Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системный анализ и машинное моделирование (САиММод)

ОТЧЕТ

По лабораторной работе №5-6

Ст. гр. 851006 Преподаватель:

Одиноченко М.И. Мельник Н.И.

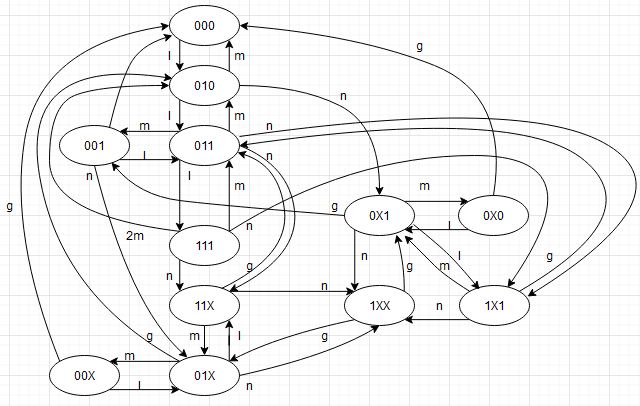
Минск 2021

**Условие**

На вход СМО вида М/M/2/1 поступает простейший поток заявок с интенсивностью l. Время обслуживания – показательное с параметром m. Работающий (обслуживающий заявки канал) может выходить из строя (отказывать). Поток отказов – простейший с параметром n. Ремонт начинается мгновенно после отказа. Время ремонта – показательное с параметром g. Заявка, находившаяся в канале в момент его выхода из строя, помещается во второй канал, если он свободен, или в очередь, если в ней есть свободные места и обслуживается заново.

а) Найти вероятности состояний канала (канал свободен, занят, ремонтируется) и абсолютную и относительную пропускную способность системы. l = 1,5, m = 0,8. n =0.01, g =0.1

**Граф состояний**



**Кодировка состояний**

Кодирование состояний:

n = {0,1} – состояние очереди (0 – свободен, 1 – занят);

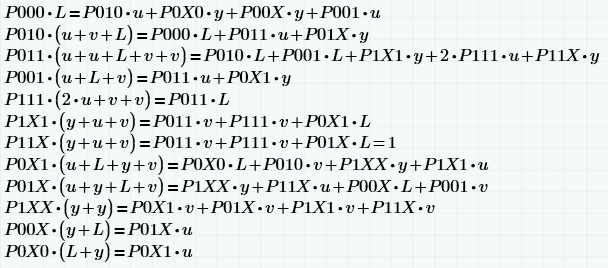
k1 = {0,1,x} – состояние первого канала (0 – свободен, 1 – занят, X – ремонт);

k2 = {0,1,x} – состояние второго канала (0 – свободен, 1 – занят, X – ремонт).

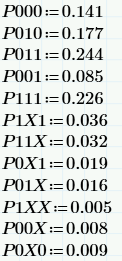
Общая кодировка состояний системы:

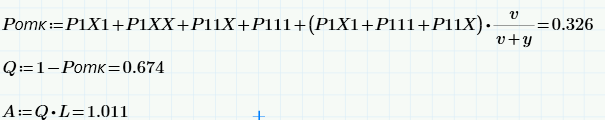
**Аналитическая модель:**

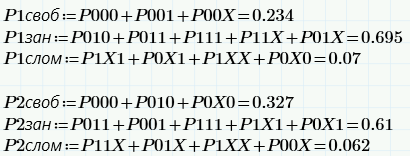




Результат:

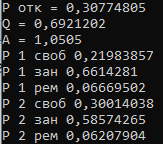






**Имитационная модель**

Результат:



Листинг

using System;

using System.Globalization;

namespace L6

{

class Program

{

static void Main(string[] args)

{

float parL, parM, parN, parG;

parL = 1.5f;

parM = 0.8f;

parN = 0.01f;

parG = 0.1f;

StateMachine sm = new StateMachine(parL, parM, parN, parG);

sm.Immitate();

sm.Print();

}

}

internal class StateMachine

{

private readonly float lambda;

private float timeBeforeGeneration;

private float[] timeUntilProcessing;

private float[] timeUntilBreaking;

private float[] timeUntilRepear;

private int[] channel;

private readonly float m;

private readonly float n;

private readonly float g;

private int processed;

private int rejected;

private int generated;

private int tried;

private float[] Free;

private float[] Busy;

private float[] Broken;

private int broken = -1;

private int repeared = -1;

private float queue;

public StateMachine(float paramLambda, float paramMu, float paramNu, float paramG)

{

lambda = paramLambda;

m = paramMu;

n = paramNu;

g = paramG;

timeBeforeGeneration = aDistribution(lambda);

timeUntilProcessing = new float[] { 0, 0 };

timeUntilBreaking = new float[] { 0, 0 };

timeUntilRepear = new float[] { 0, 0 };

channel = new int[] { 0, 0 };

Free = new float[] { 0, 0 };

Busy = new float[] { 0, 0 };

Broken = new float[] { 0, 0 };

processed = 0;

}

public void Immitate()

{

int i;

for (float time = 0; time <= 10000;)

{

if (timeBeforeGeneration == 0)

{

i = getFirstEmpty();

if (i != 2)

{

channel[i] = 1;

if (timeUntilBreaking[i] == 0)

timeUntilBreaking[i] = aDistribution(n);

timeUntilProcessing[i] = aDistribution(m);

generated++;

}

else

{

if (queue == 0)

{

queue = aDistribution(m);

generated++;

}

else

rejected++;

}

timeBeforeGeneration = aDistribution(lambda);

}

if (repeared != -1)

{

if (queue == 0)

{

channel[repeared] = 0;

}

else

{

channel[repeared] = 1;

timeUntilProcessing[repeared] = queue;

queue = 0;

}

repeared = -1;

}

if (broken != -1)

{

if (channel[broken] == 1)

{

if (channel[(broken + 1) % 2] == 0)

{

channel[(broken + 1) % 2] = 1;

timeUntilProcessing[(broken + 1) % 2] = timeUntilProcessing[broken];

}

else

{

if (queue == 0)

{

queue = timeUntilProcessing[broken];

}

else

rejected++;

}

}

timeUntilProcessing[broken] = 0;

timeUntilBreaking[broken] = 0;

channel[broken] = 2;

timeUntilRepear[broken] = aDistribution(g);

broken = -1;

}

for (i = 0; i < 2; i++)

{

if (timeUntilProcessing[i] == 0 && channel[i] == 1)

{

if (queue == 0)

{

channel[i] = 0;

processed++;

}

else

{

timeUntilProcessing[i] = queue;

processed++;

queue = 0;

}

}

}

float minTimeUntilEndOfChannel = float.MaxValue;

for (i = 0; i < 2; i++)

{

if (channel[i] == 1 && timeUntilProcessing[i] != 0 && timeUntilProcessing[i] < minTimeUntilEndOfChannel)

{

minTimeUntilEndOfChannel = timeUntilProcessing[i];

}

}

float minTimeUntilBreak = float.MaxValue;

for (i = 0; i < 2; i++)

{

if (timeUntilBreaking[i] < minTimeUntilBreak && timeUntilBreaking[i] != 0)

{

minTimeUntilBreak = timeUntilBreaking[i];

broken = i;

}

}

float minTimeUntilRepear = float.MaxValue;

for (i = 0; i < 2; i++)

{

if (timeUntilRepear[i] < minTimeUntilRepear && channel[i] == 2 && timeUntilRepear[i] != 0)

{

minTimeUntilRepear = timeUntilRepear[i];

repeared = i;

}

}

float temp;

if ((minTimeUntilEndOfChannel == float.MaxValue && minTimeUntilBreak == float.MaxValue && minTimeUntilRepear == float.MaxValue) ||

(timeBeforeGeneration < minTimeUntilBreak && timeBeforeGeneration < minTimeUntilEndOfChannel && minTimeUntilRepear == float.MaxValue) ||

(timeBeforeGeneration < minTimeUntilBreak && timeBeforeGeneration < minTimeUntilEndOfChannel && timeBeforeGeneration < minTimeUntilRepear) ||

(minTimeUntilBreak == float.MaxValue && minTimeUntilEndOfChannel == float.MaxValue && timeBeforeGeneration < minTimeUntilRepear))

{

tried++;

temp = timeBeforeGeneration;

broken = -1;

repeared = -1;

timeBeforeGeneration = 0;

}

else if ((minTimeUntilEndOfChannel < minTimeUntilBreak && minTimeUntilRepear == float.MaxValue) ||

(minTimeUntilEndOfChannel < minTimeUntilBreak && minTimeUntilEndOfChannel < minTimeUntilRepear))

{

temp = minTimeUntilEndOfChannel;

broken = -1;

repeared = -1;

timeBeforeGeneration -= temp;

}

else if (minTimeUntilBreak < minTimeUntilRepear || minTimeUntilRepear == float.MaxValue)

{

temp = minTimeUntilBreak;

repeared = -1;

timeBeforeGeneration -= temp;

}

else

{

temp = minTimeUntilRepear;

broken = -1;

timeBeforeGeneration -= temp;

}

for (i = 0; i < 2; i++)

{

if (channel[i] != 0)

{

if (timeUntilProcessing[i] != 0)

{

Busy[i] += temp;

timeUntilProcessing[i] -= temp;

}

if (timeUntilRepear[i] != 0)

{

Broken[i] += temp;

timeUntilRepear[i] -= temp;

}

if (timeUntilBreaking[i] != 0)

timeUntilBreaking[i] -= temp;

}

else

Free[i] += temp;

}

time += temp;

}

}

public void Print()

{

Console.WriteLine($"P отк = {((float)rejected / tried)}");

Console.WriteLine($"Q = {((float)processed / tried)}");

Console.WriteLine($"A = {((float)processed / 10000f)}");

Console.WriteLine("P 1 своб " + Free[0] / generated);

Console.WriteLine("P 1 зан " + Busy[0] / generated);

Console.WriteLine("P 1 рем " + Broken[0] / generated);

Console.WriteLine("P 2 своб " + Free[1] / generated);

Console.WriteLine("P 2 зан " + Busy[1] / generated);

Console.WriteLine("P 2 рем " + Broken[1] / generated);

Console.WriteLine(generated + " " + rejected + " " + processed + " " + tried);

}

private float aDistribution(float param)

{

Random rand = new Random();

return (float)(-1 \* Math.Log(rand.NextDouble()) / param);

}

private int getFirstEmpty()

{

int i;

for (i = 0; i < 2; i++)

{

if (channel[i] == 0)

{

break;

}

}

return i;

}

}

}