Assignment 6: Event Driven Simulation solution

Introduction: An *event-driven simulation* is a computer program that mimics the behavior of people or objects in a system in response to events that occur at certain times. The program must maintain a data object for each person or object (called an *actor*) and place it in a queue according to the time of its event. It then reads the queue in the order of the events and, for each event, causes the corresponding actor to do its actions scheduled for that time. The action of the actor may be to change its own state, change the state of the system, do something on behalf of another actor, or something else. Sometimes, the action will cause the actor to rejoin the event queue for a subsequent action. Sometimes, the action may add some other actor to the event queue or to another queue.

Assignment Description:

In this assignment, you will simulate customers arriving at a bank and standing in line in front of one of the tellers. People arrive at random intervals. Each person waits in his/her selected line until reaching the head of that line. When a person reaches the head of his line, the teller provides service for a random amount of time. After the service is completed, the person leaves the bank. The purpose of the simulation is to measure the average amount of time people spend between arriving at the bank and leaving the bank.

Assume that when there is a separate line for each teller, a newly arrived person joins the shortest line (or selects randomly among the set of equally short lines) and stays in that line until served. That is, no person leaves a line without being served, and no person hops from one line to another.

If a teller has finished serving a customer and there are no other customers waiting in its line, the teller selects the first customer from the line of another, randomly-chosen teller and serves that customer. If there are no customers waiting at all, the teller does other duties for a (small) random amount of time before checking the lines again.

The entire purpose of this simulation is to compare the performance of a single line serving all tellers *versus* separate lines for each teller.

Pre-requisites: knowledge of Structures and function pointer

Implementing your program

Your program should be called **qSim**. It needs to do several things:–

- Get and interpret the program parameters from the command line.
- Create a structure variable for each customer indicating his/her arrival time at the bank. Arrival times are determined from a uniform random number generator and the input parameters of the

simulation. Also create a structure variable for each teller, with a random idle time in the range 1-600 seconds. *All constants in this simulation must be defined symbolically.*

- Create a single *event queue* in the form of a linked list. The members of the linked list may be customer events or teller events.
- Place each object in the *event queue* sorted according to the time of its event. That is, the event with the earliest time is at the head of the queue, and the event with the latest time is at the tail of the queue. Note: Your program instantiates all the customer-arrival objects at the beginning of the program; as it creates each one, the object is given a random arrival time (sometime during the day) and then put into the queue.
- Play out the simulation as follows: take the first event off the *event queue*, advance a simulated *clock* to the time of that event, and invoke the action method associated with that event. Continue until the event queue is empty.
- Print out the statistics gathered by the simulation.

For this assignment, you will need to play the simulation twice — once for a bank with a single queue and multiple tellers and once for a bank with a separate queue for each teller. Draw some comparison about the average time required for a person to be served at the bank under each queue regime.

Here are some of the actions that can occur when an *event* reaches the head of the *event queue*:

- If the event represents a newly arrived customer at the bank, add that person onto the end of a teller line — either the common line (in the case of a bank with a single line for all tellers) or to the shortest teller line. If there are several equally short teller lines, choose one at random.
- If the event represents a customer whose service at the bank has been completed, collect statistics about the customer: in particular, how long has the customer been in the bank, from arrival time to completion of teller service. After collecting the statistics, the customer leaves the bank and its **Event** object is deleted.
- If the event represents a teller who has either completed serving a customer or has completed an idle time task, gather statistics about that teller. If there is no customer waiting in any line, put a teller event back into the *event queue* with a random idle time of 1-150 seconds.

If there is a customer waiting in line, remove the first customer from its line, generate a random service time according to the input parameters of the program, and add *two* events to the event queue, sorted by time. One is a customer event and represents the completion of that service. The other event is a teller event representing completion of a service and to look for the next customer (or to idle).

Implementation Details:

You must define one or more structures that allow you to represent *Events*, *Customers*, and *Tellers*. It is suggested that the most important structure of your simulation should be **Event**. How you distinguish between events associated with customers and events associated with tellers is your choice.

Two methods of an **Event** are to add it to the **event queue** and to remove it from the **event queue**. In addition, each **Event** has an **action** method that is to be invoked when an **Event** is

removed from the **event queue**. The **action** method should somehow distinguish between *customers* and *tellers* and invoke the appropriate *action* function for that kind of event. These should perform the appropriate action for a customer or teller.

Each line in front of a teller should be implemented as a variable of a structure **tellerQueue**. Implement this structures using a linked list. You will need to write methods to add customers to the end of the linked list and to remove them from the head of the list. In addition, include a *static variable* in the **tellerQueue** class that indicates which line (i.e., instance of the **tellerQueue** class) is the shortest. If more than one line is equally short, select one at random.

The **eventQueue** itself should also be implemented by a linked list. You will need to write a method to add an **Event** to the **eventQueue** in time order. This method should iterate through the list until it finds an **Event** with a time greater than the **Event** being inserted, and then it should insert the new **Event** just before that **Event**. (Note: This implies that if two events happen to have the same time, the one added to the queue second goes after the event added first. Also, remember that it is possible that the event being added will become the new head of the queue.) You will also need a method to remove an **Event** from the **event queue** and invoke the **action** method of the event.

Every time a Function Pointer is called, it must be logged to show that you using function Pointer.

Input

The command line of your program should be of the following form:

./qSim #customers #tellers simulationTime averageServiceTime

The numbers of customers and tellers should be integers, and the simulation and average service times should be floating point numbers in units of minutes. E.g

./qSim 100 4 60 2.3

should be interpreted to mean that 100 customers and four tellers should be simulated over a period of 60 simulated minutes. The service time for each teller is an *average* of 2.3 minutes.

Random Number Generation

To generate random numbers, use the function **rand()**.

To generate random arrival times with a uniform distribution, the following is suggested:–

float arrTime = simulationTime * rand()/float(RAND_MAX);

It is useful to generate all customer arrivals at the beginning of the program and put them into the **event queue** in order of arrival time.

To generate random service times, the following is suggested:–

float serviceTime = 2*averageServiceTime*rand()/float(RAND_MAX);

Output

After your simulation has completed for both types of queuing regimes, you should print out a summary with the following information:–

- Total number of customers served and total time required to serve all customers
- Number of tellers and type of queuing (one per teller or common)
- Average (i.e., mean) amount of time a customer spent in the bank and the standard deviation
- Maximum wait time from the time a customer arrives to the time he/she is seen by a teller.
- Total amount of teller service time and total amount of teller idle time.

The information that you need to print should determine the statistics that you gather during the simulations.

GNUPlot: plot a graph between average amounts of time a customer spent in bank versus number of tellers (assume number of queue = 1)

Deliverables

You must submit the following:

Folder Structure: src, include, bin, output etc

- The files of your project, including **.h** files and **.c** files to implement your simulation, and the **makefile**. The target of the makefile should be called **qSim**.
- At least three different test cases that show the behavior of the bank under both queuing regimes. Show the command line and the output.
- A document called **README.pdf** summarizing your program, how to run it, and detailing any problems that you had.
- An analysis of your results — i.e., under what circumstances a single queue is better or worse than a queue per teller. You may include this analysis in your **README**

Before submitting your assignment, *clean* your project to get rid of extraneous files.