Sushil Khanal

Section 1

Project 9

**QUEUEING SIMULATION**

**Introduction**

A queue is a sequence of elements of the same type, to which elements can be appended at one end (the end) and from which elements can be removed at the other end (the start). Queues are first-in-first-out (FIFO) structures that interprets 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.

**Data Structures**

The program contains .h file, .cpp file and main file. .h contains class, private, public and function prototypes. The main file receives instructions from the user and implements the function accordingly.

**Main Program**

All the codes from .h and .cpp files are linked with the main program. The main program interacts with the user to give desired output.

**Code**

Queue.h

//Data member

class Queue {

private:

Item data[capacity] int front;

int rev;

int count;

//Data member

//Private member function

int next\_index(int i)

{

return (i + ) % capacity;

}

Public:

type set\_Item;

static count int = 50;

//constructor

Queue()

{

front = 0;

rear = capacity - 1;

count = 0;

}

void enqueue(item entry);

Item deque;

//Constant member functions

int length()

{

return count;

}

bool empty()

{

return count ==0;

}

//friend function

friend ostram&operator(ostream& outs, deque)

}

Queue.cpp

Queue::Queue()

{

front = rear = capacity - 1;

count = 0

}

Void Queue::enqueue(Item entry)

{

if (count < CAPACITY) {

rear = next\_index(rear)

data[rear]

= entry;

++count;

}

Queue::Item Queue::dequeue()

{

Item it;

if (count > 0) {

it = data[front];

front = next\_index(front);

--count;

return it;

}

}

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

{

item it;

int m = q.length();

for (int i = 0, i < m; i++) {

it = q.dequque();

out\_s << it << "";

q.enqueue(it);

}

return out\_s;

}

Project9.cpp

const int DURATION = 120;

const int ARV\_PROB = 25;

const int MAX TRANS\_TIME = 6;

int main()

{

Queue line; //number of ints

int time;

int count; //number of customers served

int entry\_time; // when each customer arrived

int wait\_sum; //sum of wait time

int seed int trans\_time; //Time remaining in a transaction

cin >> seed;

rand(seed);

count = 0;

wait\_sum = 0;

trans\_time = 0;

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

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

line.enqueue(time); //Arrival?

if (trans.time == 0) {

if (!line.empty()) {

empty\_time = line.dequeue();

wait.sum += (time\_entry\_time);

++count;

trans.time = (rand() % MAX\_TRANS.Tne) + 1 else \_\_ trans\_time

//snapshot;

cout

<< time << " " << trans\_time << " " << line << endl;

}

}

}

}

**Conclusion**

In this project we learned about queuing simulation. This was a tough project.