1)
A thread array to hold all the possible customers.
For however many customers, create a thread. So if there are 7 customers, then there are 7 threads.

2)
Yes there is a scheduler for all the threads,

3)
Two mutex's are used, alternating between the two and occasionally locking/unlocking both.
One is used to lock then continue iterating/finding idle clerks, then the second mutex is locked to edit the queue's.

4)
Yes the main thread will be idle by pthread_join().

5)
I'm using a linked list queue to hold the customer information, this will allow to hold the customer criteira and a next,
the linked list queue will be able to iterate over the linked list.

6)
The data structures will not be modified concurrently, since I am using mutex lock. Anytime the thread is
accessed or the queue is being modified, the mutex will be locked then unlocked afterwards to ensure the structure is not modified.

7)
Two convars are used, one for economy and one for business.
a)
The convar will represent the business=convar1 and the economy=convar2.

b)
Convar1 is associated with mutex1, and convar2 is associated
with mutex1. This is to help keep track of the mutex/convar, and since mutex1 is locked in order to edit clerks.
This blocks the customer thread, and once it is free, mutex2 is used to edit the queue.

c)
Once the convar has waited and been unblocked, the mutex will be re-acquired allowing for the thread that was blocked
to acquire the mutex and begin its operation that waited.

8)
Main:

Begin program, initialize the mutex's and convar's.
Create a customer array, then parse the customers file to get the customer info.
Create 4 clerk threads, initialize as Linked list

Create the customer threads in customer_threads, wait for the customer to 'arrive' using usleep, place the customer in their class
using their priority (1 =bus, 0=econ) initialize the customer as a thread. Each customer is a thread

Since the customer threads are now created, move to customer_entry

```
Customer Entry:
    get the customer info, and start time, then request a clerk this customer

Clerk Request:
    Lock mutex1 in order to traverse clerks and change the clerk and customer info.
    Find a clerk that is idle (clerk status =0), lock mutex2
    Now remove the current customer because it is being served
    Unlock the mutex2, once the edits have been made
    Unlock mutex1 and return Customer Entry

Customer Entry:
    Wait the according time for the customer to be served
    Lock the mutex1 to change the clerk status
    Signal back to clerk worker that the customer thread is complete

If finding an idle clerk was unsuccessful in Clerk Request:
    Clerk Request:
        All clerks are busy? Jump to clerk worker

    Clerk Worker:
        Wait until the customer can be served by an idle clerk,
        ~Signalled~ from customer entry that a customer thread is free,
        set the customer to current customer, break from while

        Check for an idle clerk,
        When found begin serving the customer,
        Lock mutex1 to change the clerk status, unlock mutex1
        Lock mutex2 to remove current customer from queue, unlock mutex2

        return to customer entry

    Customer Entry:

    Exit the threads, return to main

Main:
    Waited for the threads to terminate,
    Destroy mutex and convar,
    Print the waiting times
```