Programming Assignment 2

Work done

by

Arun Prakash Themothy Prabu Vincent

NET ID : AXT161330

SPRING'17 CS/CE 6352

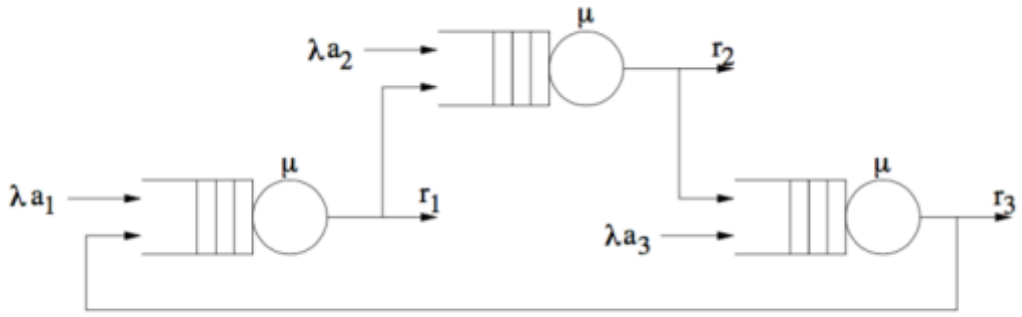
Performance of Computer Systems and Networks

CONTENTS

Chapter 1

Project Description

In this assignment, you will implement a simulation for a three-node unidirectional ring network. The queuing model for the network is illustrated below:



Packets arrive from outside the network according to a Poisson process with rate λ. With probability a_i, an arriving

packet enters the ith queue, where $a_1 + a_2 + a_3 = 1$. The service time for a packet at a Node is exponentially

distributed with an average service time of ^{1} seconds. When a packet departs from queue i, it departs from the μ

network with probability r_i, and it is forwarded to the queue of the next node with probability $1 - r_i$. The packets can only be transmitted in one direction around the ring. Each queue has infinite capacity.

Implement a discrete event simulation for the above system. For each of the experiments below, run the simulation until 500,000 packets have departed from the system.

- Let $a_1 = {}^1, a_2 = {}^1, a_3 = {}^1, r_1 = {}^1, r_2 = {}^2, r_3 = {}^1$, and $\mu = 8$. Plot the simulation and theoretical results for 244234 expected number of customers in each queue as a function of λ for $\lambda = 1, 2, \cdots, 9$. (You may have your program output numerical values, and then create plots using any plotting/graphing/spreadsheet program such as Excel.)

- Let $\lambda = 8$ and the other parameters are as above. Find the throughput and utilization for each queue. Compare the simulation results to theoretical results.

To be submitted:

1. Source code files. Be sure to include sufficient comments. Include your name and netid at the top of every source code file that you submit.

2. "Readme" file specifying OS (UNIX, Windows, etc.), compilier/platform, instructions for compiling and running the program.

3. File(s) containing output plots (PostScript, PDF, MS Word, or MS Excel). Be sure to label plots appropriately.

For C++, source code files should end with .cpp or .h extensions. For C, source code files should end with .c or .h extensions. For Java, source code files should end with a .java extension. Place all files in a single folder or directory named with your netid, e.g. xyz061000, and zip this directory into a single zip file. Upload this zipped file to eLearning.

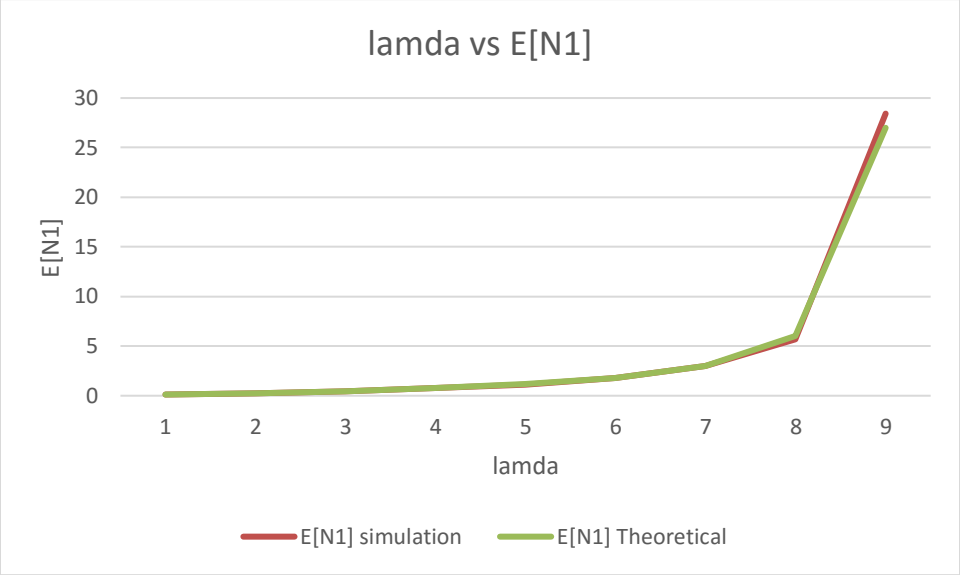
An example simulation for an M/M/1 system is available at:
http://www.utdallas.edu/%7Ejjue/cs6352/sim/

You may use this code as a template for your simulation, or you may write your own code. Under no circumstances may you use code from any other source. Programs will be checked using copy-detection software. If your code is found to be similar to another person's code, you will be subject to disciplinary action according to University policies. If you are unable to complete your project on time, submit whatever you have done. Partial credit will be given.

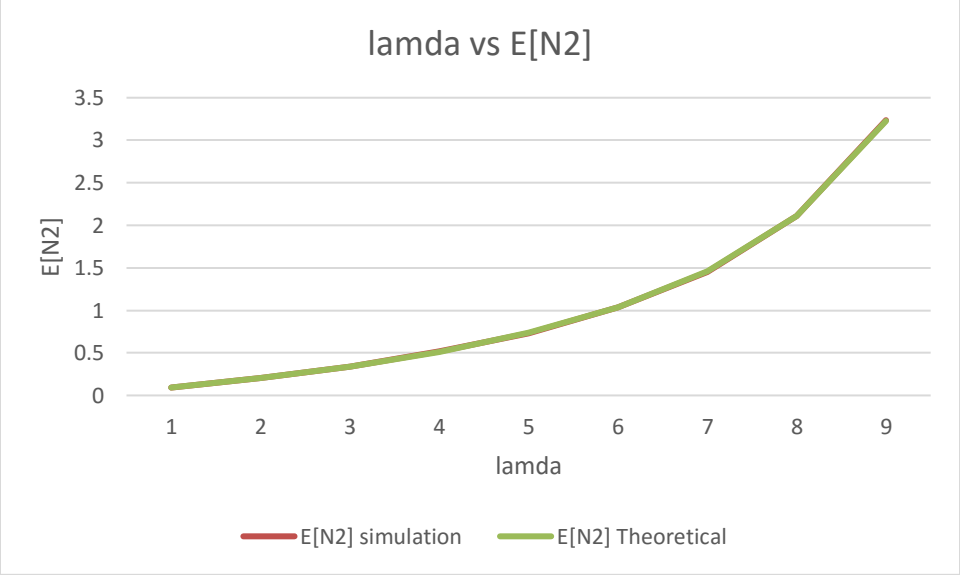
Chapter 2

Graphs

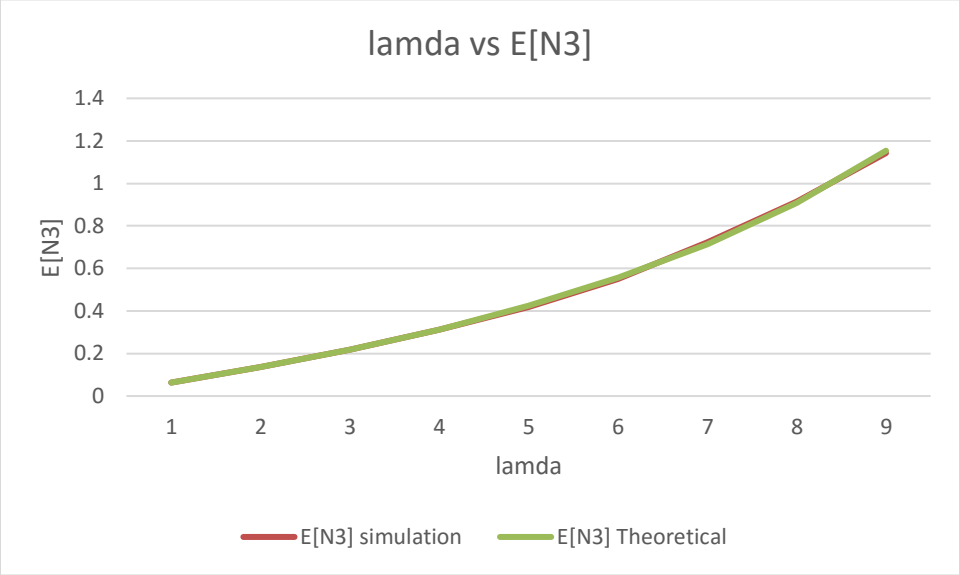
2.5 lamda(λ) vs Average Components of Queue 1 E[N1]



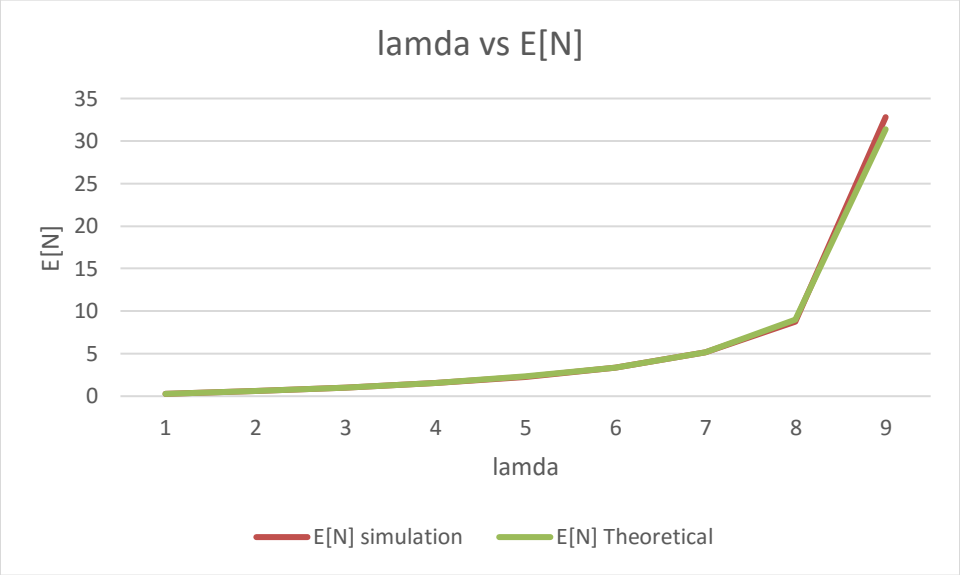
2.6 lamda(λ) vs Average Components of Queue 2 E[N2]



2.7 lamda(λ) vs Average Components of Queue 3 E[N3]



2.4 lamda(λ) vs Average Components in the system E[N]



Appendices

Appendix A – Source Code

```
package proj2;

public class Event {

        int type; /* Type of Event 1-Arrival of packet for queue 1
                              2-Arrival of packet for queue 2
                              3-Arrival of packet for queue 3
                                 0-Departure */
    double timeStamp; // Timestamp of Event

         public Event(double time, int t){
                    timeStamp = time; // Initialization
                    type = t;
          }


}


package proj2;
import java.util.LinkedList;
import java.util.List;
import java.util.Stack;

public class EventList {

        public static LinkedList<Event> eventList = new LinkedList<Event>();

        public static void listInsert( double timeStamp, int type){


        // System.out.println(timeStamp + "  " + type);
          int eventAdded = 0; // to indicate if event is added to list
          Event evnt = new Event(timeStamp, type); //creating a Event

          if(eventList.isEmpty()){ // if the event list is empty
                    eventList.add(evnt);
                    //System.out.println("Arrival Event Added to List");
          }
          else // Insert event in the correct chronological position
          {
                    int iterator = 0;
```

```
                while(iterator < eventList.size()){

                        Event temp = eventList.get(iterator);
                        if(temp.timeStamp < evnt.timeStamp){ //if arrived event has greater time
stamp
                                ++iterator ;
                        }
                        else{
                                eventList.add(iterator, evnt); //if current event has smaller
timestamp it is added to the list
                                eventAdded = 1; // set the event added status to success
                                break;
                        }
                }
            if(eventAdded != 1) // checking if the event the event is added
                eventList.addLast(evnt); /* the event has larger timestamps than
                                 all the available events in the list*/
              //  System.out.println("Arrival Event Added to List");

          }

        }

        public static Event listGetEvent(){

          if(eventList.isEmpty()) // checking if the list is empty
                return null;
          else
          {
                Event evnt = eventList.getFirst(); //Remove the element with the smallest
timeStamp
                eventList.removeFirst();
                return evnt; //return event
          }

        }


}

// Name : Arun Prakash Themothy Prabu Vincent
//NetId : AXT161330

package proj2;

import java.io.File;
import java.io.FileNotFoundException;
```

```
import java.util.Scanner;

public class EventDriven {

        public static double seed = 1111.0;

    public static void main(String[] args) throws FileNotFoundException{


          Scanner scan;
        if (args.length > 0) { // if the input file is specified
          File inputFile = new File(args[0]); // read the file
          scan = new Scanner(inputFile); // get a scanner of the input file
        } else { // the input file must be provided
          System.out.println("provide the location of the config file as the first input argument.");
          scan = null;
          System.exit(1);
        }

        double lamda = scan.nextDouble(); //arrival rate of packets
          double mu = scan.nextDouble(); //Service rate of packets by the node
          double a1 = scan.nextDouble();
          double a2 = scan.nextDouble();
          double a3 = scan.nextDouble();
          double r1 = scan.nextDouble();
          double r2 = scan.nextDouble();
          double r3 = scan.nextDouble();


          while(lamda<= 9){

          double lamda1 = a1*lamda; //arrival rate of packets for queue1
          double lamda2 = a2*lamda; //arrival rate of packets for queue2
          double lamda3 = a3*lamda; //arrival rate of packets for queue3
          int type1Arrival = 1; //type 1 for arrival for queue1
          int type2Arrival = 2; //type 2 for arrival for queue2
          int type3Arrival = 3; //type 3 for arrival for queue3
          int type1Departure = 6; //type 6 for departure from queue1
          int type2Departure = 7; //type 7 for departure from queue2
          int type3Departure = 8; //type 8 for departure from queue3

          double Rate;

          int noOfComponents = 0;
          int noOfDepartures = 0;

           int system1Size = 0 ; // components in service in node1 + wait Queue1
```

```
 int queue1Size = 0; // components waiting for service in queue1
 int system2Size = 0 ; // components in service in node 2 + wait Queue2
 int queue2Size = 0; // components waiting for service in queue2
 int system3Size = 0 ; // components in service in node3 + wait Queue3
 int queue3Size = 0; // components waiting for service in queue3

 double sysClock = 0.0;
 int noOfServer1Available = 1; // No of nodes available for packaging for type1
 int noOfServer2Available = 1; // No of nodes available for packaging for type2
 int noOfServer3Available = 1; // No of nodes available for packaging for type3

 Event currentEvnt ;

double timeStamp = exponentialRv(lamda1); //Production by machine 1
EventList.listInsert(timeStamp, type1Arrival); //Arrival of packet for queue 1

timeStamp = exponentialRv(lamda2);//Production by machine 2
EventList.listInsert(timeStamp, type2Arrival); // Arrival of packet for queue 2

timeStamp = exponentialRv(lamda3);//Production by machine 2
EventList.listInsert(timeStamp, type3Arrival); // Arrival of packet for queue 3

//Initialization of System parameters
int iter = 0;
int totalArrival = 0;
int totalEntered1 = 0;
int totalEntered2 = 0;
int totalEntered3 = 0;
double E_N1 = 0.0;
double E_N2 = 0.0;
double E_N3 = 0.0;
double Utilization = 0;
int k1=0;
int k2=0;
int k3=0;
double Util1 = 0.0; //utilization of node 1
double Util2 = 0.0; //utilization of node 2
double Util3 = 0.0; //utilization of node 3
int noOfQueue1Departures = 0;
int noOfQueue2Departures = 0;
int noOfQueue3Departures = 0;
 while(noOfDepartures < 500000){

        currentEvnt = EventList.listGetEvent(); // get event from the event List
        double prevClock = sysClock; //set the prev clock
        sysClock = currentEvnt.timeStamp; //update the system clock
        Util1 = calcUtilization(Util1,system1Size,sysClock,prevClock);
```

```
                E_N1 += (system1Size*(sysClock-prevClock));
                Util2 = calcUtilization(Util2,system2Size,sysClock,prevClock);
                E_N2 += (system2Size*(sysClock-prevClock));
                Util3 = calcUtilization(Util3,system3Size,sysClock,prevClock);
                E_N3 += (system3Size*(sysClock-prevClock));


        if(currentEvnt.type == type1Arrival){



                        ++system1Size ; //incrementing the system size
                        ++totalEntered1 ; //incrementing the arrival

                        if(noOfServer1Available >0 && system1Size>0){
                    EventList.listInsert(sysClock+exponentialRv(mu),
type1Departure);/*Creating a departure Event if server is available */
                                        --noOfServer1Available ;// setting server to busy
                                }

                        //if server is not available
                else{
                        queue1Size++ ;//component added to queue for processing
                }
                        /*Generating Arrival Event*/
                            Rate = lamda1 ; //Machine 1 generates components
                 timeStamp = sysClock + exponentialRv(Rate);
                 EventList.listInsert(timeStamp, type1Arrival);// Event added to the list
                        }

        if(currentEvnt.type == type2Arrival){



                        ++system2Size ; //incrementing the system size
                        ++totalEntered2 ; //incrementing the arrival

                        if(noOfServer2Available >0 && system2Size>0){
                    EventList.listInsert(sysClock+exponentialRv(mu),
type2Departure);/*Creating a departure Event if server is available */
                                        --noOfServer2Available ;// setting server to busy
                                }

                        //if server is not available
                else{
                        queue2Size++ ;//component added to queue for processing
                }
```

```
                    /*Generating Arrival Event*/
                        Rate = lamda2 ; //Machine 1 generates components
                timeStamp = sysClock + exponentialRv(Rate);
                EventList.listInsert(timeStamp, type2Arrival);// Event added to the list
               }

        if(currentEvnt.type == type3Arrival){


            ++system3Size ; //incrementing the system size
                    ++totalEntered3 ; //incrementing the arrival
                    if(noOfServer3Available >0 && system3Size>0){

                            EventList.listInsert(sysClock+exponentialRv(mu),
type3Departure);/*Creating a departure Event if server is available */
                            --noOfServer3Available ;// setting server to busy
                     }

                       //if server is not available
              else{
                 queue3Size++ ;//component added to queue for processing
               }
                    /*Generating Arrival Event*/
                        Rate = lamda3 ; //Machine 1 generates components
                timeStamp = sysClock + exponentialRv(Rate);
                EventList.listInsert(timeStamp, type3Arrival);// Event added to the list
                    }

         if(currentEvnt.type == type1Departure){

              --system1Size ; // Updating the system size - Decrementing it by 1
              ++k1;
              ++noOfServer1Available; // making a server available
              ++noOfQueue1Departures;
              double p = uniformrv();
              if(p <= r1) //the packet departs from the system
                ++noOfDepartures ; // incrementing the number of departures by 1
              else{
                    ++system2Size;
                    ++totalEntered2;
                    if(noOfServer2Available >0 && system2Size>0){
                    EventList.listInsert(sysClock+exponentialRv(mu),
type2Departure);/*Creating a departure Event if server is available */
                                --noOfServer2Available ;// setting server to busy
                    }
                            else
                                ++queue2Size; // it enters the queue 2 for processing
```

```
                }


            //checking if any component is waiting in the queue for service
            if(queue1Size>0){

              //Checking if the Server is available
            if(noOfServer1Available >0){
            timeStamp = sysClock + exponentialRv(mu);
                        EventList.listInsert(timeStamp, type1Departure); //Generating
departure Event
                    --noOfServer1Available ; //making the server busy
                    --queue1Size; // updating the queue
              }
            }
                 }

        if(currentEvnt.type == type2Departure){

              --system2Size ; // Updating the system size - Decrementing it by 1
              ++k2;
              ++noOfServer2Available; // making a server available
              ++noOfQueue2Departures;
              double p = uniformrv();
              if(p <= r2) //the packet departs from the system
                ++noOfDepartures ; // incrementing the number of departures by 1
              else{
                      ++system3Size;
                      ++totalEntered3;
                      if(noOfServer3Available >0 && system3Size>0){
                      EventList.listInsert(sysClock+exponentialRv(mu),
type3Departure);/*Creating a departure Event if server is available */
                                    --noOfServer3Available ;// setting server to busy
                      }
                              else
                                    ++queue3Size; // it enters the queue 2 for processing
              }


            //checking if any component is waiting in the queue for service
            if(queue2Size>0){

              //Checking if the Server is available
             if(noOfServer2Available >0){
             timeStamp = sysClock + exponentialRv(mu);
                        EventList.listInsert(timeStamp, type2Departure); //Generating
```

```
departure Event
                        --noOfServer2Available ; //making the server busy
                        --queue2Size; // updating the queue
                    }
                }
                     }

            if(currentEvnt.type == type3Departure){

                    --system3Size ; // Updating the system size - Decrementing it by 1
                    ++k3;
                    ++noOfServer3Available; // making a server available
                    ++noOfQueue3Departures;
                    double p = uniformrv();
                    if(p <= r3) //the packet departs from the system
                      ++noOfDepartures ; // incrementing the number of departures by 1
                    else{
                            ++system1Size;
                            ++totalEntered1;
                            if(noOfServer1Available >0 && system1Size>0){
                            EventList.listInsert(sysClock+exponentialRv(mu),
type1Departure);/*Creating a departure Event if server is available */
                                        --noOfServer1Available ;// setting server to busy
                            }
                                    else
                                        ++queue1Size; // it enters the queue 2 for processing
                    }


                  //checking if any component is waiting in the queue for service
                  if(queue3Size>0){

                     //Checking if the Server is available
                   if(noOfServer3Available >0){
                   timeStamp = sysClock + exponentialRv(mu);
                            EventList.listInsert(timeStamp, type3Departure); //Generating
departure Event
                        --noOfServer3Available ; //making the server busy
                        --queue3Size; // updating the queue
                    }
                }
                     }
            }
            double EN = (E_N1/sysClock) + (E_N2/sysClock) + (E_N3/sysClock)  ; // calculating the
Average Number of Components
            Utilization = (Util1/sysClock)+(Util2/sysClock)+(Util3/sysClock);
            double throughput = (noOfQueue1Departures/sysClock) +
```

```
(noOfQueue2Departures/sysClock) + (noOfQueue3Departures/sysClock);
        // double Et = (E_N1/noOfQueue1Departures) + (E_N2/noOfQueue2Departures) +
(E_N3/noOfQueue3Departures);

        // System.out.println(totalEntered1 + " " +  totalEntered1 + " " + totalEntered1);
        // System.out.println(k1 + " " +  k2 + " " + k3);

        System.out.println("_____
___");
            System.out.println("For Lamda = " + lamda);
            System.out.println();
            System.out.println("Simulation Result");
            System.out.println();
            System.out.println("Queue 1 - Parameters");
            System.out.println("E[N1] = " + (E_N1/sysClock));
            System.out.println("Utilization = " + (Util1/sysClock) );
            System.out.println("Throughput = " + (noOfQueue1Departures/sysClock));
            System.out.println("E[t1] = " + (E_N1/noOfQueue1Departures));
            System.out.println();
            System.out.println("Queue 2 - Parameters");
            System.out.println("E[N2] = " + (E_N2/sysClock));
            System.out.println("Utilization = " + (Util2/sysClock) );
            System.out.println("Throughput = " + (noOfQueue2Departures/sysClock));
            System.out.println("E[t2] = " + (E_N2/noOfQueue2Departures));
            System.out.println();
            System.out.println("Queue 3 - Parameters");
            System.out.println("E[N3] = " + (E_N3/sysClock));
            System.out.println("Utilization = " + (Util3/sysClock) );
            System.out.println("Throughput = " + (noOfQueue3Departures/sysClock));
            System.out.println("E[t] = " + (E_N3/noOfQueue3Departures));
            System.out.println();

            System.out.println("System - Parameters");
            System.out.println("E[N] = " + EN);
            System.out.println("Utilization = " + Utilization);
            System.out.println("Throughput = " + throughput);
             System.out.println();
             theoreticalCalculation(lamda,a1,a2,a3,r1,r2,r3,mu);
             lamda = lamda +1;
        }
        }



        /*function to generate uniform random variable */
        static double uniformrv(){
```

```
        int k = 16807;
        int m = 2147483647;
        seed = ((k*seed) % m);
    double r = seed / m;
        return r;
   }

       static double exponentialRv(double rate)
       {
          double expRV;
          expRV = ((-1) / rate) * Math.log(uniformrv());
          return(expRV);
       }

       static double calcUtilization(double util,int systemSize ,double sysClock,double
prevClock){
          double Utilization = util ;
          if(systemSize >= 1){
                  Utilization += 1 * (sysClock-prevClock);
          }
           return Utilization;
       }


       static void theoreticalCalculation(double lamda,double a1,double a2,double a3,double
r1,double r2,double r3, double mu)
       {
          double theta2=(0.6785)*lamda;
          double theta3=(0.25*lamda)+(theta2/3);
          double theta1=(0.5*lamda)+(0.75*theta3);

          double rho1=theta1/mu;
          double rho2=theta2/mu;
          double rho3=theta3/mu;

          double E_N1=(rho1)/(1-rho1);
          double E_N2=(rho2)/(1-rho2);
          double E_N3=(rho3)/(1-rho3);

          double ET1=E_N1/theta1;
          double ET2=E_N2/theta2;
          double ET3=E_N3/theta3;

          System.out.println("Theoretical Result");
          System.out.println();
          System.out.println("Queue 1 - Parameters");
          System.out.println("E[N1] = " + (E_N1));
```

```
            System.out.println("Utilization = " + rho1);
            System.out.println("Throughput = " + theta1);
            System.out.println("E[t1] = " + ET1);
            System.out.println();
            System.out.println("Queue 2 - Parameters");
            System.out.println("E[N2] = " + E_N2);
            System.out.println("Utilization = " + rho2 );
            System.out.println("Throughput = " + theta2);
            System.out.println("E[t2] = " + ET2);
            System.out.println();
            System.out.println("Queue 3 - Parameters");
            System.out.println("E[N3] = " + E_N3);
            System.out.println("Utilization = " + rho3);
            System.out.println("Throughput = " + theta3);
            System.out.println("E[t] = " + ET3);
            System.out.println();

            System.out.println("System - Parameters");
            System.out.println("E[N] = " + ( E_N1 + E_N2 + E_N3) );
            System.out.println("Utilization = " + (rho1 + rho2 + rho3));
            System.out.println("Throughput = " + (theta1 + theta2 + theta3));
             System.out.println();
            }

}
```