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

- 5 clerk threads, these threads will serve the customers.
- N customer threads where N is the number of customers, these threads will contain information for each customer and simulate the customers behavior.
- 1 Main thread, this thread will read the input file, parse the input file, create the customers, create the clerks, initialize and destroy the mutexes, initialize and destroy the condition variables, initialize the global variables, and calculate the statistics when all customers have been served.

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

The threads will work independently no controller thread will be used as a controller thread produces a very coupled design that will be less efficient. Each clerk using mutexes to coordinate with the other clerks to serve the customers.

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

5 mutexes will be used. Each one is described below:

- timeLock

  Guards access to the waiting time variable. This variable is updated by all customers once their waiting time is complete.

- queueLock

  Guards access to the business and economy queue along with their corresponding variables like length, and index (everything before the index has been served). These variables are accessed by all customers to add themselves to the queue once they have arrived, and by the clerks to determine who to serve. The clerks are the only threads to update the index as they use it to keep track of the queues progression. The customers are the only ones to update the length variable as they do this when they add themselves to the queue. The corresponding variables are accessed the same time the queues are hence the reason I chose to have them use the same mutex.

- clerkAvailable

  Guards the customer when they check if they are next to be served. The main use of this mutex is to be paired with the conditional variable that indicates a clerk is serving a customer.

- nextClerk

  Guards access to the serving variables which indicates the customer to serve next and the serving clerk ID. Since only one clerk can serve the next customer this guards the clerk and allows it to serve the next customer without other clerks trying to do the same thing.

- clerkOccupied (one for each clerk i.e. array of mutexes)

  Guards the clerk from being used by another customer. This is also paired with the done serving conditional variable. This is used because the mutex can always be obtained back by the clerk when the customer signals they are done. If another mutex is used then it might not be able to be obtained as easily and the conditional wait will not return.

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

The main thread will loop to ensure every customer has joined. Following all customers rejoining the main thread will print the statistics and destroy the resources (mutexes, threads, and conditional variables).

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

A struct will be used and casted to void* and passed to the thread as an argument. The struct will contain the following information: - customer id - class the customer belongs to - 1 indicates Business class - 0 indicates Economy class - arrival time - service time

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

Each customer will have it's own struct. So an array of structs will be used. Clerk id's will also be passed in different variables so they don't get changed before the clerk can save it. All other global data is protected by the mutexes as described above.

7. How many conditional variables are you going to use?

serving

(a) Describe the condition that the conditional variables will represent.

- Alerts customers a clerk is ready to serve customer that's id is stored in the toBeServed variable.

(b) Which mutex is associated with the conditional variables? Why?

- clerkAvailable. That is because this mutex is intended to Guard the customer when they check if they are next to be served.

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

- The customer will check if their id is being served. If they are not being served they will conditionally wait again. If they are being served they will save their servers ID and lock that clerk. They then unlock the clerkAvailable mutex so the line can move up.

queueChanged

(a) Describe the condition that the conditional variables will represent.

- This represents a customer has just joined the line. This allows the clerk if waiting to serve someone to start serving the customer. If there is no clerk to serve the customer when one is ready they will take the customer and no signal is needed. More efficient than the clerk sleeping and continually waiting to check if a customer arrived.

(b) Which mutex is associated with the conditional variables? Why?

- queueLock. This is because the clerk then has to check the queue and update the queue index so the queue lock must be obtained to do this.

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

- The waiting clerk will choose a customer to serve.

customerServed (one for each clerk i.e. array of conditionals)

(a) Describe the condition that the conditional variables will represent.

- This represents that a customer served by clerk i is done being served.

(b) Which mutex is associated with the conditional variables? Why?

- clerkOccupied[i-1] is used because the clerk can obtain its mutex back. This then can be unlocked for the next customer they serve to obtain.

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

- The clerk will try to obtain the server mutex to be the next server.

8. Briefly sketch the overall algorithm you will use.

Clerk

- Save clerk id
- While 1:
    - Lock the server mutex so the clerk can be the next server
    - Set serverID to the clerk id to indicate this clerk is the next server
    - Lock queue
    - While no customer to serve
        - Conditionally wait for the queue to change and obtain the queue lock again
    - If business queue has customer to serve:
        - Set toBeServed to the next business customer to be served
        - Increment the business served index
    - Otherwise:
        - Set toBeServed to the next economy customer to be served
        - Increment the economy served index
    - Unlock queue mutex
    - Set status to busy
    - Alert customers that clerk is serving customer indicated by the toBeServed variable
    - usleep for 10 to allow customers to have already got back to call pthread_cond_wait
    - Unlock the next clerk mutex to allow another clerk to serve another customer
    - While clerk is busy:
        - Conditionally wait for the customer served variable and obtain back the clerkOccupied mutex
- pthread_exit(NULL);

Customer

- Cast argument into struct customerInformation
- usleep for the arrivalTime * 100000
- Print the customer has arrived
- Lock queue mutex
- If customer is in the economy class
    - Increment economy queue length
    - Print customer has entered economy queue and the current length of th queue
    - Add to the economy queue at the back
- Otherwise if in business class

    - Increment business queue length
    - Print customer has entered business queue and the current length of th queue
    - Add to the business queue at the back
- Unlock the queue mutex
- Get the start time
- Signal the queue has changed if a clerk is waiting to serve a customer
- While my id is not being served:
    - Conditionally wait on serving and try to obtain the clerkAvailable mutex
- Save serverID
- Lock specified clerkOccupied mutex
- Unlock the clerkAvailable mutex
- Get the end time
- Print clerk with respective ID has started serving the customer with ID at respective time
- Lock the timeLock mutex
- Add waiting tme to the specified class wait time
- Unlock the timeLock mutex
- usleep for serviceTime * 100000
- Clear clerks busy flag
- Unlock clerkOccupied mutex
- Signal clear that customer is served
- Print clerk with respective ID has finished serving the customer with ID at respective time
- pthread_exit(NULL);