УО «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет инновационного непрерывного образования

Кафедра ПОИТ

Дисциплина: Системный анализ и машинное моделирование

ОТЧЕТ

по контрольной работе

Построение аналитической и имитационной

модели дискретно-стохастической СМО.

Выполнил: Проверил:

студент гр. 891051 Мельник Н.И.

Наумович Р.А.

Минск 2021

**Задание**

Построить граф состояний P-схемы. Смысл кодировки состояний раскрыть (время до выдачи заявки, число заявок в накопителе и т.д.). По графу построить аналитическую модель и, решив ее, определить вероятности состояний. Рассчитать теоретическое значение показателей эффективности, заданного целью исследования.

Для СМО, конфигурация которой задана вариантом, построить имитационную модель и исследовать ее (разработать алгоритм и написать имитирующую программу, предусматривающую сбор и статистическую обработку данных для получения оценок заданных характеристик СМО). Распределение интервалов времени между заявками во входном потоке и интервалов времени обслуживания – геометрическое с соответствующим параметром (ρ, π1, π2). Если ρ не задано, то входной поток – регулярный (с указанным в обозначении источника числом тактов между заявками).

**Вариант 11**

****

1. **Граф состояний P-схемы.**  
   Состояние системы определяется четырехкомпонентным вектором: tjc1c2.  
   t – число тактов, оставшихся до появления заявки на выходе источника (t1 = 1,2).  
   j – количество заявок, находящихся в накопителе (длина очереди) (j =0,1,2).  
   c1 – определяет состояние канала обслуживания 1, может принимать два значения: c1 = 0 – канал свободен, c1 =1 – канал занят обслуживанием заявки.  
   c2 – определяет состояние канала обслуживания 2, может принимать два значения: c2 = 0 – канал свободен, c2 =1 – канал занят обслуживанием заявки.



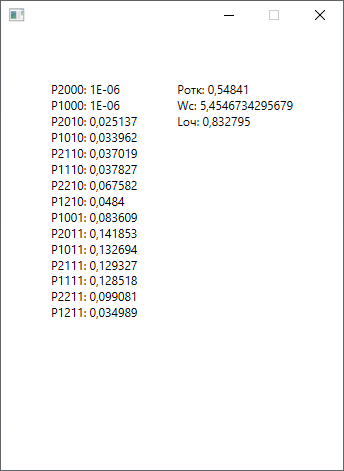
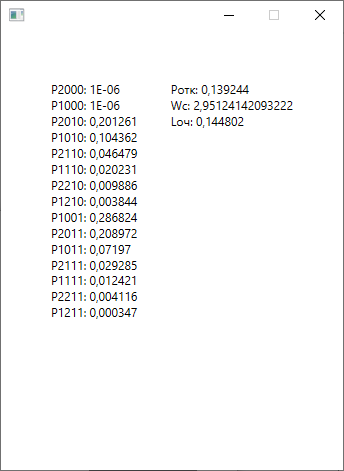
**Расчет параметров**

Построим систему уравнений для стационарных вероятностей состояний.

P2000 = 0  
P1000 = P2000  
P2010 = P1000 + P1001(1-π2)  
P1010 = P2010\*π1 + P2011\*π1(1-π2)  
P2110 = P1010\*π1 + P1011\*π1(1-π2)  
P1110 = P2110\*π1 + P2111\*π1(1-π2)  
P2210 = P1110\*π1 + P1111\*π1(1-π2) + P1210\*π1 + P1211\*π1(1-π2)  
P1210 = P2210\*π1 + P2211\*π1(1-π2)  
P1001 = P2010(1-π1) + P2011(1-π1)(1-π2) + P2011(1-π1)π2  
P2011 = P1010(1-π1) + P1001\*π2 + P1011(1-π1)(1-π2) + P1011(1-π1)π2  
P1011 = P2110(1-π1) + P2011\*π1\*π2 + P2111(1-π1)(1-π2) + P2111(1-π1)π2  
P2111 = P1110(1-π1) + P1011\*π1\*π2 + P1111(1-π1)(1-π2) + P1111(1-π1)π2  
P1111 = P2210(1-π1) + P2111\*π1\*π2 + P2211(1-π1)(1-π2) + P2211(1-π1)π2  
P2211 = P1210(1-π1) + P1111\*π1\*π2 + P1211\*π1\*π2 + P1211(1-π1)(1-π2) + P1211(1-π1)π2  
P1211 = P2211\*π1\*π2

Для π1=0,5, π2=0,7:  
P2000 = 0  
P1000 = 0  
P2010 = 0,025  
P1010 = 0,034  
P2110 = 0,037  
P1110 = 0,038  
P2210 = 0,068  
P1210 = 0,049  
P1001 = 0,083  
P2011 = 0,142  
P1011 = 0,133  
P2111 = 0,13  
P1111 = 0,129  
P2211 = 0,099  
P1211 = 0,035

Исходя из полученных данных, рассчитаем теоретические значения вероятности отказа Pотк, среднюю длину очереди Lоч, среднее время пребывания заявки в системе Wc:  
1.  
Pотк = 1 - Q = 1 - A/λ  
A = (P1001+P2011+P1011+P2111+P1111+P2211+P1211) \* (1-π2)  
**Pотк = 0,5494**2.  
Lоч = P2110 + P1110 + 2P2210 + 2P1210 + P2111 + P1111 + 2P2211 + 2P1211**Lоч = 0,836**  
3.  
Wc = Lc/AобщLc = P2010 + P1010 + 2P2110 + 2P1110 + 3P2210 + 3P1210 + P1001 + 2P2011 + 2P1011 + 3P2111 + 3P1111 + 4P2211 + 4P1211 = 2,506  
Аобщ = А + Аотк  
Аотк = P1211(1-π1)\*π2 + P2211(1-π1)π2 + P1111(1-π1)π2 + P2111(1-π1)π2 + P1011(1-π1)π2 + P2011(1-π1)π2 = 0,2338  
Aобщ = 0,2253 + 0,2338 = 0,4591  
**Wc = 5,46**

1. π1 = 0,5, π2 = 0,7 ****  
     
   π1 = 0,3, π2 = 0,3  
   ****

Значения показателей эффективности при имитационном моделировании оказались близки к значениям, полученным с помощью аналитической модели.

**Код**

internal class ImitationModel

{

private const int steps = 1000000;

private readonly double pi1;

private readonly double pi2;

private int denialCount;

// для получения среднего времени нахождения заявки в системе подсчитывается общее количество заявок requestCount, прошедших через систему,

// и общее количество нахождения заявки в системе в тактовый интервал requestInSystemCount, которое делится на requestCount

private int requestCount;

private int requestInSystemCount;

private int queueCount;

private Dictionary<string, int> stateCount = new Dictionary<string, int>()

{

{ "2000", 0 }, { "1000", 0 }, { "2010", 0 }, { "1010", 0 }, { "2110", 0 }, { "1110", 0 }, { "2210", 0 }, { "1210", 0 },

{ "1001", 0 }, { "2011", 0 }, { "1011", 0 }, { "2111", 0 }, { "1111", 0 }, { "2211", 0 }, { "1211", 0 }

};

private Dictionary<string, double> stateProbability = new Dictionary<string, double>();

private double denialProbability;

private double avgTimeInSystem;

private double avgQueueLength;

public ImitationModel(double pi1, double pi2)

{

this.pi1 = pi1;

this.pi2 = pi2;

}

public double this[string s]

{

get => stateProbability[s];

}

public double DenialProbability => denialProbability;

public double AvgTimeInSystem => avgTimeInSystem;

public double AvgQueueLength => avgQueueLength;

public void Simulate()

{

Random random1 = new Random((int)DateTime.Now.Ticks);

// прибавляется число для того, чтобы сгенерированные величины не были одинаковы

Random random2 = new Random((int)DateTime.Now.Ticks + 120121);

int stepsToSupply = 3;

int channel1 = 0;

int channel2 = 0;

int queue = 0;

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

{

if (channel2 == 1)

{

if (random2.NextDouble() <= 1 - pi2)

{

channel2 = 0;

++requestCount;

}

}

if (channel1 == 1)

{

if (random1.NextDouble() <= 1 - pi1)

{

channel1 = 0;

if (channel2 == 0)

{

channel2 = 1;

}

else

{

++denialCount;

++requestCount;

}

}

}

if (queue > 0 && channel1 == 0)

{

channel1 = 1;

--queue;

}

if (--stepsToSupply == 0)

{

stepsToSupply = 2;

if (queue == 1)

{

queue = 2;

}

else if (queue == 2)

{

++denialCount;

}

else if (queue == 0)

{

if (channel1 == 1)

queue = 1;

else

channel1 = 1;

}

}

++stateCount[$"{stepsToSupply}{queue}{channel1}{channel2}"];

requestInSystemCount += queue + channel1 + channel2;

queueCount += queue;

}

Summarize();

}

private void Summarize()

{

denialProbability = (double)denialCount / (double)(steps / 2);

avgQueueLength = (double)queueCount / (double)steps;

avgTimeInSystem = (double)requestInSystemCount / (double)requestCount;

foreach (var pair in stateCount)

{

stateProbability[pair.Key] = (double)pair.Value / (double)steps;

}

}

}