Name: Jingyao Yu

ID: V00972776

Design Document

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

Two threads.
Customer Thread: make the customer enter the queue or wait.
Clerk Thread: send the clerk to the customer, check available

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

Work independently.

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

Two mutexes. I create one mutex for customers(queue), which can control the clerk to serve the customers. I create one mutex for clerks, which can control the clerk to serve if customers no one served.

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

Yes.
It will initialize the metux and conditional variable. After the "pthread_join()", the main thread will go to sleep.

5. How are you going to represent customers? what type of data structure will you use?

```
struct customer_info{
        int user_id;
        int class_type; //0: customer, 1: business
        int arrival_time;
        int service_time;
        int customer_status; // 1: cleerk1, 2: cleerk2, 3: cleerk3, 4: cleerk4, 5: cleerk5
        int position_in_line; // position in the all_customer[]
        double start_waiting; // customer start waiting time
        double end_waiting; // customer end waiting time
};
```

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

Use the lock to lock the mutex.

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

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

When waking up the customer and selecting from the queue.

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

The mutex of the queue will need a convar to control the clerk serving the correct customers. And also make sure the clerk lock until it finishes the current customer.

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

The customer will be serviced, and update total waiting time for econ and business class customers, and then Sleep to get some serving time and then send a signal to tell is able to serve the next customer

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

In customers, lock the queue and add the customer into the match queue (econ or business queue), then, print the sample format depending on different class types and print the information, then, store the time that customer starts waiting, if the customer is not served by clerk, depends on the class type, wait for conditional variable 0 or 1, and mutex, store the current time as the time finish waiting and get service, update total waiting time for econ and business customer, start to the server and unlock the queue, sleep to give time to serve and send the signal to the clerk, and able to serve next customer.

In clerks, lock the queue, if the econ line is not empty, get the econ customer from the queue, pop a customer from econ_queue, set the clerk is busy, send a clerk to the customer, update customer_status, send the signal to queue, and able to serve next customer, send the signal to the clerk, unlock the queue. Elseif, the business line is not empty, get the business customer from the queue, set the clerk as busy, send a clerk to the customer, update customer_status, end the signal to queue, and be able to serve the next customer, send the signal to the clerk, unlock the queue. Lock the busy clerk and if the clerk is busy then wait.

In main, initialize queue_mutex and queue_convar, initialize clerk_mutex and clerk_convar, create clerk thread and set value for clerk, create customer thread, then join the customer thread, wait for one by one.