Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных сетей

ОТЧЕТ

по лабораторной работе №5-6

на тему

«Построение аналитической и имитационной модели и сравнение результатов исследования»

|  |  |
| --- | --- |
| Выполнили |  |
| студенты гр. 751003: | Стубеда В. Д. |
|  |  |
| Проверил | Мельник Н. И. |

Минск 2021

**Задание.**

12. Автозаправочная станция (АЗС) имеет n колонок; площадка возле нее допускает одновременное ожидание не более m автомашин. Поток автомашин, пребывающих на АЗС простейший с интенсивностью λ. Время обслуживания показательное со средним значением tобсл.

а) λ=1 маш/мин, tобсл = 2 мин.

Найти вероятности отказа и среднее значение длины очереди при n = 2 для значений m от 3 до 7.

**Решение.**

1. *Введение мест ожидания*

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

Интенсивность:

Среднее время обслуживания одной машины:

Значения m:

Среднее количество машин обслуженных за 1 мин:

Вероятность отказа:

Среднее число заявок в очереди:

Результаты:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Длина очереди | 3 | 4 | 5 | 6 | 7 |
| Вероятность отказа | 0.(18) | 0.15 | 0.1(3) | 0.12 | 0.11 |
| Среднее значение длины очереди | 1.(09) | 1.54 | 2 | 2.47 | 2.95 |

Статистическое исследование:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Длина очереди | 3 | 4 | 5 | 6 | 7 |
| Вероятность отказа | 0.18 | 0.15 | 0.14 | 0.12 | 0.10 |
| Среднее значение длины очереди | 1.10 | 1.55 | 2.03 | 2.51 | 2.99 |

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace Lab5

{

public partial class Form1 : Form

{

private const int time = 10000;

private const int n = 2;

private int m;

private const int Lambda = 1;

private const double Mu = 0.5;

private const double w = 2;

private int QueueCount;

private double CurrentTime;

private double TimeWhenNewRequestComes;

private List<double?> FinishTimeList;

private int QueueCountAll;

private int QueueCountCount;

private int GeneratedRequests;

private int DroppedRequests;

public Form1()

{

InitializeComponent();

}

private void trackBar1\_ValueChanged(object sender, EventArgs e)

{

m = trackBar1.Value;

FinishTimeList = new List<double?>();

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

{

FinishTimeList.Add(null);

}

Run();

double Potk = Math.Pow(w, n + m) / (Math.Pow(n, m) \* 2) \* Math.Pow(1 + w + Math.Pow(w, 2) / 2 + m \* Math.Pow(n, n) / 2, -1);

double Loch = m \* (m + 1) \* Math.Pow(1 + w + Math.Pow(w, 2) / 2 + m \* Math.Pow(n, n) / 2, -1);

label13.Text = ((double) QueueCountAll / (QueueCountCount)).ToString();

label14.Text = Loch.ToString();

label15.Text = ((double)DroppedRequests / GeneratedRequests).ToString();

label2.Text = Potk.ToString();

label3.Text = label3.Text.Substring(0, label3.Text.Length - 1) + m;

}

public void Run()

{

QueueCount = 0;

CurrentTime = 0;

QueueCountAll = 0;

QueueCountCount = 0;

GeneratedRequests = 0;

DroppedRequests = 0;

while (CurrentTime < time)

{

TimeWhenNewRequestComes = CurrentTime + SimpleStream.Next(Lambda);

var nextProcessedIndex = NextRequestProcessedIndex();

if (FinishTimeList[nextProcessedIndex] < TimeWhenNewRequestComes)

{

CurrentTime = FinishTimeList[nextProcessedIndex].Value;

if (QueueCount > 0)

{

QueueCount--;

FinishTimeList[nextProcessedIndex] = CurrentTime + SimpleStream.Next(Mu);

}

else

{

FinishTimeList[nextProcessedIndex] = null;

}

}

else

{

CurrentTime = TimeWhenNewRequestComes;

GeneratedRequests++;

var freeIndex = FreeConsumerIndex();

if (freeIndex == -1)

{

if (QueueCount >= m)

{

DroppedRequests++;

}

else

{

QueueCount++;

}

}

else

{

FinishTimeList[freeIndex] = CurrentTime + SimpleStream.Next(Mu);

}

}

QueueCountAll += QueueCount;

QueueCountCount++;

}

}

private int FreeConsumerIndex()

{

return FinishTimeList.IndexOf(null);

}

private int NextRequestProcessedIndex()

{

var minIndex = 0;

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

{

if (FinishTimeList[i] < FinishTimeList[minIndex])

{

minIndex = i;

}

}

return minIndex;

}

private void Form1\_Activated(object sender, EventArgs e)

{

trackBar1\_ValueChanged(sender, e);

}

}

class SimpleStream

{

private static readonly Random random = new Random();

public static double Next(double lambda)

{

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

}

}

}