JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA SECTOR-62

B.TECH. (V SEMESTER ODD 2024)

OPERATING SYSTEM AND SYSTEMS PROGRAMMING LAB

(COURSE CODE : 15B17CI472)
PROJECT REPORT

TITLE OF THE PROJECT

Synchronous Restaurant Management System

(BATCH : B8)

Submitted To :

Dr. Alka Singhal
Department of Computer Science and Engineering
Jaypee Institute of Information Technology,
Noida Sector 62

Submitted By :

Asmit Kumar Tyagi-B8-(22103227)
Ayush Sharma-B8-(22103228)
Hardik Gupta-B8-(22103242)
Prashant Kesarwani-B8-(22103240)

Abstract

The **Multithreaded Synchronous Restaurant Management System** is a C++ program simulating a restaurant environment using **operating system concepts** like threads, process synchronization, and semaphores. The system integrates clients, servers, and a priority queue for order management. It demonstrates synchronization between threads to handle concurrent requests efficiently, ensuring mutual exclusion, order priority, and resource allocation.

Introduction

In modern operating systems, **multithreading** and **process synchronization** are essential concepts for achieving concurrency and resource management. This project uses these principles to simulate a restaurant scenario where clients place orders, servers process them, and a thread-safe priority queue ensures proper synchronization.

Key features include:

- **Client threads** to place orders.
- **Server threads** to process orders.
- **Semaphores** to manage synchronization.
- **Priority queues** to handle orders based on priority.

Objectives

1. To simulate a restaurant management system using threads and synchronization techniques.
2. To implement a priority-based order queue.
3. To demonstrate mutual exclusion using semaphores.
4. To achieve concurrency in a multithreaded environment.

System Design

1. Components

- **Menu**: A collection of items with preparation and eating times.
- **Clients**: Threads representing customers placing orders.
- **Servers**: Threads handling orders based on priority.
- **Order Queue**: A priority queue ensuring high-priority orders are processed first.

2. Features

- Dynamic menu loading from a file or manual input.

- Randomized order priority for realistic simulation.
- Thread-safe communication between clients and servers.

3. Tools and Technologies

- **Programming Language**: C++
- **Libraries**:
    - <pthread.h> for thread management.
    - <semaphore.h> for synchronization.
    - <queue> for priority queue implementation.

Methodology

1. **Menu Initialization**:
    - Load menu from a file or manual input.
    - Display menu items with preparation and eating times.
2. **Order Queue**:
    - Implemented as a thread-safe priority queue.
    - Synchronization achieved using semaphores:
        - **mutex**: Ensures mutual exclusion during queue operations.
        - **full**: Tracks the number of filled slots in the queue.
        - **empty**: Tracks the number of available slots in the queue.
3. **Thread Management**:
    - Client threads:
        - Randomly assign priorities to orders.
        - Simulate eating after order completion.
    - Server threads:
        - Process orders based on priority.
        - Sleep for preparation time to simulate cooking.
4. **Synchronization**:
    - Semaphores prevent race conditions and ensure thread-safe operations.
5. **Execution Flow**:
    - Clients place orders in the queue.
    - Servers process orders, respecting their priority.
    - Threads terminate gracefully once all clients are served.

Code Implementation

Key Functions

- **Menu Class**: Loads and displays menu items.
- **OrderQueue Class**: Implements thread-safe priority queue operations.
- **Server and Client Classes**: Represent threads for handling restaurant operations.

Synchronization

- Semaphores manage critical sections, preventing simultaneous access to shared resources.

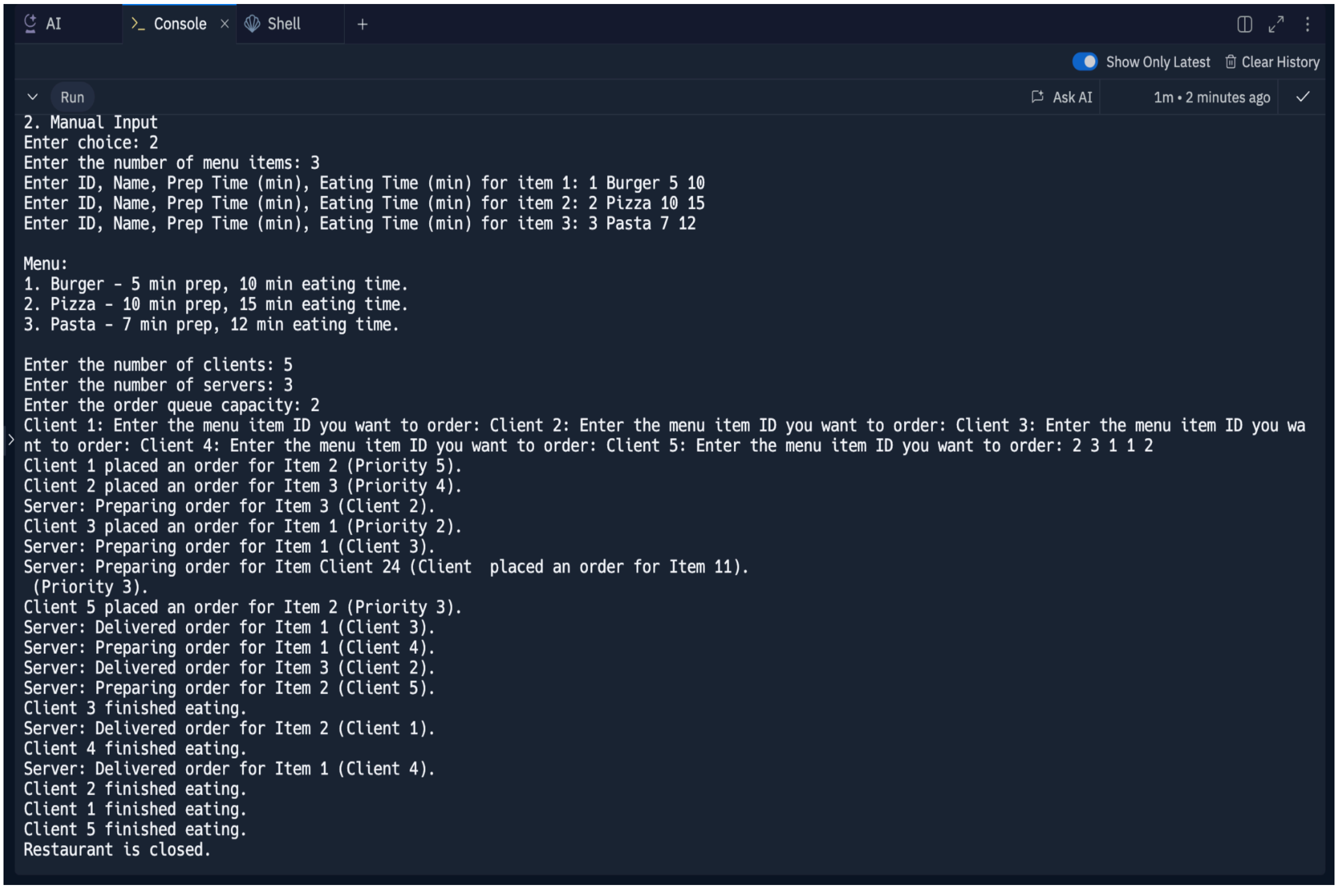
Code Snippet

sem_wait(&mutex);

orders.push(order);

sem_post(&mutex);

Screenshot



```
2. Manual Input
Enter choice: 2
Enter the number of menu items: 3
Enter ID, Name, Prep Time (min), Eating Time (min) for item 1: 1 Burger 5 10
Enter ID, Name, Prep Time (min), Eating Time (min) for item 2: 2 Pizza 10 15
Enter ID, Name, Prep Time (min), Eating Time (min) for item 3: 3 Pasta 7 12

Menu:
1. Burger - 5 min prep, 10 min eating time.
2. Pizza - 10 min prep, 15 min eating time.
3. Pasta - 7 min prep, 12 min eating time.

Enter the number of clients: 5
Enter the number of servers: 3
Enter the order queue capacity: 2
Client 1: Enter the menu item ID you want to order: Client 2: Enter the menu item ID you want to order: Client 3: Enter the menu item ID you wa
nt to order: Client 4: Enter the menu item ID you want to order: Client 5: Enter the menu item ID you want to order: 2 3 1 1 2
Client 1 placed an order for Item 2 (Priority 5).
Client 2 placed an order for Item 3 (Priority 4).
Server: Preparing order for Item 3 (Client 2).
Client 3 placed an order for Item 1 (Priority 2).
Server: Preparing order for Item 1 (Client 3).
Server: Preparing order for Item Client 24 (Client  placed an order for Item 11).
 (Priority 3).
Client 5 placed an order for Item 2 (Priority 3).
Server: Delivered order for Item 1 (Client 3).
Server: Preparing order for Item 1 (Client 4).
Server: Delivered order for Item 3 (Client 2).
Server: Preparing order for Item 2 (Client 5).
Client 3 finished eating.
Server: Delivered order for Item 2 (Client 1).
Client 4 finished eating.
Server: Delivered order for Item 1 (Client 4).
Client 2 finished eating.
Client 1 finished eating.
Client 5 finished eating.
Restaurant is closed.
```

Results

1. **Efficient Order Handling**:
   - High-priority orders processed first.
   - Clients served concurrently without deadlock or starvation.
2. **Scalability**:
   - The system accommodates varying numbers of clients, servers, and queue capacities.
3. **Synchronization**:
   - Semaphores ensured mutual exclusion and proper thread coordination.
4. **Thread Termination**:
   - All threads terminated gracefully after completing their tasks.

Conclusion

The Multithreaded Restaurant Management System demonstrates the effective use of **operating system concepts** like threads, semaphores, and synchronization to handle concurrent tasks. The project highlights the importance of priority queues and mutual exclusion in managing shared resources.

Future Scope

1. Extend the system to support real-time menu updates.
2. Add more realistic features like multiple order queues for different cuisines.
3. Implement advanced scheduling algorithms for better resource utilization.

References

1. C++ Standard Library Documentation
2. POSIX Threads Programming
3. Operating System Concepts, Silberschatz.