Project 8

Queue Simulation

Full Name: Prashul Shrestha

Section #: 2

Project #: 8

Due Date: April 6, 2017

**Design** **Document**

**Introduction**

A **queue** is a sequence of elements, all the same type, to which elements can be appended at one end (the **rear** of the queue) and from which elements can be removed at the other end (the **front**). Queues are first-in-first-out (FIFO) structures. They mimic the behavior of such systems as people in lines, cars at traffic lights, and files to be printed.

Systems that involve queues are called **queueing systems**. A queueing system consists of one or more queues of elements waiting to be **served** by one or more **servers**. When an element is removed from the front of a queue, a server serves that element. How queues and servers interact and parameters such as the numbers of queues and servers, how often new elements arrive, and when items are dequeued and served determine the behavior of a queueing system.

A **queueing simulation** is a program that simulates a queueing system. A **probabilistic** simulation calls a pseudo-random number generator to determine if events occur at each tick of the simulation's clock and the details of those events.

**Data** **Structures**

The program uses **array of Queues []** which depends upon the user to enter the number of server as **queuecount** and **array of transaction time[]** to store the number of customer entering in a probabilistic time and gets enqueued in the array of Queues[] and the array of transaction time[] stores transaction time for each line.When the teller is free and the queue is not empty, the customer is dequeued from the corresponding queue and served and customer is added in the shortest **queue** and one or more customer is served at one time if the multiple server is free at same time. If the teller is not free, then the transaction time for that customer and that queue gets decremented each time it goes through the for loop of Duration set by the user such that we know how much time is taken for each customer. Each time we check if the server is free, new customer gets added in a random manner using **rand() % 100** which will be less than our set probability %. This way we create a queue simulation and get a snap or cout of each tick.

**Functions**

The program uses **four** functions to implement the Queue Simulation. The functions are called from main() and some are member function to return the result within the function which called it. The list of the functions are given below:

* Store\_Queue() : A constructor to initialize front, rear to NULL.
* ~Store\_Queue (): A destructor to empty the queues.
* Void enqueue(Item Time)– This will insert the item into the Queue of corresponding array..
* Item dequeue(): Returns the item front of the queue.
* Size\_t length(): Return the length of the queue;
* Int shortest\_queue()– to return the shortest queue of all.
* Ostream operator << (): to print the items in the queue.

**Menu**() is a the main function from where we first create an object or an array instance of class **Store**\_**Queue** called **Line[queuecount]** and which helps invoke the functions such as Line[i]**.enqueue**(time);

**The Main Program**

**Menu**() is a the main function from where we first create an array of object or an instance of class **Store\_Queue** and which helps invoke the functions such as **Line[i].enqueue**(time); to enqueue items into the stack. The program uses **array of Queues []** which depends upon the user to enter the number of server as **queuecount** and **array of transaction time[]** to store the number of customer entering in a probabilistic time and gets enqueued in the array of Queues[] and the array of transaction time[] stores transaction time for each line.When the teller is free and the queue is not empty, the customer is dequeued from the corresponding queue and served and customer is added in the shortest **queue** and one or more customer is served at one time if the multiple server is free at same time. If the teller is not free, then the transaction time for that customer and that queue gets decremented each time it goes through the for loop of Duration set by the user such that we know how much time is taken for each customer. Each time we check if the server is free, new customer gets added in a random manner using **rand() % 100** which will be less than our set probability %. This way we create a queue simulation and get a snap or cout of each tick.

User Document

A **queue** is a sequence of elements, all the same type, to which elements can be appended at one end (the **rear** of the queue) and from which elements can be removed at the other end (the **front**). Queues are first-in-first-out (FIFO) structures. They mimic the behavior of such systems as people in lines, cars at traffic lights, and files to be printed.

A **queueing simulation** is a program that simulates a queueing system. A **probabilistic** simulation calls a pseudo-random number generator to determine if events occur at each tick of the simulation's clock and the details of those events.

The main program named **main**.**cpp** can be compiled and run, using the code:

**g++ main**.**cpp**

**a.out**

**g++** function will compile the function and make it ready to be run using **a.out**. The function will prompt the following output:

**Ouptut:**

**Hello user. Lets use Queue Simulation to carry out Simulation for a Grocery Store.**

**--------------------------------------------------**

**Enter the program Simulation datas.**

**The number of queue/server pairs: 4**

**The probability that a customer arrives in one tick (%): 80**

**The maximum duration of a transaction in ticks: 12**

**The duration of the simulation in ticks: 100**

**Enter the Seed value ->3**

**0 8**

**0**

**0**

**0**

**1 7**

**0**

**0**

**0**

**2 6**

**0**

**0**

**0**

**3 5**

**9**

**0**

**0**

**4 4**

**8**

**7**

**0**

**5 3**

**7**

**6**

**0**

**.**

**.**

**96 3 77 78 79 80 87 88**

**2 81 82 83 84 89 90**

**6 85 91 92 93 94 95**

**1 52 64 65 76 86 96**

**97 2 77 78 79 80 87 88 97**

**1 81 82 83 84 89 90**

**5 85 91 92 93 94 95**

**0 52 64 65 76 86 96**

**98 1 78 79 80 87 88 97**

**0 81 82 83 84 89 90**

**4 85 91 92 93 94 95**

**9 52 64 65 76 86 96**

**99 0 79 80 87 88 97**

**8 81 82 83 84 89 90**

**3 85 91 92 93 94 95**

**8 52 64 65 76 86 96**

**60 customers waited an average of 10ticks.**

**The longest time a customer waited was 81 ticks.**

**23 customers remain in the lines.**

**--------------------------------------------------**

**Thank you for using Queue Simulation Program.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Summary**

Queue Simulation is used in real-world experiences therefore it is very helpful in generating a sequence of events happening in and around us. This program helped me explore my knowledge further in the use of simple Queue programming to cumulate events which can be used later to develop software for day to day use for calculating the average earnings and the growth rate of a store.

Since, this program uses simplistic array of Queues to hold data for each line or servers, we can use it to implement the current database of students in a university or of a general retail store. Queues can be used for any situation where you want to efficiently maintain a First-in-first out order on some entities and are used literally in every type of software development. For instance, if we have a web-site which serves files to thousands of users. We cannot service all requests, you can only handle say 100 at once. A fair policy would be first-come-first serve: serve 100 at a time in order of arrival. In that case, using Queue would be the most appropriate data structure. Similarly, in a multitasking operating system, or in Android devices, small chips cannot handle all request at once, all requests may be batched up in one bundle and things may be carried out using Queue.