БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра программного обеспечения информационных технологий

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Отчет

по лабораторным работам № 5-6

по дисциплине «Системный анализ и машинное моделирование»

Вариант №8

Выполнил студент: Верещагин Н. В.

группы 851006

Проверил: Мельник Н.И.

Минск 2021

1. **Задание**

На вход 2 – канальной СМО с отказами поступает простейший поток заявок с интенсивностью λ = 4 заявки в час. Среднее время обслуживания одной заявки 0.8 часа. Каждая обслуженная заявка приносит доход 4у.е. Содержание одного канала обходится 2 у.е./час. Определить, что экономически целесообразнее – увеличение числа каналов до 3, или введение мест ожидания, если содержание одного места обходится в 0.3 у.е./час.

1. **Решение**

Рассмотрим оба случая:

1. Увеличение числа каналов до 3

В этом случае система превращается в 3-канальную СМО с отказами

Интенсивность:

Среднее время обслуживания одной заявки:

Доход с обслуживания одной заявки:

Содержание одного канала:

Среднее количество заявок обслуженных за 1 час:

Вероятность отказа:

Относительная пропускная способность:

Абсолютная пропускная способность:

Доход:

Расход:

Чистая прибыль:

1. Введение мест ожидания

В этом случае система превращается в 2-канальную СМО с ограниченной очередью (m)

Интенсивность:

Среднее время обслуживания одной заявки:

Доход с обслуживания одной заявки:

Содержание одного канала:

Содержание одного места в очереди

Среднее количество заявок обслуженных за 1 час:

Вероятность отказа:

Относительная пропускная способность:

Абсолютная пропускная способность:

Доход:

Расход:

Чистая прибыль:

Результаты:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Длина очереди | 0 | 1 | 2 | 3 | 4 | 5 |
| Вероятность отказа | 0,549356 | 0,467794 | 0,428071 | **0,406498** | 0,394085 | 0,386705 |
| Относительная пропускная способность | 0,450643777 | 0,532206487 | 0,571928724 | **0,593502087** | 0,605914944 | 0,613295226 |
| Абсолютная пропускная способность | 1,802575 | 2,128826 | 2,287715 | **2,374008** | 2,42366 | 2,453181 |
| Доход, у.е. | 7,2103 | 8,515304 | 9,15086 | **9,496033** | 9,694639 | 9,812724 |
| Расход, у.е. | 4 | 4,3 | 4,6 | **4,9** | 5,2 | 5,5 |
| Чистая прибыль, у.е. | 3,2103 | 4,215304 | 4,55086 | **4,596033** | 4,494639 | 4,312724 |

Итог: выгоднее ввести 3 места ожидания.

Статистическое исследование:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | k=2 | k=3 | m=1 | m=2 | m=3 | m=4 | m=5 |
| Чистая прибыль, у.е. | 3,2172 | 4,1476 | 4,7392 | 5,1040 | 5,0052 | 4,7632 | 4,4880 |

1. **Листинг программы**

namespace Lab5

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

int queueCount = Convert.ToInt32(textBox3.Text);

int n = Convert.ToInt32(textBox2.Text);

double time = Convert.ToInt32(textBox1.Text);

var model = new Model(time, n, queueCount);

ModelResult modelResult = model.Run();

double spend = (n \* 2 + 0.3 \* queueCount) \* time;

double get = modelResult.ProcessedRequests \* 4;

label14.Text = spend.ToString();

label13.Text = get.ToString();

label15.Text = (get - spend).ToString();

}

private void label13\_Click(object sender, EventArgs e)

{

}

}

public class Model

{

private int \_queueMax = 0;

private int \_queueCount = 0;

private double \_lambda = 4;

private double \_currentTime = 0;

private double \_endTime;

private double \_nextRequestGenerationTime;

private const double \_processTime = 0.8;

private List<double?> \_consumersFinishTime = new List<double?>();

private int lamb = 0;

ModelResult modelResult = new ModelResult();

public Model(double endTime, int countConsumers, int queueCount)

{

\_endTime = endTime;

\_queueMax = queueCount;

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

{

\_consumersFinishTime.Add(null);

}

}

public ModelResult Run()

{

CalculateNextGeneration();

while (\_currentTime < \_endTime)

{

var nextProcessedIndex = NextRequestProcessedIndex();

if (\_consumersFinishTime[nextProcessedIndex] < \_nextRequestGenerationTime)

{

\_currentTime = \_consumersFinishTime[nextProcessedIndex].Value;

modelResult.ProcessedRequests++;

if (\_queueCount > 0)

{

\_queueCount--;

\_consumersFinishTime[nextProcessedIndex] = \_currentTime + \_processTime;

}

else

{

\_consumersFinishTime[nextProcessedIndex] = null;

}

}

else

{

\_currentTime = \_nextRequestGenerationTime;

modelResult.GeneratedRequests++;

var freeIndex = FreeConsumerIndex();

if (freeIndex == -1)

{

if (\_queueCount >= \_queueMax)

{

modelResult.DroppedRequests++;

}

else

{

\_queueCount++;

}

}

else

{

\_consumersFinishTime[freeIndex] = \_currentTime + \_processTime;

}

CalculateNextGeneration();

}

}

return modelResult;

}

private void CalculateNextGeneration()

{

\_nextRequestGenerationTime = \_currentTime + SimpleStream.Next(\_lambda);

lamb++;

}

private int FreeConsumerIndex()

{

return \_consumersFinishTime.IndexOf(null);

}

private int NextRequestProcessedIndex()

{

var minIndex = 0;

for (int i = 1; i < \_consumersFinishTime.Count(); i++)

{

if (\_consumersFinishTime[i] < \_consumersFinishTime[minIndex])

{

minIndex = i;

}

}

return minIndex;

}

}

public class ModelResult

{

public int GeneratedRequests = 0;

public int ProcessedRequests = 0;

public int DroppedRequests = 0;

}

class SimpleStream

{

private static readonly Random random = new Random();

public static double Next(double lambda)

{

return -1 \* Math.Log(random.NextDouble(), Math.E) / lambda;

}

}

}