Simulation Home Assignment 2

Louise Söderström and Zuyao Li
May 16, 2014

Task1

In order to get a more stable result, we simulate the system for 100000 seconds.
Because the each queue in the queuing system is symmetric, so we only measure the the number of jobs in the first queue.

When the mean arrival time is 0.11 seconds, the mean number of jobs in the queue 1 with random algorithm is 8.6.

When the mean arrival time is 0.11 seconds, the mean number of jobs in the queue 1 with round algorithm is 5.7.

When the mean arrival time is 0.11 seconds, the mean number of jobs in the queue 1 with smallest algorithm is 2.3.

When the mean arrival time is 0.15 seconds, the mean number of jobs in the queue 1 with random algorithm is 1.8.

When the mean arrival time is 0.15 seconds, the mean number of jobs in the queue 1 with round algorithm is 1.2.

When the mean arrival time is 0.15 seconds, the mean number of jobs in the queue 1 with smallest algorithm is 0.96.

When the mean arrival time is 2 seconds, the mean number of jobs in the queue 1 with Round algorithm is 0.051.

When the mean arrival time is 2 seconds, the mean number of jobs in the queue 1 with random algorithm is 0.048.

When the mean arrival time is 2 seconds, the mean number of jobs in the queue 1 with smallest algorithm is 0.28.

From above, we see that to send the job to queueing system with the smallest number of jobs is the best algorithm.

Task2

We used simulation program to simulate the procedure that pharmacist fill coming prescriptions. We simulate this process for 100000 hours.

1.The average time his work will have finished is 18:10 every day.

2.The average time from the arrival of a prescription until it has been filled in is 15 minutes.

Task3

We used simulation program to simulate 36952 “runs” of the system, the mean time until the system breaks down is 2.76.

Code for all 3 tasks:

```
// This is an abstract class which all classes that are used for defining real
// process types inherit. The puropse is to make sure that they all define
the
// method treatSignal, which is needed in the main program.


public abstract class Proc extends Global{
        public abstract void TreatSignal(Signal x);
}



// This class defines a signal. What can be seen here is a mainimum. If
one wants to add more
// information just do it here.

class Signal{
        public Proc destination;
        public double arrivalTime;
        public int signalType;
        public Signal next;
}


// This class defines the signal list. If one wants to send more information
than here,
// one can add the extra information in the Signal class and write an extra
sendSignal method
// with more parameters.

public class SignalList{
        private  static Signal list, last;

        SignalList(){
        list = new Signal();
        last = new Signal();
        list.next = last;
        }

        public static void SendSignal(int type, Proc dest, double arrtime){
        Signal dummy, predummy;
        Signal newSignal = new Signal();
        newSignal.signalType = type;
        newSignal.destination = dest;
        newSignal.arrivalTime = arrtime;
        predummy = list;
        dummy = list.next;
        while ((dummy.arrivalTime < newSignal.arrivalTime) & (dummy !=
last)){
                predummy = dummy;
                dummy = dummy.next;
        }
        predummy.next = newSignal;
        newSignal.next = dummy;
 }

        public static Signal FetchSignal(){
                Signal dummy;
                dummy = list.next;
                list.next = dummy.next;
                dummy.next = null;
                return dummy;
        }
}
```

Code for especially for task1:

```
public class Global{
        public static final int ARRIVAL = 1, READY = 2, MEASURE = 3;
        public static double time = 0;
}

import java.util.*;
import java.io.*;

//It inherits Proc so that we can use time and the signal names without dot
notation

class Gen extends Proc{

        //The random number generator is started:
        Random slump = new Random();

        //There are two parameters:
        public Proc sendTo;    //Where to send customers
```

```
	public double lambda;  //How many to generate per second



	 //What to do when a signal arrives
	public void TreatSignal(Signal x){
		switch (x.signalType){
			case READY:{
				SignalList.SendSignal(ARRIVAL, sendTo, time);
				SignalList.SendSignal(READY, this, time +
(2.0/lambda)*slump.nextDouble());}
				break;
		}
	}
}
import java.util.*;
import java.io.*;

// This class defines a simple queuing system with one server. It inherits
Proc so that we can use time and the
// signal names without dot notation
class QS extends Proc{
	public int numberInQueue = 0, accumulated, noMeasurements;
	public Proc sendTo;
	Random slump = new Random();

	/*Function to pick a random number from an exponential distribution
with
	mean mu*/
	public double expRandom(double mu){
		double u = slump.nextDouble();
		return -1*mu*Math.log(1-u);
	}
```

```
        public void TreatSignal(Signal x){
                switch (x.signalType){

                        case ARRIVAL:{
                                numberInQueue++;
                                if (numberInQueue == 1){
                                        double waittime=expRandom(0.5);
                                        SignalList.SendSignal(READY,this, time +
    waittime);
                                }
                        } break;

                        case READY:{
                                numberInQueue--;
                                if (sendTo != null){
                                        SignalList.SendSignal(ARRIVAL, sendTo,
    time);
                                }
                                if (numberInQueue > 0){
                                        double waittime=expRandom(0.5);
                                        SignalList.SendSignal(READY, this, time +
    waittime);
                                }
                        } break;

                        case MEASURE:{
                                noMeasurements++;
                                accumulated = accumulated + numberInQueue;
                                SignalList.SendSignal(MEASURE, this, time +
    2*slump.nextDouble());
                        } break;
                }
        }
}
import java.util.*;
import java.io.*;


public class MainSimulation extends Global{

    public static void main(String[] args) throws IOException {
        // The signal list is started and actSignal is declaree. actSignal is the
latest signal that has been fetched from the
        // signal list in the main loop below.

        Signal actSignal;
        new SignalList();

        // Here process instances are created (two queues and one
generator) and their parameters are given values.
       ArrayList<QS> qsList = new ArrayList<QS>();


       for (int i=0; i<5 ; i++)
       {
          QS Q = new QS();
          Q.sendTo = null;
          qsList.add(Q);
       }
       Random genNum = new Random();

        Gen Generator = new Gen();
        Generator.lambda = 1/0.12;  //Generator shall generate 9 customers
per second
```

```
        Generator.sendTo = null;   // The generated customers shall be sent
to Q1

        //To start the simulation the first signals are put in the signal list

        SignalList.SendSignal(READY, Generator, time);
        SignalList.SendSignal(MEASURE, qsList.get(1), time);

       int index = 0;

        // This is the main loop

        while (time < 100000){
                actSignal = SignalList.FetchSignal();
                time = actSignal.arrivalTime;
          if (actSignal.destination==Generator)
          {
             if (args[0].equals("Random"))
             {
                index = genNum.nextInt(qsList.size());
                Generator.sendTo = qsList.get(index);
             }
             else if (args[0].equals("Round"))
             {
                index = (index+1)%qsList.size();
                Generator.sendTo = qsList.get(index);
             }
             else if (args[0].equals("smallest"))
             {
                double min = Double.POSITIVE_INFINITY;
                index = -1;
                for (int i=0; i<qsList.size(); i++)
                {
                    if (qsList.get(i).numberInQueue < min)
                    {
                        index = i;
                        min = qsList.get(i).numberInQueue;
                    }
                }
                Generator.sendTo = qsList.get(index);
            }
            else
            {
                Generator.sendTo = qsList.get(0);
            }

        }
              actSignal.destination.TreatSignal(actSignal);
       }

       //Finally the result of the simulation is printed below:
      int accumulated=0,noMeasurements=0;
      for (int i=0; i<qsList.size(); i++)
      {
          accumulated+=qsList.get(i).accumulated;
          noMeasurements+=qsList.get(i).noMeasurements;
      }
       System.out.println("Mean number of customers in queuing system: "
+ 1.0*qsList.get(1).accumulated/qsList.get(1).noMeasurements);

  }
}
```

Code for especially for task2:

```
public class Global{
        public static final int ARRIVAL = 1, READY = 2, MEASURE = 3;
        public static double time = 0;
}

import java.util.*;
import java.io.*;

//It inherits Proc so that we can use time and the signal names without dot
notation

class Gen extends Proc{

        //The random number generator is started:
        Random slump = new Random();

        //There are two parameters:
        public Proc sendTo;   //Where to send customers
        public double lambda;   //How many to generate per second
        public double expRandom(double mu){
                double u = slump.nextDouble();
                return -1*mu*Math.log(1-u);
        }


        //What to do when a signal arrives
        public void TreatSignal(Signal x){
                switch (x.signalType){
                        case READY:{
                                if (time%24>=9 && time%24<=17){
                                        SignalList.SendSignal(ARRIVAL, sendTo,
time);
                                }
```

```
                    SignalList.SendSignal(READY, this, time +
expRandom(0.25));
                }
                    break;
            }
        }
}

import java.util.*;
import java.io.*;

// This class defines a simple queuing system with one server. It inherits
Proc so that we can use time and the
// signal names without dot notation
class QS extends Proc{
        public int numberInQueue = 0, accumulated, noMeasurements;
        public Proc sendTo;
        Random slump = new Random();

        public void TreatSignal(Signal x){
                switch (x.signalType){

                        case ARRIVAL:{
                                numberInQueue++;
                                if (numberInQueue == 1){
                                        SignalList.SendSignal(READY,this, time +
1.0/6+1.0/6*slump.nextDouble());
                                }
                        } break;

                        case READY:{
                                numberInQueue--;
                                if (sendTo != null){
                                        SignalList.SendSignal(ARRIVAL, sendTo,
time);
                                }
                                if (numberInQueue > 0){
                                        double
dealtime=1.0/6+1.0/6*slump.nextDouble();
                                        //System.out.println(dealtime);
                                        SignalList.SendSignal(READY, this, time +
dealtime);
                                }
                                if (numberInQueue == 0 && time%24>17)
                                        System.out.println(time%24);
                        } break;

                        case MEASURE:{
                                noMeasurements++;
                                accumulated = accumulated + numberInQueue;
                                SignalList.SendSignal(MEASURE, this, time +
2*slump.nextDouble());
                        } break;
                }
        }
}

import java.util.*;
import java.io.*;

//It inherits Proc so that we can use time and the signal names without dot
notation


public class MainSimulation extends Global{
    public static void main(String[] args) throws IOException {

        // The signal list is started and actSignal is declaree. actSignal is the
latest signal that has been fetched from the
        // signal list in the main loop below.

        Signal actSignal;
        new SignalList();

        // Here process instances are created (two queues and one
generator) and their parameters are given values.

        QS Q1 = new QS();
        Q1.sendTo = null;

        Gen Generator = new Gen();
        Generator.lambda = 4; //Generator shall generate 9 customers per
second
        Generator.sendTo = Q1;   // The generated customers shall be sent
to Q1

        //To start the simulation the first signals are put in the signal list

        SignalList.SendSignal(READY, Generator, time);
        SignalList.SendSignal(MEASURE, Q1, time);


        // This is the main loop

        while (time < 100000){
                actSignal = SignalList.FetchSignal();
                time = actSignal.arrivalTime;
                actSignal.destination.TreatSignal(actSignal);
        }

        //Finally the result of the simulation is printed below:

        //System.out.println("Mean number of customers in queuing system:
" + 1.0*Q1.accumulated/Q1.noMeasurements);

    }
}
```

Code for especially for task3:

```
public class Global{
        public static final int READY = 1, DEAD = 2;
        public static double time = 0,previousTime=0,brokenNumber=0;
}

import java.util.*;
import java.io.*;

// This class defines a simple queuing system with one server. It inherits
Proc so that we can use time and the
// signal names without dot notation
class Component extends Proc{
        Random slump = new Random();
        public Proc sendTo;
        public void TreatSignal(Signal x){
                switch (x.signalType){
                        case DEAD:{
                                brokenNumber+=1;
                                if (brokenNumber==5) {
                                        SignalList.SendSignal(READY, this, time);
                                        System.out.println(time-previousTime);
                                        previousTime=time;
```

```
                    }
                } break;
                case READY:{
                    SignalList.SendSignal(DEAD, this,
time+1+4*slump.nextDouble());
                } break;

            }
        }
}

import java.util.*;
import java.io.*;

//It inherits Proc so that we can use time and the signal names without dot
notation


public class MainSimulation extends Global{

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

        // The signal list is started and actSignal is declaree. actSignal is the
latest signal that has been fetched from the
        // signal list in the main loop below.

        Signal actSignal;
        new SignalList();

        // Here process instances are created (two queues and one
generator) and their parameters are given values.

       Random slump = new Random();
```

```
Component C1 = new Component();
C1.sendTo = null;
Component C2 = new Component();
C2.sendTo = null;
Component C5 = new Component();
C5.sendTo = null;
Component C3 = new Component();
C1.sendTo = null;
Component C4 = new Component();
C2.sendTo = null;



 //To start the simulation the first signals are put in the signal list

 SignalList.SendSignal(READY, C1, time);
SignalList.SendSignal(READY, C2, time);
SignalList.SendSignal(READY, C5, time);
SignalList.SendSignal(READY, C3, time);
SignalList.SendSignal(READY, C4, time);


 // This is the main loop

 while (time < 100000){
         actSignal = SignalList.FetchSignal();
         time = actSignal.arrivalTime;
   if (actSignal.signalType==READY && brokenNumber==5){
       SignalList.SendSignal(READY, C1, time);
       SignalList.SendSignal(READY, C2, time);
       SignalList.SendSignal(READY, C5, time);
       brokenNumber=0;
   }
```

```
            else{

                if (actSignal.signalType==DEAD && actSignal.destination==C1) {
                    SignalList.SendSignal(DEAD, C2, time);
                    SignalList.SendSignal(DEAD, C5, time);
                }
                if (actSignal.signalType==DEAD && actSignal.destination==C3) {
                    SignalList.SendSignal(DEAD, C4, time);
                }
                      actSignal.destination.TreatSignal(actSignal);
            }
          }


      }
  }
```