SJSU FA 18
CS 149

Project 3 Multithreading
Group 2
Ranying Cai, Tianyun Chen, Roger Kuo, Sijing Xie

This project is a simulation of multi-threaded ticket selling, where each thread is represented as a ticket seller of high, middle, or low pricing. In the concert hall of the simulation, there is a total of 100 seat tickets to be sold to a specified number of buyers who arrive at random times within a 60-minute window. Each ticket purchase will also be given a random time for completion, and the range of the time is dependent on the seller type.

To initialize the simulation, we begin by representing all 100 seats in a global 10-by-10 2D string array, to which all 10 seller threads have access. This array is the shared information which the mutex locks and unlocks as the program switches between threads. As seats are assigned, each element in the array is filled with its respective buyer's name. We use priority queues to hold each seller's line of customers to simplify the sorting process, and the customers in each queue are initialized within a seller's thread. To simplify tracking a customer's information, we defined a Customer class to hold a customer's ID, arrival time, and completion time. Additionally, we defined a Compare class which is passed into a priority queue to compare buyers and sort them by arrival time.

The `main` function prompts command line input to define the number of customers per queue. After initializing a pthread mutex and condition, it creates all 10 seller threads and begins their start routines. Once all the threads have completed their work, the main function joins all the threads back into the main thread, prints the final seating chart, and exits the program.

The start routine for each thread, `eachSeller`, performs all of the ticket transactions and seat assignments, thus being the critical region. Pthread locking, unlocking, broadcasting, and condition waiting all occur within the while loop of this function and allow seats to be assigned by only one thread at a time, though the actual assignments are completed via multiple helper functions. Process synchronization takes place here, as the seating chart is being accessed concurrently by multiple threads. The pthread functions help maintain data consistency and ensures synchronized execution of simultaneous processes. The start routine begins by generating the customer queue for its thread and initializing the current time to zero. Within an hour margin, it loops through the queue to update the current time, lock the seat array to perform transactions, unlocking the array after sale completion, and report the arrival and exit time of each customer and the status of the sale. Finally, the thread is closed and control is returned to the `main` function.

Though the simulation is in a controlled environment, we had to adjust certain parameters for it to run more realistically. The customers' arrival times were randomized values set within an hour. Each seller type was also given a range of completion time for selling a ticket, simulating the unpredictable time of interaction between a buyer and seller. Though the number of customers per seller is a static variable defined by the user, its role in creating a realistic simulation is minimal.