Nicolas Seto

Vinh Doan

Parth Vyas

CS 149 – Hw 3 Report

Note: We implemented the extra credit.

Designing this program took quite some time. At first, we were thinking about how to have a separate queue that is initialized within each seller; however, with this implementation, we could not figure out how to have each customer add themselves to the queues if they do not have access to them in the first place. In the end, most of the shared data structures and variables are global.

We knew around which areas the critical regions were located, but we were unsure of where exactly to begin locking and unlocking mutexes; this took some time. For example, there was one hiccup in our design regarding the seller’s access to queues as we were testing our program. In the beginning, we had it so each seller thread locked the particular mutex and then went to sleep. This prevented other sellers from proceeding through the ticket sale because there was still a lock on the mutex. We had to unlock the mutex before sleeping, and we re-locked it after sleeping. In this sense, the sellers who work faster (or sleep less), will regain access to the mutex faster in order to process the ticket sale.

The following are the data that was shared between sellers and customers: queues, the leave count, and the printing. The queues were shared between both sellers and customers because the customers had to know whether they were processed and given a seat or turned away. When a customer added themselves to a queue, they placed their name (like H001) in the queue. If the customer was processed by the seller or left on his/her own accord, the element’s value would be changed to 1, so that the seller would know if the customer left already due to the 10+ minute waiting policy. The areas where the seats and events were printed are critical regions because the results would be out of order otherwise.

The following are the data that was shared between the sellers: an array of indices that indicate the next customer in line, the seats that hold all the customers, a seats taken count, and an array that consists of the number of customers given a seat based on which seller type.

The design of the program is as follows:

1. The seller threads are created. When the last seller is created, the sale period starts.
2. The customer threads are created and sleep a random number of minutes before adding themselves to the queues. The queue size of each seller is recorded so that each customer knows where to insert themselves in the queue. A customer is added to the queue and increments the queue size. The actual queue itself holds the names of the customers.
3. Each seller records which customer is next in line. All sellers begin at index 0. If there is an empty string at index i, then there is no customer waiting in the queue. If there is a 1 at index i, then the customer has already left. If there is a customer, the seller checks to see if there are open seats.
   1. If there are no seats, the leave count is incremented, and the element’s value at index i is changed to 1 to indicate that the customer has been processed.
   2. If there are open seats, seat the customer. Increment the seats taken count and the seller’s count of how many customers it has processed.

Since the customer has been processed, go to index i + 1 to serve the next customer.

1. For each customer, if time’s up or the wait has been 10 minutes or longer, stop waiting. Check the queue to see if they have been processed or not.
   1. If the value at the respective index is 1, then the customer has been processed. Do nothing.
   2. If the value at the respective index is still the customer’s name, then the customer has not been processed. Increment the leave count and print out the correct event: either the time has run out, or the customer waited 10 minutes or longer.