Rochester Institute of Technology

SWEN - 563

Project 4

Devon Bagley

[dxb4606@rit.edu](mailto:dxb4606@rit.edu)

Edward Maskelony

[exm8563@rit.edu](mailto:exm8563@rit.edu)

11/13/2017

**Project 2 Overview:**

Design and implement an embedded, stand-alone QNX Neutrino program to simulate the workflow in a typical banking environment -- single queue with queuing to a multi-threaded server or three separate server processes.

**Area of Focus:**

Devon Bagley - initial design / creation of metric functions / Help with teller and queue creation/functions

Edward Maskelony - Qnx system clock functions / Queue and teller interactions / displaying metrics

**Design/Analysis:**

* Timer
  + The QNX system clock was used to track the simulation time. The clock runs in realtime at a nanosecond resolution, but the simulation runs at a resolution of 100ms. To obtain times during the simulation, the system clock was read exclusively using a function that takes the system timespec (a struct containing seconds and nanoseconds) and converts it to an int that corresponds to the simulation time in 30-second “ticks.” At the start of the simulation, the system clock is set to the equivalent of 9AM in simulation time.
* Customers
  + These customers are structures that contain their queue entry time and their transaction start time. With these values we can calculate their total waiting time in the queue, and also the time spent with tellers based on the qnx system clock.
* Queue
  + The queue is made up of a list of customers, 2 semaphores, a mutex, and attributes of the queue. The queue knows its depth, front position, and rear position. The mutex prevents the queue from being modified by multiple threads at once; the queue elements are accessed exclusively with “enter” and “leave” functions; both of these functions lock the mutex before modifying the queue and unlock the mutex when it’s done. One semaphore is used to stop the main thread from adding customers to the queue when it is full, and the other semaphore stops teller threads from dequeueing customers from an empty queue.
* Metrics
  + The metrics is a struct used to track, collect, and display the metrics of the program. Using the total number of customers and the total values of customer wait time and transaction time we can calculate the average customer metrics. The maximum metrics will all be updated upon either a customer entering the queue, leaving the queue, or finishing a transaction. Each time a metric is updated by a teller, the teller will need to lock a mutex on the shared information in the queue. This mutex will keep the tellers from interfering with each other.
* Tellers
  + The tellers are 3 separate threads with the same attributes. They have an id, a timestamp to track how long they spend waiting for the next customer, and pointers to both the queue and metric structs. The tellers will be waiting for an available customer represented by the semaphore. Once the semaphore is updated one teller will grab the customer and update the corresponding metrics. This is when the teller will perform the customer’s random transaction time before looking for a new customer. Once done with the transaction they update the corresponding metrics and look for another customer. The teller tracks the time between finishing a transaction and beginning another to calculate their waiting time.

**Test Plan:**

For this bank simulation we are looking to simulate the concurrency of bank transactions using threading. We were looking for a very small amount of waiting time for the customers and a slightly larger wait time for the tellers to ensure that the flow of the program is constant and not being hung up by a single teller. The average times we calculate are all around the expected average time based on the maximum and minimum random values for a transaction and customer entry. We ensure that each teller only interacts with one customer at a time and updates the metrics without interference from other tellers. We also tested various versions of the bank with different number of tellers, different intervals for random entry time, and different intervals for random transaction time to ensure our simulation tracks and displays the desired effect.

**Results:**

The bank is opened at 9:00 AM and customers all enter at random intervals and placed in the queue successfully. The teller threads wait until a customer is available and perform all the necessary metric updates. The teller then waits the random transaction time to deal with the customer and then further updates the metrics. The bank closes at the correct time of 4:00 PM and the tellers complete the remaining customers left over in the queue for the day. The calculation for the average times are done and the metrics are then displayed.

Sample output:

Total customers: 188

Average customer wait time: 00:00:00

Average transaction time: 00:03:30

Average teller wait time: 00:02:30

Maximum customer wait time: 00:02:00

Maximum teller wait time: 00:08:00

Maximum transaction time: 00:07:30

Maximum queue depth: 2

**Lesson Learned:**

This project took more initial design and though before we were able to build our bank. When putting together the bank we would need to have almost all of the interactions fully thought through before being able to build the project. This combined with little experience with semaphores and mutexes within qnx lead to us spending much more time working on this project than expected. The use of the system clock was also a new concept that took extra time to understand and implement correctly. In other projects for microcontrollers, we had more control over the resolution and speed of our timers. Using the QNX system time was reliable once we started working with it correctly, but it was less flexible than other timers.