National University of Computer and Emerging Sciences



**Warehouse Supply Chain Management**

**Group Members**

Zain Saqib …………………….K23-0588

Tasbiha Nasir………………….K23-0528

Muhammad Uzair Hussain……K23-0638

Madiha Aslam.………...………K23-0846

**Supervised by : Anaum Hamid**

**FAST School of Computing**

**National University of Computer and Emerging Sciences**

**Karachi, Pakistan**

**30 April,2025**

**Warehouse Supply Chain Management (Producer-Consumer Problem)**

**1. Introduction:**

The purpose of this project is to simulate a real-world warehouse system that uses multithreaded programming to efficiently manage the production and completion of customer orders. In this system, the classic producer-consumer model is implemented where **producers act as suppliers** and **consumers function as retailers**. Each supplier thread is responsible for manufacturing a specific item (chair, sofa, or bed) and placing it into a shared buffer, simulating the stocking of goods in a warehouse. Retailer threads pick up these items one at a time to fulfill customer orders. Priority orders are given preference through a separate high-priority buffer, ensuring faster delivery for smaller orders. Synchronization tools like mutexes and semaphores are used to manage concurrent access to these buffers, preventing race conditions and ensuring smooth interaction between suppliers and retailers. This mirrors a real warehouse environment where multiple departments operate in parallel while competing for shared space and resources.

**2. Project Scope:**

The system accepts multiple orders, each consisting of three types of items: Chairs, Sofas, and Beds. Each order is categorized as normal or priority based on the total quantity. The project uses multiple producer and consumer threads, two separate buffers (normal and priority), and includes logging, progress tracking, and status monitoring.

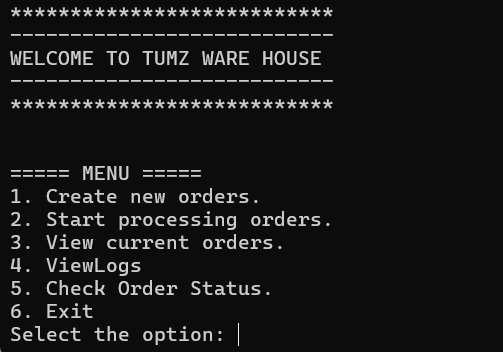
**3.Features:**

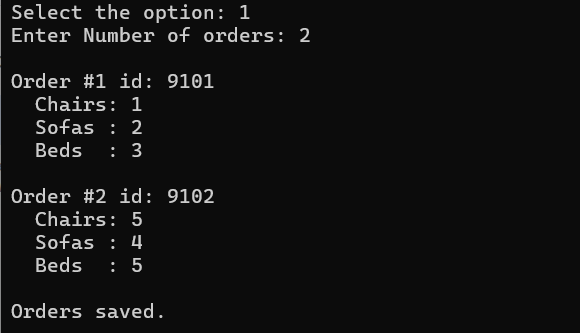
* **Multithreading**: Separate producer and consumer threads for real-time parallelism.
* **Priority Scheduling:** Orders with ≤10 total items are treated as high-priority and processed faster.
* **Synchronization:** Uses mutexes and semaphores to manage access to shared buffers safely.
* **Two Buffers**: One for normal orders, one for priority orders.
* **Logs:** Detailed order lifecycle (creation to completion) is logged in a file.
* **Status Tracking:** Live order status —Completed / Not Started.
* **Interactive Console:** Menu-driven system with order creation, processing, and logs.
* **Alert System:**alerts are triggered dynamically inside the **Producer()** and **Consumer()** functions based on current buffer conditions using **sem\_getvalue()** and real-time checks.

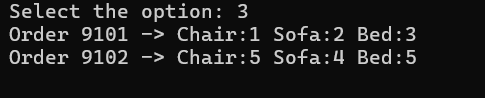
**4.** **Technology Used:**

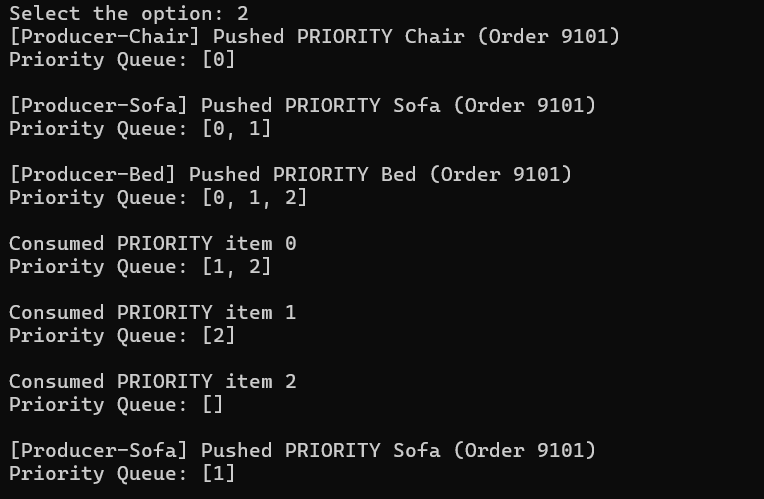
| **Category** | **Tools/Language** |
| --- | --- |
| Programming | C |
| Multithreading | pthread.h |
| Synchronization | pthread\_mutex, semaphore.h |
| Time and Logging | time.h, file I/O |
| OS Compatibility | Linux (Ubuntu) |

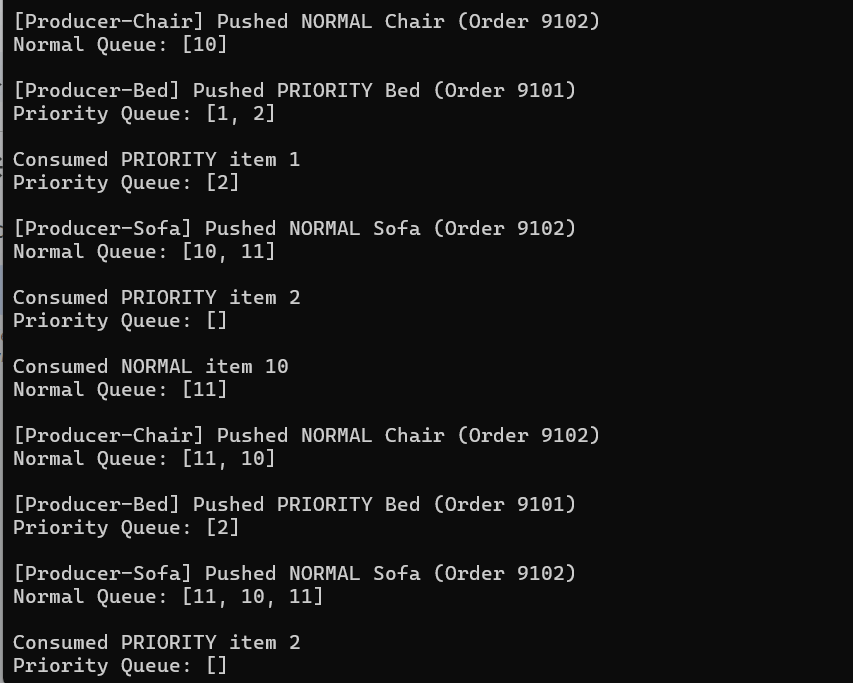
**4.** **Code Snippets:**

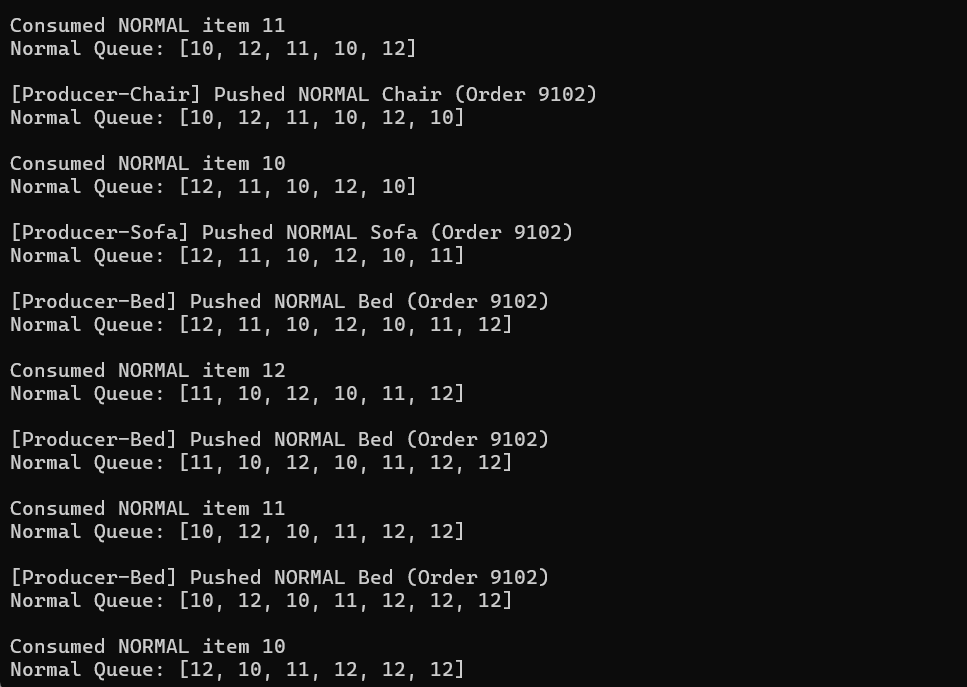
****

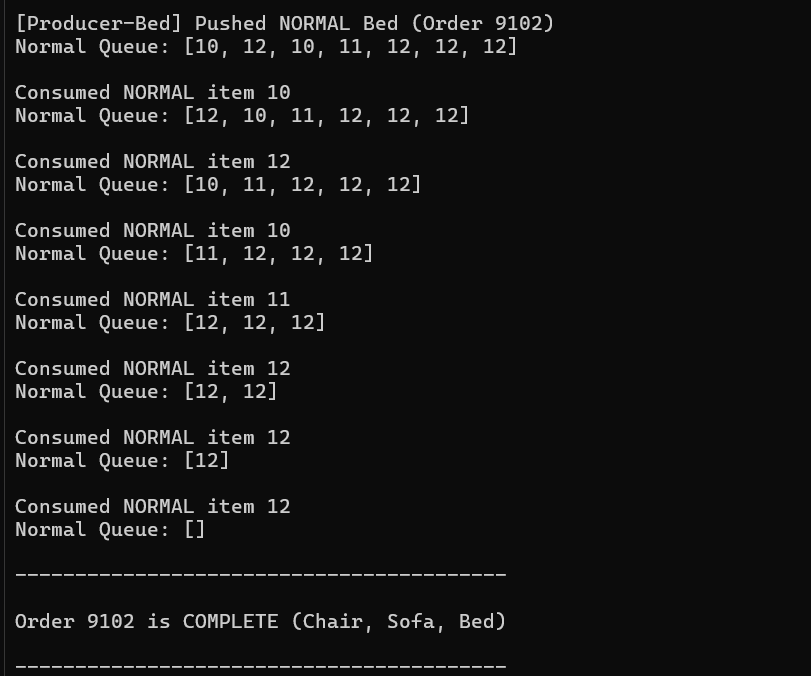
****

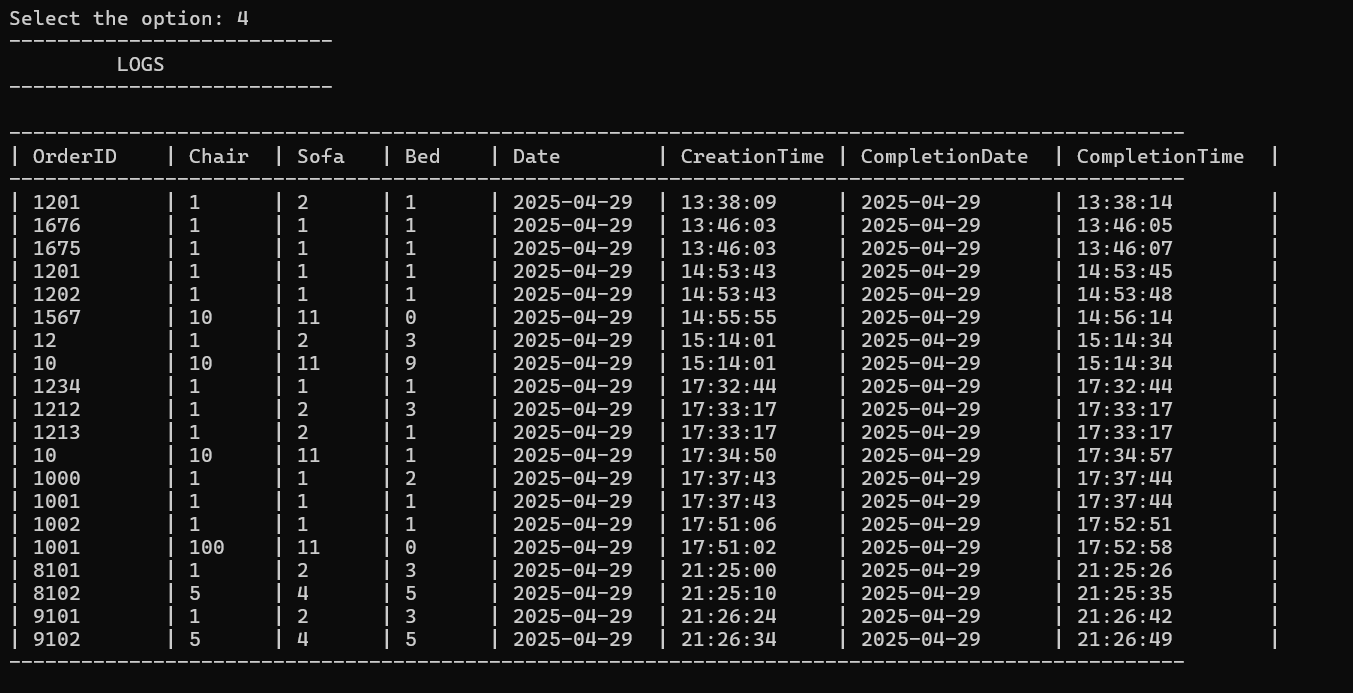
****

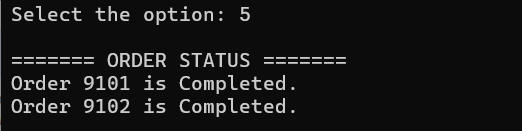
****

****

****

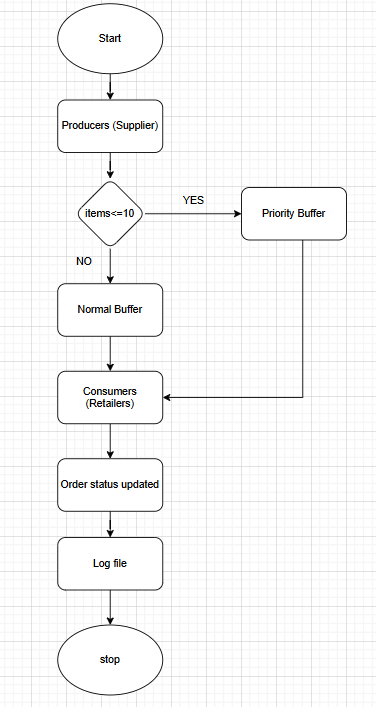
****

****

****

****

**5. Flow Chart:**

****

**6. Dry Run:**

1. Order 1 → ID: 101, Chair: 1, Sofa: 2, Bed: 1 → Priority.

(Added in Priority queue because items are less than 10).

1. Order 2 → ID: 102, Chair: 5, Sofa: 5, Bed: 5 → Normal.
2. Encodes 0 \* 10 + 0 = 0 → Chair for Order 101.
3. Pushed into Priority Queue: [0].
4. Encodes 0 \* 10 + 1 = 1 → Sofa for Order 101.
5. Pushed into Priority Queue: [0, 1].
6. Consumed item 0 → Chair (Order 101).
7. Consumed item 1 → Sofa (Order 101).
8. All items for Order 101 processed → Order 101 marked COMPLETE.
9. Encodes 1 \* 10 + 0 = 10 → Chair for Order 102.
10. Pushed into Normal Queue: [10].
11. Processes all items for Order 102 from Normal Queue.
12. Once all are matched → Order 102 marked COMPLETE.

**7. Conclusion:**

The TUMZ Warehouse project successfully demonstrates the practical application of multithreaded programming using the classic producer-consumer model. By incorporating priority handling, semaphores, mutexes, and real-time alerts, the system ensures efficient and synchronized processing of orders. The use of buffers and item-specific threads mimics a real-world warehouse with limited storage and concurrent workflows. All core features including logging, order tracking, and dynamic status updates have been implemented and tested thoroughly. The project is on track for timely completion, with no major technical obstacles remaining. Overall, it provides a strong demonstration of synchronization concepts in operating systems.