ECM2433- Coursework 1 report

My program does not have the functionality to be run multiple times to repeat the simulation, I had a limited version of it working but I ultimately removed it as I ran out of time and I wanted to make sure that what I submitted was a working program even if it does not have the full functionality of repeated simulations.

Design choices

I made the decision to implement my queue system as a singularly linked list as a linked list seemed like the best method of implementation for a FIFO queue and I could not seen any potential benefits of doing it as a doubly linked list or circular linked list.

I made a NODE struct that would represent each customer as a single NODE, this struct included the information of customerNo; the customers unique identifier, tolerance; the amount of time the customer was willing to wait in the queue for before leaving and currentWait; the current number of time intervals the customer had been in queue for. The NODE struct also contained the link to the next NODE in the list.

The QUEUE struct represents the queue of the post office as the singularly linked list, it contains information about the number of NODEs in it as well as markers for the first and last NODEs in the queue.

The SERVER struct represents each post office worker as a server/cashier/counter, each of which serves one customer at a time. The struct has an int variable of serviceTime representing a countdown of how long it will take to serve the current customer e.g. It could start at 3 and each time interval it would decrease by 1. It also has a bool variable called active that represents whether the SERVER is currently working/active or not.

The initialise function is used to initialise an empty QUEUE struct of the name provided as an argument by setting the count of the QUEUE to 0 and pointing both the front and rear NODEs of the QUEUE to NULL.

The isEmpty function is used to check if a queue is empty or not and is used later in the joinQueue function. It returns a value of 0 or 1 depending on if the rear element of the QUEUE given as an argument is equal to NULL.

The joinQueue function is used to add a new NODE with the values given as the 2nd and 3rd arguments to the QUEUE given as the 1st argument. It involves creating a temporary NODE and assigning it the values provided as well as giving it a currentWait value of 0 as it represents a customer who has just joined the queue. It then points to the next NODE as NULL as the new NODE will be at the back of the QUEUE. An if statement utilising isEmpty is then used to check if the QUEUE has any NODEs in it yet, if it does, the rear of the QUEUE is pointed to the new NODE as next and the new node becomes the rear of the QUEUE. If the queue is empty both the front and the rear of the QUEUE are pointed to the new NODE as it will be the only one in the QUEUE. Finally, the count of the QUEUE is incremented.

The deleteByKey function is used to delete the NODE which matches the key given as the 2nd argument from the QUEUE given as the 1st argument. The function creates 2 new NODEs, prev (previous) and cur (current), it then uses a while loop to iterate through the QUEUE searching for a NODE with customerNo that is equal to the key. If it finds a NODE fitting this criteria it points the NODE with the current NODE as next to the NODE that the current NODE has as next, effectively removing the NODE from the linked list as it is no longer linked to any other nodes. If I were to do this again, I would probably implement the QUEUE as a doubly linked list as it might make this function simpler and clearer.

The first thing in the main function is a check that the 4 required arguments are provided when running the executable, 1: the executable name (simQ), 2: the input file (e.g. testInput.txt), 3: the number of times to simulate (unused), and 4: the name of the output file(e.g. testOutput.txt). If 4 arguments are not provided an error message is sent to stderr. The next thing is the creation of the GSL random number generator which is used for various uses requiring pseudo random numbers. All the variables used in the main function are then initialised and some are assigned initial values. The QUEUE used is then created, assigned memory to and initialised, if there is not enough memory for the QUEUE then an error message will be sent to stderr and the program will exit. Line 130: remove(argv[3]); is used to remove any currently existing text files with the same name as the output file as to write to the output fopen append mode is used to prevent overwriting the contents each time interval. The input file is then opened in read mode and checked that it is not NULL, if it is, an error message is sent to stderr. The values provided in the unput file are then read from the file using fscanf and assigned to the appropriate variables and the file closes, if the file is not in the correct format then an error is sent to stderr, the file closes and the program exits. An array of SERVER structs is then initialised and each struct in the array has its initial values assigned.

The first major while loop of the main function is active when the current time is less than the closing time of the post office, each loop newCust is set to 0 and a while loop for adding new customers to the QUEUE is true while newCust is less than a random number generated from the Poisson distribution of the average number of new customers given in the input file.

If there is space in the QUEUE a new customer with the provided values including tolerance as a random number generated from the Poisson distribution of the average tolerance provided in the input file is added and count, numQueue and newCust are all incremented. If there is no space in the queue newCust and unfulfilled are both incremented to represent a customer who arrived and left before joining the post office queue due to lack of space. A new NODE temp is assigned to be the same as the front NODE in the QUEUE.

A for loop is used to iterate through each server checking if it is active and if it is, decrementing the service time and checking if the new service time is equal to 0, if it is this represents that the customer has finished being served so the active variable of the SERVER is set to false, busyCounters is decremented and fulfilled is incremented.

Another while loop is then used to iterate through each customer, incrementing their currentWait and then checking each SERVER to see if any are false and if so setting active of the SERVER to true and the service time to be equal to a random number generated by Poisson distribution based on the average service time provided by the input file, deleting the NODE and incrementing/decrementing appropriate variables.

The currentWait of the customer is then incremented and is this new value is equal to the tolerance of the customer the NODE is deleted, appropriate variables incremented/decremented, and temp is pointed to the NODE after temp. If the values are not equal temp is pointed to the NODE after temp.

The output file is opened in append+ mode to prevent overwriting each loop and using fprintf the necessary information is written to the file which is then closed and currentTime incremented.

The second major while loop is similar to the first but the code for adding to the QUEUE is removed as the second major loop represents the time after closing time where no new customers are arriving and it is just the remaining customers in the queue being served. I set this loop to be true until the extra time past closing time was equal to the closingTime divided by 5 as I thought that realistically the post office would have to close sometime after closing even if people were still in the queue and I thought for this purpose closingTime/5 would be a good approximation. The other differences are that each loop as well as incrementing the current time, the extra time is also incremented as well as being written to the output file.

Finally, the QUEUE q and the GSL RNG r are freed.

Some assumptions I made were that the times to join and leave the queue were 0-time intervals as well as the time to start being served and leave after being served were also 0 time intervals.