**Project-8**

**QUEUEING SIMULATION**

**Name: Pawan Dhungana**

**Project no: 8**

**Due Date: November 08, 2018**

**Design Document**

**Introduction**

A queue is a sequence of elements, all of 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 that mimic the behavior of such systems as people in lines, cars at traffic lights, and files to be printed.

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 how often servers remove elements from queues 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, and their parameters, at each tick of the simulation's clock. Here, a program is designed where we are using multiple server probabilistic simulation like in banks so that more users can be served in a given time duration.

**Data Structures**

Here, we used data[CAPACITY] array to hold the capacity of the queue. We then used array-based implementation to enqueue, dequeue and print the items in the program. There are multiple queues for multiple servers for which we have an array of objects of queue. Each queue represents different line waiting from different servers. Another array is used to hold transaction time for each queue.

**Functions**

The program uses the default constructor that initializes the queue to be empty. The function enqueue() is used to enter the user in the queue. dequeue() is used to remove the user from the queue. The size() function returns the size of the queue. The friend function is used to print out the contents in the queue. The inline function empty() is a boolean function that checks if the queue is empty or not. We also use the next\_index() function defined under private to hold the array in a circle.

**Main Program**

In the main program, simulation works for each tick of time. There are two arrays in the main program. The first array represents an array of queue and the other represents an array of transactions. To find the smallest queue, first trace of bubble sort and for loop is used to allow the multiple server being used at a same time.

**User Document**

A queue is a sequence of elements, all of 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 that 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, and their parameters, at each tick of the simulation's clock.

Here, a program is designed where we are using multiple server probabilistic simulation like in banks so that more users can be served in a given time duration.

The program's name is Project8.cpp, to compile and run it, simply enter:

g++ Project8.cpp

a.out

**A run of the program might look like this:**

Enter these parameters of the simulation:

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: 120

Enter a random number seed: 3

1 4

0

0

0

2 3

9

0

0

3 2

8

5

0

.

.

.

118 1 79 82 91 96 100 102 109 114 116

6 80 87 88 97 101 103 110 115

3 90 92 94 98 104 107 111 112

12 93 95 99 105 108 113 118

119 0 79 82 91 96 100 102 109 114 116

5 80 87 88 97 101 103 110 115

2 90 92 94 98 104 107 111 112

11 93 95 99 105 108 113 118 119

120 3 82 91 96 100 102 109 114 116

4 80 87 88 97 101 103 110 115

1 90 92 94 98 104 107 111 112

10 93 95 99 105 108 113 118 119

68 customers waited an average of 16 ticks.

The longest time a customer waited was 41 ticks.

32 customers remain in the lines.

**Code Listing:**

#include<iostream>

#include<cstdlib>

using namespace std;

class Queue

{

public:

typedef int Item;

static const int CAPACITY = 30;

//constructor

Queue()

{

front = 0;

rear = CAPACITY-1;

count = 0;

};

//constant member function

bool empty() {return count == 0;}

int size() {return count;}

//Modification member functions

void enqueue(Item entry);

Item dequeue();

//friend function

friend ostream& operator <<(ostream & out\_s, Queue q);

private:

//Data members

Item data[CAPACITY];

int front;

int rear;

int count;

//private member function

int next\_index(int i)

{

return (i+1) % CAPACITY;

}

};

int main()

{

int number;

int ARV\_PROB;

int MAX\_TRANS\_TIME;

int DURATION;

//Asking for details from the user

cout<<"\nEnter these parameters of the simulation: "<<endl;

cout<<"The number of queue/server pairs: "<<endl;

cin>>number;

cout<<"The probability that a customer arrives in one tick (%) : "<<endl;

cin>>ARV\_PROB;

cout<<"The maximum duration of a transaction in ticks: "<<endl;

cin>>MAX\_TRANS\_TIME;

cout<<"The duration of the simulation in ticks: "<<endl;

cin>>DURATION;

//An array of Queue

Queue line[number];

//An array of transaction

int trans\_time[number];

//Initialization

for(int l=0; l<number; ++l)

{

trans\_time[l] = 0;

}

int count = 0;

int entrytime;

int wait\_sum = 0;

int wait = 0;

int seed;

int rCustomer = 0;

int longWait = 0;

cout<<"Enter a random number seed : "<<endl;

cin>>seed;

cout<<"\n \n \n";

srand(seed);

for(int time=0; time<DURATION; ++time)

{

int least =0;

if(rand()%100 < ARV\_PROB)

{

int start = 1;

while (start < number)

{

//getting the line with the smallest size

if(line[least].size() > line[start].size())

{

least = start;

}

++start;

}

line[least].enqueue(time);

}

for (int i = 0;i<number; ++i)

{

if(trans\_time[i] == 0)

//tailor is free

{

if(!line[least].empty())

{

entrytime = line[least].dequeue();

wait\_sum += time - entrytime;

wait = time - entrytime;

if(wait>longWait)

longWait = wait;

++count;

trans\_time[i] = rand() % MAX\_TRANS\_TIME + 1;

}

}

else

--trans\_time[i];

cout<<endl;

}

cout<<time<<" ";

for(int j = 0; j<number; ++j)

{

cout<<"\t"<<trans\_time[j]<<"\t"<<line[j]<<endl;

}

}

for(int i = 0; i<number; ++i)

{

rCustomer = rCustomer + line[i].size();

}

cout<<count<<"customers waited an average of "<<wait\_sum/count<<"ticks."<<endl;

cout<<"The longest time a customer waited was "<<longWait<<"ticks."<<endl;

cout<<rCustomer<<"customers remain in the lines. "<<endl;

return 0;

}

void Queue::enqueue(Item entry)

{

rear = next\_index(rear);

data[rear] = entry;

++count;

}

Queue::Item Queue::dequeue()

{

Item dequed;

dequed=data[front];

front=next\_index(front);

--count;

return dequed;

}

ostream& operator<<(ostream& out\_s, Queue q)

{

int j = q.front;

for(int i = 0; i<q.count; ++i)

{

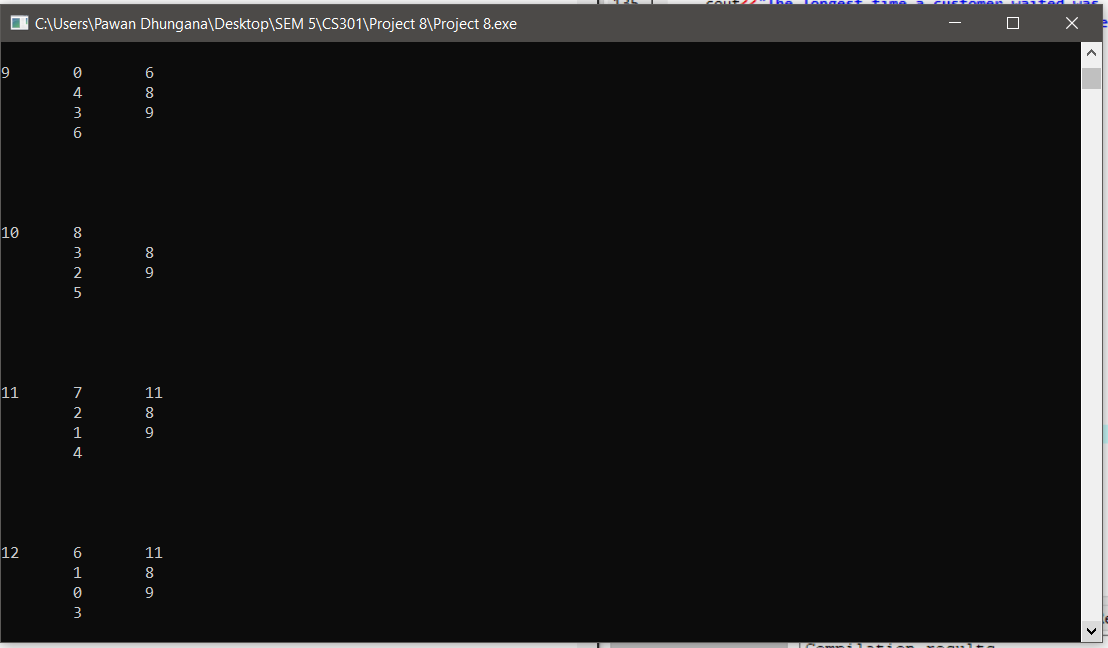
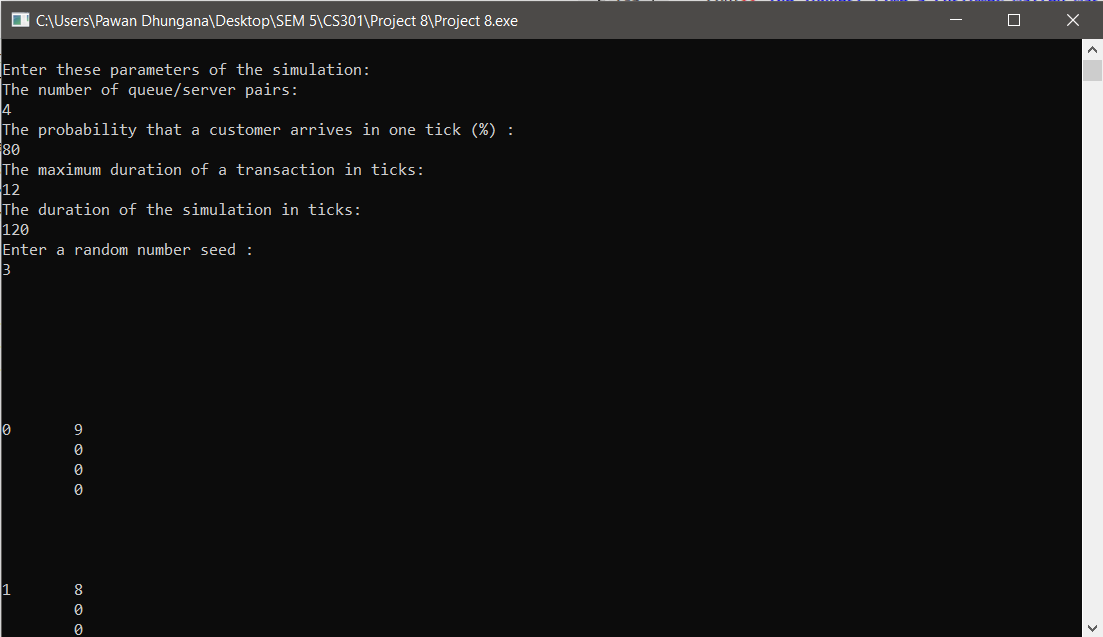
out\_s<<q.data[j]<<' ';

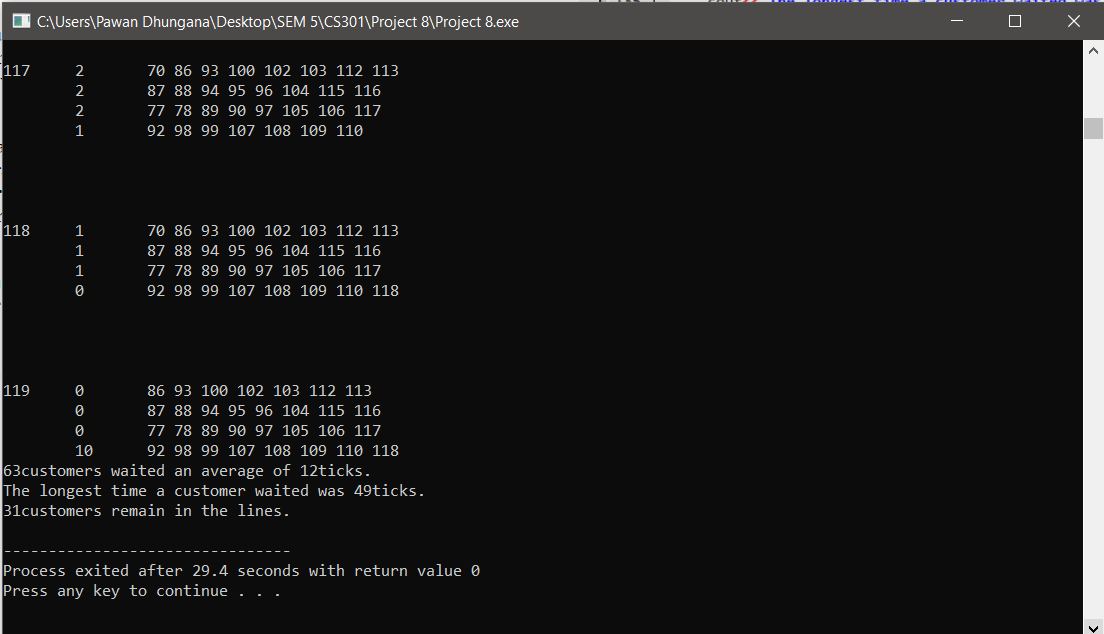
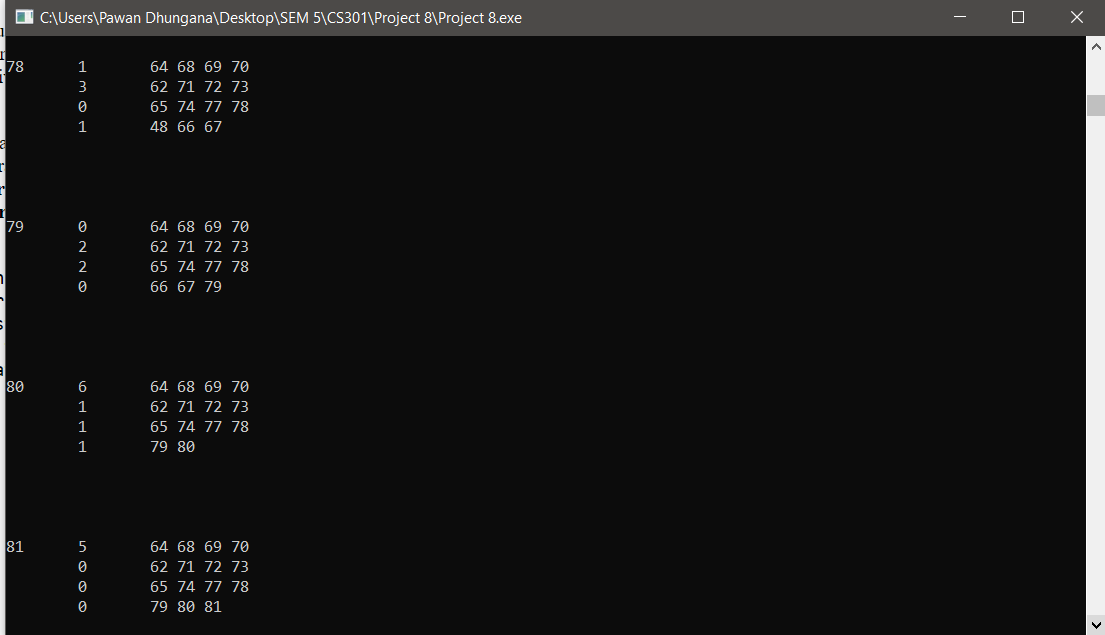
j = q.next\_index(j);

}

return out\_s;

}

**Test Document**

****

**Summary**

In this project, we implemented a queuing simulation program to multi user multi server transaction at the given amount of time. Here, I learned the importance of array and for loop to create multiple servers and let the simulation work at every tick of time. We learned the real-time application of the queue ADT by applying in multiple server and multi user program.