Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Отчет по лабораторной работе № 3

«Аналитическое моделирование дискретно-стохастической СМО и построение её имитационной модели»

Выполнил: В.И. Кириллов

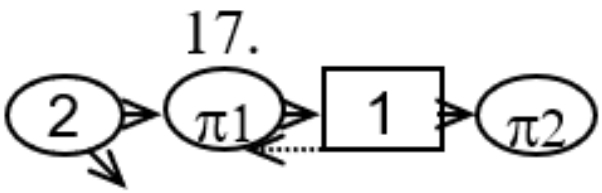
Проверил: И.Г. Алексеев

МИНСК 2023

Задание 1.

Расчет варианта: 05050065 % 32 = 17.

Построить граф состояний Q-схемы.



j = {1, 2} – число тактов до новой заявки
q = {0, 1} – кол-во заявок в очереди
t1 = {0, 1} – кол-во заявок в канале №1
t2 = {0, 1} – кол-во заявок в канале №2

Общий вид кодировки состояния системы: {j, q, t1, t2}.

По графу построим аналитическую модель и, решив ее, определим вероятности состояний. $P_{2000} = 0$, т.к. 2000 — невозвратное состояние.

$P_{2100} = \pi1(1-\pi2)\, P_{1101} + (1-\pi2)\, P_{1001} + \pi1\, P_{1100}$

$P_{1001} = (1-\pi1)\, P_{2100} + (1-\pi1)(1-\pi2)\, P_{2101}$

$P_{2101} = \pi2\, P_{1001} + (1-\pi1)\, P_{1100} + \pi1\pi2\, P_{1101} + (1-\pi1)(1-\pi2)\, P_{1101} + (1-\pi2)\, P_{1011}$

$P_{1011} = (1-\pi1)\pi2\, P_{2101} + (1-\pi1)(1-\pi2)\, P_{2111}$

$P_{2111} = \pi2\, P_{1011} + (1-\pi1)\pi2\, P_{1101} + (1-\pi2)\, P_{1111} + \pi2\, P_{1111}$

$P_{1111} = (1-\pi1)\pi2\, P_{2111} + \pi1\pi2\, P_{2111}$

$P_{1101} = \pi1(1-\pi2)\, P_{2111} + \pi1\pi2\, P_{2101}$

$P_{1100} = \pi1\, P_{2100} + \pi1(1-\pi2)\, P_{2101}$

Решив систему уравнений ($\mathbf{p_1}$=0,5 $\mathbf{p_2}$=0,5), получили:

$P_{2100} = 0{,}15$ $P_{1111} = 0{,}07$

$P_{1001} = 0{,}13$ $P_{1101} = 0{,}09$

$P_{2101} = 0{,}22$ $P_{1100} = 0{,}13$

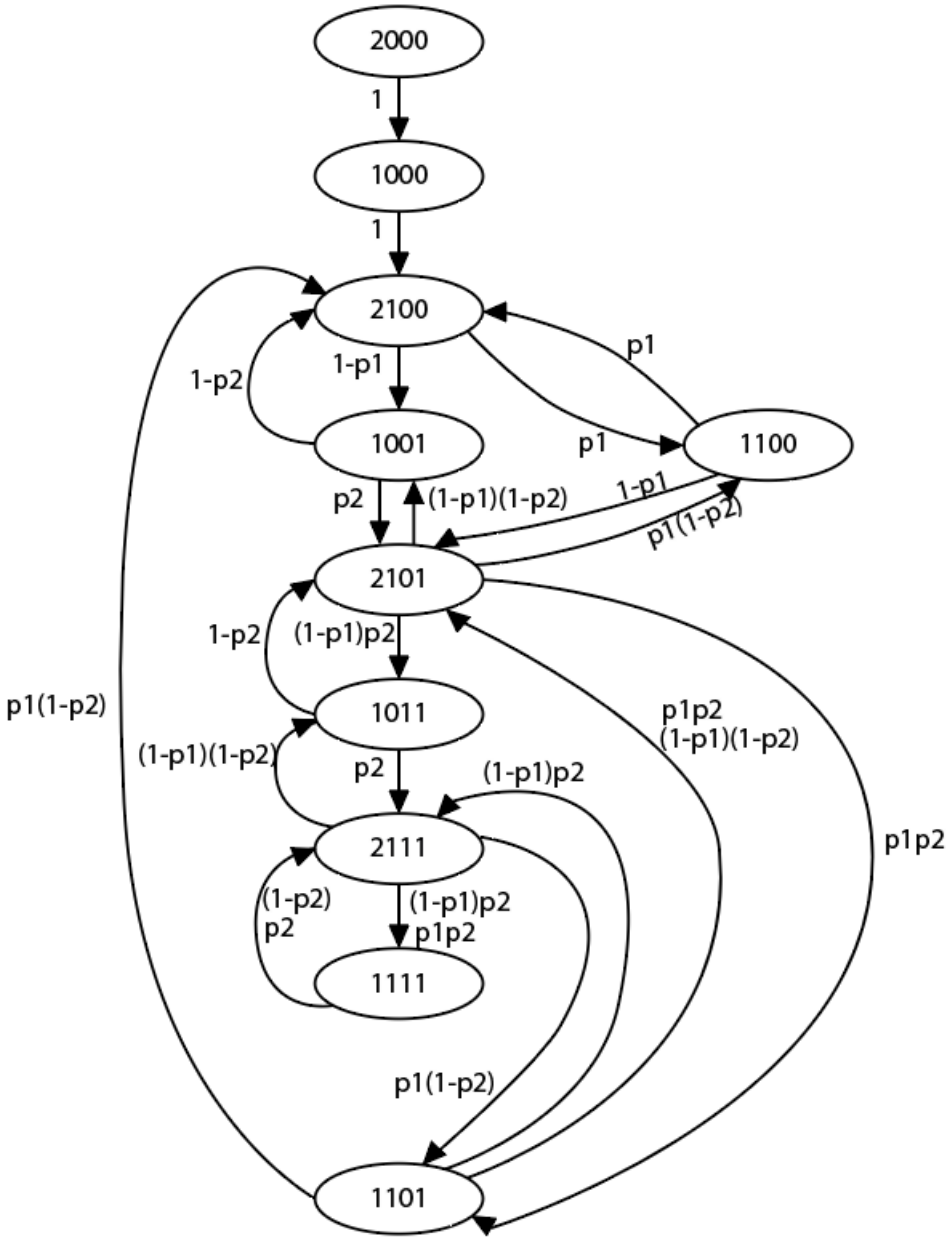
$P_{1011} = 0{,}09$

$P_{2111} = 0{,}13$

Исходя из полученных данных, рассчитаем теоретические значения средней длины очереди $L_{оч}$ и среднее число обслуживаемых заявок в такт A:

$L_{оч} = 1^* P_{1011} + 1^* P_{2111} + 2^* P_{1111} = 0{,}09 + 0{,}13 + 0{,}07 = 0{,}29$

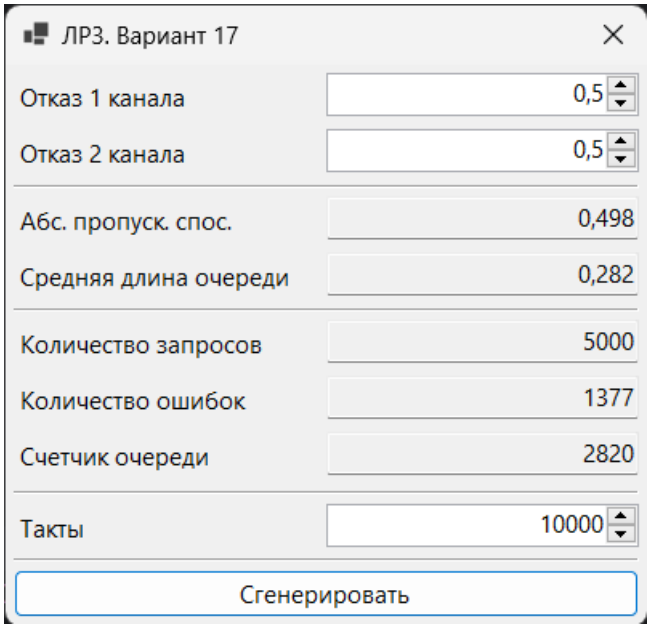
A = (1* P_{2100} + 1* P_{1100} + 1* P_{1001} + 2* P_{2101} + 2* P_{1011} + 2* P_{2111} + 2* P_{1111} + 2* P_{1101})/2 = 0.494



Задание 2

Для СМО из задания 1 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 (p, p_1, p_2).

Результат работы программы:



Вывод:

В ходе лабораторной работы была аналитически смоделирована дискретно-стохастическая СМО и разработана программа, имитирующая поведение данной СМО. Построенная модель позволяет статистически подсчитать характеристики СМО. Статистическое значение искомой характеристики оказывается близким к теоретически рассчитанному. Значит имитационная модель построена верно. Было также замечено, что на выходные данные влияют параметры СМО, такие как **p**, **p_1**, **p_2**.

Листинг программы:

Класс MainForm:

```
public partial class MainForm : Form
   {
      public MainForm()
      {
         InitializeComponent();
      }

      private void ButtonClick(Object sender, EventArgs args)
      {
         Double first = (Double) upDown1.Value;
         Double second = (Double) upDown2.Value;

         Queue queue = new Queue(first, second);

         Int32 ticks = (Int32) upDown3.Value;

         for (Int32 i = 0; i < ticks; i++)
         {
            queue.NextState();
         }

         textBox1.Text = $"{1.0 / queue.RequestFrequency * (1.0 - queue.First) * (1.0 -
queue.Second):0.000}";
         textBox2.Text = $"{(Double) queue.QueueCount / (queue.RequestCount * 2):0.000}";

         textBox3.Text = $"{queue.RequestCount}";
         textBox4.Text = $"{queue.FailureCount}";
         textBox5.Text = $"{queue.QueueCount}";
      }
   }
```

Класс Queue:

```
public class Queue
   {
      private readonly Random _random = new();

      private Byte _requestCounter;

      private Byte Channel1 { get; set; }

      private Byte _buffer;

      private Byte Channel2 { get; set; }
```

```
private bool _blocked;

public readonly byte RequestFrequency = 2;

private const int BufferSize = 1;

public Double First { get; }
public Double Second { get; }

public Int32 RequestCount { get; private set; }
public Int32 FailureCount { get; private set; }
public Int32 QueueCount { get; private set; }

public Queue(Double first, Double second)
{
   _requestCounter = RequestFrequency;
   Channel1 = 0;
   _buffer = 0;
   Channel2 = 0;

   First = first;
   Second = second;

   RequestCount = 0;
   FailureCount = 0;
   QueueCount = 0;
}

public void NextState()
{
   if (Channel2 == 1)
   {
      if (_random.NextDouble() > Second)
      {
         Channel2 = 0;
      }
   }

   if (_buffer > 0)
   {
      if (Channel2 == 0)
      {
         _buffer--;
         Channel2 = 1;
      }
```

```
}

if (Channel1 == 1)
{
   if (_blocked)
   {
      if (_buffer < BufferSize)
      {
         _buffer++;
         _blocked = false;
         Channel1 = 0;
      }
   }
   else if (_random.NextDouble() > First)
   {
      if (_buffer == BufferSize)
      {
         _blocked = true;
      }
      else
      {
         Channel1 = 0;
         _buffer++;
      }
   }
}

if (_buffer > 0)
{
   if (Channel2 == 0)
   {
      _buffer--;
      Channel2 = 1;
   }
}

if (_requestCounter != 1)
{
   _requestCounter--;
}
else
{
   _requestCounter = RequestFrequency;
   RequestCount++;

   if (Channel1 == 0)
```

```
            {
                Channel1 = 1;
            }
            else
            {
                FailureCount++;
            }
        }

        if (_buffer != 0)
        {
            QueueCount++;
        }

        Console.WriteLine($"{_requestCounter} {Channel1} {_buffer} {Channel2}");
    }
}
```