**Report - COMP2006 - Operating Systems – Assignment**

**Thilina Prasad Athukorala**

**ID: 21038875**

## README

1. To be sure that the program is compiled in the relevant system first give command “make clean” and recompile using the command “make”.

2. Type the run command as “./execute<space><command line arguments>”

   Command line arguments in the order of insertion is as follows

```
<File name to execute>  <Size of customer queue>  <Periodic time for adding customers>  <Time duration to serve withdrawal>  <Time duration to serve deposit>  <Time duration to give information>
```

   Ex:

```
./execute c_file4.txt 5 1 3 4 5
```

3. Give a proper file name including the extension as shown above.

4. Check r_log.txt file for the output log.

5. Note that no output is shown in the terminal until the whole process is completed and all the functions will be logged.

## Core implementations

- First-in First-out (FIFO) customer queue and discussion on how mutual exclusion is achived when accessing the queue.
- customer(), adddToQueue() functions which periodically gets a customer from the c_file and puts the information in c_queue ().
- adddToQueue() function log whenever it receives a customer.
- A function tellerProcess() is written that simulates the operations of each teller, and four threads are created each of which runs function teller().
- logStatus() function logs to r_log.txt whenever it receives a customer and completes a service.
- Four tellers are waiting for customers and starts when the queue is full initially.
- Three services are provided by each teller: Cash-withdraw, Cash-deposit, and Provide-information.
- All tellers terminate when the queue is empty.
- All activities of the queue and the tellers are recorded in a file named r_log.txt and mutual exclusion is achieved when accessing the r_log.txt shared memory.
- All mutex locks are stored in mutexLocks.h and initiated in mutexLocks.c so that thay can be accesed by any file.

- **First-in First-out (FIFO) customer queue and discussion on how mutual exclusion is achived.**

queue is located in queue.c such that user is able to define the queue size by specifying it with a command line input value denoted as 'm' which is the second argument. It stores customer struct variables. No customers will be added further after the queue has reached maximum number of customers until a teller serves at least one customer. This process is handled by calling the below condition (by addToQueue() method located in readFile.c) in a while loop when queue is full so that no more customers are added into the queue. Before any of these conditions are triggered queue acquires the lock 'queue_mutex' nad releases when the relevant is inserted.

```
pthread_mutex_lock(&queue_mutex);

 while (queue->queueCount==userQueueSize) {

        pthread_cond_wait(&queue_not_full, &queue_mutex);
    }
```

when atlest one customer is removed by using removeFromQueue() method located in main.c then the signal confirming queue is not full

```
    pthread_cond_signal(&queue_not_full);
  pthread_mutex_unlock(&queue_mutex);
```

is sent to addToQueue() method(located in readFile.c) to initiate addition of customers to queue again releasing 'queue_mutex' lock.

In the same manner when queue is empty after serving all the customers (if tellers have served all the customers before arrival of a new customer) then below condition is looped in removeFromQueue() method located in main.c

```
   while (queue->queueCount==0) {

        pthread_cond_wait(&queue_not_empty, &queue_mutex);

   }
```

And even if one customer is added to queue by addToQueue() method(located in readFile.c) it signals the removeFromQueue() method to initiate it's process if any of it is waiting releasing the 'queue_mutex' lock.

```
   pthread_cond_signal(&queue_not_empty);
pthread_mutex_unlock(&queue_mutex);
```

queue is accessed during 2 instances when one customer is considered

- When a new customer is added to queue.
- When a customer is removed from the queue.

**In each of these instance's 'queue_mutex' lock is used to satisfy mutual exclusion so that no 2 above functions/instances use queue simultaneously.**

- **customer(), adddToQueue() functions**

Customer function located in readfile.c gets a customer periodically into the queue according to the time given as a command line argument (3rd argument) defined as $t_c$ in the specification. This process occurs by a sleep function called after addition of each customer to queue by giving the input to it as the customer defined value in the sub function created for customer addition (located in readFile.c).

```
void addToQueue(int customer_num,char service_type,  Queue* queue, int
userQueueSize, int customerInPeriod)
```

```
sleep(customerInPeriod);
```

- **<u>adddToQueue() function log whenever it receives a customer.</u>**

As soon as a customer is added to queue **adddToQueue()** method aquires the mutex lock (log_mutex) and logs the arrival time in the following format - was extracted from the log file created in test run  and accurate to best of my knowledge

```
-----------------------------------------------------------------
Customer 14: W
Arrival time: 11:48:48
-----------------------------------------------------------------
```

Which then releases the lock allowing any other function which access the same memory location (shared).

- **tellerProcess() function**

tellerProcess() function is the core function used to create the 4 threads used in serving customers which technically drives the programme to be successful. This function simply checks which service is required to be fulfilled and, in this simulation, it uses sleep function as the service fulfill time and leaves the function after it's done. All arguments given by user as the time duration for each service to be fulfilled is stored in Teller struct located in tellerProcess.h as shown below

```
typedef struct Teller {
  Customer* customer;
  char tellerName[20];
  int withdrawalDuration;
  int depositDuration;
  int informationDuration;



}Teller;
```

- **logStatus () function log**

tellerProcess() function calls the logStatus() method which is also located in tellerProcess.h file (uses log_mutex lock before accessing r_log shared memory) which is responsible for logging details of customer whenever a customer is gained by a teller and after completion of the service of customer by calling the same function in both instances as shown below- sample extracted from r_log.txt during a test run.

```
------------------------------------------------------------
Teller: 3
Customer: 5
Arrival time: 11:48:30
Response time: 11:48:48
------------------------------------------------------------


------------------------------------------------------------
Teller: 3
Customer: 5
Arrival time: 11:48:30
Completion time: 11:48:53
```

```
------------------------------------------------------------------
```

- **All tellers terminate when there are no remaining customers**

The core loop which runs has the conditions so that it will not exit until the file is read to until the end is reached and also the queue is empty therefore even though tellers sleep until new customers are added when queue is empty, the loop which serves customers will only end after all customers in the file are served which follows to termination of tellers after the loop has reached to an end.

Whenever a teller thread terminates r_log.txt file is accessed again to log the teller performance using logTermination () function (located in main.c) which gives log output as shown below – sample gained during the test run

```
------------------------------------------------------------------
Termination: Teller-1
#served customers: 4
Start time: 11:48:44
Termination time: 11:48:55
------------------------------------------------------------------
Termination: Teller-2
#served customers: 3
Start time: 11:48:46
Termination time: 11:48:57
------------------------------------------------------------------
```

when all threads are terminated final teller thread triggers the logging of all teller statistics giving a total of all tellers as shown below – sample gained during the test run.

```
------------------------------------------------------------------
Teller Statistics

Teller-1 served 4 customer(s).
Teller-2 served 3 customer(s).
Teller-3 served 4 customer(s).
Teller-4 served 4 customer(s)

Total number of customers served: 15
------------------------------------------------------------------
```

**Note: Each time a function is accessing r_log.txt file it aquires the mutes lock(log_mutex) ans releases when logging is completed.**

- **<u>All activities of the queue and the tellers are recorded in a file named r_log.txt and mutual exclusion is achieved when accessing the r_log.txt shared memory.</u>**

r_log.txt file is accessed during 5 instances when one customer is considered

- When a new customer is added to queue.
- When a teller gains a customer to serve.
- When a teller completes serving a customer.
- When a teller terminates.
- When final teller terminates while logging all the teller statistics.

**In each of these instances log_mutex lock is used to satisfy mutual exclusion so that no 2 above functions/instances use r_log file simultaneously.**

Complete journey of a customer (customer number 2, Service type W) in this program as logged in r_log.txt file can be denoted as below – extracted from the test run

```
---------------------------------------------------------------
Customer 2: W
Arrival time: 11:48:24
---------------------------------------------------------------
Teller: 4
Customer: 2
Arrival time: 11:48:24
Response time: 11:48:42
---------------------------------------------------------------
Teller: 4
Customer: 2
Arrival time: 11:48:24
Completion time: 11:48:45
---------------------------------------------------------------
```

**Note: withdrawal process was considered to be executed within 3 seconds.**

- **All mutex locks are stored in mutexLocks.h and initiated in mutexLocks.c so that thay can be accesed by any file.**

Shown below are all locks and conditions used as per mutexLocks.h which are freed by destroying at the end.

```
extern  pthread_mutex_t queue_mutex;
extern pthread_mutex_t log_mutex;
extern  pthread_cond_t queue_not_full;
extern  pthread_cond_t queue_not_empty;
```

- **Complete output for the below mentioned run command with arguments is shown below**

**Run command (c_file4 contains 5 customers):**

```
./execute c_file4.txt 5 1 3 4 5
```

**Output:**

```
Customer 1: W
Arrival time: 08:37:00
-----------------------------------------------------------------
Customer 2: W
Arrival time: 08:37:02
-----------------------------------------------------------------
Customer 3: I
Arrival time: 08:37:04
-----------------------------------------------------------------
Customer 4: D
Arrival time: 08:37:06




-----------------------------------------------------------------
```

```
Customer 5: I
Arrival time: 08:37:08
-----------------------------------------------------------------
Teller: 1
Customer: 1
Arrival time: 08:37:00
Response time: 08:37:08
-----------------------------------------------------------------
Teller: 2
Customer: 2
Arrival time: 08:37:02
Response time: 08:37:08
-----------------------------------------------------------------
Teller: 3
Customer: 3
Arrival time: 08:37:04
Response time: 08:37:08
-----------------------------------------------------------------
Teller: 4
Customer: 4
Arrival time: 08:37:06
Response time: 08:37:08
-----------------------------------------------------------------
Teller: 1
Customer: 5
Arrival time: 08:37:08
Response time: 08:37:08
-----------------------------------------------------------------
Teller: 1
Customer: 1
Arrival time: 08:37:00
Completion time: 08:37:11
-----------------------------------------------------------------
Teller: 2
Customer: 2
Arrival time: 08:37:02
Completion time: 08:37:11
-----------------------------------------------------------------
Teller: 4
Customer: 4
Arrival time: 08:37:06
Completion time: 08:37:12



-----------------------------------------------------------------
```

```
Teller: 3
Customer: 3
Arrival time: 08:37:04
Completion time: 08:37:13
-----------------------------------------------------------------
Teller: 1
Customer: 5
Arrival time: 08:37:08
Completion time: 08:37:13
-----------------------------------------------------------------
Termination: Teller-1
#served customers: 2
Start time: 08:37:08
Termination time: 08:37:13
-----------------------------------------------------------------
Termination: Teller-2
#served customers: 1
Start time: 08:37:08
Termination time: 08:37:13
-----------------------------------------------------------------
Termination: Teller-3
#served customers: 1
Start time: 08:37:08
Termination time: 08:37:13
-----------------------------------------------------------------
Termination: Teller-4
#served customers: 1
Start time: 08:37:08
Termination time: 08:37:13
-----------------------------------------------------------------
Teller Statistics

Teller-1 served 2 customer(s).
Teller-2 served 1 customer(s).
Teller-3 served 1 customer(s).
Teller-4 served 1 customer(s)

Total number of customers: 5
-----------------------------------------------------------------
```

**All requirements as mentioned in the specification are reached and are functioning perfectly according to best of my knowledge.**

Curtin University – Department of Computing

# Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

| | | | |
|---|---|---|---|
| Last name: | Athukorala | Student ID: | 21038875 |
| Other name(s): | Thilina Prasad Athukorala | | |
| Unit name: | Operating Systems | Unit ID: | COMP2006 |
| Lecturer / unit coordinator: | Dr. Sie Teng Soh | Tutor: | U.U. Samantha Rajap |
| Date of submission: | 2023/05/08 | Which assignment? | (Leave blank if the unit has only one assignment.) |

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature:  Date of signature: 2023/05/08

*(By submitting this form, you indicate that you agree with all the above text.)*