Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ-7)

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

НА ТЕМУ:

Программная имитация i-го прибора

Студент ИУ7-78 А.В. Иванников
(Группа) (И.О.Фамилия)
Вариант 4

Преподаватель И.В. Рудаков
(И.О.Фамилия)

2020 г.

Выбор языка программирования приложения

В качестве языка программирования приложения был выбран язык C#.

Цель работы

Смоделировать систему, состоящую из генератора, источника информации, блока памяти и обслуживающего аппарата (ОА). Определить объем памяти, при котором не будет потерянных сообщений.

Закон генерации заявок - равномерный (параметры настраиваются и варьируются). В ОА – нормальный закон распределения.

Управляющую программу имитационной модели реализовать по двум принципам:

- принципу Δt;

- событийному принципу.

Методика построения программной модели вычислительной системы

Для разработки программной модели исходная вычислительная система (далее в тексте – ВС) должна быть представлена как стохастическая система массового обслуживания. Это можно объяснить следующим: информация от внешней среды поступает в случайные моменты времени, длительность обработки различных типов информации может быть в общем случае различна. Внешняя среда является генератором сообщений. Комплекс вычислительных устройств – обслуживающими устройствами.

Обобщенная структурная схема ВС.

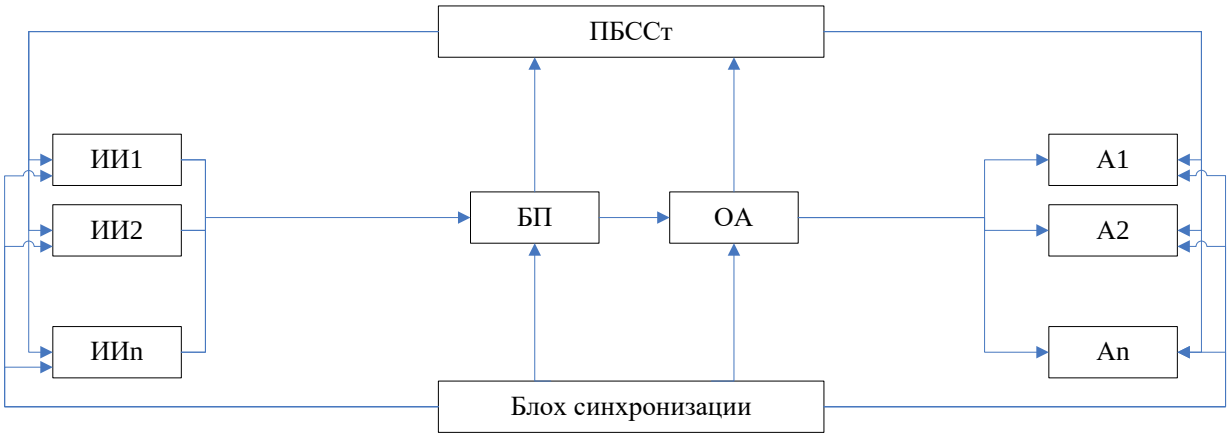

Рисунок 1 – Обобщенная структурная схема ВС

ИИ (источники информации) – выдают на вход буферной памяти (БП) независимые друг от друга сообщения. Закон появления сообщений – произвольный, но задан на перед.

В БП (буферной памяти) сообщения записываются и выбираются по одному в обслуживающий аппарат (ОА) по принципу FIFO/LIFO. Длительность обработки одного сообщения в ОА в общем случае так же может быть случайной, но закон обработки сообщений должен быть задан. Так как. быстродействие ОА ограничено, то на входе системы в БП возможно сложение данных ожидающих обработки.

А – абоненты.

Моделирование работы источника информации

Поток сообщений обычно имитируется моментами времени, отображающими появление очередного сообщения в потоке. Схема для расчета времени работы источника информации приведена на рисунке 2.

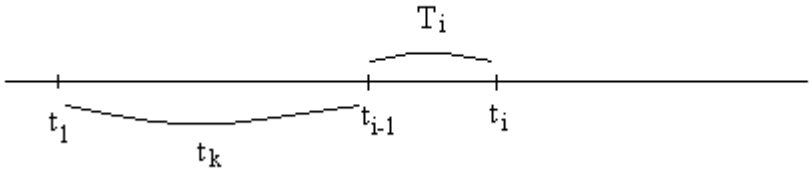

Рисунок 2 – Схема для расчета времени работы источника информации

$$t_i = \sum_{k=1}^{i-1} T_k + T_i,$$

где T_i – интервал времени между появлением i-го и (i-1)-го сообщения.

Программа – имитатор выработки интервалов:

1. Обратиться к генератору равномерно распределенных случайных величин на [a,b].
2. T_i – по заданному закону.
3. К текущему времени + T_i.

Выражения для вычисления времени с различным распределением приведены в таблице 1.

Таблица 1 – Выражения для вычисления времени с различным распределением

Вид распределения	**Выражение**
Равномерное на [a,b]	$T_i = a + (b-a)R$
Нормальное	$T_i = \sigma_X \sqrt{\frac{12}{n}}(\sum_{i=1}^{n} R_i - \frac{n}{2}) + M_X$, n ~= 12
Экспоненциальное	$T_i = -\frac{1}{\lambda}\ln(1-R)$
Эрланга	$T_i = \frac{1}{k\lambda}\sum_{i=1}^{k}\ln(1-R_i)$

Моделирование работы обслуживающего аппарата

Программа-имитатор работы ОА представляет собой комплекс, вырабатывающий случайные отрезки времени, соответствующие длительностям обслуживания требований. Например, если требования от источника обрабатываются в ОА по нормальному закону с параметрами M_x и σ_x, то длительность обработки i-ого требования:

$$T_{обр} = M_x + (\sum_{i=1}^{12} R_i - 6) \cdot \sigma_x$$

Схема алгоритма имитатора приведена на рисунке 3.

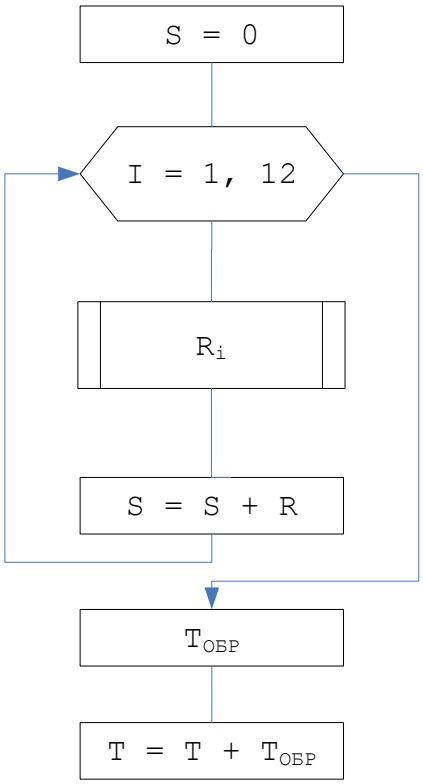

Рисунок 3 – Схема алгоритма имитатора

R_i – случайное число с равномерным законом распределения

$T_{ОБР}$ – время обработки очередного сообщения

T – время освобождения ОА

Управляющая программа имитационной модели

Если программа-имитатор работы источника или буферной памяти обслуживающего аппарата имитируют работу отдельных устройств, то управляющая программа имитирует алгоритм взаимодействия отдельных устройств системы.

Управляющая программа реализуется в основном по двум принципам:

- принципу Δt;
- событийному принципу.

Принцип Δt

Принцип Δt заключается в последовательном анализе состояний всех блоков в момент $t + \Delta t$ по заданному состоянию блоков в момент t. При этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием с учетом действующих случайных факторов, задаваемых распределениями вероятности. В результате такого анализа принимается решение о том, какие общесистемные события должны имитироваться программной моделью на данный момент времени.

Основной недостаток этого принципа: значительные затраты машинного времени на реализацию моделирования системы. А при недостаточно малом Δt появляется опасность пропуска отдельных событий в системе, что исключает возможность получения адекватных результатов при моделировании.

Достоинство: равномерная протяжка времени.

Событийный принцип

Характерное свойство систем обработки информации то, что состояние отдельных устройств изменяются в дискретные моменты времени, совпадающие с моментами времени поступления сообщений в систему,

временем поступления окончания задачи, времени поступления аварийных сигналов и т.д. Поэтому моделирование и продвижение времени в системе удобно проводить, используя событийный принцип, при котором состояние всех блоков имитационной модели анализируется лишь в момент появления какого-либо события. Момент поступления следующего события определяется минимальным значением из списка будущих событий, представляющего собой совокупность моментов ближайшего изменения состояния каждого из блоков системы.

Пример работы приложения

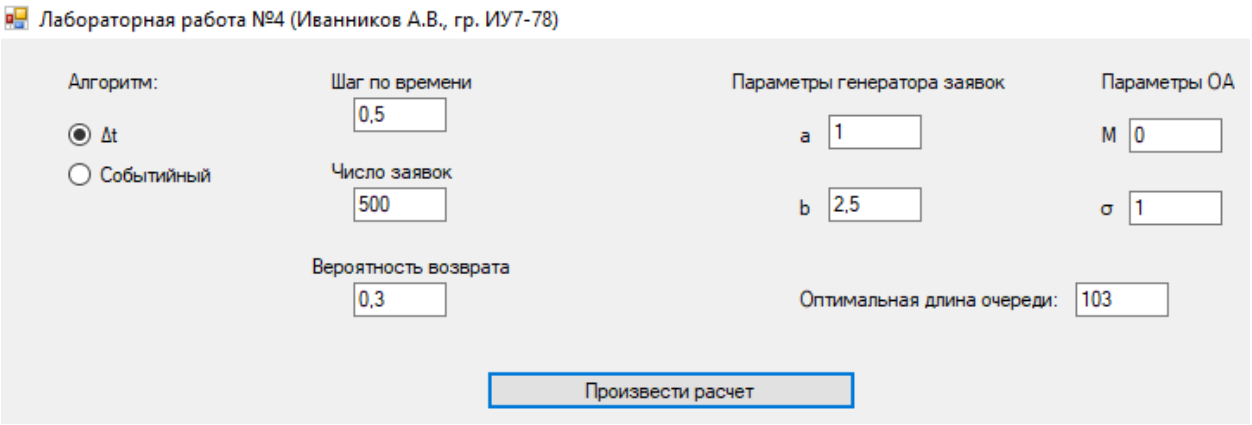

Рисунок 4 – Пример работы приложения

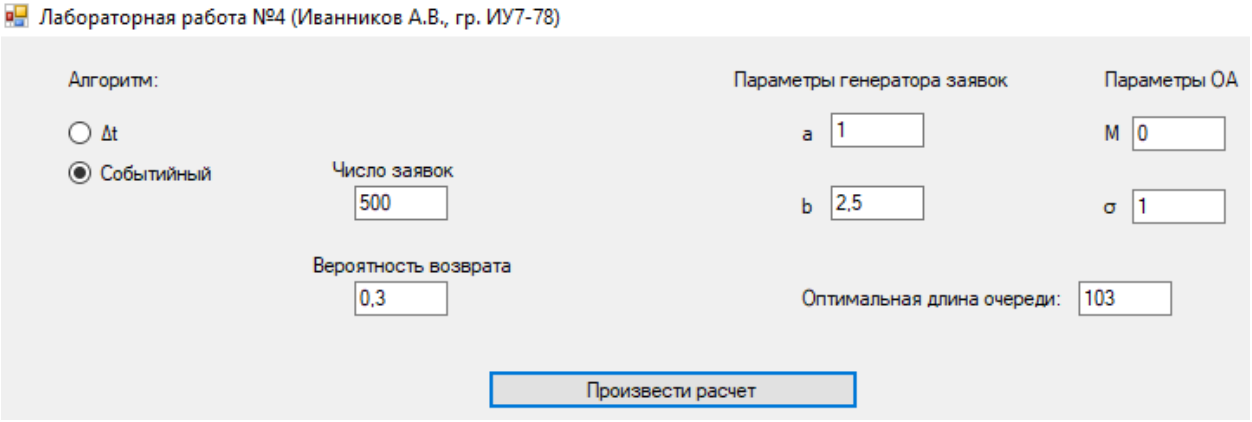

Рисунок 5 – Пример работы приложения

Листинг программного кода приложения

```
/*ModelController.cs*/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace СМО_с_очередью
{
    abstract class ModelController
    {
        protected int MAX_QUEUE_LENGTH; //Максимальная длина очереди

        protected int numRequests; //Число заявок

        private UniformTimeRandomizer recvTimeGenerator; //Генератор заявок
        private NormalTimeRandomizer handlTimeGenerator; //ОА

        private DecisionRandomizer loopBackDecisionRandomizer; //Вовзращение заявки

        protected double[] recvTime; //Время получения заявки

        protected bool requestIsBeingHandled; //Заявка обрабатывается

        protected double handlingFinishTime; //Время обработки заявки

        public double CurrentTime { get; protected set; } //Текущее время
        public int QueueLength { get; protected set; } //Длина очереди
        public int MaxQueueLength { get; protected set; } //Максимальная длина очереди

        public int NumSentRequests { get; protected set; } //Количество отправленных
заявок
        public int NumReceivedRequests { get; protected set; } //Количество полученных
заявок
        public int NumDeclinedRequests { get; protected set; } //Количество отклоненных
заявок
        public int NumHandledRequests { get; protected set; } //Количество обработанных
заявок
        public int NumLoopedBackRequests { get; protected set; } //Количество
возвращенных заявок

        public ModelController(int nReq, UniformTimeRandomizer recvTimeGen,
            NormalTimeRandomizer handlTimeGen, DecisionRandomizer lbDecisionRand)
        {
            numRequests = nReq;

            recvTime = new double[numRequests];

            recvTimeGenerator = recvTimeGen;
            handlTimeGenerator = handlTimeGen;
            loopBackDecisionRandomizer = lbDecisionRand;

            MAX_QUEUE_LENGTH = nReq;
        }

        /*Инициализация*/
        public void Initialize()
        {
            recvTime[0] = recvTimeGenerator.NextValue();
```

```
            /*Время работы генератора заявок*/
            for (int i = 1; i < numRequests; i++)
            {
                recvTime[i] = recvTime[i - 1] + recvTimeGenerator.NextValue();
            }

            requestIsBeingHandled = false;

            QueueLength = 0;
            MaxQueueLength = 0;
            CurrentTime = 0;

            NumSentRequests = 0;
            NumReceivedRequests = 0;
            NumDeclinedRequests = 0;
            NumHandledRequests = 0;
            NumLoopedBackRequests = 0;
        }

        public void MoveOn()
        {
            PassTime();

            MaxQueueLength = Math.Max(MaxQueueLength, QueueLength);

            if (requestIsBeingHandled) //Если заявка в обоаботке
            {
                if (handlingFinishTime <= CurrentTime)
                {
                    NumHandledRequests++;

                    if (loopBackDecisionRandomizer.ShouldPerformAction())
                    {
                        NumLoopedBackRequests++;
                        handlingFinishTime = CurrentTime +
handlTimeGenerator.NextValue();
                    }
                    else
                    {
                        requestIsBeingHandled = false;
                    }
                }
            }
            else if (QueueLength > 0) //Если в буфере есть заявка
            {
                QueueLength--;
                handlingFinishTime = CurrentTime + handlTimeGenerator.NextValue();
                requestIsBeingHandled = true;
            }

            if (NumSentRequests < numRequests &&
                recvTime[NumSentRequests] <= CurrentTime)
            {
                NumSentRequests++;
                if (QueueLength <= MAX_QUEUE_LENGTH)
                {
                    NumReceivedRequests++;
                    QueueLength++;
                }
                else
                {
                    NumDeclinedRequests++;
                }
            }
        }
```

```
        public abstract void PassTime();

        public bool Finished()
        {
            return NumSentRequests == numRequests &&
                NumHandledRequests == NumReceivedRequests + NumLoopedBackRequests;
        }
    }

    class TimeModelController : ModelController
    {
        public double TimeStep { get; private set; }

        public TimeModelController(int nReq, UniformTimeRandomizer recvTimeGen,
        NormalTimeRandomizer handlTimeGen, DecisionRandomizer lbDecisionRand, double
timeStep):
        base(nReq, recvTimeGen, handlTimeGen, lbDecisionRand)
        {
            TimeStep = timeStep;
        }

        public override void PassTime()
        {
        CurrentTime += TimeStep;
        }
    }

    class EventModelController : ModelController
    {
        private const double ERROR = 1e-7;

        public EventModelController(int nReq, UniformTimeRandomizer recvTimeGen,
            NormalTimeRandomizer handlTimeGen, DecisionRandomizer lbDecisionRand) :
            base(nReq, recvTimeGen, handlTimeGen, lbDecisionRand)
        { }

        public override void PassTime()
        {
            if (NumReceivedRequests == numRequests)
            {
                CurrentTime = handlingFinishTime + ERROR;
            }
            else
            {
                if (requestIsBeingHandled)
                {
                    CurrentTime = Math.Min(handlingFinishTime,
                        recvTime[NumSentRequests]) + ERROR;
                }
                else
                {
                    CurrentTime = recvTime[NumSentRequests] + ERROR;
                }
            }
        }
    }
}
```

```
/*TimeRandomizer.cs*/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace СМО_с_очередью
{
    interface TimeGenerator
    {
        double NextValue();
    }

/*Моделирование генератора заявок (равномерное распределение)*/
    class UniformTimeRandomizer: TimeGenerator
    {
        private double a, b;

        private Random rnd = new Random();

        public UniformTimeRandomizer(double leftValue, double rightValue)
        {
            a = leftValue;
            b = rightValue;
        }


        public double NextValue()
        {
            double T = 0;

            T = a + (b - a) * rnd.NextDouble();

            return T;
        }
    }

    /*Моделирование обслуживающего аппарата (нормальное распределение)*/
    class NormalTimeRandomizer : TimeGenerator
    {
        private double M, sigma;

        private Random rnd = new Random();

        public NormalTimeRandomizer(double m, double s)
        {
            M = m;
            sigma = s;
        }

        public double NextValue()
        {
            double s = 0, T = 0;

            for(int i = 1; i < 12; i++)
            {
                s += rnd.NextDouble();
            }

            T = Math.Abs(M + (s - 6) * sigma);

            return T;
        }
    }
```

```
}


/*DecisionRandomizer.cs*/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace СМО_с_очередью
{
    class DecisionRandomizer
    {
        private double probability; //Вероятность возврата заявки

        private Random rnd = new Random();

        public DecisionRandomizer(double prob)
        {
            probability = prob;
        }

        public bool ShouldPerformAction()
        {
            return rnd.NextDouble() <= probability;
        }
    }
}
```