Design of Assignment 2 code structure – ACS

By Victoria Shaw
V020304

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

There will be 6 threads. One each for each clerk and a 6th for the "dispatcher" that will add customers to the queues.

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

The threads are all sort of independent, but there is a single thread that controls when customers

get added to the queues that the other threads interact with. The threads all interact with the same queues, but there is no direct interaction beyond the dispatcher thread signaling the clerks when a new customer arrives.

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

Initially I planned to have many mutexes, with one each for the statistics associated with each Queue, another for anything using time and one other. That lead to several deadlocks when implemented. Instead there are two mutexes. One that is used for both B and E queues, and another that is used for “global statistics”. By separating these two it means that unrelated processes aren’t waiting on each other, but protects the critical “customersToProcess” variable (and others) and the queues.

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

Yes, it will be mostly idle after the setup. At the end it will compute the averages and other statistics

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

I will make a struct for customers. Before they are released to the queues they will be stored in an array of structs, and then they will be put into one of two queues based on their class. This structure contains the following information:

Customer Id
Customer arrival time (in 10ths of seconds)
Customer expected processing time (in 10ths of seconds)
and customer wait time (in seconds) -this ended up not being used.

There is another data structure called customerQueue that has an array of customer pointers, an index for the "head" and "tail" of the queue (used in the "back end" for push and pop) and a quantity value. This is used to keep the queue working smoothly.

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

(when I was sketching the answer to this ahead of time I just included something along the lines of I will have an array that "only one thread can access" and two mutex protected queues. But that proved to be incorrect since the array kept getting overwritten)

I had serious issues with data structures having their data modified. I tried several methods using a variety of mutexes and several ways of storing and passing data including:

Using a "Queue.c" implementation based on a linked list,
Using that same queue implementation with linked lists with the functions all in the main file.
Using that queue implementation with the array and queues declared in main and passed to the threads.
Using that same queue implementation with the data structures declared in global scope

All of the above caused some serious issues with overwriting data they shouldn't, although having the array and queues in global scope had the fewest errors. (For fun I have included a screenshot of when the memory got so messed up that it started printing my actual code. I have never in my life seen that)

All of the above eventually had the same very simple mutex algorithm of two mutexes and one convar that I was fairly certain was correct.

I eventually tried bringing all of the shared data structures into the global scope since passing an array through (void *) was causing most of the issues. This solved some of them. I knew it was the fact that passing it through a (void *) was the issue because it didn't overwrite data when I had the dispatcher as a function and passed the array and queue addresses the usual way.

The final design:

The array is in global scope and (finally) can only be accessed by the dispatcher thread, so race conditions are highly unlikely. The values associated with the array are protected by the Global Numbers mutex. The two queues are similarly located in global scope and can legally be accessed by all of the threads. Due to this, they are both protected by the same queue mutex, and their associated variables are protected by global numbers. The single mutex for the queue means that deadlocks no longer happen.

The original response: There are 4 primary structures: two queues, 1 array, and one collection of global statistics. The two arrays will have the same mutex to protect them, the array is only accessible by one thread, meaning that there shouldn't be any race conditions.

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

Again, my original design reflected my lack of knowledge of convar and mutext. My original plan was to have up to 3 condition variables. One each for the queues, and another for the global variables to have the average wait times computed by main throughout the process.

The final design I simplified to uses only one condition variable.

Convar 1 - condQueue:

> This condition variable will represent the changing state of the queues, and it is associated with the queue mutex. This is so that the dispatcher can “wake up” the clerks when a new customer is added to the queue just in case there was a gap in the queue and the clerks had gone into wait mode. It also serves to let the clerks alert each other when one realizes that there are no more customers left to process. This removes the infinite wait that happens when the customersToProcess variable still indicates that there will be more customers, so the clerk goes to sleep waiting for dispatch to indicate that a new customer was added, but then a different clerk gets the last customer, so the other clerk is forever waiting for another customer. By having the clerks signal when they process the last customer, they wake up and release the other clerks. Note that this does not kill clerks if they are still processing customers, and this can be seen in the output. Clerks will signal that they are “done” while some other clerks are still finishing up with their last customers. Once the clerks are signaled, they will check whether there are more customers to come or in the queues, and process customers, sleep or finish as needed.

(a) Describe the condition that the convar will represent.
(b) Which mutex is associated with the convar? Why?
(c) What operation should be performed once pthread cond wait() has been unblocked and reacquired the mutex?

8. Briefly sketch the overall algorithm you will use. You may use sentences such as: If clerk i finishes

service, release clerkImutex.

There are three primary pieces of code, but only two that run concurrently. The other is only setup and calculations

at the end.

The clerk functionality and algo is as follows:

Get queue mutex lock, then, if there is a customer in the business queue, pop that off, if not check the economy queue. Pop customer from the economy queue if there is one, if not, wait on the convar (there is only one, condQueue) to indicate that a customer has arrived at one of the queues. (dispatcher sends the signal with broadcast)

Once the clerk has popped a customer from one of the queues and has a customer to work with, release the queue mutex and print the appropriate information (Start time, clerk ID, customer ID), decrement the CustomersToProcess variable and log the starting time.

Usleep for the time dictated by the customers service time, then print the appropriate information (Start time, end time, clerk ID, customer ID) and calculate wait time. Add this wait time to the appropriate sum (economy or business as dictated by the customer) and then repeat.

If the variable CustomersToProcess is zero, signal to the other clerks that there are no more customers to process and end. If not, then search for more customers, and wait until they arrive.

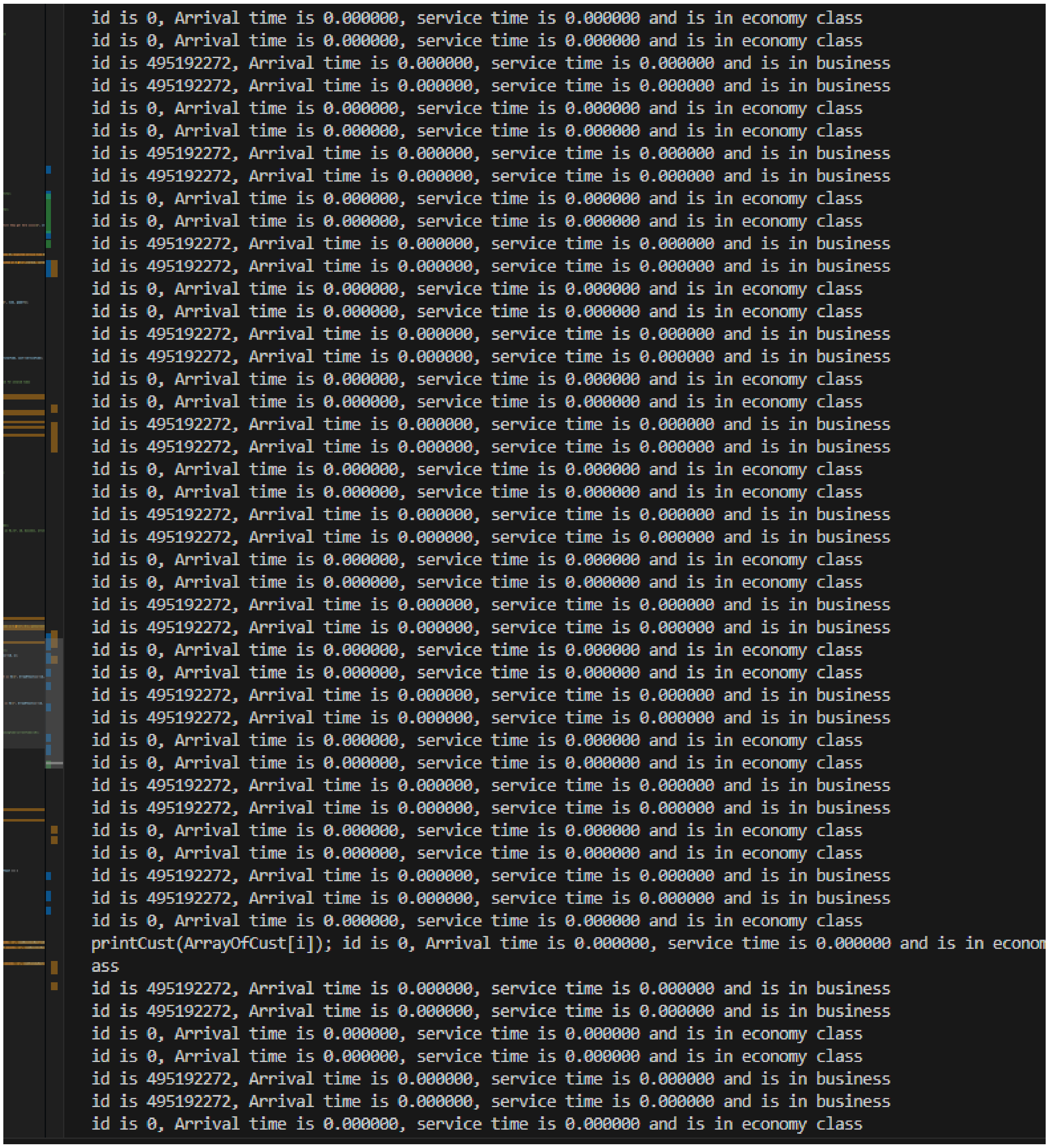
The dispatcher (which was honestly the hardest part) functionality is as follows:

After the contents of the file have been converted into an array of customers, and then sorted in order of arrival time by the sorting function, the dispatcher thread takes over.

The dispatcher repeatedly checks the current time, and compares it to the arrival time of the next customer in the array. If the current time is later than, or the same as the arrival time then the dispatcher sends that customer to the appropriate queue after getting the queue mutex lock and increments the array index. Note that this does not increment the “CustomersToProcess” variable. That starts at N, the return value of loadCustomers, and is decremented by the clerk only. After sending out the customer the dispatcher signals to the clerks that a new customer has arrived (using condvar) and releases the queue mutex so that the clerks can use it. If there are no customers ready (indicated by a counter for each Queue data structure), then it loops back and rechecks that the array index is less than the total amount of customers (which comes from the return value of the load customers function, same as the CustomersToProcess, the difference is that it doesn’t change). If the array index is not less than the total then it will end, and print a statement indicating that no more customers will be arriving. If the array index is less than the total, then the process repeats and checks the time again.

To see the various iterations of this code you can see that github repo I am using (I think, it's my first time using github for assignments) https://github.com/VShawFluenta/CSC360_A2

The promised image of my program printing part of its own code. (I was printing the "next" customer in the array in each iteration to see at what point it got corrupted and had an incorrect id and lost or corrupted all remaining data)



```
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
printCust(ArrayOfCust[i]); id is 0, Arrival time is 0.000000, service time is 0.000000 and is in econon
ass
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 495192272, Arrival time is 0.000000, service time is 0.000000 and is in business
id is 0, Arrival time is 0.000000, service time is 0.000000 and is in economy class
```