Hoksolinvan Chhun V01024951

Airline Control System (ACS) Design Document:

Essential Files:

ACS.c: The main program responsible for reading in current customer check-in lists, along with the customer and clerk thread functions.

Modifiedqueue.c: Queue implementation using linkedlists. Featuring operations such as: `enqueue()`, `dequeue()`, and `countNodes()`.

Modifiedqueue.h: Header file including function prototypes implemented in Modifiedqueue.c

Makefile: essential for compiling the programs necessary to create ACS

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

The program will include 5 clerk threads (according to the assignment details) and x amount of customer threads, depending on the total amount of customers that would be arriving for each session. The program will simulate a typical producer-consumer situation where the customers will be the producer (providing relevant information) and the clerk will be the consumer of those data (making judgments and decision based on information provided).
In the customer thread, initially there would be a certain wait period (based on their arrival time) and after this wait period has been settled using Usleep(). Thereafter, the thread will indicate that the customer has arrived and after that it goes through two different if conditions (one for economy queue and one for business queue) and checks which queue it would go into. According to the relevant if condition it would then get into the relevant critical section code encapsulated by the necessary mutex lock, then increment the total number of `economy_num` or `business_num`, then enqueue itself into the correct queue depending on its designation, and then finally signals to the clerk thread that it has finished providing the necessary information. Finally finishing the thread and then join the main thread.
In the clerk thread, the clerk will be waiting for customers and checks to make sure that there would still be customers left to be serviced. If there aren't any customers in any of the queue at the moment it would wait until a customer signal that they are in the queue. Then it would stop idling and check the Business queue first and if there are customers there it would dequeue and service them. Otherwise, the clerk thread would check the Economy queue if it is empty and perform the same procedure. At the end, after all customers have been serviced throughout the session it would finish the thread and deallocate any memory that might have been used for the functionality of this thread's function.

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

Throughout the program it will be implemented in a way that does not require a manager thread to manage the different responsibilities and processes of each of the threads. The threads will work independently from one another but they are coordinated through shared resources and synchronization mechanisms during access of the queues or global variables.

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

There will be 4 mutex locks:

- **economy_mutex**: responsible for guarding access of the economy queue through `enqueue()`, `dequeue()`, and keeping track of the number of economy customers enqueued.
- **business_mutex**: responsible for guarding access of the business queue through `enqueue()`, `dequeue()`, and keeping track of the number of business customers enqueued.
- **secondary_wait_mutex:** responsible for keeping track of the total amount of time for all customers, business customers, and economy customers.
- **primary_wait_mutex:** responsible for the purpose of simulating clerks waiting for customers to appear assisted with `pthread_cond_wait()` when both queues are currently empty but there are still remaining customers.

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

Throughout the program, the main thread will not be idle as it will be responsible for several critical tasks throughout the execution such as:

- **Initial Setup & Input Parsing:** processing command-line arguments, input-validation, reading customer file, initializing & populating customer data structures with values that were read from.
- **Thread management:** Creating customer and clerk threads. Additionally, waiting for all customer and clerk threads to complete using the `pthread_join` operation
- **Cleanup**: manages and destroys mutex locks and condition variables.
- **Final Report:** Prints out the final statistics of the simulation, including the total and average waiting times for both business and economy-class customers.

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

The customers will be represented using a `customers` struct that involves necessary details regarding their class_type, arrival_time, and service_time as well as additional information like their IDs and the moment in time where they got into the queue. There would be two queue data structures for `Economy` and `Business` queues. The queue data structure will be implemented using linkedlist for the customer struct that features three essential operations such as: `enqueue()`, `dequeue()`, and `countNodes()` to simulate the behavior of the queue.
The struct of the customer would be as follow:

```
struct Customers{

int customer_id;
int class_level; // 0 for economy, 1 for business
int arrival_time; // in 1/10th of second
int service_time; // in 1/10th of second
double customer_enqueue_time; // time for when the customer enqueued

};
```

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

To ensure that the data structures in the program are not modified concurrently which could lead to issues similar to race conditions and inconsistencies in data, each of the queues will have its own unique mutex lock that can only be accessed one at a time by customers and clerks for the purpose of enqueuing and dequeuing within critical section area(s). Additionally, regarding the global variable which keeps track of the total amount of waiting time it will be managed by its own unique mutex lock as well for this specific purpose. With proper mutex handling and implementation of proper synchronization mechanism this ensures that the data structure will not be modified concurrently.

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

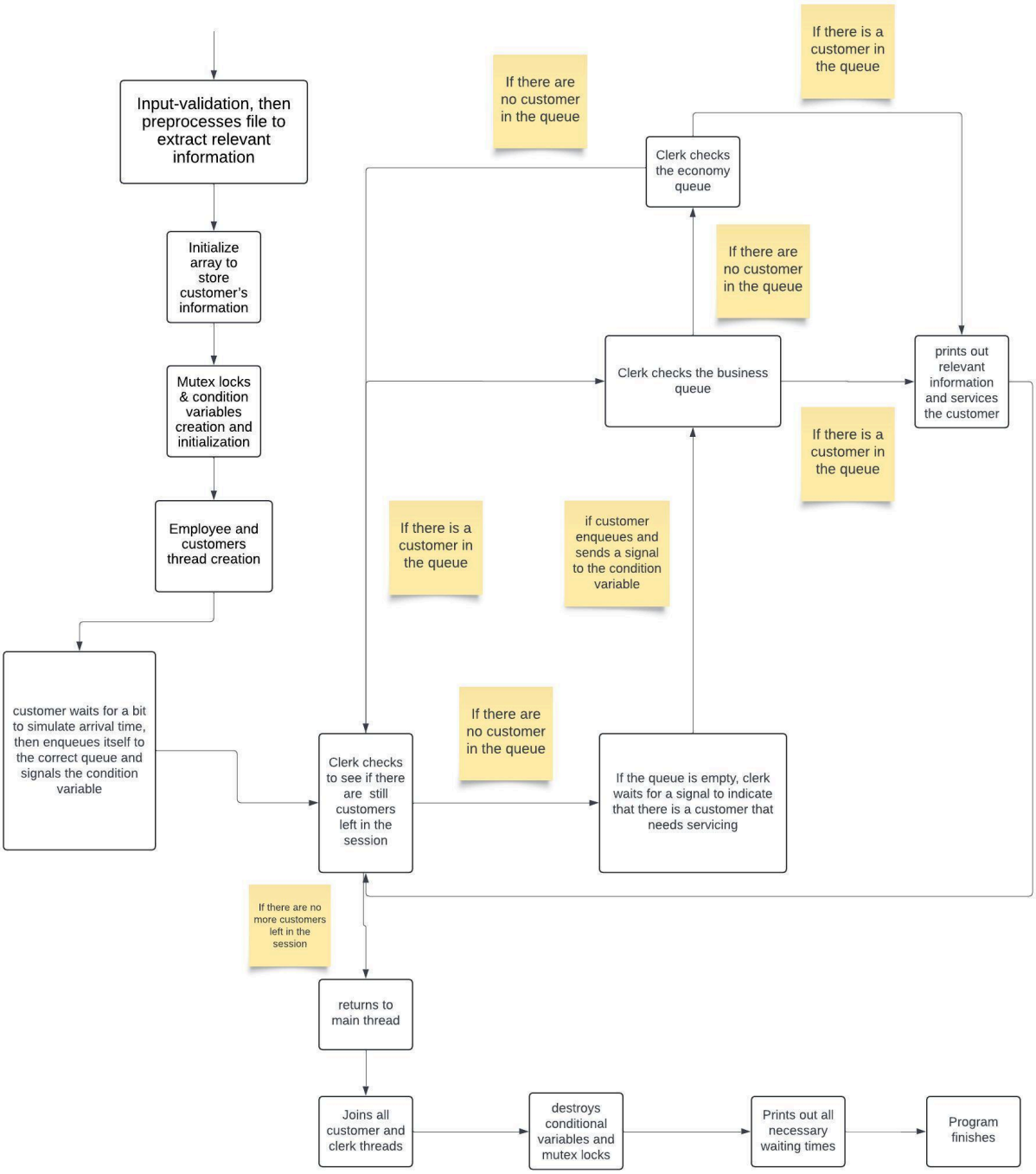
Throughout the entire program there will only be one condition variable (ACS_cond)

a. The condition that the convar will represent is the state of the queue and will be used for signaling when a customer has entered the queue.

b. The mutex that would be associated with this condition variable would be the primary_mutex, this mutex is used when the queue is empty and the clerks are busy waiting for a customer to arrive. Therefore, when a customer arrives a signal will be sent to the clerk through this convar and it tells the clerk that there are customers currently waiting to be serviced In one of the queues

c. Immediately following after pthread_cond_wait() there would be a pthread_mutex_unlock(&primary_mutex); and after all this there will be if conditions checking the business queue first to see if there are any customers that needs servicing and then checking the economy queue right after if there weren't any customers in the initial business queue.

8. Sketch


Input-validation, then preprocesses file to extract relevant information
Initialize array to store customer's information
Mutex locks & condition variables creation and initialization
Employee and customers thread creation
customer waits for a bit to simulate arrival time, then enqueues itself to the correct queue and signals the condition variable
Clerk checks to see if there are still customers left in the session
If there are no customer in the queue
If the queue is empty, clerk waits for a signal to indicate that there is a customer that needs servicing
If there are no more customers left in the session
returns to main thread
Joins all customer and clerk threads
destroys conditional variables and mutex locks
Prints out all necessary waiting times
Program finishes
Clerk checks the economy queue
If there is a customer in the queue
If there are no customer in the queue
Clerk checks the business queue
prints out relevant information and services the customer
If there is a customer in the queue
If there is a customer in the queue
if customer enqueues and sends a signal to the condition variable