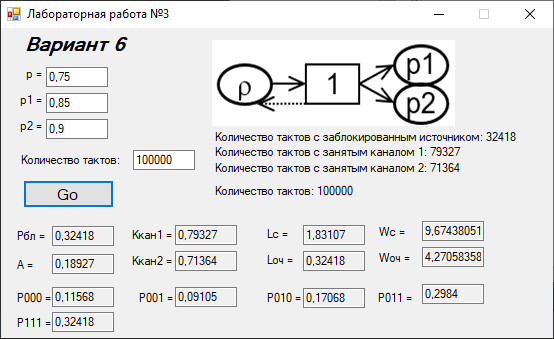
**Задание 2**

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, p1, p2).

**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**, **p1**, **p2**.

**Листинг программы:**

public partial class Form1 : Form  
**{** public Form1()  
 {  
 InitializeComponent();  
 }  
  
 private void GoButton\_Click(object sender, EventArgs e)  
 {  
 QueuingSystem smo = new QueuingSystem(double.Parse(textBox\_p.Text), double.Parse(textBox\_p1.Text), double.Parse(textBox\_p2.Text));  
 int ticksCount = int.Parse(textBox\_ticks.Text);  
  
 for (int i = 0; i < ticksCount - 1; i++)  
 smo.GenerateNextState();  
   
   
 textBox\_block.Text = (smo.P111 / (double)ticksCount).ToString(); // Pбл - вероятность блокировки  
 textBox\_k1.Text = (smo.Counter1/(double) ticksCount).ToString(); // Коэффициент загрузки канала 1  
 textBox\_k2.Text = (smo.Counter2 /(double)ticksCount).ToString(); // Коэффициент загрузки канала 2  
   
 textBox\_P000.Text = (smo.P000 / (double)ticksCount).ToString(); // P000  
 textBox\_P001.Text = (smo.P001 / (double)ticksCount).ToString(); // P001  
 textBox\_P010.Text = (smo.P010/ (double)ticksCount).ToString(); // P010  
 textBox\_P011.Text = (smo.P011 / (double)ticksCount).ToString(); // P011  
 textBox\_P111.Text = (smo.P111 / (double)ticksCount).ToString(); // P111  
  
 textBox\_A.Text = (smo.AdmissionCounter / (double)ticksCount).ToString();   
 textBox\_Lo.Text = (smo.P111 / (double)ticksCount).ToString(); // P001  
 textBox\_Wo.Text = (smo.P111/ (double)smo.queryCounter ).ToString(); // P010  
 textBox\_Lc.Text = ((smo.P001+smo.P010+2\*smo.P011+3\*smo.P111)/ (double)ticksCount).ToString(); // P011  
 textBox\_Wc.Text = (smo.colAplication / (double)smo.AdmissionCounter ).ToString(); // P111  
   
  
 label\_block.Text = "Количество тактов с заблокированным источником: " + smo.P111.ToString();  
 label\_k1.Text = "Количество тактов с занятым каналом 1: " + smo.Counter1.ToString();  
 label\_k2.Text = "Количество тактов с занятым каналом 2: " + smo.Counter2.ToString();  
 label\_ticks.Text = "Количество тактов: " + ticksCount.ToString();  
 }  
   
}

namespace lab3  
{  
 class QueuingSystem  
 {  
 private readonly Random \_rand, \_rand1, \_rand2;  
 private readonly double \_p, \_p1, \_p2;  
   
 public int colAplication { get; private set; } // Состояние очереди {0, 1}  
   
 public int queryCounter { get; private set; } // Счётчик поступлений в очередь  
  
 public byte J { get; private set; } // Состояние очереди {0, 1}  
 public byte T1 { get; private set; } // Состояние канала 1 {0, 1}  
 public byte T2 { get; private set; } // Состояние канала 2 {0, 1}  
  
 public int AdmissionCounter { get; private set; } // Счётчик поступлений заявок  
 public int Counter1 { get; private set; } // Счётчик занятости канала 1  
 public int Counter2 { get; private set; } // Счётчик занятости канала 2  
   
 public int P000 { get; private set; }   
 public int P001 { get; private set; }   
 public int P010 { get; private set; }   
 public int P011 { get; private set; }   
 public int P111 { get; private set; }   
  
 public QueuingSystem(double p, double p1, double p2)  
 {  
 J = 0;  
 T1 = 0;  
 T2 = 0;  
 //BlockCounter = 0;  
 colAplication = 0;  
 AdmissionCounter = 0;  
 queryCounter = 0;  
 Counter1 = 0;  
 Counter2 = 0;  
 P000 = 0;  
 P001 = 0;  
 P010 = 0;  
 P011 = 0;  
 P111 = 0;  
 \_rand = new Random();  
 \_rand1 = new Random(101);  
 \_rand2= new Random(4242440);  
 \_p = p; // Вероятность не выдачи заявки  
 \_p1 = p1; // Вероятность не обслуживания заявки каналом 1  
 \_p2 = p2; // Вероятность не обслуживания заявки каналом 2  
 }  
  
 // Была ли не выдана заявка  
 private bool IsNoRequest()  
 {  
 return \_rand.NextDouble() < \_p;  
 }  
  
 // Была ли не обслужена заявка каналом 1  
 private bool IsNoService1()  
 {  
 return \_rand1.NextDouble() < \_p1;  
 }  
  
 // Была ли не обслужена заявка каналом 2  
 private bool IsNoService2()  
 {  
 return \_rand2.NextDouble() < \_p2;  
 }  
  
  
 public void GenerateNextState()  
 {  
 colAplication += J + T1 + T2;  
 if (J == 0)  
 {  
 if (T1 == 0)  
 {  
 if (T2 == 0)  
 {  
 P000++;  
 }  
 else  
 {  
 P001++;  
 }  
 }  
 else  
 {  
 if (T2 == 0)  
 {  
 P010++;  
 }  
 else  
 {  
 P011++;  
 }  
 }  
 }  
 else  
 {  
 P111++;  
 }  
   
   
 if (T1 == 1) // Если в канале 1 есть заявка  
 {  
 if (IsNoService1()) Counter1++;  
 else T1 = 0;  
 }  
   
 if (T2 == 1) // Если в канале 2 есть заявка  
 {  
 if (IsNoService2()) Counter2++;  
 else T2 = 0;  
 }  
   
 if(J == 1) // Если есть заявка в очереди  
 {  
 if (T1 == 0)  
 {  
 J = 0;  
 T1 = 1;  
 Counter1++;  
 }  
 else if (T2 == 0)  
 {  
 J = 0;  
 T2 = 1;  
 Counter2++;  
 }  
 }  
   
   
  
 if (IsNoRequest()) // Если источник не выдал заявку  
 return;  
   
   
  
 if (J == 0) // Если нет заявок в очереди  
 {  
 AdmissionCounter++;  
 if (T1 == 0)  
 {  
 T1 = 1;  
 Counter1++;  
 }  
 else if (T2 == 0)  
 {  
 T2 = 1;  
 Counter2++;  
 }  
 else  
 {  
 J = 1;  
 queryCounter++;  
 }  
 }  
 else // Если есть заявка в очереди  
 {  
 //BlockCounter++; // Увеличиваем счётчик блокировок  
 return;   
 }  
 }  
 }  
}