SWEN 563

Real Time & Embedded Systems

Project 4

Christopher Broderick *cjb9317@rit.edu*

Muhammad Adib Yahaya *my3197@rit.edu*

March 27, 2018

Overview

QNX is a real-time operating system. Using QNX and Diamond Systems Helios single board computer, we implemented a program that simulates the workflow in a typical banking environment. There are one queue for customers and three tellers to serve customers. The queue and tellers are all running in separate processes. This means there are four threads are running concurrently to queue customers and to serve customers.

Areas of Focus

Muhammad:

1. Create the threads.
2. Create the queue thread function.
3. Create the teller thread function.
4. Create the teller struct to store data.
5. Make program runs at simulation clock time.
6. Create a delay when customer enters the queue and when teller is doing transaction with a customer.
7. Get the time difference between when the customer enters the queue and when it gets to a teller.
8. Lock mutexes when a resource needs to be protected.
9. Create metric functions for sign offs (1, 2, 5, 7)

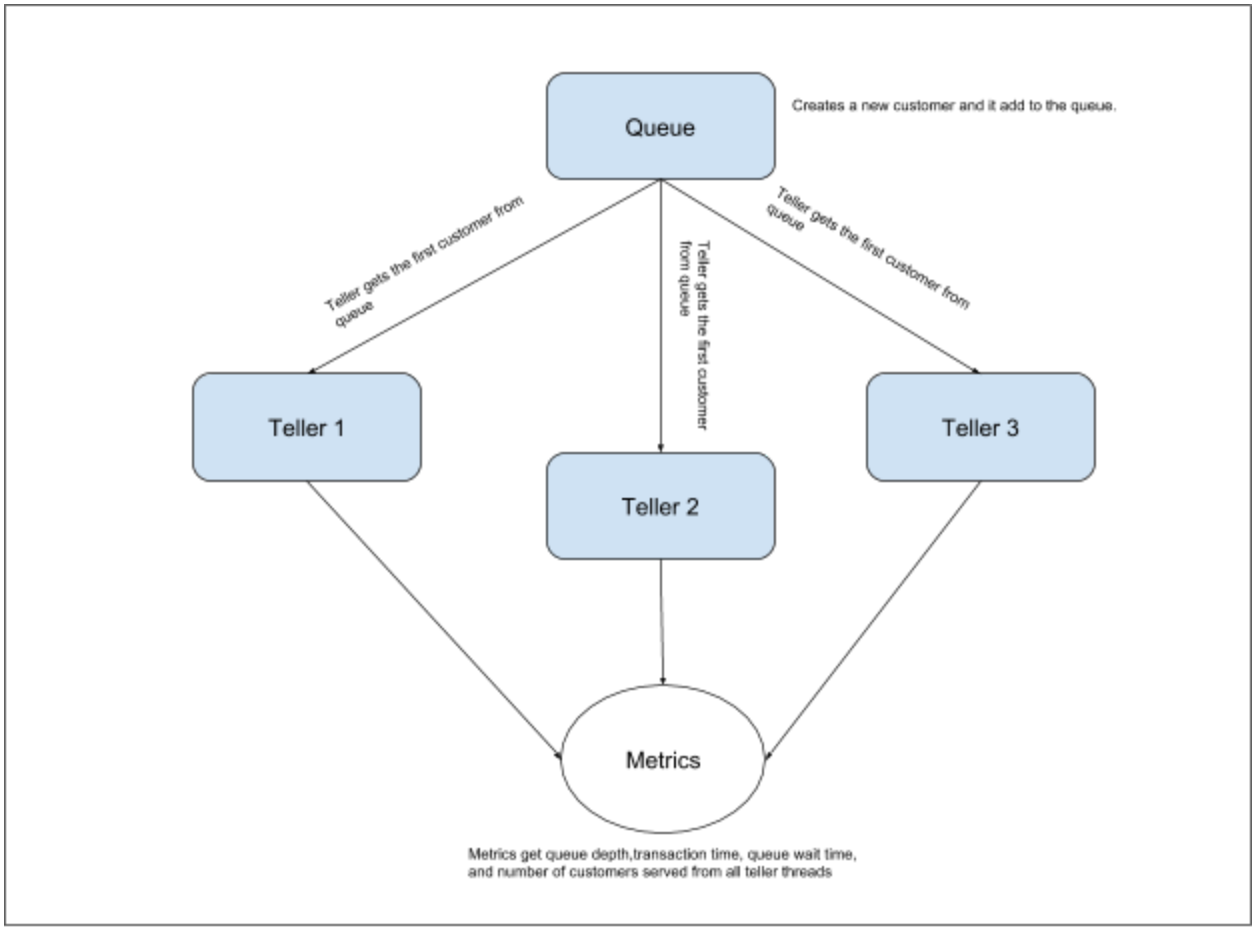
Chris:

1. Create queue stack.
2. Create queue methods.
3. Create the random number generator for customer arrival time and teller transaction time.
4. Create metric functions for sign offs (3, 6, 4, 8, 9)

Analysis / Design

1. To begin a function was made to create the threads to be used for the queue and 3 tellers. The separation into 4 threads allowed the program to take in customers into the queue while simultaneously having 3 tellers pull from the queue uninterrupted which simulated a more real life situation.
2. The queue and its supporting files were created first because each teller needed a "line"(queue) to pull from. A structure named "queue_t" was created to record the actual array of values, number of spots available in the queue, number of customers in the queue, first customer in queue, and last customer in queue. Using a structure seemed to be the easiest way to store these value that could now be called on to either be changed or used in other functions. The queue files contained the functions for enqueueing (taking in customers), dequeueing (removing customers), and two function to get the rear (last person in line)/front (first person in line).
3. The teller and its supporting files were created next. Similarly to the queue, a structure named "teller_t" was made to record the information required at the teller. First a function is used to initialize a structure values for each teller so that they can have their own independent information. With this newly created information the tellers had a function that would take in a customer's time they enter the queue and which teller is taking the customer. With this information the teller run function calculated the metrics needed those being time waited in queue, how long the teller waited for a customer, and determine depth of the queue.

4. Lastly the metrics files were created to get the information needed for the demo. This task was simply done by taking in values already recorded in the teller and queue files. Global values were created for all max values needed, all average values needed, and total values needed to compute averages. Each set of data had its own function that would compare a value from each customer to be compared against the global max to see if it was greater than it. If it was greater the max was set to this new value. The value taken in was also added to a total that would be used in another function that would divide the total by the amount of customers served in the work day. These functions were then placed throughout the code in the correct place to get the values needed for comparison and averaging. This allowed a simply and easy way to get the metrics that needed to be printed for the lab demo.



Block diagram of the program showing how each teller thread communicates with the queue thread, and how to get the statistics.

Test Plan

1. Create a thread for each process - queue, teller 1, teller 2, and teller 3.
2. Lock mutex when a resource is shared between multiple threads, and unlock it immediately after finishing to avoid mutex deadlock or other problems.
3. Make the program run for 42 seconds to simulate 7 hours of business time.
4. Create a random number generator that can generate uniformly distributed numbers for customers' arrival time and transaction time.
    a. Generated numbers for customer arrival time to enter the queue should be between one to four minutes.
    b. Generated numbers for customer transaction time with a teller should be between 30 seconds to 8 minutes.
5. Get the current clock time when a customer enters the queue and when a customer meets a teller. This is to calculate the amount of time a customer waited in the queue before getting served by a teller.
6. Get the current clock time when a teller is done serving a customer. This is to calculate the amount of time a teller waited for a customer.
7. Create a queue stack for the customers to come in. The queue stack should have methods to add and remove a customer, and to get the first customer in the queue.
8. Create methods to get the required metrics and print them to the console.

Project Results

Below is the result metrics when the program was ran for 42 seconds or 7 hours in simulation time. The time unit shown below is in simulation minutes.

```
1. The total number of customers serviced during the day      : 160
2. Customers served by Teller 1, 2, and 3 respectively        :
   Teller1 = 53 Teller2 = 55 Teller3 = 52
3. The average time each customer spends waiting in the queue : 0.080000
4. The average time each customer spends with the teller      : 4.030000
5. The average time tellers wait for customers                : 6.750000
6. The maximum customer wait time in the queue                : 3.980000
7. The maximum wait time for tellers waiting for customers    : 13.360000
8. The maximum transaction time for the tellers               : 8.000000
9. The maximum depth of the customer queue                    : 1
```

All results were within the expected values. The number of customers served by each teller was uniformly distributed. This shows each thread was running concurrently with one another.

The average time customers spend waiting in the queue and the maximum queue depth were very low because when a teller thread is busy, another teller thread would be available before other threads finish their transactions.

The maximum transaction time for the tellers was 8 minutes, which is also the maximum time a teller should interact with a customer.

Lessons Learned

1. Learned how to create a thread in QNX real time operating system.
2. Learned how to properly use mutex lock/unlock to protect a shared resource.
    a. We had a problem understanding mutex lock. We thought mutex should be locked at the start of a thread. That caused the threads to not run simultaneously due to the queue stack is being locked by a single thread and could not be accessed by other threads.
3. Learned more about function pointers and how to use them effectively.
4. Learned more about pointers and how to pass arguments to a function effectively.
5. Learned more about passing by reference.
6. Learned how to use clock and timer in a POSIX-compliant OS.