CSC 360 – Operating Systems

Summer 2024

Assignment 2: Design Document

Karanbir Gosal

V00979752

1. **How many threads are you going to use? Specify the task that you intend each thread to perform.**

   - **Customer Threads**: One thread per customer to simulate their arrival, waiting, and being served.
   - **Clerk Threads**: One thread per clerk to handle customer service.

2. **Do the threads work independently? Or, is there an overall "controller" thread?**

   Customer threads work independently based on their arrival and service needs. While each **clerk thread** waits and services customers independently (using mutexes and condition variables), there is likely some synchronization or coordination needed among clerks as well as customers. For example, clerks checks if there is another clerk is serving some customer from the queue and customer signals the clerk while it is done. Mostly, we can say they work independently of each other except some basic synchronization.

   There is no specific thread but **overall "controller" thread** can be considered as the **main thread which is our program** that initializes the system, manages shared resources (like queues), monitors the overall state of customers and clerks, and ensures proper termination of the program.

3. **How many mutexes are you going to use? Specify the operation that each mutex will guard.**

   - **Queue Mutexes**: One per queue (business and economy) to protect queue operations.

- **Clerk Mutexes**: One per clerk to manage access to clerk's service state.
- **Common Use Mutex**: One for the common use like counting the total wait time etc.

4. **Will the main thread be idle? If not, what will it be doing?**

   The main thread will initialize necessary data structures such as queues, mutexes, and condition variables. It will create threads to handle concurrent operations. It will continuously monitor the overall program state and manage termination processes effectively. Upon termination, it will free up resources, including destroying condition variables and mutexes, and it will output the statistics of airline customer service times.

5. **How are you going to represent customers? what type of data structure will you use?**
   The **customer_info** struct encapsulates details about each customer within a system. The **User ID (int)** uniquely identifies customers and **Class Type (int)** categorizes customers based on service class that is *business (1)* or *economy (0)*. **Arrival Time (int)** indicates when a customer enters the system. **Service Time(int)** specifies how long it takes for a customer for the service by a clerk.

6. **How are you going to ensure that data structures in your program will not be modified concurrently?**

   To ensure data structures are not modified concurrently, we use mutexes to guard access to shared data structures such as queues and clerk states. This ensures that only one thread modifies data at a time, thereby preventing race conditions.

7. **How many convars are you going to use? For each convar:**

   (a) Describe the condition that the convar will represent.

   (b) Which mutex is associated with the convar? Why?

   (c) What operation should be performed once pthread cond wait() has been unblocked and reacquired the mutex?

Number of Condition Variables:

- **Queue Condition Variables**: One per queue to signal when new customers arrive.
- **Clerk Condition Variables**: One per clerk to signal when a clerk is available for service.

a) Queue condition variables are used to wake up customers in the queue so whichever the first one in the queue can be served. Clerk condition variables are used to signal the clerks that the customer is served and the clerk can now serve another customer.

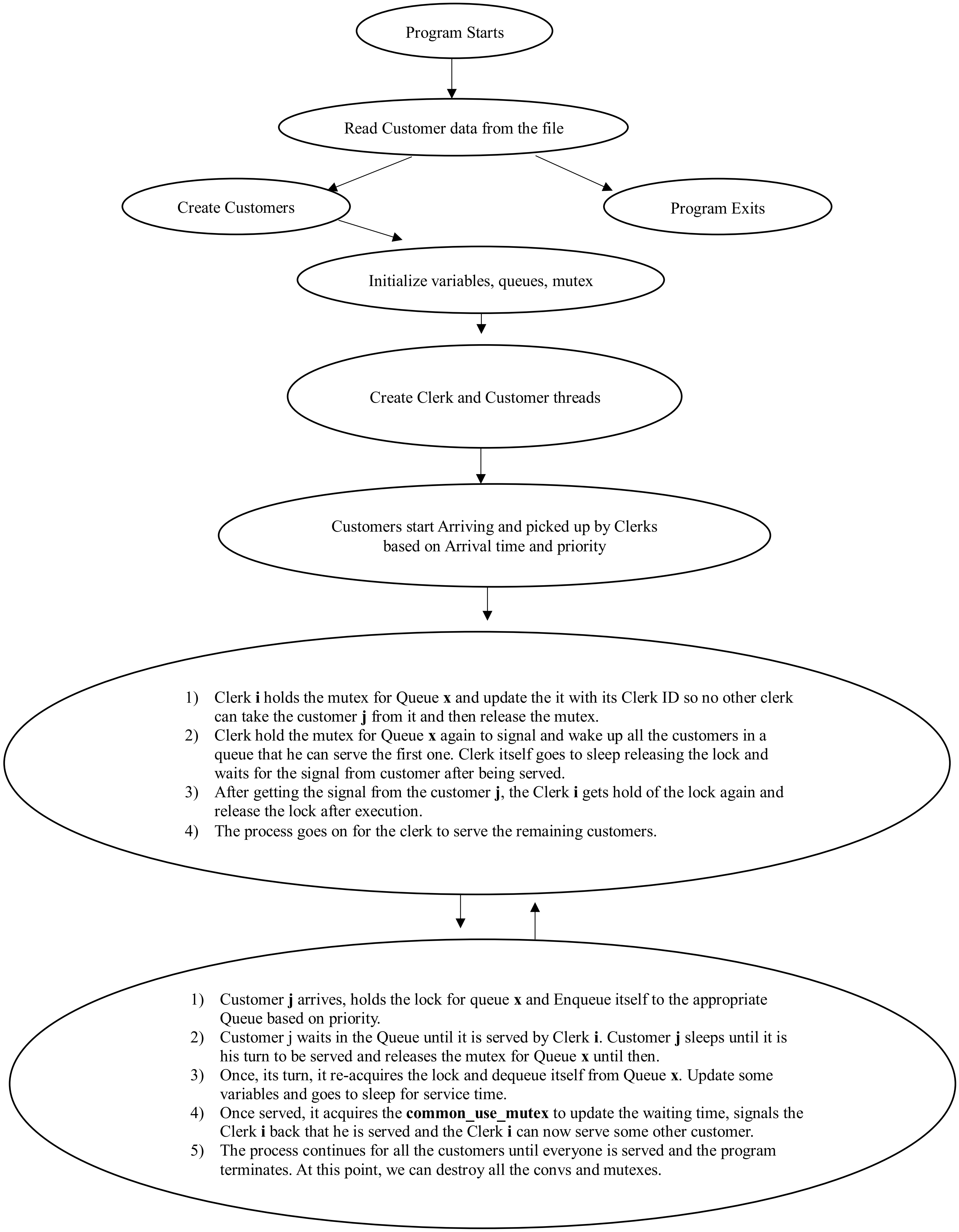
b) Queue mutex is associated with both Clerk condition variables and Queue condition variables as we want to make sure we release the lock to allow other threads to modify the data structure as well.

Clerk condition variables are associated with the Queue mutex when Clerk is waiting for the customer to signal back once customer is done being served.

Queue condition variables are associated when the customer thread is waiting for being served.

c) After being unblocked and waking up from *pthread_cond_wait*, the thread reacquires the associated mutex and continues execution.

8. **Briefly sketch the overall algorithm you will use. You may use sentences such as: *If clerk i finishes service, release clerkImutex*.**


Program Starts
Read Customer data from the file
Create Customers
Program Exits
Initialize variables, queues, mutex
Create Clerk and Customer threads
Customers start Arriving and picked up by Clerks based on Arrival time and priority

1) Clerk **i** holds the mutex for Queue **x** and update the it with its Clerk ID so no other clerk can take the customer **j** from it and then release the mutex.
2) Clerk hold the mutex for Queue **x** again to signal and wake up all the customers in a queue that he can serve the first one. Clerk itself goes to sleep releasing the lock and waits for the signal from customer after being served.
3) After getting the signal from the customer **j**, the Clerk **i** gets hold of the lock again and release the lock after execution.
4) The process goes on for the clerk to serve the remaining customers.

1) Customer **j** arrives, holds the lock for queue **x** and Enqueue itself to the appropriate Queue based on priority.
2) Customer j waits in the Queue until it is served by Clerk **i**. Customer **j** sleeps until it is his turn to be served and releases the mutex for Queue **x** until then.
3) Once, its turn, it re-acquires the lock and dequeue itself from Queue **x**. Update some variables and goes to sleep for service time.
4) Once served, it acquires the **common_use_mutex** to update the waiting time, signals the Clerk **i** back that he is served and the Clerk **i** can now serve some other customer.
5) The process continues for all the customers until everyone is served and the program terminates. At this point, we can destroy all the convs and mutexes.