memory:** At least 8 GB RAM to handle multiple threads efficiently.
- **Network:** A LAN or localhost setup for testing purposes.

Software Requirements

- **Operating System:** Linux (preferred for multithreading) or Windows.
- **Programming Language:** C for backend; HTML, CSS, and JavaScript for frontend.
- **Libraries:** POSIX Threads (pthread), Sockets API for server operations.

SYSTEM ARCHITECTURE

Modules

1. Client Module (Frontend)

- A web-based form for customers to place orders.
- Sends HTTP POST requests to the server with the customer ID and serving time.
- Displays server responses, including order status and errors.

2. Server Module (Backend)

- A multithreaded server listens on port 8080.
- Handles client requests, validates input, and queues orders.
- Manages waiter threads to serve queued orders.

3. Waiter Module

- Each waiter is a separate thread that picks up orders from the queue.
- Simulates serving time and marks the waiter as available afterward.

Flow Diagram

1. Customer places an order via the web interface.
2. The server validates the request and adds it to the order queue.
3. Waiter threads pick orders, simulate service, and notify completion.
4. Customers receive order status updates in real time**.**

IMPLEMENTATION

The implementation of the **Restaurant Order Management System** is divided into two primary components: the **backend server** and the **frontend interface**. Below is a detailed explanation of the steps, technologies, and techniques used to build this system.

1. Backend Implementation

The backend is a multithreaded server written in C that handles customer orders and manages waiters. It employs socket programming, thread synchronization, and HTTP request handling.

1.1 Key Functionalities

- Order Queue Management:
  The server maintains a queue of customer orders using an array, ensuring thread-safe access with a mutex lock.
- Waiter Allocation:
  A pool of waiter threads continuously serves orders, with availability managed using semaphores.
- HTTP Server:
  The server listens on a designated port (8080) and processes incoming HTTP POST requests from clients.

1.2 Modules in Backend

a. Order Handling Module:

- When a client sends an HTTP POST request, the server extracts the customer_id and serving_time from the request body.
- If the queue is not full, the order is added to the queue, and the client receives an acknowledgment. If full, an error message is returned.
- Synchronization is achieved using a mutex lock to prevent concurrent writes to the order queue.

b. Waiter Module:

- Each waiter thread waits for an order to be available in the queue.
- After picking an order, it simulates serving by sleeping for the specified serving_time.
- Once serving is completed, the waiter becomes available again, indicated by signaling the waiter semaphore.

c. HTTP Response Handling:

- The server responds to clients with proper HTTP headers, including CORS headers to allow cross-origin communication with the frontend.

1.3 Key Backend Features

1. **Multithreading:**
   Each waiter is a separate thread, and client connections are handled in their respective threads, ensuring concurrent processing.
2. **Synchronization Mechanisms:**
   - Mutex (pthread_mutex_t): Ensures safe access to the shared order queue.
   - Semaphores (sem_t):
     - Order Semaphore: Tracks pending orders.
     - Waiter Semaphore: Tracks the availability of waiters.
3. **Server Robustness:**
   - Handles multiple client connections and processes HTTP POST requests efficiently.
   - Handles edge cases such as an empty queue or a full order list.

2. Frontend Implementation

The frontend is a **web-based interface** that allows customers to place orders conveniently and displays responses from the server. It is built using **HTML, CSS, and JavaScript**.

2.1 Structure of the Frontend

a. HTML (User Interface):

- A simple form allows users to input Customer ID and Serving Time.
- A section displays the server's response dynamically.

b. CSS (Styling):

- Enhances the visual appeal of the form with modern design elements.
- Ensures responsiveness so the system can be accessed on various devices.

c. JavaScript (Dynamic Functionality):

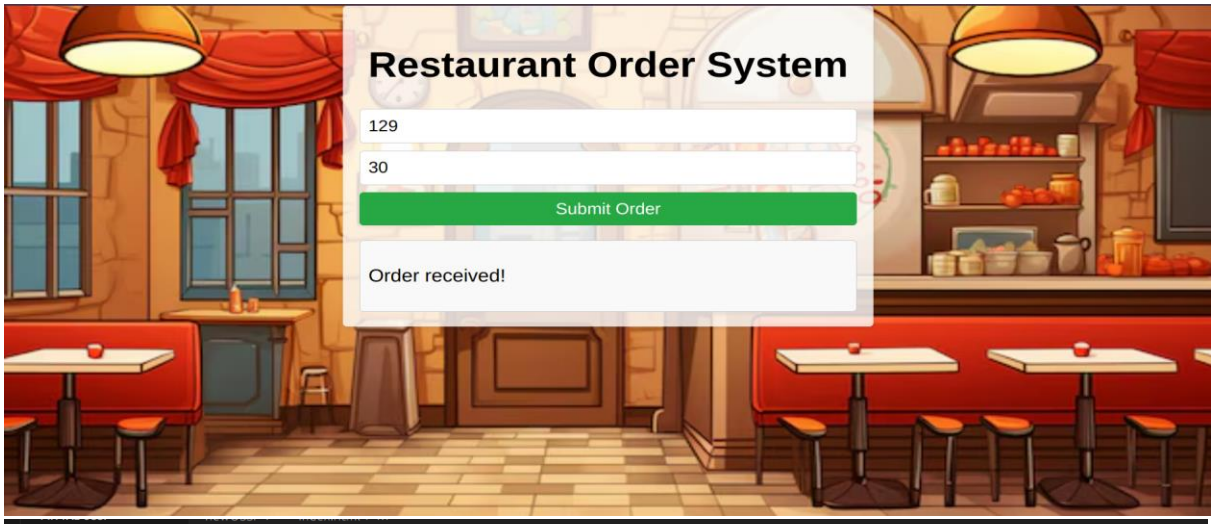
- Event Listener: Captures form submissions and prevents default behavior.
- Fetch API: Sends HTTP POST requests to the server with the order details in plain text.
- Response Handling: Displays the server's response (e.g., order acknowledgment, error messages) dynamically on the webpage.

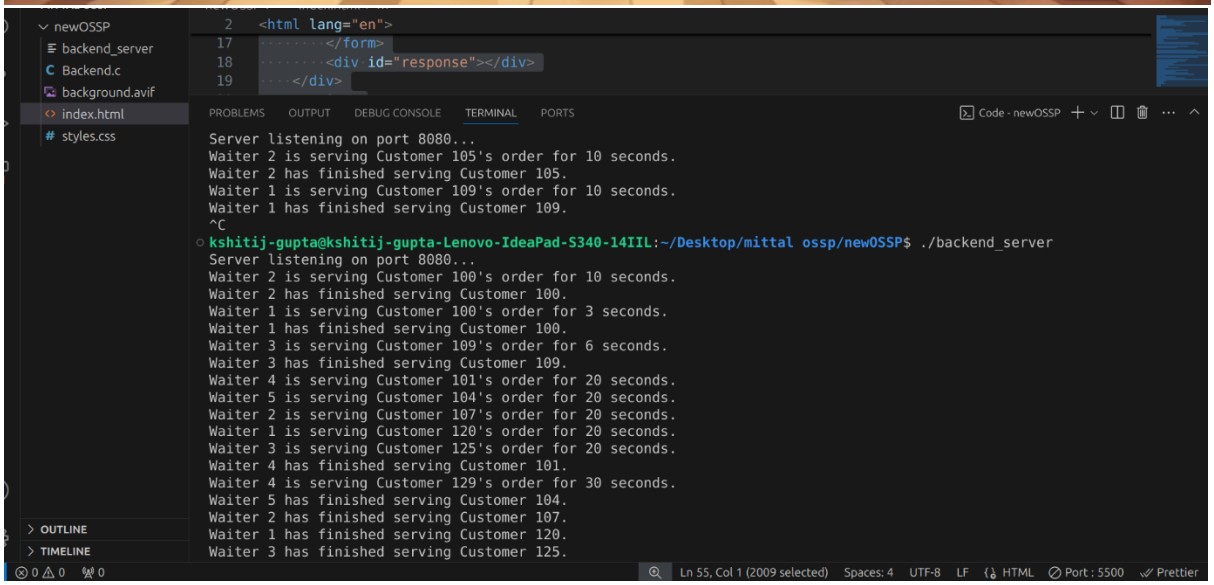
3. Integration of Backend and Frontend

3.1 Data Flow

1. The frontend sends an HTTP POST request to http://localhost:8080 containing the customer_id and serving_time.
2. The backend processes the request, validates the data, and updates the order queue.
3. The backend responds with appropriate HTTP messages:
    - Success: "Order received" or "Order placed but waiters are busy."
    - Error: "Invalid order format" or "Order queue is full."
4. The frontend receives and displays the response dynamically.

OUTPUT SCREENSHOTS





FUTURE SCOPE

The **Restaurant Order Management System** has significant potential for expansion and improvement, allowing it to evolve into a highly sophisticated and scalable platform. Below are the areas where the system can be enhanced:

1. Integration with Databases

- Implement a database layer (e.g., MySQL, PostgreSQL, or MongoDB) to store customer information, order history, and server logs.
- Enable persistent data storage, allowing restaurants to analyze past orders for decision-making and improve operational efficiency.

2. Scalability

- Dynamic Scaling: Extend the system to support a larger number of customers and waiters dynamically based on real-time load.
- Deploy the backend server to cloud platforms like AWS, Azure, or Google Cloud for global accessibility and better performance under heavy loads.

3. Advanced Resource Management

- Order Prioritization: Introduce a priority system where urgent or VIP orders are handled first.
- Dynamic Waiter Assignment: Use AI-based algorithms to assign waiters based on their availability and proximity to the order.

4. Enhanced Frontend Features

- Mobile App: Develop a mobile app version for greater accessibility.
- Interactive Dashboard: Provide real-time tracking of orders and waiters, along with analytics for managers.
- Multilingual Support: Add language options to cater to diverse customer bases.

5. Notifications and Alerts

- Integrate with SMS, email, or push notifications to inform customers about their order status or estimated wait times.
- Enable real-time alerts for waiters when a new order is ready to be served.

Conclusion

The **Restaurant Order Management System** successfully bridges the gap between efficient backend processing and intuitive frontend interaction, addressing the common challenges faced in restaurant operations. The system's multithreaded backend ensures smooth handling of concurrent orders, while the user-friendly web interface simplifies order placement for customers.

This project demonstrates how core concepts like multithreading, synchronization, and client-server communication can be applied to real-world scenarios. It also highlights the importance of integrating frontend and backend components seamlessly to provide a complete solution.

In its current form, the system:

- Manages customer orders efficiently using a robust thread-based model.
- Allocates resources (waiters) dynamically based on availability.
- Provides a responsive and engaging interface for users to interact with.

The future scope outlines significant areas for improvement and expansion, making this project a solid foundation for developing a scalable, intelligent, and versatile order management platform. With continued development, the system can transform into a comprehensive solution catering to the hospitality industry and beyond, aligning with modern technological advancements and evolving customer expectations.

REFERENCES

[1] https://www.geeksforgeeks.org/posix-threads-in-os/

[2 https://www.geeksforgeeks.org/semaphores-in-process-synchronization/

[3] https://www.javatpoint.com/multithreading-in-cpp-with-examples