CSEN 383

Advanced Operating System

Project 3 Report

Group 5
Vibha Nayak, Parth Negi, Dhanshree Narendra Pajankar, Nilkumar Patel

We developed our project using the boilerplate code that was included in the project preview. On top of that, we used the main thread to simulate a clock tick and the child threads meant for simulating a ticket sale to control the critical region and sell process.

In order to simulate a minute of time, we used the main thread in our project to generate a clock tick every minute. For the simulation, we made the following assumptions:

1. ClockTick: Every child's thread and the lowest measurable time quantity of one minute have been designed to simulate tasks that require one minute. Serving customers, holding off on making a sale, or closing the deal are a few examples for it.

2. Seller's thread state: At any given time, each seller's thread is presumed to be in one of the following states.
▪ Waiting: Waiting for new customer
▪ Serving: Serving new customer from the seller's queue
▪ Processing: Processing and taking time to process the sell
▪ Completing: Completing sell for the customer

3. To preserve time synchronization, a new clock is generated once each seller's thread has completed their work for that time quanta.

4. To prevent seat assignment conflicts, a two-dimensional matrix has been used to simulate concert seating. It is assumed that only one thread will be modifying the matrix.

The project's overall workflow looks like this:

1. Initialization: We have set up the concert seat matrix, lock mutex, customer queue for every seller, and threads.

2. After seller threads were created, all of the threads were initialized and then placed in waiting for clock tick mode.

3. Simulating the Clock Tick: This was one of the more difficult parts because we had to make sure that the clock tick signal would only be sent once every thread had completed its task for that particular time quanta. By keeping track of the number of threads that are still running, we have managed to make the main thread wait.

4. Depending on their current states, each seller's thread races to obtain a lock on the concert seat matrix when the main thread sends a clock tick.

5. A seller thread first looks for a new customer based on their arrival time before beginning to serve one at a time.

6. We have to maintain the time delay and reduce it with each clock tick in order to simulate a delay in finishing sales. A random time delay is created when a new customer arrives. We finish selling that thread when the time delay reaches zero.

7. Termination Condition: If the concert is sold out or the simulation time runs out, each seller thread will stop processing. The consumer must depart in both scenarios.

It should be noted that the project's design makes sure that seat assignments are serialized while real processing occurs concurrently. Thus, it's feasible that all ten sellers are handling client requests concurrently. The output makes this evident since it shows that up to 8 clients are being served concurrently at one point.