Лабораторная работа №1

студента группы ИТ – 42

Курбатовой Софьи Андреевны

Выполнение: _____ Защита _____

ОПРЕДЕЛЕНИЕ ВЕРОЯТНОСТЕЙ СОСТОЯНИЙ СИСТЕМЫ МАССОВОГО ОБСЛУЖИВАНИЯ И ФИНАЛЬНЫХ ВЕРОЯТНОСТЕЙ

Цель работы: построение имитационной модели системы массового обслуживания, параметры которой являются детерминированными величинами.

Содержание работы

Система обработки информации содержит мультиплексный канал и N ЭВМ. Сигналы поступают на вход канала через t1(мкс).

В канале они предварительно обрабатываются в течение t2 (мкс). Затем они поступают на обработку в ту ЭВМ, где наименьшая очередь. Емкости входных накопителей в каждой ЭВМ - E. Время обработки сигнала в каждой из ЭВМ - t3 (мкс).

Смоделировать процесс обработки 1000 сигналов.

Данные для детерминированной модели СМО: N=3, t1=10, t2=10 , t3=33, E=4.

Данные для стохастической модели СМО: интервал t1 распределен по показательному закону с параметром λ1=0,1, интервалы t2, t3 распределены нормально с параметрами m2=10, m3=33, σ2=1,5, σ3=3; вследствие возмущающих воздействий емкости входных накопителей каждой из ЭВМ непрерывно меняются, поэтому величина E является стационарным случайным процессом с нормальным законом распределения и интервалом разброса [2... 6] (сигналы, находившиеся в накопителе до изменения его емкости и не вмещающиеся в него после изменения его емкости, уничтожаются).

Варьируемые параметры: N.

Показатели работы: производительность системы, стоимость обработки, вероятность переполнения накопителей.

Условные обозначения: Aij – активность, ФДi – функциональное действие, УЗij – условие запуска.

В системе наблюдаются следующие функциональные действия (ФД):

ФД1 – приход сигнала с интервалом t1

ФД2 – обработка сигнала внутри канала

ФД3 – поступление на обработку в ЭВМ с наименьшей очередью

Предполагается наличие следующих активностей:

A10 – Поступление сигнала в канал

A21 – Обработка сигнала внутри канала

A22 – Конец обработки и переход к следующему

A31 – Определение ЭВМ с меньшей очередью (где емкость больше)

A32 – Выполнение обработки сигнала в ЭВМ

Кобрбсигн – количество обработанных сигналов.

Квх – количество принятых(входных) сигналов
Кпотерсигнал – количество сигналов, которые были потеряны

Содержание активностей предполагается следующее:

Таблица 1

Активность	Условие запуска	Алгоритм
A10: Прием сигнала	(t0-время появления входящего сигнала) > время прихода сигнала	Квх := Квх + 1 (увеличение счётчика количества сигналов проходящих в канал) время появления входящего сигнала:= t0 (вычисление времени прихода следующего сигнала, собирающегося пройти по каналу);
A20: Обработка в канале	(t0 – время обработки записанное в канале) > время на обработку сигнала	Квых := Квх + 1 (увеличение счётчика количества сигналов выходящих из канала)
A21: Поиск ПК с наименьшей очередью		для ПК от i=1 до N если Emin > Ei, то min = i
A30	(t0 – время обработки записанное в канале) > время на обработку сигнала	
A31	Если ЭВМmin не занят и его очередьMIN >=0	время обработки записанное в канале = t0.
		Кобрбсигн= Кобрбсигн +1;
		Если очередь ЭВМ > 0, то очередьMIN = очередьMIN -1
A32	Если ЭВМmin занят и его очередьMIN <4	очередьMIN = очередьMIN +1
A33	Если ЭВМmin занят и его очередьMIN= 4	Кпотерсигнал = Кпотерсигнал +1;

На рисунке 1.1 – 1.3. приведены алгоритмы определения запуска активностей, в соответствии с описанными в таблице 1 условиями. Применяются следующие обозначения:

к_ОС - количество обработанных сигналов
к_ЗОС - количество затребованных для обработки сигналов
ArivT - время прибытия сигнала
prepT - время обработки сигнала в канале
prepTC - время обработки сигнала в ЭВМ
ОчередьMin – ЭВМ с наименьшей очередью

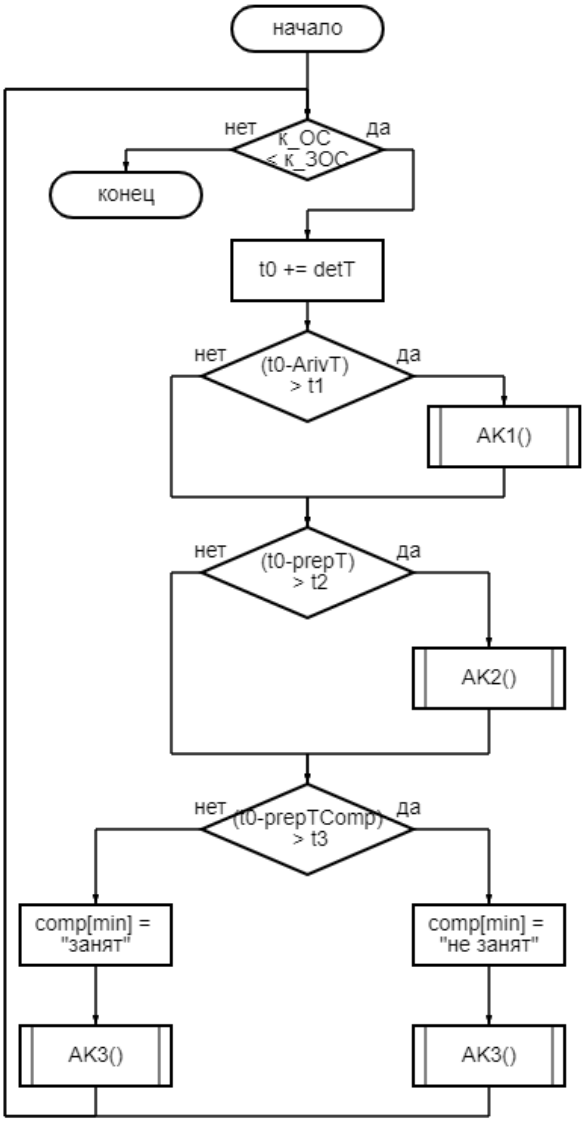

Рис. 1.1. Блок-схема алгоритма моделирования движения сигнала

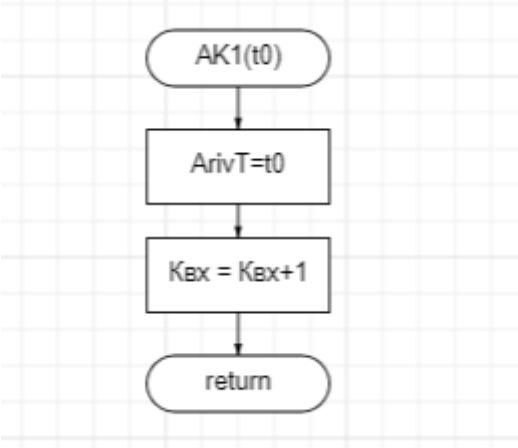

Рис. 1.2. Блок-схема алгоритма первой активности

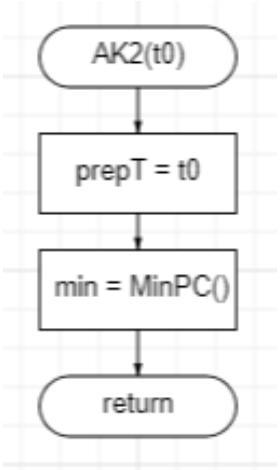

Рис. 1.3. Блок-схема алгоритма второй активности

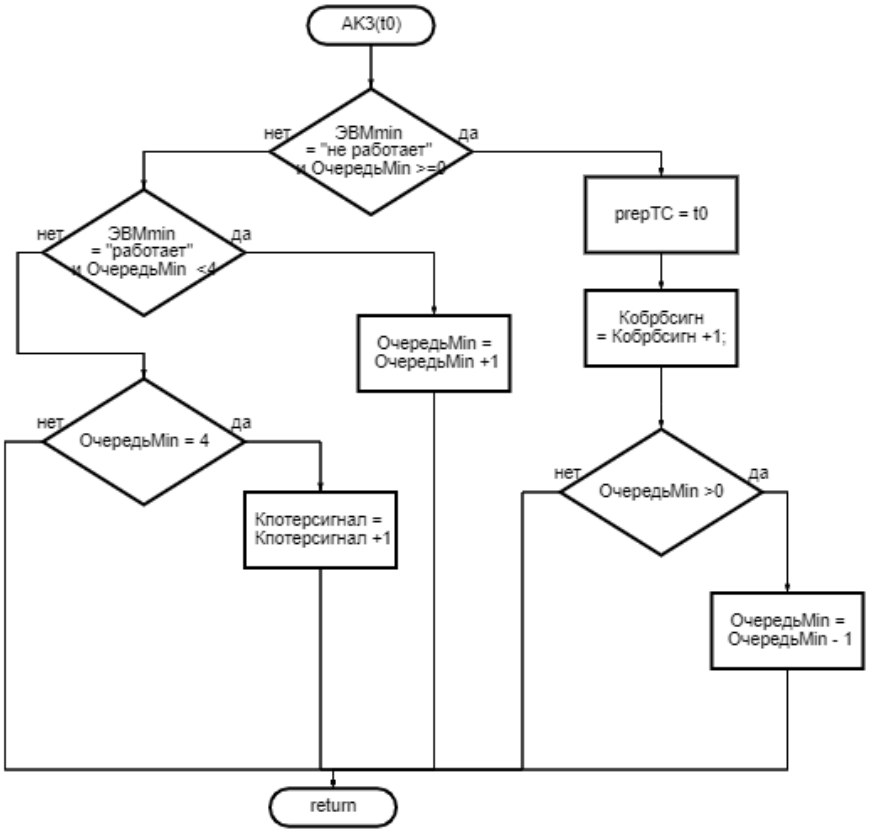

Рис. 1.4. Блок-схема алгоритма третьей активности

Процесс выполнения обработки сигналов представлен на рисунках 1.5 – 1.10.

При запуске приложения заполнение параметров моделирования происходит автоматически в соответствии с указанными в задании параметрами.

На вкладке данные представлен протокол моделирования, который отображает текущее время t0 и какое в этот момент времени выполняется действие.

Вывод: Таким образом в ходе выполнения лабораторной работы было осуществлено построение имитационной модели системы массового обслуживания, параметры которой являются детерминированными величинами. Результатом выполненной работы стало настольное приложение позволяющее смоделировать процесс обработки входящих сигналов.

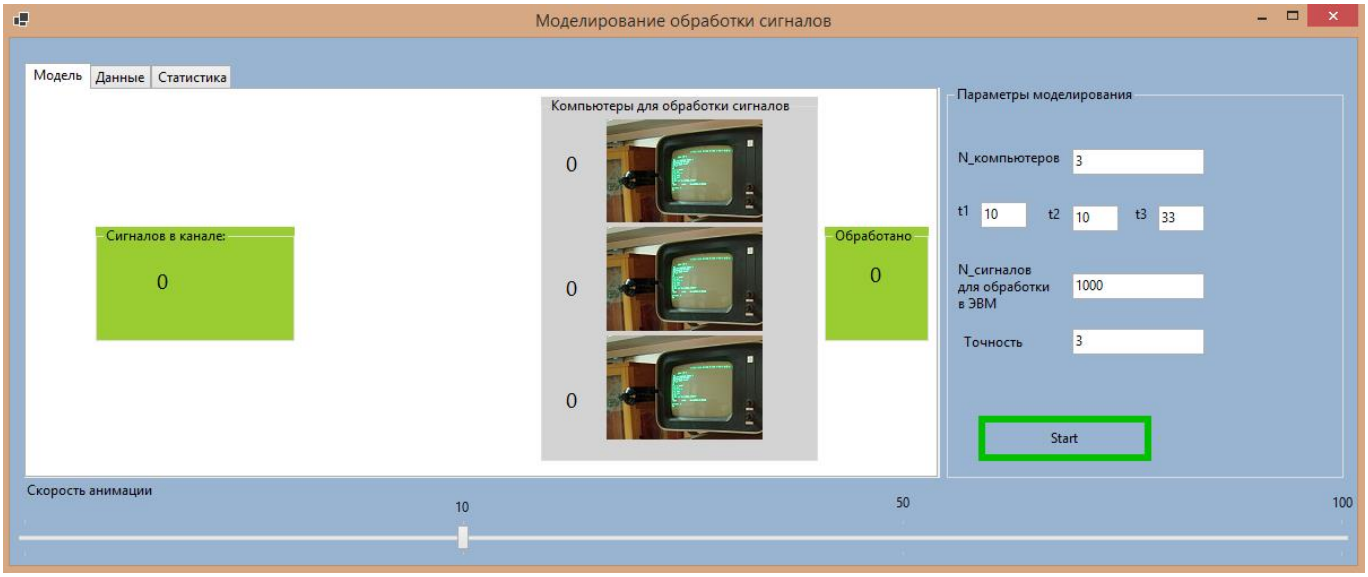

Рис. 1.5. Окно запуска приложения

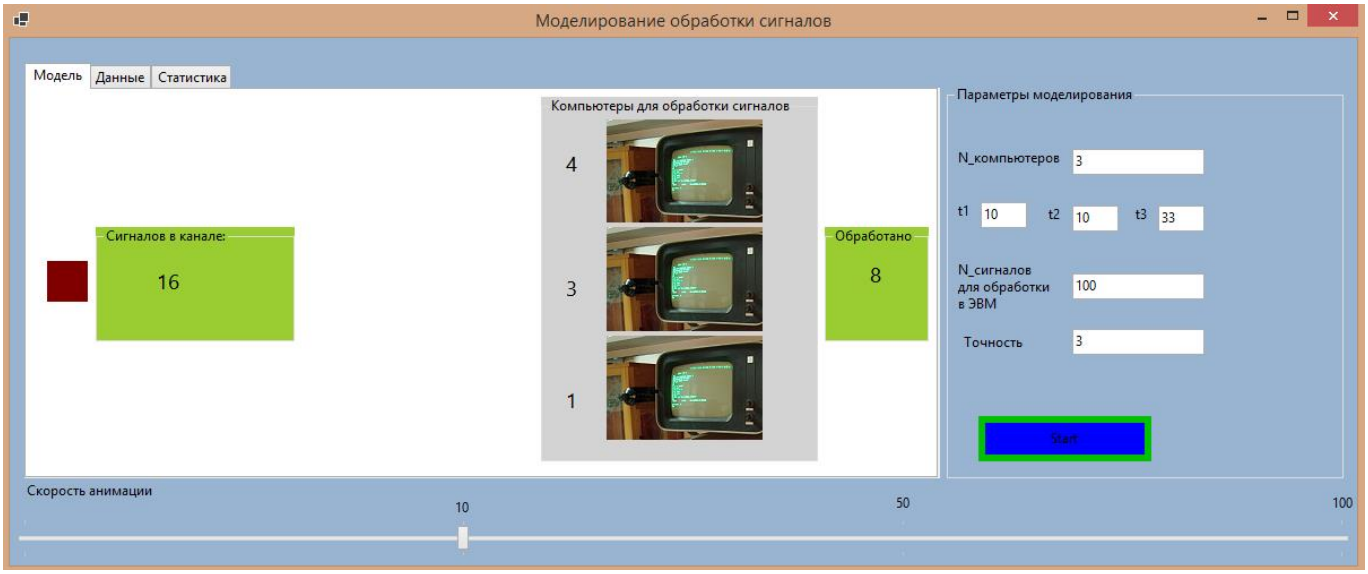

Рис. 1.6. Движение сигнала

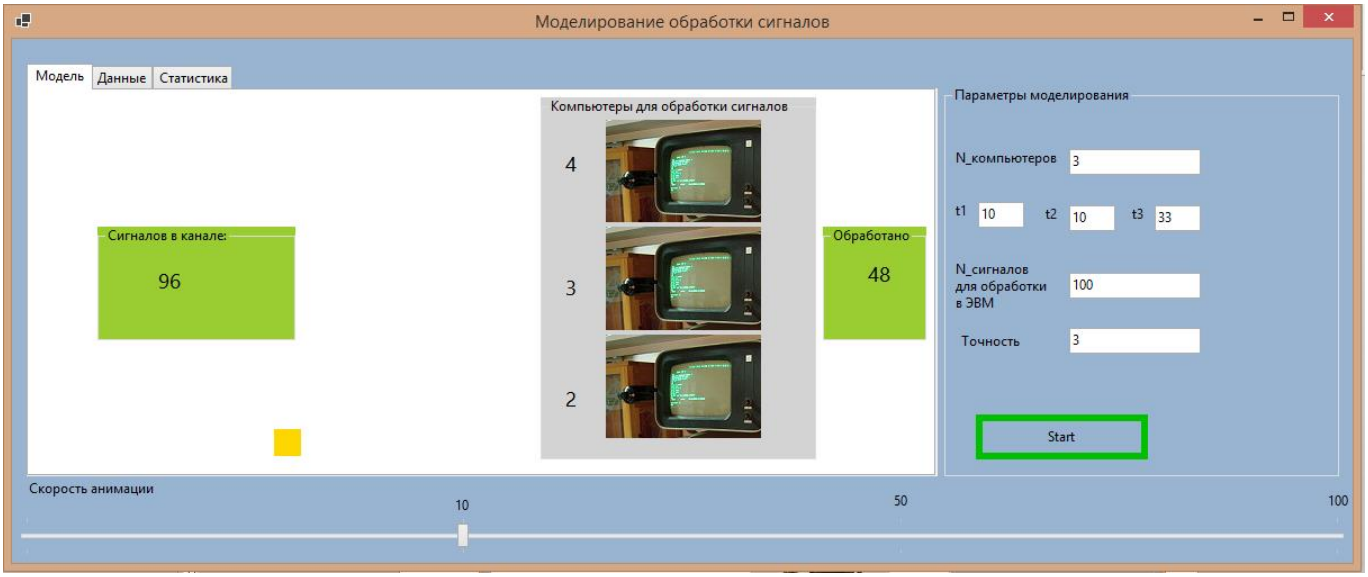

Рис. 1.7. Перемещение сигнала к ЭВМ с меньшей очередью

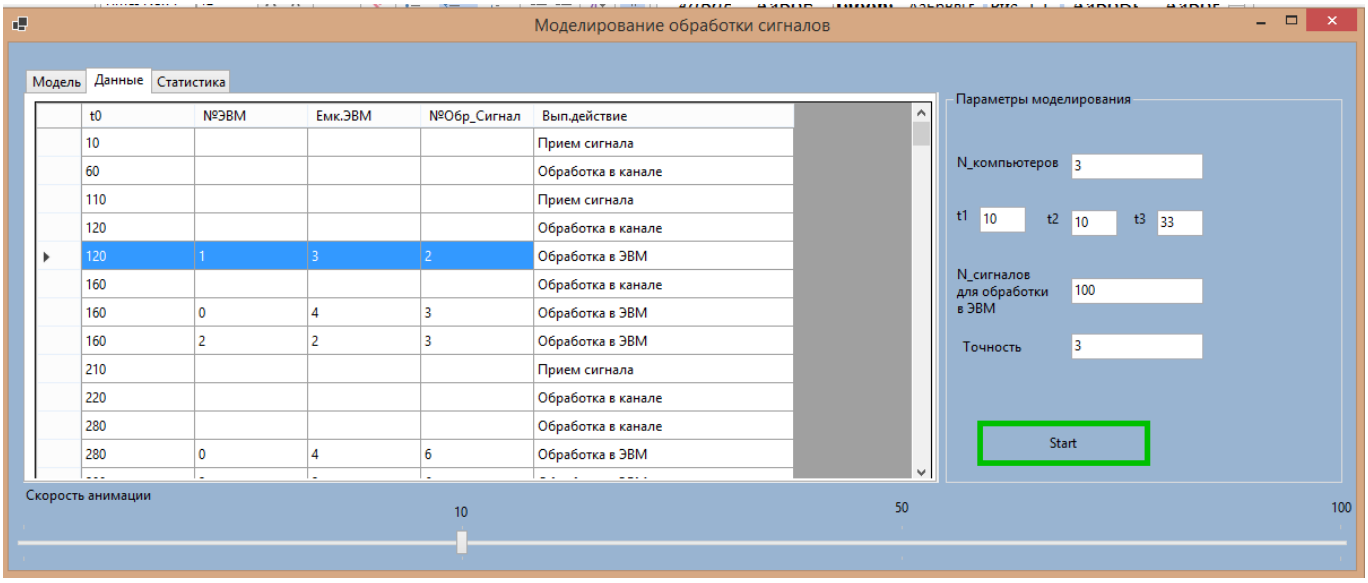

Рис. 1.8. Протокол моделирования

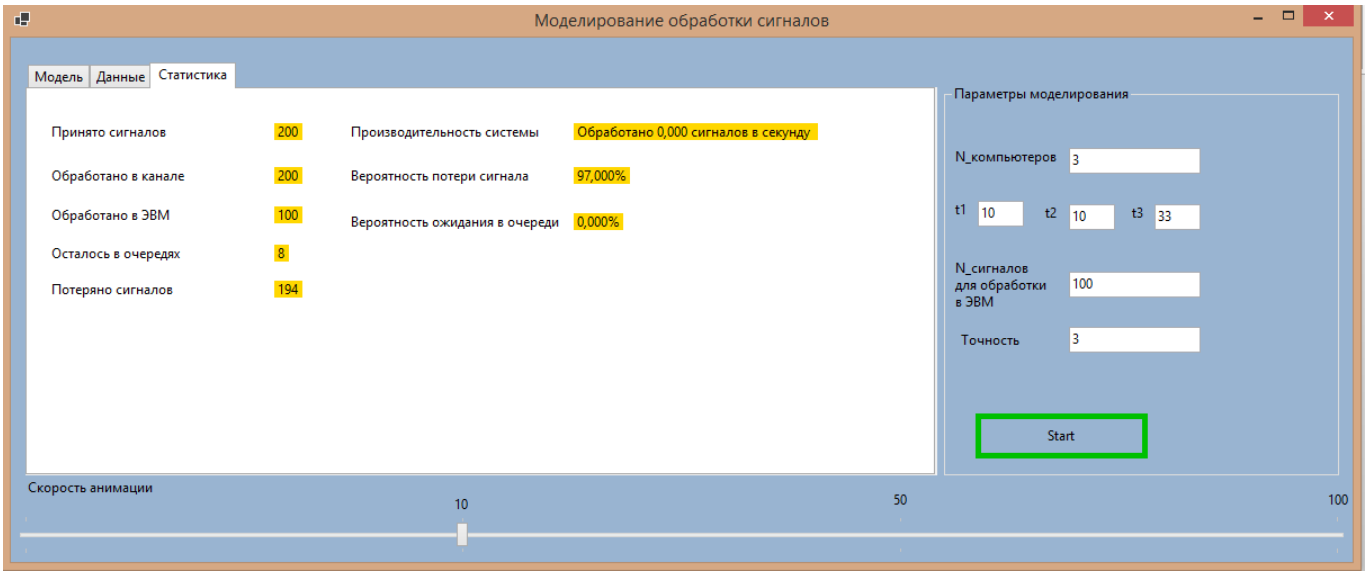

Рис. 1.9. Показатели работы системы при исходных t

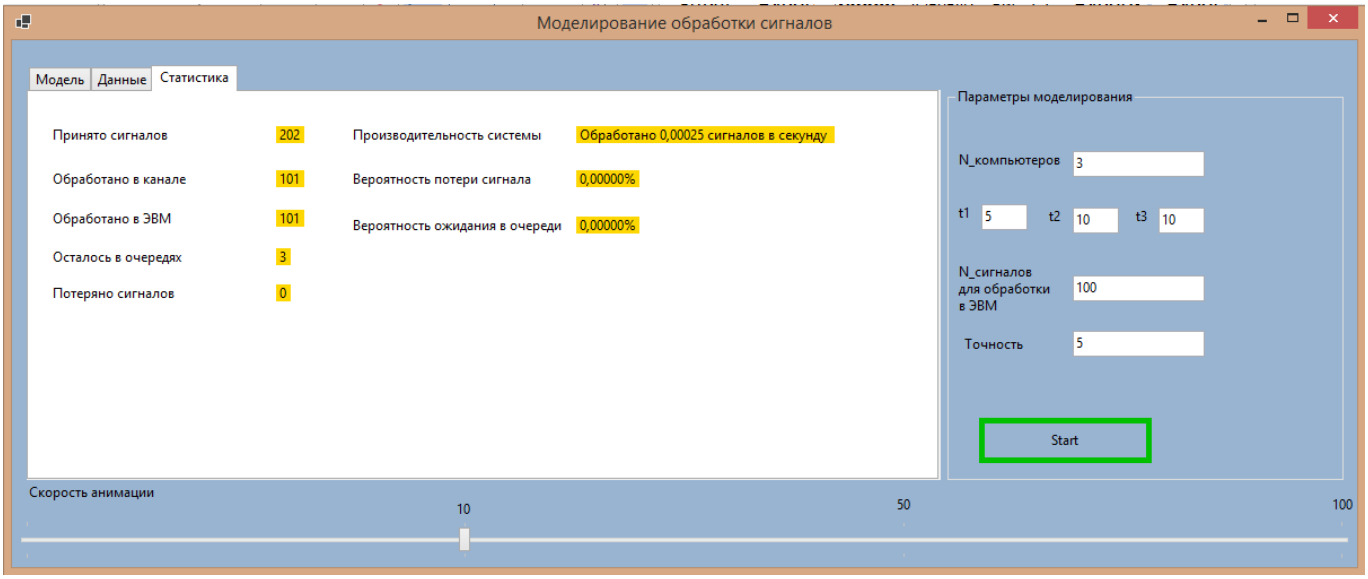

Рис. 1.10. Показатели работы системы при новых t

Листинг 1 Класс для ЭВМ

```
public class comp: IComparer<comp>
{
    public string compName { get; set; } //название группы верхнего уровня
    public int capacity;
    public bool in_work;
    public comp()
    {
        this.compName = "comp_";
        this.capacity = 0; //изначально емкость незаполнена
        this.in_work = false;
    }
    public comp(int Capacity,string compName)
    {
        this.compName = compName;
        this.capacity = Capacity;
    }
    public int Compare(comp o1, comp o2)
    {
        if (o1.capacity > o2.capacity)
        {
            return 1;
        }
        else if (o1.capacity < o2.capacity)
        {
            return -1;
        }
        return 0;
    }
}
```

Листинг 2

```
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 lw_sm_1
{
    public partial class Form1 : Form
    {
        public static List<comp> compList = new List<comp>();
        readonly Timer tmr = new Timer();
        //счетчик канала
        int min = 0, signalCounter = 0, OutSignalCounter = 0;
        bool Visible = false;
        //для детерминированной СМО
        double t0 = 0, N = 3, t1 = 10, t2 = 10, t3 = 33, E = 4, detT = 10;
        //Локальное время обработки
        double arivTime = 0, prepTime = 0, checkTime = 0, prepTimeComp = 0, prepSignal = 0,
lostSignal=0;

        //Для координат
        private int StartX = 0;
        private readonly int StartY = 0;
        private int SignalWidth = 0;
        private int SignalHeight = 0;
        private int Delay = 40;
        public Form1()
        {
            InitializeComponent();
            tmr.Interval = 10;
            tmr.Tick += new EventHandler(tmr_Tick);
```

```
            route.Text = signalCounter.ToString();
            StartX = 16;
            StartY = 151;
            SignalWidth = 36;
            SignalHeight = 36;
            tbT1.Text = "10";
            tbT2.Text = "10";
            tbT3.Text = "33";
            tbEpsilon.Text = "3";
            tbNumComp.Text = "3";
            tbNumSignal.Text = "1000";
            tbSpeed.Scroll += tbSpeed_Scroll;

        }

        private async void DrawSignal()
        {
            Bitmap mybit = new Bitmap(signal.Width, signal.Height);
            Graphics g = Graphics.FromImage(mybit);
            SolidBrush grBrush = new SolidBrush(Color.Green);
            g.FillRectangle(grBrush, 0, 0, signal.Width, signal.Height);
            g.DrawImage(mybit, StartX, StartY);
            SignalMovement();
        }

        private void SignalMovement()
        {
                if (signal.Location.X < 62)
                {
                    SlideLeft();
                }
                else
                {
                    if (signal.Location.X >= 65 && (signal.Location.Y >= 150) &&
signal.Location.X < 366)
                    {
                        ChangeBox(Color.Gold);
                        switch (min)
                        {
                            case 0:
                                while (signal.Location.Y >= 50)
                                {
                                    SlideUp();
                                }
                                while (signal.Location.X <= 366)
                                {
                                    SlideLeft();
                                }
                            //if (signal.Location.X > 366)
                            //{
                            //    SlideStart();
                            //}
                            //SlideStart();
                            break;

                            case 1:
                                while (signal.Location.X <= 366)
                                {
                                    SlideLeft();
                                }
                                //if (signal.Location.X > 340)
                                //{
                                //    SlideStart();
                                //}
                            //SlideStart();
                            break;
                            case 2:
                                while (signal.Location.Y <260)
                                {
                                    SlideDown();
                                }
```

```
                                while (signal.Location.X <= 366)
                                {
                                    SlideLeft();
                                }
                                //if (signal.Location.X > 366)
                                //{
                                //    SlideStart();
                                //}
                                //SlideStart();
                                break;
                            default:
                               // SlideStart();
                            break;
                        }
                        //SlideUp();
                    }
                    else
                        {
                            SlideStart();
                        }
                }
            }


            private void btnStart_Click(object sender, EventArgs e)
            {
                signalCounter = 0;
                t0 = 0;
                arivTime = 0; prepTime = 0; checkTime = 0; prepTimeComp = 0; prepSignal = 0;
OutSignalCounter = 0;lostSignal = 0;

                if (tmr.Enabled==false)
                {
                    t1 = Convert.ToDouble(tbT1.Text.Trim());
                    t2 = Convert.ToDouble(tbT2.Text.Trim());
                    t3 = Convert.ToDouble(tbT3.Text.Trim());
                    tmr.Enabled = true; //старт/стоп
                    compList.Clear();
                    logTable.Rows.Clear();

                    for (int i = 0; i < Convert.ToDouble(tbNumComp.Text.Trim()); i++)
                    {
                        compList.Add(new comp());
                    }
                     lbComp1.Text = Convert.ToString(compList[0].capacity);
                     lbComp2.Text = Convert.ToString(compList[1].capacity);
                     lbComp3.Text = Convert.ToString(compList[2].capacity);

                    Task.Run(() =>
                        {
                            Count();
                        });
                }
                else
                {
                    tmr.Enabled = false; //старт/стоп
                    //StatData();
                }

            }

            private void SlideLeft()
            {
                signal.Location = new Point(signal.Location.X + 50, signal.Location.Y);
                Task.Delay(Delay).Wait();
            }

            private void ChangeBox(Color color)
            {
                signal.BackColor = color;
                signal.Width = Convert.ToInt32( SignalWidth/1.5 );
```

```
    signal.Height = Convert.ToInt32(SignalHeight/1.5);
}

private void SlideUp()
{
    signal.Location = new Point(signal.Location.X, signal.Location.Y - 50);
    Task.Delay(Delay).Wait();
}
private void SlideDown()
{
    signal.Location = new Point(signal.Location.X, signal.Location.Y + 50);
    Task.Delay(Delay).Wait();
}
private void SlideStart()
{
    signal.Location = new Point(StartX, StartY);
    signal.Width = SignalWidth;
    signal.Height = SignalHeight;
    signal.BackColor = Color.Maroon;
    Task.Delay(Delay).Wait();
}
private void AK1()
{
    arivTime = t0;
    signalCounter++;
    Task.Delay(Delay).Wait();
}
private void AK2()
{
    checkTime += t2;//для фиксации общего времени проверки в канале
    prepTime = t0;
    OutSignalCounter++;//считаем количесво исходящих сигналов

    //Определим где наименьшая очередь
    min = MinPC();
    Task.Delay(Delay).Wait();
}

private int MinPC()
{
    int locMin = 0;
    //Определим где наименьшая очередь
    for (int i = 0; i < compList.Count(); i++)
    {
        if (compList[min].capacity > compList[i].capacity)
        {
            locMin = i;
        }
    }
    return locMin;
}

private void AK3()
{

    if ((compList[min].in_work == false)&&(compList[min].capacity>=0))
    {
        prepTimeComp = t0;
        prepSignal++;
        if(compList[min].capacity >0)
        {
            compList[min].capacity--;
            Visible = false;
        }
    }
    else
    {
        if((compList[min].in_work == true)&&(compList[min].capacity<4))
        {
            compList[min].capacity++;
            Visible = true;
```

```
        }
        else
        {
            if ((compList[min].in_work == true) && (compList[min].capacity == 4))
            {
                lostSignal++;
            }
        }
    }
     Task.Delay(Delay).Wait();
}

private void Count()
{
        for (; prepSignal <= Convert.ToDouble(tbNumSignal.Text.Trim()); t0 += detT)
        {
            if ((t0 - arivTime) > t1)
            {
                AK1();//выполнение первого действия
            }
            if ((t0- prepTime)>t2)
            {
                AK2();//выполнение обработки в канале
            }
            if ((t0 - prepTimeComp) > t3)
            {
               compList[min].in_work = false;
               AK3();
            }
            else
            {
                compList[min].in_work = true;
                AK3(); //обработка в ПК
            }
        }
}
private void tbSpeed_Scroll2(object sender, EventArgs e)
{
    if (tbSpeed.Value >= 20)
    {
       detT = 30;
    }
    else
    {
        if ((tbSpeed.Value >= 10) || (tbSpeed.Value <= 20))
        {
            detT = 20;
        }
        else
        {
            if ((tbSpeed.Value >= 0) || (tbSpeed.Value <= 10))
            {
                detT = 10;
            }
        }
    }
}
async void tmr_Tick(object sender, EventArgs e)
{

    route.Text = signalCounter.ToString();
    lbPrepVisual.Text = prepSignal.ToString();
    signal.Visible = true;
    if ((t0 - arivTime) == t1)
    {
        logTable.Rows.Add(Convert.ToString(t0), Convert.ToString(arivTime),
            " ", " ", " ", " ", " ", "Прием сигнала");
        signal.Visible = true;
    }
    if ((t0 - prepTime) > t2)
    {
```

```
                    logTable.Rows.Add(Convert.ToString(t0), " ",
                    Convert.ToString(prepTime), " ", " ", " ", " ", "Обработка в канале");
                    signal.Visible = false;
                }
                if ((t0 - prepTimeComp) > t3)
                {
                    logTable.Rows.Add(Convert.ToString(t0), "", "", Convert.ToString(prepTimeComp),
                    Convert.ToString(min),               Convert.ToString(compList[min].capacity),
Convert.ToString(prepSignal), "Обработка в ЭВМ");
                }
                SignalMovement();
                switch (min)
                {
                    case 0:
                        lbComp1.Text = Convert.ToString(compList[0].capacity);
                        break;
                    case 1:
                        lbComp2.Text = Convert.ToString(compList[1].capacity);
                        break;
                    case 2:
                        lbComp3.Text = Convert.ToString(compList[2].capacity);
                        break;
                    default:
                        break;
                }
                if (prepSignal >= Convert.ToDouble(tbNumSignal.Text.Trim()))
                {
                    tmr.Enabled = false; //старт/стоп
                    StatData();
                }
            }

            private async void StatData()
            {
                lbSignalCounter.Text = signalCounter.ToString();
                lbPrepSignal.Text = prepSignal.ToString();
                lbOuterSignal.Text = OutSignalCounter.ToString();
                lbLostSignal.Text = lostSignal.ToString();
                var    waitPosition    =    compList[0].capacity    +    compList[1].capacity    +
compList[2].capacity;
                lbAllCapacity.Text = waitPosition.ToString();
                var lost = (lostSignal / signalCounter) * 100;
                if (lost > 100)
                {
                    lost = 100;
                }
                lbPSignal.Text = lost.ToString("F" + tbEpsilon.Text.Trim()) + "%";
                //Вероятность ожидания в очереди сигнала
                var wait = 100 * ((signalCounter- waitPosition) / signalCounter);
                if (wait > 100)
                {
                    wait = 100;
                }
                lbWait.Text = wait.ToString("F" + tbEpsilon.Text.Trim()) + "%";
                lbProd.Text = "Обработано " + ((prepSignal / signalCounter) / t0).ToString("F" +
tbEpsilon.Text.Trim()) + " сигналов в секунду ";
            }
        //двойная буферизация
            protected override CreateParams CreateParams
            {
                get
                {
                    var cp = base.CreateParams;
                    cp.ExStyle |= 0x02000000;    // WS_EX_COMPOSITED
                    return cp;
                }
            }
        }
    }
```