ECE 358: Computer Networks

Winter 2015

Project 1: Queue simulation

Date submitted: Friday, February 6, 2015

Submitted by: Haodong Huang

Student ID: 20442592

Student name: Haodong Huang

Email: [h53huang@uwaterloo.ca](mailto:h53huang@uwaterloo.ca)

Marks received:

Marked by

Contents

[M/M/1 Queue 3](#_Toc410994656)

[Question 1: 3](#_Toc410994657)

[Question 2: 4](#_Toc410994658)

[Question 3: 7](#_Toc410994659)

[Question 4: 8](#_Toc410994660)

[M/M/1/K Queue 9](#_Toc410994661)

[Question 5: 9](#_Toc410994662)

[Question 6 12](#_Toc410994663)

[Compiling Instructions 15](#_Toc410994664)

[M/M/1 15](#_Toc410994665)

[M/M/1/K 15](#_Toc410994666)

M/M/1 Queue

# Question 1:

int numbersTotal = 1000;

double Random::generateRanNum()

{

return (rand() / (double)(RAND\_MAX));

}

double Random::generateExponentialRanVar(double lambda)

{

return ((double)(-1.0)/lambda)\*log(1.0-generateRanNum());

}

void Random::variableTest()

{

double number, total, total2, mean, variance;

total = 0;

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

{

number = generateExponentialRanVar(75.0);

total += number;

generatedNumbers[i] = number;

}

mean = total / numbersTotal;

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

{

total2 += (generatedNumbers[i] - mean) \* (generatedNumbers[i] - mean);

}

variance = total2 / numbersTotal;

std::cout<< "Mean: " << mean << "\n";

std::cout<< "Variance: " << variance;

}

int main()

{

srand(time(NULL));

Random \*random = new Random();

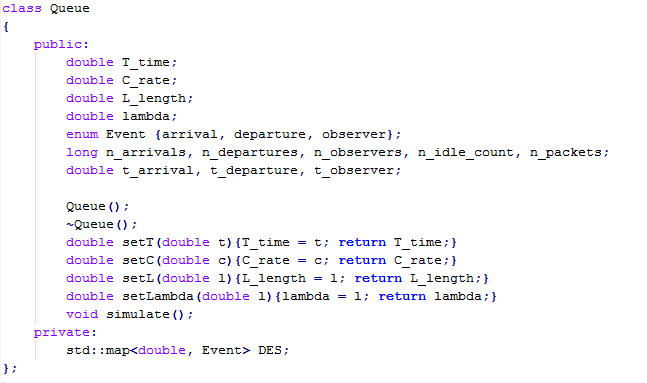
std::cout<< random->generateExponentialRanVar(75)<< std::endl;

delete random;

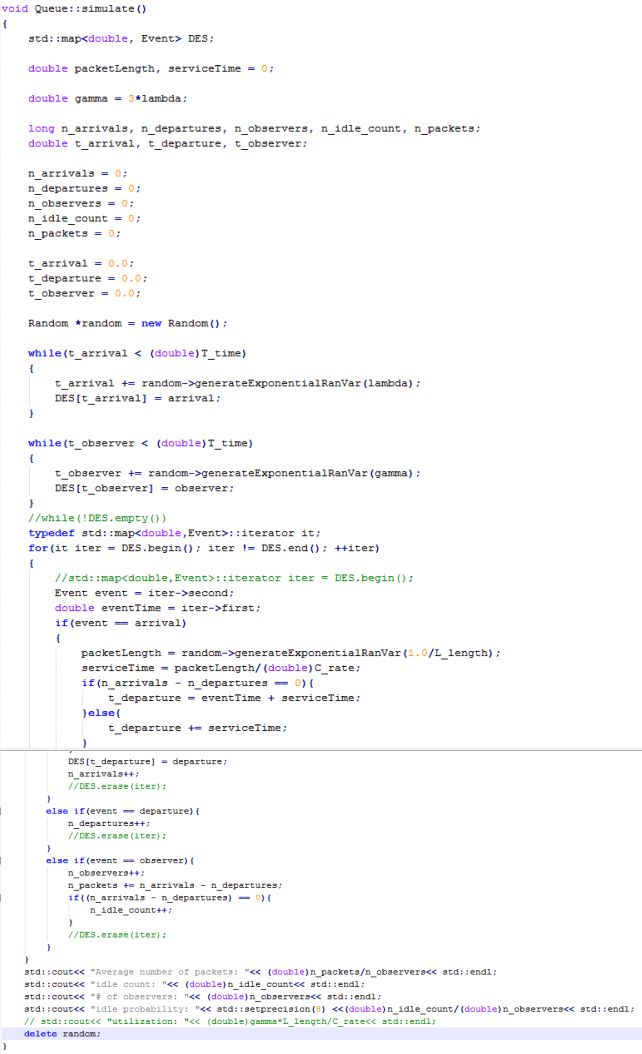
}

The expected value (0.013276) and variance (0.000179143) are close to 1/75 and 1/75^2 as expected.

# Question 2:



The variables n\_arrivals, n\_departures and n\_observers are used to count their respective events. n\_idle\_count is to keep track of how often the queue is empty, and n\_packets is used to count the number of packets in the queue at a given observation (this number needs to be averaged). T\_time is the length of the simulation in seconds.



The random number generator and exponential random variable generator are used to generate incoming packet lengths and times. The random number is uniform between 0 and 1. The exponential variable is given by var = -1/λ \* ln(1- U(0,1)) where U(0,1) is our random number.

First, we initialize some variables such as the simulation time (T), bit rate (C) and the average bit length (L) and lambda (ρ\*C/L). We let the simulation run for T seconds and generate random arrivals and observers based on the exponential generator using lambda as the parameter for arrivals and 3\*lambda for observers. These events are put in our DES and ordered in an ordered map based on their times. We then check these events in order. If it’s an arrival, then a departure time is created, based on the previous departure time plus service time. We increment our counter for arrival. If it’s a departure event that’s first in the DES, we increment our count for departures. If it’s an observer, we increment our counter for observers. We make note of how many number of packets are in the DES for later. If the buffer is empty, i.e. arrival counter = departure counter, then we increment our idle counter.

# Question 3:

|  |  |  |
| --- | --- | --- |
| Ρ (utilization) | E[N] | P\_idle |
| 0.25 | 0.333608 | 0.749799 |
| 0.35 | 0.535664 | 0.650774 |
| 0.45 | 0.820634 | 0.549453 |
| 0.55 | 1.22489 | 0.448976 |
| 0.65 | 1.8273 | 0.352883 |
| 0.75 | 2.99545 | 0.250727 |
| 0.85 | 5.76602 | 0.148064 |
| 0.95 | 18.5054 | 0.0505985 |

The code in question 2 was used to find E[N] and P\_idle. A for loop was used to run through each size of rho. The value of T is 10000.

The average number of packets (E[N]) was calculated by dividing the sum of the number of packets observed every time divided by the number of observations. As ρ increases, the average number of packets in the system goes up because more packets are being generated in between observations without being serviced because service time is fixed. Idle time goes down because you are unlikely to have all the packets serviced at the time of the observation.

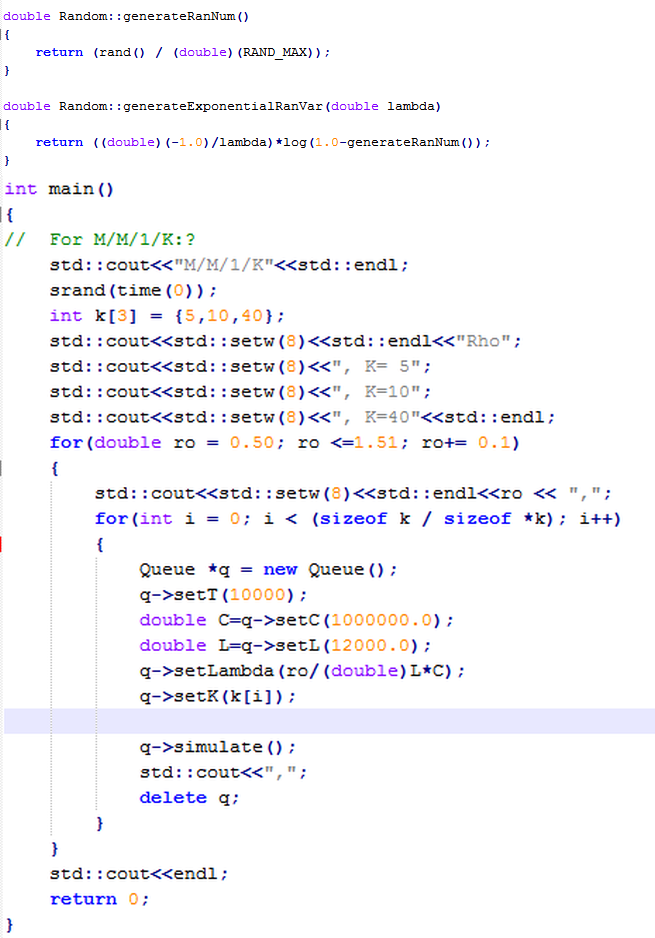
# Question 4:

|  |  |  |
| --- | --- | --- |
| ρ | E[N] | P\_idle |
| 1.2 | 83437.4 | 3.3360678e-06 |

As ρ increased to 1.2, the number of packets increased by a significant amount and idle time dropped to almost zero. The service time has not changed but the number of packets that get generated keep going up. The service cannot keep up with the number of packets being generated so more and more packets fill up the DES. The average number of packets is expected to increase as time goes on.

M/M/1/K Queue

# Question 5:



#ifndef QUEUE\_H

#define QUEUE\_H

#include <map>

#include <stdlib.h>

class Queue

**{**

public**:**

double T\_time**;**

double C\_rate**;**

double L\_length**;**

double lambda**;**

enum Event **{**arrival**,** departure**,** observer**};**

long n\_arrivals**,** n\_departures**,** n\_observers**,** n\_idle\_count**,** n\_packets**;**

double t\_arrival**,** t\_departure**,** t\_observer**;**

int k\_size**;**

long n\_generated**,** n\_loss**;**

Queue**();**

**~**Queue**();**

double setT**(**double t**){**T\_time **=** t**;** **return** T\_time**;}**

double setC**(**double c**){**C\_rate **=** c**;** **return** C\_rate**;}**

double setL**(**double l**){**L\_length **=** l**;** **return** L\_length**;}**

double setLambda**(**double l**){**lambda **=** l**;** **return** lambda**;}**

int setK**(**int k**){**k\_size **=** k**;** **return** k\_size**;}**

void simulate**();**

private**:**

std**::**map**<**double**,** Event**>** DES**;**

**};**

#endif



There were few code changes in the M/M/1/K compared to the M/M/1 code. Three variables were introduced. One for the size of the buffer K, and two for the number of packets generated vs lost. This is used to calculate p\_loss. One if statement was added when an arrival event was created to make sure that the number of packets in the buffer did not exceed the buffer size.

# Question 6

Result of E[N] values at T = 10000

|  |  |  |  |
| --- | --- | --- | --- |
| Rho | K= 5 | K=10 | K=40 |
| 0.5 | 0.90379 | 0.999397 | 0.997412 |
| 0.6 | 1.20719 | 1.46071 | 1.51142 |
| 0.7 | 1.52798 | 2.11875 | 2.34013 |
| 0.8 | 1.86107 | 2.96647 | 3.96916 |
| 0.9 | 2.19958 | 3.93314 | 8.5155 |
| 1 | 2.49427 | 4.97542 | 19.8857 |
| 1.1 | 2.78063 | 5.95609 | 31.0436 |
| 1.2 | 3.0256 | 6.71212 | 35.0274 |
| 1.3 | 3.23774 | 7.32074 | 36.6071 |
| 1.4 | 3.42146 | 7.77141 | 37.5001 |
| 1.5 | 3.58455 | 8.13184 | 37.9814 |

We know that we can’t have more packets than the buffer size K. We notice that as ρ increases and more packets are generated, there are more and more packets left in the queue. It seems that when ρ = 1, the average number of packets is half of K, which means that the arrival and departure rate are about equal. That means that when ρ > 1, the amount of packets in our queue is potentially up to K. At lower values of K, E[N] compares to when K= ∞. At higher values of ρ, the values of E[N] differ because when K < ∞, E[N] cannot increase without bounds.

|  |  |  |  |
| --- | --- | --- | --- |
| ρ | K= 5 | K=10 | K=40 |
| 0.4 | 0.005872 | 7.80E-05 | 0 |
| 0.5 | 0.016178 | 0.000415 | 0 |
| 0.6 | 0.032954 | 0.002215 | 0 |
| 0.7 | 0.058186 | 0.008633 | 0 |
| 0.8 | 0.088698 | 0.024052 | 3.90E-05 |
| 0.9 | 0.125905 | 0.051177 | 0.001455 |
| 1 | 0.167149 | 0.090887 | 0.026005 |
| 1.1 | 0.207663 | 0.138106 | 0.094224 |
| 1.2 | 0.24984 | 0.194652 | 0.16937 |
| 1.3 | 0.291122 | 0.242383 | 0.229986 |
| 1.4 | 0.329068 | 0.291015 | 0.285416 |
| 1.5 | 0.365611 | 0.337174 | 0.333473 |
| 1.6 | 0.397911 | 0.37538 | 0.3744 |
| 1.7 | 0.429019 | 0.413055 | 0.410643 |
| 1.8 | 0.458521 | 0.445654 | 0.445619 |
| 1.9 | 0.485123 | 0.474818 | 0.473904 |
| 2 | 0.50664 | 0.500607 | 0.500628 |
| 2.2 | 0.550888 | 0.545776 | 0.545744 |
| 2.4 | 0.58766 | 0.583284 | 0.583752 |
| 2.6 | 0.617571 | 0.616258 | 0.614524 |
| 2.8 | 0.644913 | 0.644011 | 0.643688 |
| 3 | 0.668267 | 0.667159 | 0.666012 |
| 3.2 | 0.688549 | 0.687515 | 0.68756 |
| 3.4 | 0.705756 | 0.705383 | 0.705688 |
| 3.6 | 0.722191 | 0.722261 | 0.722024 |
| 3.8 | 0.737254 | 0.736857 | 0.73701 |
| 4 | 0.74992 | 0.750196 | 0.749903 |
| 4.2 | 0.762448 | 0.76172 | 0.762194 |
| 4.4 | 0.772413 | 0.77278 | 0.772397 |
| 4.6 | 0.782514 | 0.782402 | 0.782314 |
| 4.8 | 0.791667 | 0.791916 | 0.791639 |
| 5 | 0.80039 | 0.799808 | 0.800024 |
| 5.4 | 0.814708 | 0.815235 | 0.81479 |
| 5.8 | 0.82754 | 0.827818 | 0.827215 |
| 6.2 | 0.838656 | 0.838743 | 0.838441 |
| 6.6 | 0.848548 | 0.84855 | 0.848352 |
| 7 | 0.857217 | 0.857369 | 0.856729 |
| 7.4 | 0.864715 | 0.864983 | 0.864765 |
| 7.8 | 0.871652 | 0.871898 | 0.871869 |
| 8.2 | 0.877828 | 0.878019 | 0.87805 |
| 8.6 | 0.883691 | 0.883849 | 0.883757 |
| 9 | 0.888674 | 0.888979 | 0.888983 |
| 9.4 | 0.893122 | 0.893652 | 0.893442 |
| 9.8 | 0.897984 | 0.898064 | 0.897946 |

P\_loss was obtained by dividing the number of packets lost by the number of packets generated. As ρ gets higher, P\_loss should approach 1 because packets are generated so fast that very few packages can be serviced compared to the amount generated. For very low values of ρ, no packages are lost because if our buffer has enough size, it is very unlikely that no packages have been serviced before filling the buffer. The general relationship can be described as (1 - P\_loss)\* ρ = 1 for values of ρ > 1.

Compiling Instructions

## M/M/1

For question 1: “make q1” to compile, followed by “./q1.o” to run the code. The output will print the numbers as well as the mean and variance.

For questions 2, 3, 4: “make q3” to compile, followed by “./q3.o” to run the code. The output will print E[N] and P\_idle for all rho values from 0.25-0.95 and 1.2.

## M/M/1/K

For questions 5 and 6 part 1: “make q61” to compile, followed by “./q61.o” to run the code. The code will print out the E[N] values for different values of ρ and K. Running this code will take ~3 min.

For question 6 part 2: “make q62” to compile, followed by “./q62.o” to run the code. This will print out the P\_loss values for different values of ρ and K from ρ=0.4 – 4.8. Running this code will take ~15 min.

For values of ρ >= 5, type “make qlarge” to compile, followed by “./qlarge.o” to run. This will also take ~15 min to run and prints P\_loss for those values of K and.