Homework #3 Simulation of a Simple Queueing System

**Due on 2022-01-19 請將書面報告及程式碼上傳到TronClass**

**Objective:** This laboratory is to familiarize the students with the use of simulation to study the performance of a simple queueing system under various conditions.

**Background:** Suppose that there is a router with *one* incoming link. Assume that on the link the packets arrive following a Poisson process, i.e., the interarrival time between two consecutive packets on each link is exponentially distributed with probability density function equal to *a(t) = e****-t***, *t >= 0* (The average interarrival time is equal to *1/,* and ** [packets/sec] is called the arrival rate). Packets arriving at the router enter a queue and are served by the CPU sequentially. The queue is of infinite capacity. Assume that the service time a packet requires is also exponentially distributed with probability density function *b(t) = e****-t*** , *t >= 0* (The average service time is equal to *1/,* and ** [packets/sec] is called the service rate). A packet exits the system after it is processed by the CPU.

Buffer



CPU

**Procedures**:

We first define two parameters:

## *N* = Average number of packets in the system

*T* = System Time = the average time from the moment a packet arrives to the moment when it leaves the system (theoretically, *T = N/ ***)**

** = System utilization = */*

## (theoretically, *N = / (1-)* )

Run each of the following simulation for a **simulation time** of 10,000 sec

1. Plot the *T* vs** curves of the following four cases on the same chart.
2. ** = 10, ** = 0.05, 1, 3, 5, 7, and 9
3. ** = 20, ** = 0.1, 2, 6, 10, 14, and 18
4. ** = 50, ** = 0.25, 5, 15, 25, 35, and 45
5. Assume the packet service time is equal to 0.02 sec (which is comparable to **= 50), which is constant. ** = 0.25, 5, 15, 25, 35, and 45
6. Plot the *N* vs** curves of the above four cases on the same chart.

**Results and Discussions:**

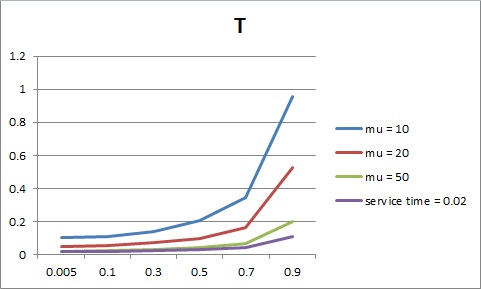
1. Compare cases 1(a), 1(b), and 1(c) and discuss the results.
2. Compare cases 1(c) and 1(d). Which case has better performance? Why?

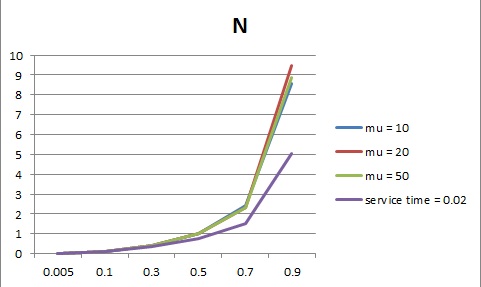
**What to turn in:**

Your report must include: graphs, and discussions, listing of source code

**Example Outputs:**

The graphs look like the following:





**Note:**

To produce random samples that are drawn from an exponential distribution with parameter **, use

*exptime (* *)* = -1.0 \* *log****e***(random\_number()) /** where 0 ≤ random\_number() ≤ 1

模擬程式結構流程大致如下(以Java為範例，使用single thread即可，不需要multi-thread):

**class** Event {

double eventTime;

int type; // 0 = packet arrival, 1 = packet departure

}

**class** Packet {

double arrTime, svcTime;

}

**public class** QueueingSimulation { //以**為例

**public static void** main(String[] args) {

simulate (0.05, 10);

for (int i=0; i<5; i++) {

double ** i\*double **

simulate(**



}

**public static void** simulate (double **, double **) {

double currTime = prevTime = 0.0; boolean cpuBusy = false;

int numPacketsInSystem = 0; double timePacketProduct = 0.0;

int numPacketsServed = 0; double totalSystemTime = 0.0;

Packet currPacket;

//產生下一個packet到達的時間

Event e = new Event( ); e.type = 0; e.eventTime = exptime (** );

將e 插入到event queue中，queue裡的event必須保持依照eventTime的順序由小到大

while (currTime < ENDTIME) { // 這裡ENDTIME設為10000

Event e = 從event queue取出first event;

prevTime = currTime; currTime = e.eventTime;

if (e.type == 0) { // 處理packet arrival

timePacketProduct += numPacketsInSystem\*(currTime – prevTime);

Packet p = new Packet( ); numPacketsInSystem++;

p.arrTime = currTime; p.svcTime = exptime(** ); //或是 = 0.02固定值

if (cpuBusy == false) { // CPU可以處理packet

cpuBusy = true; currPacket = p;

Event e2 = new Event( );

e2.eventTime = currTime + p.svcTime; e2.type = 1;

將e2 依照它的eventTime插入到event queue中適當位置;

}

else 將p 插入到packet queue的尾巴;

//產生下個packet的到達時間

Event e3 = new Event( );

e3.eventTime = currTime + exptime (** ); e3.type = 0;

將e3 依照它的eventTime插入到event queue中適當位置;

}

else { // 處理packet departure

timePacketProduct += numPacketsInSystem\*(currTime – prevTime);

numPacketsInSystem--;

numPacketsServed++; totalSystemTime += (currTime – currPacket.arrTime);

if (packet queue 是空的) cpuBusy = false;

else { // CPU處理下一個packet

currPacket =從packet queue取出first packet;

Evenet e4 = new Event( );

e4.eventTime = currTime + currPacket.svcTime; e4.type = 1;

將e4 依照它的eventTime插入到event queue中適當位置;

}

}

}

印出 N = timePacketProduct / ENDTIME;

印出 T = totalSystemTime/ numPacketsServed;

}

}