COMP2006 - OPERATING SYSTEMS
CUSTOMER QUEUE

Jacob Arvino

19817082

Contents

- **Synchronization 3**
- **Errors 4**
- **Demonstration 3**

Synchronization

Synchronization has been achieved in the customer queue function through the use mutexes, specifically using the pthreads library. Two kinds of 'mutex locks' have been used in the program: simple mutex lock and a broadcast mutex lock.

Firstly, the simple mutex lock has been used to protect the tellerNameArray and customerServedArray. This mutex works on a simple pthread_mutex_lock/unlock call, where a thread will acquire the mutex lock before inserting the necessary data into the arrays. This is important as a shared variable, *threadsDone*, is used as the array index. If two threads were to concurrently increment this variable, it would throw the array reference out of order and try to access memory not allocated to the array, resulting in a segmentation fault.

Secondly, the broadcast mutex lock is used across both the customer() and teller() functions. It specifically uses the broadcast function over the signal, due to at any one point there may be 1 or more threads waiting for the mutex. This occurs because of the 1-to-many natures of how the program runs. You have 1 thread moving customers into the queue and 4 threads moving customers out of the queue.

To achieve synchronization the traditional pthread_mutex_lock/unlock is used, however once a function acquires the mutex lock, it must ensure no other function is accessing the shared variable, *c_queue*. To assist in this, pthread_cond_signal is used in the writer function, *customer()*, to alert sleeping threads that it has finished writing to c_queue and that one of the threads is able to remove a customer from it. On the contrary, the reader function, *teller()*, acquires the mutex but still has to wait, symbolised by the pthread_cond_wait function, for the writer function to be done with the shared variable.

Errors

The major error within the program is that it doesn't write to a file. The issue I had with implementing it was getting synchronization between the threads that needed to write to the file. This was an issue as I couldn't get the mutex block of code to work around the writer functions. Each time I tried to set it up it resulted in a deadlock of the program, so it has been omitted.

I have included the code I wrote in an attempt to implement it, but it has been commented out and has no effect of the programs execution.

To make up for lack of output, I have instead printed to the terminal screen what I would have written to the "r_log" file.

Demonstration

After compiling the program with the *make* command, input the following terminal command:

```
Jacobs-MacBook-Air:OS Assignment jacobarvino$ ./cq c_file.txt 1 2 4 6
```

The program will begin to execute:

```
------------------------
 Customer number 1 : W
 Arrival time: 13:25:29
------------------------

------------------------
 Customer number 2 : I
 Arrival time: 13:25:30
------------------------

        -------------------------
        Teller 1
        Customer Number: 1
        Arrival time: 13:25:29
        Response time: 13:25:30
        -------------------------

------------------------
 Customer number 3 : D
 Arrival time: 13:25:31
------------------------
```

Note:

The difference in arrival times is 1 second, as specified in the command line.

For the sake of the demonstration, we will follow customer number 47:

```
------------------------
 Customer number 47 : W
 Arrival time: 13:26:15
------------------------
```

After being moved into *c_queue*, customer 47 is served by teller 3 after waiting 8 seconds in *c_queue*

```
------------------------
Teller 3
Customer Number: 47
Arrival time: 13:26:15
Response time: 13:26:23
------------------------
```

Customer 47 has requested for a withdrawal, which we have specified in the command-line, takes 2 seconds to complete:

```
------------------------
Teller 3:
Customer Number: 47
Arrival time: 13:26:15
Completion time: 13:26:25
------------------------
```

NOTE: The 2 second completion time is the time between response time and completion time.

Now that customer 47 has been served, teller 3 moves onto the next customer. We will follow teller 3 for the rest of the demonstration:

```
------------------------
Teller 3
Customer Number: 49
Arrival time: 13:26:17
Response time: 13:26:26
------------------------
```

Teller 3 continues to serve customers until *c_queue* is depleted, upon which teller 3 will display his termination message:

```
------------------------
Termination: Teller 3
Served: 26 customers
Start time: 13:25:29
Finish time: 13:27:29
------------------------
```

Once all tellers have terminated, the final customers served for each teller and total customers served are displayed:

```
Teller statistics:
Teller 4 serves 26 customers
Teller 1 serves 24 customers
Teller 3 serves 26 customers
Teller 2 serves 24 customers

Total customers served: 100
```