CS20 – Section 093 – Fall 2019
Programming Assignment 6 – Due Friday October 25 at 11:59PM

Event processing simulation using a transaction queue (60 points)
error – queue empty, queue full throw? no catch 4 places. remove server speed?

Part 1 – ADT definitions

Transaction ADT

Use a struct to define the data members of a Transaction. A transaction has a transaction ID and the number of work units needed to process it. The data members are:

transaction ID	-	string
work units	-	int

Server ADT

Use a struct to define the data members of a Server. A server has a flag telling whether it's free or busy, and its speed in processing units per clock tick. When a transaction is started by a server, the time that transaction will end is recorded. The data members are:

free (false) or busy (true)	-	bool
the speed of the server, in work units per clock tick	-	int
time that the current transaction will end	-	int

Queue ADT

Create a class named Queue. A queue element contains one transaction. The queue elements are physically stored in an array. The size of the array is specified by the user when a Queue object is instantiated. Model your queue using the description of Malik's structure and functions provided separately.

Place the structs and class declaration into a header file, your member functions into a .cpp file, and your simulation main into a second .cpp file.

Part 2 – Testing (45 points)

When you have coded the Queue class, test it using main # 1 (provided). Make sure your queue is works correctly before continuing.

Part 3 - The Simulation (15 points)

The simulation is driven by a clock, which starts at zero. Think of the clock in terms of seconds…1, 2, 3, and so on. As the simulation progresses, the clocks ticks, one tick at a time. The clock runs for a user-specified number of ticks, at which time the simulation ends.

Transactions arrive into the system at random times and are added to the transaction queue. The queue grows and shrinks as transactions enter and leave. Transactions are processed by a server. The server can process only one transaction at a time. If transactions arrive faster than the server can process them, the

excess transactions build up in the queue. If transactions arrive more slowly, the server will find itself with nothing to do, and will wait for a new transaction to arrive.

When the server is free, and if there is a transaction waiting in the queue, the server takes the transaction off the queue and begins processing it. The server processes one transaction at a time. When a transaction finishes running, the server becomes free.

The server is capable of processing *n* work units per clock tick. *n* is a whole number. A transaction consumes *w* number of work units. For example, if a transaction starts, and needs to consume 10 work units before it's finished, and the server is capable of processing 2 work units per second, the transaction will be in process for 5 clock ticks.

Prompt the console user for the following information:

number of clock ticks to simulate
probability that a transaction will be generated each new clock tick
server speed: number of work units per clock tick
highest and lowest number of work units per transaction

Simulation overview:
When the clock ticks, the first step in the simulation process is to generate a new transaction based on the probability entered by the user. For example, if the user enters 50, there is a 50% chance that a transaction will be generated each time the clock ticks. When you generate a transaction, give it a unique ID. Randomly generate the number of work units for the transaction, using the high/low range specified by the user.

When the clock ticks, the second step in the simulation is to check the server. It might be processing a transaction which finished this clock tick. If the transaction is done, the server is marked free. A transaction will always consume a whole number of clock ticks: this means that during one clock tick, a transaction will either start, or complete, but not both. Your simulation will have one server of the speed specified by the user.

When the clock ticks, the third step in the simulation is to make the server busy, if possible. This happens when the server is free and there is a transaction in the queue. Mark the server busy. Remove the transaction, calculate the time that the transaction will end, and save it in the server. The ending time is calculated as follows: current time + (number of work units / server speed). Fractions of work units are ignored.

When the simulation is over, produce a report similar to the following:

***** Simulation ended *****
Number of clock ticks – 1000
Number of transactions generated – 543
Number of transactions processed – 387
Number of work units consumed by the completed transactions - 1086
There was (or was not) a transaction being processed when time was up
Number of unprocessed transactions left in the queue – 155

There are many ways to design an event processing simulator. For a system with one server, try the following:

```
main program:
read input parameters from user:
        number of clock ticks
        probability of a transaction arriving on a clock tick
        low and high ranges for processing units per transaction
        number of work units per clock tick for the server
clock = 0
server busy = false
while (clock < number of clock ticks)

        add transaction:
        if we're adding a transaction to queue
                generate transaction
                add to queue
        endif

        process transaction:
        if server not busy
                if there is an available transaction
                        put transaction into server
                        remove transaction from queue
                        put transaction's ending time into server
                        mark server busy
                endif
        endif

        end transaction:
        if server busy and transaction's ending time = now
                remove transaction
                record data related to transaction finishing
                mark server free
        endif

        clock ++
end while
display ending statistics on console:
        number of clock ticks
        number of completed transactions
        number of processing units used
        was there a transaction in progress when clock expired?
        number of unprocessed transactions in queue
exit
```