Max Khan
Andrew Li
Drhuvi Kothari
Inderpreet Kaur

CSEN 383 Project 3: Multi-Threaded Ticket Sellers

Our project implements a multi-threaded ticket sellers' simulation in C using the Pthreads library to model 10 ticket sellers (1 high priced, 3 medium priced, and 6 low priced) selling 100 concert seats in a 10x10 grid over a one hour period. Each seller manages a queue of N customers who arrive at random times between 0 and 59 minutes. The simulation ensures realistic seat assignments and tracks events (arrivals, service starts, seat assignments, completions, and turn-aways). Our design leverages Pthreads per the project requirements for concurrent seller operations, with synchronization mechanisms to prevent race conditions and ensure accurate seat allocation.

The program is structured around a main thread that controls simulation time and 10 seller threads – each managing a linked list of customers sorted by arrival time. The main thread advances time in 1-minute increments, waking all seller threads via a condition variable broadcast. Each seller thread processes customer arrival times, serves customers whose arrival time has passed, and assigns seats based on seller type (H: front to back, L: back to front, and M: middle outward). The simulation terminates after 60 minutes or when all seats are sold, with remaining customers turned away and final statistics computed.

To ensure the simulation runs realistically, we tuned several parameters:

- Number of customers (N): The command line parameter N determines the number of customers per seller, resulting in 50, 100, 150 total customers for the values 5, 10, and 15, respectively.
    - N=5 ensures all customers are likely served, testing what we would believe to be low contention.
    - N=10 approximates full capacity, balancing success and turn-aways.
    - N=15 creates high resource contention, with many customers turned away – mimicking what would seem like an oversubscribed event. We believe that this would be most realistic of the three values to simulate a variety of scenarios while engaging in ticket sales in the real world.
- Customer arrival times: Each customer's arrival time is randomly generated between 0 and 59 minutes. This uniform distribution simulates unpredictable customer arrivals throughout the hour-long simulation and does a decent job at mimicking real-world ticket sales where customers arrive sporadically. Perhaps we would have chosen a more complex randomization process including a mix of normal and Poisson distributions but opted not to in this project.
- Service times: Service times are randomized to reflect different customer interactions.
    - High priced tickets take between 1 and 2 minutes randomly.
    - Medium priced tickets take between 2 and 4 minutes randomly.
    - Low priced tickets take between 4 and 7 minutes randomly.
- Simulation time: Time is modeled in 1-minute increments over 60 minutes, with each minute triggering a condition variable broadcast to wake seller threads. This granular time unit balances computational efficiency with realistic timing.

The simulation involves shared data accessed by multiple seller threads. To prevent race conditions from occurring, we used locks – which more specifically comes from the seat_mutex variable in our code. The key shared data and their critical regions are:

- Seat matrix: A 10x10 array tracking seat status (0=unsold, 1=sold).
- Seat owners: A 10x10 array storing customer IDs (e.g., H001, M101) for assigned seats.
- Total seats sold: A counter tracking the number of sold seats, used to detect if we have sold out of tickets.
- Simulation time: Keeps track of the current time ranging from 0-59.
- Sold-out flag: A flag set to 1 when all seats are sold out, signaling threads to exit.

We also needed robust synchronization to manage concurrent access to shared data and coordinate thread execution. We used the following mechanisms:

- Mutex for seat assignment (seat_mutex).
    - A pthread_mutex_t protects the critical region in assign_seat, ensuring that only one seller thread can access or modify seats, seat_owners, total_seats_sold, and sold_out at a time. This prevents race conditions, such as two sellers assigning the same seat or incorrectly counting sold seats.
- Condition variable for time synchronization (cond).
    - A pthread_cond_t synchronizes seller threads with the simulation clock. The main thread advances current_time each minute and calls wakeup_all_seller_threads, which uses pthread_cond_broadcast to wake up all seller threads waiting on pthread_cond_wait.
    - This ensures all sellers process the current minute's events simultaneaously, while maintaining a consistent simulation timeline.
- Seller-specific customer queues.
    - Each seller maintains a private linked list of customers, sorted by arrival time. Since each seller thread exclusively manages is own queue, no additional synchronization is needed for queue operations.
- Thread termination.
    - Seller threads exit when current_time reaches 60 or sold_out is set, coordinated via the condition variable broadcast.

In total, the synchronization methods we chose helped ensure thread-safe access to our well-defined shared resources, prevent seat double booking, and maintain a relatively realistic simulation. The use of mutexes for critical regions and condition variables for time coordination aligned with Pthread best practices (that we found online), which balanced concurrency and correctness.

Simulation results for N=5:

Average Response Time (H)	0.00 Minutes
Average Response Time (M)	0.20 Minutes
Average Response Time (L)	1.31 Minutes
Average Turnaround Time (H)	2.00 Minutes
Average Turnaround Time (M)	3.20 Minutes
Average Turnaround Time (L)	6.31 Minutes
Throughput (H)	0.08 Customers/Minute
Throughput (M)	0.25 Customers/Minute
Throughput (L)	0.48 Customers/Minute

Simulation results for N=10:

Average Response Time (H)	0.30 Minutes
Average Response Time (M)	0.66 Minutes
Average Response Time (L)	4.51 Minutes
Average Turnaround Time (H)	1.70 Minutes
Average Turnaround Time (M)	3.59 Minutes
Average Turnaround Time (L)	10.19 Minutes
Throughput (H)	0.17 Customers/Minute
Throughput (M)	0.48 Customers/Minute
Throughput (L)	0.88 Customers/Minute

Simulation results for N=15:

Average Response Time (H)	0.29 Minutes
Average Response Time (M)	1.46 Minutes
Average Response Time (L)	7.88 Minutes
Average Turnaround Time (H)	1.71 Minutes
Average Turnaround Time (M)	4.23 Minutes

Average Turnaround Time (L)	13.53 Minutes
Throughput (H)	0.23 Customers/Minute
Throughput (M)	0.58 Customers/Minute
Throughput (L)	0.85 Customers/Minute