



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 №5
НА ТЕМУ:

Моделирование работы информационного центра

Студент ИУ7-78
(Группа)
Вариант 4

А.В. Иванников
(И.О.Фамилия)

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 информационный центр со следующими характеристиками:

- интервал подачи заявок клиентами – 10 ± 2 минуты;
- количество операторов, обслуживающих заявки клиентов – 3;
- время обработки заявок операторами:
 - 1) первый оператор – 20 ± 5 минут;
 - 2) второй оператор – 40 ± 10 минут;
 - 3) третий оператор - 40 ± 20 минут;
- количество компьютеров, обслуживающих запросы операторов – 2;
- время обработки запросов компьютерами:
 - 1) первый компьютер – 15 минут;
 - 2) второй компьютер – 30 минут.

Если все три имеющихся оператора заняты, клиенту отказывают в обслуживании. Клиенты стремятся занять свободного оператора с максимальной производительностью. Полученные операторами запросы сдаются в накопитель, откуда выбираются на обработку. На первый компьютер запросы от первого и второго операторов, на второй – запросы от третьего.

В процессе взаимодействия клиентов с информационным центром возможны:

- режим нормального обслуживания, т.е. клиент выбирает одного из свободных операторов, отдавая предпочтение тому, у которого меньше номер.
- режим отказа в обслуживании клиента, когда все операторы заняты

Схема информационного центра приведена на рисунке 1.



Рисунок 1 – Схема информационного центра

В информационный центр поступает 300 заявок от клиентов. За единицу системного времени выбрать 0,01 минуты.

Теоретическая часть

Для решения задачи моделирования информационного центра необходимо создать концептуальную модель системы, определить экзогенные и эндогенные переменные.

Под концептуальной моделью понимаются сведения о выходных и конструктивных параметрах системы, её структуре, особенности работы каждого ресурса (элемента системы), характере взаимодействия между ресурсами.

Значения экзогенных переменных задаются извне, а значения эндогенных переменных формируются внутри модели.

Переменные модели информационного центра:

- экзогенные переменные - число обслуженных клиентов и число клиентов получивших отказ.
- эндогенные переменные: время обработки задания i -ым оператором, время решения этого задания j -ым компьютером.

Структурная схема информационного центра приведена на рисунке 2.

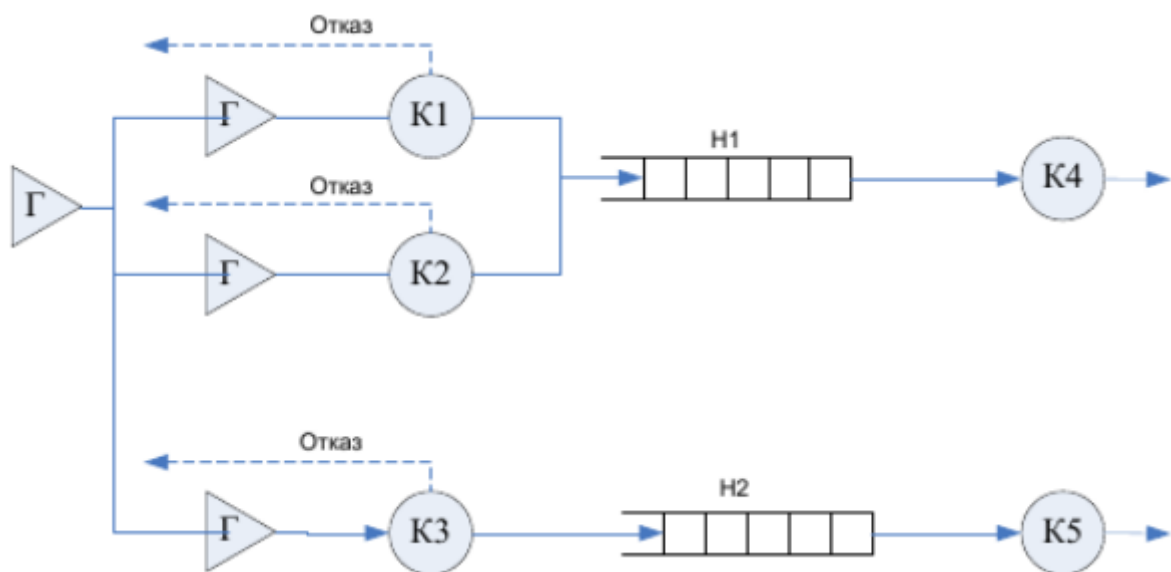


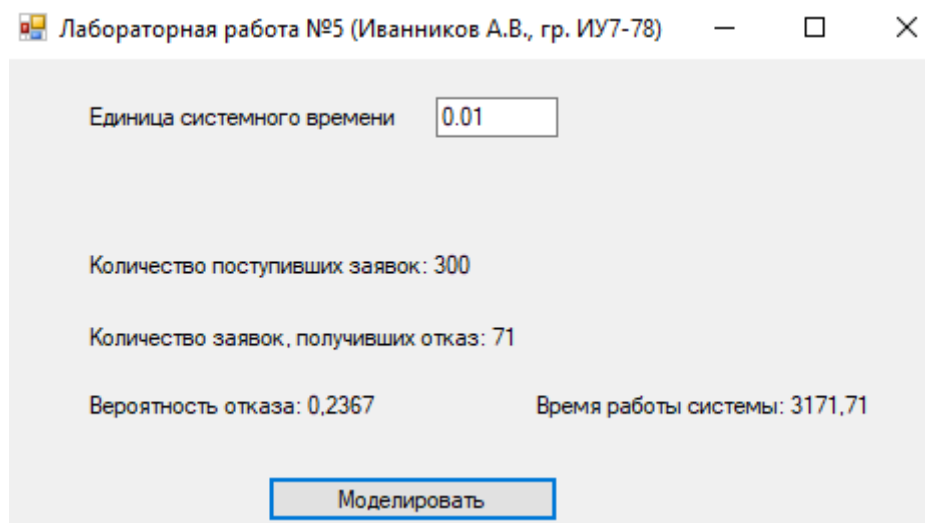
Рисунок 2 – Структурная схема информационного центра

Вероятность отказа вычисляется по формуле $P_{\text{отк}} = \frac{C_{\text{отк}}}{C_{\text{отк}} + C_{\text{обсл}}}$, где

$C_{\text{отк}}$ – количество заявок, получивших отказ,

$C_{\text{обсл}}$ – количество обслуженных заявок.

Пример работы приложения



Лабораторная работа №5 (Иванников А.В., гр. ИУ7-78)

Единица системного времени

Количество поступивших заявок: 300

Количество заявок, получивших отказ: 71

Вероятность отказа: 0,2367 Время работы системы: 3171,71

Рисунок 3 – Пример работы приложения

Листинг программного кода приложения

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

namespace Моделирование_информационного_центра
{
    class Client
    {
        /*Интервал времени, через который будет подана следующая заявка*/
        public double NextRequestInterval { get; private set; } = 0;

        private readonly int minRequestTimeInterval, maxRequestTimeInterval;

        private readonly ITimeRandomizer nextRequestTimeRandomizer;

        public Client(int minReqTimeInt, int maxReqTimeInt,
            ITimeRandomizer timeRand = null)
        {
            minRequestTimeInterval = minReqTimeInt;
            maxRequestTimeInterval = maxReqTimeInt;
            nextRequestTimeRandomizer = timeRand ?? new TimeRandomizer();
        }

        public bool MoveOn(double dt)
        {
            /*Счетчик времени до следующей заявки*/
            NextRequestInterval -= dt;

            if (NextRequestInterval <= 0)
            {
                NextRequestInterval = nextRequestTimeRandomizer
                    .NextValue(minRequestTimeInterval, maxRequestTimeInterval);
                return true;
            }
            return false;
        }
    }
}

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

namespace Моделирование_информационного_центра
{
    class Model
    {
        private Client client = new Client(minReqTimeInt: 8, maxReqTimeInt: 12);

        private readonly Operator[] operators;
```

```

private readonly ProcessingSystem[] processingSystems =
{
    new ProcessingSystem(minProcTime: 15, maxProcTime: 15),
    new ProcessingSystem(minProcTime: 30, maxProcTime: 30)
};

public double TimeStep { get; set; }

public double CurrentTime { get; private set; }

public bool Active
{
    get
    {
        return operators.Any(o => o.Active) ||
            processingSystems.Any(s => s.Active);
    }
}

public int NumHandledRequests { get; private set; }

public int NumDeclinedRequests { get; private set; }

public Model(double timeStep = 0.01)
{
    TimeStep = timeStep;

    operators = new Operator[]
    {
        new Operator(minServTime: 15, maxServTime: 25, procSys:
processingSystems[0]),
        new Operator(minServTime: 30, maxServTime: 50, procSys:
processingSystems[0]),
        new Operator(minServTime: 20, maxServTime: 60, procSys:
processingSystems[1])
    };
}

public void Restart()
{
    CurrentTime = 0;
}

public void MoveOn()
{
    CurrentTime += TimeStep;

    bool newRequestReceived = client.MoveOn(TimeStep);

    if (newRequestReceived)
    {
        var availableOperators =
            from o in operators
            where o.Active == false
            orderby o.ProcessingSystem.QueueLength, o.MinServiceTime
            select o;

        if (Enumerable.Count(availableOperators) == 0)
        {
            NumDeclinedRequests++;
        }
        else
        {

```



```

namespace Моделирование_информационного_центра
{
    class ProcessingSystem
    {
        public bool Active { get; private set; } = false;

        /*Длина очереди*/
        public int QueueLength { get; private set; } = 0;

        private readonly ServiceUnit computer;

        public ProcessingSystem(int minProcTime, int maxProcTime)
        {
            computer = new ServiceUnit(minProcTime, maxProcTime);
        }

        /*Поставить заявку в очередь*/
        public void EnqueueRequest()
        {
            Active = true;

            /*Пока компьютер занят, заявка остается в накопителе*/
            if (computer.Active)
            {
                QueueLength++;
            }
            /*В противном случае компьютер начинает обработку заявки*/
            else
            {
                computer.StartServing();
            }
        }

        public bool ContinueServing(double dt)
        {
            if (computer.Active)
            {
                computer.ContinueServing(dt);

                if (!computer.Active)
                {
                    return true;
                }
            }
            else if (QueueLength > 0)
            {
                computer.StartServing();
            }
            else
            {
                Active = false;
            }

            return false;
        }

        public void StopServing()
        {
            Active = false;

            computer.StopServing();
        }
    }
}

```

```

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

namespace Моделирование_информационного_центра
{
    class ServiceUnit
    {
        /*Статус оператора в данный момент*/
        public bool Active { get; private set; } = false;

        public double ? TimeLeft { get; private set; } = null;

        private readonly int minServiceTime, maxServiceTime;

        public int MinServiceTime { get { return minServiceTime; } }

        public int MaxServiceTime { get { return maxServiceTime; } }

        private readonly ITimeRandomizer timeRandomizer;

        public ServiceUnit (int minServTime, int maxServTime, ITimeRandomizer timeRand =
null)
        {
            minServiceTime = minServTime;
            maxServiceTime = maxServTime;
            timeRandomizer = timeRand ?? new TimeRandomizer();
        }

        /*В момент начала обработки заявки оператором, его статус Active становится true,
        * задается время обработки заявки*/
        public virtual void StartServing()
        {
            Active = true;

            TimeLeft = timeRandomizer.NextValue(minServiceTime, maxServiceTime);
        }

        /*Оператор выполняет обработку заявки, пока не закончилось время*/
        public virtual void ContinueServing(double dt)
        {
            if (Active)
            {
                TimeLeft -= dt;

                Active = TimeLeft > 0;
            }
        }

        /*Когда оператор закончил обработку заявки, его статус Active становится false*/
        public virtual void StopServing()
        {
            Active = false;

            TimeLeft = null;
        }
    }
}

```

```

/*TimeRandomizer.cs*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Моделирование_информационного_центра
{
    interface ITimeRandomizer
    {
        double NextValue(int minValue, int maxValue);
    }

    class TimeRandomizer:ITimeRandomizer
    {
        private readonly Random rnd = new Random();

        public double NextValue(int minValue, int maxValue)
        {
            return rnd.Next(minValue, maxValue) + rnd.NextDouble();
        }
    }
}

```