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

Each customer is a thread. The customer will record the time it enters the queue, when its service begins and when its service ends. Clerks will not be threads, but just a global variable keeping track of how many customers are being serviced at once.

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

Threads work independently and use the shared global variables to communicate.

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

Customer_enqueue_lock: guards the queues so that only one customer will modify them at a time

Customer_clerk_communication: guards the clerk global variables so that only one customer can see / modify them at any given time.

Total_wait_lock: guards the total_wait_time, business_wait_time and econ_wait_time globals so that only one customer can modify them at a time.

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

The main thread will be idle, everything is contained in the customer threads. Once each customer thread terminates the main thread will calculate the average wait times.

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

Customers will be represented by a struct, with attributes user_id, class_type, arrival_time and service_time.

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

Mutex locks will protect any code segment that modifies a global variable (outside of the main thread)

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

I am using one convar. It represents the condition that there are no clerks available to serve a customer. The associated mutex is customer_clerk_communication. The customer will lock that mutex and go into the serving logic, but if there are no clerks it will hit the cond_wait statement and wait to be signalled by a customer who has just finished being serviced. Once it's unlocked it will search for the free clerk and begin being serviced.

8. Briefly sketch the overall algorithm you will use.

Two global variables represent the clerks:

Int available_clerks : Represents the number of clerks who are not busy. Initialized to 5 at the top of main, and decremented when a customer is ready to be serviced, and incremented when that customer is finished being serviced

Int clerks[5]: Slot i in this array represents the clerk with id = i. clerks[i] is equal to 0 if that clerk is available to service a customer, and 1 if that clerk is currently serving a customer. So for example if Clerks 2 and 3 are busy and the rest are free then clerks = {0,0,1,1,0 }.

The customers are represented by a thread. The thread passes the customer_info struct on initialization and uses that to insert the customer into the correct queue. Once the customer is queued the thread will idle until a clerk is available, and then it will claim that clerk by altering available_clerks and the clerks array. Then it will call usleep for its service time, then free the clerk and the thread will terminate.

General Algorithm:

Read input file, create customer struct for each input customer

Initialize a thread for each customer

Customer thread adds itself to queue, checks if clerks are available

If clerk is not available: idle, check again

If clerk is available:

Remove clerk from available clerks, sleep for service time, return clerk, do pthread_cond_broadcast to wake up other customers so they can find the clerk.