Министерство образования республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Институт информационных технологий

Кафедра ПОИТ

**Отчет по лабораторной работе №4**

**по курсу САиММод**

**Разработка, отладка и исследование программной модели непрерывно-стохастической СМО**

Выполнили:

студенты группы 381062

Радивил Д.В.

Черник М.Л.

Шкундалёва О.В.

Проверила: Лашкевич Е.М.

Минск 2016

**Задание**

Два наладчика обслуживают 6 станков. Станок требует наладки в среднем через каждые 0,5 часа. Наладка занимает у рабочего в среднем 10 минут. Все потоки событий – простейшие. Определить, как изменятся следующие показатели:

- среднее число занятых работников;

- абсолютная пропускная способность;

- среднее число неисправных станков,

если рабочие будут налаживать станки совместно, затрачивая при этом на наладку одного станка в среднем 5 минут.

Анализ исходных данных

Данная система является многоканальной СМО с фиксированным числом мест ожидания (с ограниченной очередью) – M/M/n/m.

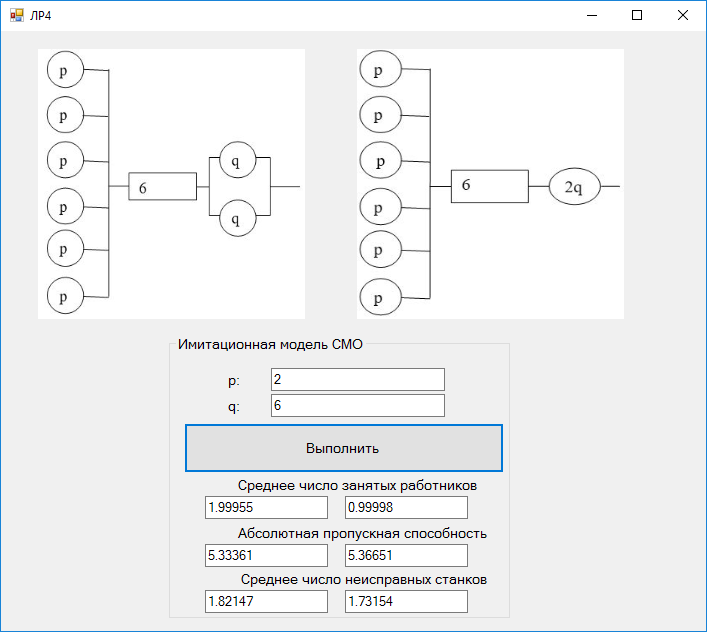
В первом случае, когда наладчики ремонтируют станки по отдельности, получим модель СМО, изображенную на рисунке 1-а.

Во втором случае, когда наладчики работают вместе, получим модель, изображенную на рисунке 2-б.

|  |  |
| --- | --- |
| C:\Users\radivil\Desktop\1---.JPG | C:\Users\radivil\Desktop\2---.JPG |
| Рис 1-а | Рис 1-б |
|  |  |

Примем во внимание, что если станок сломан и подал заявку на ремонт, то в этом случае он далее заявки не подает, пока не будет отремонтирован.

**Результат выполнения программы:**



**Код программы:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lr4

{

public partial class Form1 : Form

{

private readonly Random random = new Random();

public Form1()

{

InitializeComponent();

}

Random R = new Random();

private void button1\_Click(object sender, EventArgs e)

{

TwoChannelsWork();

OneWorkerWorks();

}

private double ExpDistributionGetInterval(double lambda) { return Math.Ceiling(-(1 / lambda) \* Math.Log(random.NextDouble())); }

public void TwoChannelsWork()

{

double p = 1 / (60 \* double.Parse(tbP.Text));

double q = 1 / (60 \* double.Parse(tbQ.Text));

Queue<Request> RequestQueue = new Queue<Request>();

int WorkingWorker = 0;

int BreakingSource = 0;

int SourcesGot = 0;

Request[] Worker = new Request[2];

int[] Source = new int[6];

for (int i = 0; i < Source.Length; i++)

Source[i] = (int)ExpDistributionGetInterval(p);

for (int i = 0; i < Worker.Length; i++)

Worker[i] = new Request(-1, 0);

for (int Quant = 0; Quant < 100000; Quant++)

{

for (int i = 0; i < Source.Length; i++)

{

if (Source[i] > 0)

Source[i]--;

if (Source[i] == 0)

{

RequestQueue.Enqueue(new Request(i, (int)ExpDistributionGetInterval(q)));

Source[i] = -1;

}

}

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

{

if (Worker[i].time == 0)

{

if (Worker[i].index != -1)

Source[Worker[i].index] = (int)ExpDistributionGetInterval(p);

if (RequestQueue.Count > 0)

Worker[i] = RequestQueue.Dequeue();

else

Worker[i].index = -1;

}

else

Worker[i].time--;

}

for (int i = 0; i < Source.Length; i++)

if (Source[i] > 0)

SourcesGot += (int)(p \* 5000);

for (int i = 0; i < Worker.Length; i++)

{

if (Worker[i].index >= 0)

{

WorkingWorker++;

BreakingSource++;

}

}

BreakingSource += RequestQueue.Count;

}

tbAverageWorkersA.Text = ((double)WorkingWorker / 100000).ToString("0.#####");

tbBreakingA.Text = ((double)(BreakingSource) / 100000).ToString("0.#####");

tbSourceGotA.Text = ((double)SourcesGot / 100000 / 15).ToString("0.#####");

}

public void OneWorkerWorks()

{

double p = 1 / (60 \* double.Parse(tbP.Text));

double q = 1 / (60 \* double.Parse(tbQ.Text));

Queue<Request> RequestQueue = new Queue<Request>();

int WorkingWorker = 0;

int BreakingSource = 0;

int SourcesGot = 0;

int[] Source = new int[6];

for (int i = 0; i < Source.Length; i++)

Source[i] = (int)ExpDistributionGetInterval(p);

Request Worker = new Request(-1, 0);

for (int Quant = 0; Quant < 100000; Quant++)

{

for (int i = 0; i < Source.Length; i++)

{

if (Source[i] > 0)

Source[i]--;

if (Source[i] == 0)

{

RequestQueue.Enqueue(new Request(i, (int)ExpDistributionGetInterval(2 \* q)));

Source[i] = -1;

}

}

if (Worker.time == 0)

{

if (Worker.index != -1)

Source[Worker.index] = (int)ExpDistributionGetInterval(p);

if (RequestQueue.Count > 0)

Worker = RequestQueue.Dequeue();

else

Worker.index = -1;

}

else

Worker.time--;

for (int i = 0; i < Source.Length; i++)

if (Source[i] > 0)

SourcesGot += (int)(p \* 5000);

if (Worker.index >= 0)

{

WorkingWorker++;

BreakingSource++;

}

BreakingSource += RequestQueue.Count;

}

tbAverageWorkersB.Text = ((double)WorkingWorker / 100000).ToString("0.#####");

tbBreakingB.Text = ((double)(BreakingSource) / 100000).ToString("0.#####");

tbSourceGotB.Text = ((double)SourcesGot / 100000 / 15).ToString("0.#####");

}

public class Request

{

public int time;

public int index;

public Request(int ind, int tim)

{

time = tim;

index = ind;

}

}

}

}