Document version: 1.1 (2015-11-15)

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:	Zhenqi Zhang	Student ID:	20080833
Other name(s):			
Unit name:	Operating System	Unit ID:	COMP2006
Lecturer / unit coordinator:	Soh	Tutor:	Fri 11 to 12
Date of submission:	07/05/2023	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: Zhenqi Zhang

Date of signature: 07/05/2023

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

Report For Operating System Assignment One.

Introduction:

In this assignment, shared resource access and synchronisation strategies in a multi-threaded programme are looked at. Two functions, **customer()** and **teller(),** which stand for the customer and teller threads, respectively, are the main operating components of the code. In order to assure the program's functionality, I made several choices about synchronisation and shared resource access. This report will look at the code and go over the choices that were taken.

Mutex Explanation:

Reader Mutex -This mutex is used as the **reader** mutex. It controls access to the customer queue (**args->c_queue**) during enqueue and dequeue operations. It ensures that only one thread can access the queue at a time, maintaining data consistency. The reader mutex is locked using the **pthread_mutex_lock()** function before performing enqueue and dequeue operations on the customer queue (**args->c_queue**).

By acquiring the lock, a thread can safely add or remove customers from the queue without conflicting with other threads. Once the operation is complete, the reader mutex is unlocked using **pthread_mutex_unlock()** to allow other threads to access the queue. This Mutex is important to resolve multiple problems such as:

1. As Enqueue and Dequeue are occurring concurrently, they may both access and modify the queue data structure. Inconsistent data may come from this, including lost or overwritten customer entries, an incorrectly sized queue, or an improper queue status. Additionally, if there is no condition wait mutex and dequeue is faster than enqueue, the dequeue will fail and cause issues for the program.
2. Several dequeues are taking place at once; if the same client is removed from the queue as a result, inconsistent data will result. Incorrect results may come from the multiple processing or skipping of some consumers.

Writer Mutex - This mutex is used as the **writer** mutex. It controls access to the output file (**args->write**) when writing customer information. It ensures that only one thread can write to the file at a time, preventing concurrent writes that may lead to data corruption. The writer mutex is locked using the pthread_mutex_lock() function before writing to the output file (r_log). By acquiring the lock, a thread can safely write its output to the file without interference from other threads. Once the write operation is complete, the writer mutex is unlocked using pthread_mutex_unlock() to allow other threads to access the file.

The writer mutex is essential for preserving the consistency and integrity of the output file. Concurrent writes to the file without sufficient synchronisation may cause data corruption, overlapping content, or both. The writer mutex ensures that only one thread has access to the file, ensuring that each thread's output is written in a sequential manner and free from other threads' interference. This Mutex is important to resolve multiple problems such as:

1. If many threads write to the same file concurrently without synchronisation, their output may overlap. It may be difficult to appropriately analyse the data as a result of data that is illegible or corrupted. In order to avoid overlapping output and preserve the integrity of the data, the writer mutex makes sure that only one thread can write to the file at once.
2. Data corruption can occur when multiple threads write to the same file at the same time and their output overlaps or interferes with one another. The output could be inaccurate or lose information as a result, which would make it unreliable. Each thread can write its output sequentially because the writer mutex guarantees exclusive access to the file, guarding against data corruption problems.

C_queue_not_empty Mutex - This mutex is used as a **condition variable** to signal that the customer queue is not empty. It allows threads to wait until there is at least one customer in the queue before they proceed with dequeuing. In this case, the condition is the availability of customers in the queue. Using the pthread_cond_signal() function, a thread signals the condition variable when a customer is enqueued. Other threads waiting on the condition variable are alerted by this signal that a client is available for processing. Using the pthread_cond_wait() function, threads that need to dequeue customers from the queue will wait on the c_queue_not_empty mutex. Other threads are now able to access the queue because this call releases the associated reader mutex. Until a signal on the condition variable awakens the thread, it is in a waiting state. The thread re-acquires the reader mutex after being awakened, then performs the dequeue operation. This Mutex is important to resolve multiple problems such as:

1. Threads that dequeue customers need to wait until there are customers available in the queue. Without proper synchronization, threads might continuously check the queue for available customers, resulting in high CPU usage and inefficient resource utilization. The c_queue_not_empty mutex provides a mechanism for threads to wait efficiently until there are customers to dequeue, reducing unnecessary resource consumption.
2. When the queue is empty, several threads may attempt to simultaneously dequeue customers. This situation can result in racial tensions and unpredictable behaviour. Threads can securely wait for customers without interfering with one another by using the c_queue_not_empty mutex and condition variable. Multiple threads cannot participate in the dequeue process at the same time because the mutex guarantees exclusive access to the condition variable.
3. Enqueue and dequeue activities can be synchronised thanks to the c_queue_not_empty mutex. The signalling of the condition variable guarantees that waiting threads are swiftly informed when a customer is enqueued. Through this signalling, threads can dequeue customers as soon as they become available and save needless waiting. Threads may overlook customer availability without the c_queue_not_empty mutex and condition variable, resulting in errors or delays.

These mutexes are part of the **mutex_args** structure, which is passed as an argument to the customer and teller threads. The structure encapsulates the necessary mutexes for synchronization in the program.

The programme makes use of these mutexes to shield critical section from concurrent access, such as enqueuing and dequeuing customers and writing to the output file. When several threads are using the same shared resources, this preserves data integrity and avoids race problems.

Synchronization for Shared Resource Access:

In my code, I have implemented synchronization mechanisms to facilitate the access and manipulation of shared resources. One important shared resource is the customer queue (args->c_queue), where customers are enqueued and dequeued by multiple threads concurrently. To ensure the integrity of the customer queue and prevent data inconsistencies, I have used a reader mutex.

The reader mutex, also known as the enqueue/dequeue mutex, plays a crucial role in synchronizing access to the customer queue. Before enqueuing a customer, a thread locks the reader mutex using pthread_mutex_lock(). This ensures that only one thread can access the queue at a time, preventing conflicts and preserving the integrity of the data structure. Once the enqueue operation is complete, the mutex is unlocked using pthread_mutex_unlock(), allowing other threads to enqueue customers.

Similarly, before dequeuing a customer, threads must first acquire the reader mutex by locking it. This guarantees exclusive access to the customer queue. By doing so, I have effectively prevented multiple threads from dequeueing the same customer simultaneously, which could lead to data inconsistencies and incorrect results. After removing a customer from the queue, the reader mutex is unlocked, enabling other threads to access the queue concurrently.

1. Customer Queue (args->c_queue):

    - Enqueue Operation: When a thread enqueues a customer, it first locks the reader mutex using pthread_mutex_lock(). This ensures exclusive access to the customer queue, preventing other threads from enqueueing or dequeueing simultaneously. After adding the customer to the queue, the mutex is unlocked using pthread_mutex_unlock() to allow other threads to access the queue.
    - Dequeue Operation: Threads that need to dequeue customers first acquire the reader mutex by locking it. This guarantees exclusive access to the queue. The thread then removes a customer from the front of the queue and unlocks the reader mutex to enable concurrent access by other threads.

Another shared resource in my code is the output file (args->write), where customer information is written. To ensure that only one thread writes to the file at a time and to maintain the integrity of the output, I have employed a writer mutex.

Before writing customer information to the output file, threads lock the writer mutex using pthread_mutex_lock(). This ensures exclusive access to the file, preventing conflicts and ensuring that the data is written correctly. Once the write operation is complete, the writer mutex is unlocked using pthread_mutex_unlock(), allowing other threads to write to the file.

By incorporating synchronization mechanisms, such as the reader mutex and the writer mutex, I have controlled access to shared resources in a thread-safe manner. This approach eliminates race conditions, data corruption, and inconsistencies that could arise if multiple threads accessed and modified the shared resources simultaneously. The synchronization ensures that each thread completes its operation before another thread accesses the shared resources, promoting data consistency and enabling reliable program execution.

2. Output File (args->write):

    - Write Operation: To write customer information to the output file, threads lock the writer mutex using pthread_mutex_lock() before accessing the file. This ensures that only one thread can write to the file at a time, avoiding conflicts and maintaining data integrity. After completing the write operation, the writer mutex is unlocked using pthread_mutex_unlock(), allowing other threads to write to the file.

Cases that fail the Program & How can the program work perfectly:

In the implemented program, the variables **m** and **tc** are used to control the number of customers to be inserted into the queue and the time delay between reading each customer and inserting them into the queue, respectively. To ensure the program runs perfectly without encountering any problems, it is crucial to set these variables appropriately based on the desired scenario and system constraints.

The value of **m** should be chosen carefully, considering factors such as the system's capacity, processing capabilities, and the desired duration of the simulation. Setting **m** to a reasonable number ensures that the program does not overwhelm the system with an excessive number of customers or cause memory-related issues. It also allows for a realistic representation of customer arrival patterns and queue dynamics.

Similarly, the value of **tc** should be determined based on the expected rate of customer arrivals and the desired pace of the simulation. Choosing a suitable **tc** value ensures that customers are inserted into the queue at a realistic rate, avoiding an unrealistic flood of customers or an unreasonably slow progression. It allows the program to simulate real-world scenarios accurately and provides a balanced workload for the system.

By setting appropriate values for **m** and **tc**, the program can run flawlessly, accurately simulating the expected customer arrivals and their insertion into the queue. It ensures that the program operates within the desired parameters, delivering reliable results without encountering any issues related to the number of customers or the timing of their arrival.

In the implemented program, it is essential to ensure that the values of the variables **tw**, **td**, and **ti**, representing the serving time for the tellers, delay time for customers, and idle time for tellers, respectively, are all integers greater than zero. This precaution is in line with common sense, as it is highly unlikely for the serving time to be zero in a real-world scenario. By enforcing this constraint, the program avoids any potential issues that may arise from zero or negative values, such as infinite loops or incorrect calculations. Consequently, the program operates with reliability and accuracy, providing a realistic simulation of customer and teller interactions.

Sample Input & Output:

Sample Input:

```
valgrind --leak-check=full --track-origins=yes -s ./cq 10 1 5 5 5
```

Sample Output: (Brief Explanation)

```
========================================================================
|                         Customer Log                         |
========================================================================
```

Enqueue First Customer

```
----------------------------------Enqueue---------------------------------
Customer 1 : W
Arrival time: 22:38:21
------------------------------------------------------------------------
```

Process the First Customer

```
----------------------------------Process---------------------------------
Teller: 1
Customer: 1
Arrival time: 22:38:21
Response time: 22:38:21
------------------------------------------------------------------------
```

Enqueue the Second Customer

```
----------------------------------Enqueue---------------------------------
Customer 2 : W
Arrival time: 22:38:22
------------------------------------------------------------------------
```

Second Teller start Processing the Second customer.

```
----------------------------------Process---------------------------------

Teller: 4

Customer: 2

Arrival time: 22:38:22

Response time: 22:38:22

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

Enqueue the Third Customer

```
----------------------------------Enqueue---------------------------------

Customer 3 : W

Arrival time: 22:38:23

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

Third Teller start Processing the Third customer.

```
----------------------------------Process---------------------------------

Teller: 3

Customer: 3

Arrival time: 22:38:23

Response time: 22:38:23

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

Enqueue the Fourth Customer

```
----------------------------------Enqueue---------------------------------

Customer 4 : W

Arrival time: 22:38:24

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

Fourth Teller start Processing the Fourth customer.

```
----------------------------------Process---------------------------------

Teller: 2

Customer: 4

Arrival time: 22:38:24

Response time: 22:38:24

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

Enqueue the Fifth Customer

```
----------------------------------Enqueue---------------------------------

Customer 5 : D

Arrival time: 22:38:25

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

Here, Teller One Have Finished processing the First Customer and now can go on to the next Customer

```
---------------------------------Finish---------------------------------

Teller: 1

Customer: 1

Arrival time: 22:38:21

Completion time: 22:38:26

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

Now that Teller One is free, and customer Five is in the queue, Teller One starts processing Customer Five

```
----------------------------------Process---------------------------------

Teller: 1

Customer: 5

Arrival time: 22:38:25

Response time: 22:38:26

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

Enqueue the Sixth Customer

```
----------------------------------Enqueue---------------------------------

Customer 6 : I

Arrival time: 22:38:26

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

Here, Teller Four (Second Woke) Have Finished processing the Second Customer and now can go on to the next Customer

```
---------------------------------Finish---------------------------------

Teller: 4

Customer: 2

Arrival time: 22:38:22

Completion time: 22:38:27
```

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

Now that Teller Four (Second Woke) is free, and customer Six is in the queue, Teller Four starts processing Customer Six

```
---------------------------------Process--------------------------------
Teller: 4
Customer: 6
Arrival time: 22:38:26
Response time: 22:38:27
-----------------------------------------------------------------------
```

Enqueue the Seventh Customer

```
---------------------------------Enqueue--------------------------------
Customer 7 : D
Arrival time: 22:38:27
-----------------------------------------------------------------------
```

Teller Three (Third Woke) have Finished its process Customer and can go next.

```
--------------------------------Finish--------------------------------
Teller: 3
Customer: 3
Arrival time: 22:38:23
Completion time: 22:38:28
-----------------------------------------------------------------------
```

Now that Teller Three is free, It starts processing the next Customer in the queue.

```
---------------------------------Process--------------------------------
Teller: 3
Customer: 7
Arrival time: 22:38:27
Response time: 22:38:28
-----------------------------------------------------------------------
```

Enqueue the Eighth Customer

```
---------------------------------Enqueue--------------------------------

Customer 8 : D

Arrival time: 22:38:28

----------------------------------------------------------------------


--------------------------------Finish--------------------------------

Teller: 2

Customer: 4

Arrival time: 22:38:24

Completion time: 22:38:29

----------------------------------------------------------------------


---------------------------------Process--------------------------------

Teller: 2

Customer: 8

Arrival time: 22:38:28

Response time: 22:38:29

----------------------------------------------------------------------


---------------------------------Enqueue--------------------------------

Customer 9 : D

Arrival time: 22:38:29

----------------------------------------------------------------------

---------------------------------Enqueue--------------------------------

Customer 10 : D

Arrival time: 22:38:30

----------------------------------------------------------------------


--------------------------------Finish--------------------------------

Teller: 1

Customer: 5

Arrival time: 22:38:25

Completion time: 22:38:31
```

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



----------------------------------Process---------------------------------
Teller: 1

Customer: 9

Arrival time: 22:38:29

Response time: 22:38:31

------------------------------------------------------------------------


---------------------------------Finish---------------------------------
Teller: 4

Customer: 6

Arrival time: 22:38:26

Completion time: 22:38:32

------------------------------------------------------------------------


----------------------------------Process---------------------------------
Teller: 4

Customer: 10

Arrival time: 22:38:30

Response time: 22:38:32

------------------------------------------------------------------------


---------------------------------Finish---------------------------------
Teller: 3

Customer: 7

Arrival time: 22:38:27

Completion time: 22:38:33

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

All the customers have enqueued and have been processed, now there are no more left and Teller can Terminate.

```
-----------------------------------Terminate---------------------------

Teller: 3

Customer Served: 2

Arrival time: 22:38:21

Completion time: 22:38:33

---------------------------------------------------------------------


--------------------------------Finish--------------------------------

Teller: 2

Customer: 8

Arrival time: 22:38:28

Completion time: 22:38:34

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

Second Teller Terminates

```
-----------------------------------Terminate---------------------------

Teller: 2

Customer Served: 2

Arrival time: 22:38:21

Completion time: 22:38:34

---------------------------------------------------------------------


--------------------------------Finish--------------------------------

Teller: 1

Customer: 9

Arrival time: 22:38:29

Completion time: 22:38:36

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

Third Teller Terminates

```
-----------------------------------Terminate---------------------------

Teller: 1

Customer Served: 3

Arrival time: 22:38:21

Completion time: 22:38:36

----------------------------------------------------------------------


--------------------------------Finish--------------------------------

Teller: 4

Customer: 10

Arrival time: 22:38:30

Completion time: 22:38:37

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

Last Teller Terminate

```
-----------------------------------Terminate---------------------------

Teller: 4

Customer Served: 3

Arrival time: 22:38:21

Completion time: 22:38:37

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

Display How many Customers each Teller Served.

```
=======================================================================


Teller Statistic


Teller-1 serves 3 customers.


Teller-2 serves 2 customers.


Teller-3 serves 2 customers.
```

```
Teller-4 serves 3 customers.

================================================================================
```