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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

О Т Ч Е Т

по лабораторной работе № 5

Название: Моделирование работы информационного центра

Дисциплина: Моделирование

Студент ИУ7-72Б _____ Е.В. Брянская
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ И.В. Рудаков
(Подпись, дата) (И.О. Фамилия)

Москва, 2021

1 Задание

В информационный центр приходят клиенты через интервал времени 10 ± 2 минуты. Если все три имеющихся оператора заняты, клиенту отказывают в обслуживании.

Операторы имеют разную производительность и могут обеспечивать обслуживание среднего запроса пользователя за 20 ± 5; 40 ± 10; 40 ± 20.

Клиенты стремятся занять свободного оператора с максимальной производительностью.

Полученные запросы сдаются в накопитель. Откуда выбираются на обработку. На первый компьютер запросы от 1 и 2-ого операторов, на второй – запросы от 3-его. Время обработки запросов первым и 2-м компьютером равны соответственно 15 и 30 мин. Промоделировать процесс обработки 300 запросов.

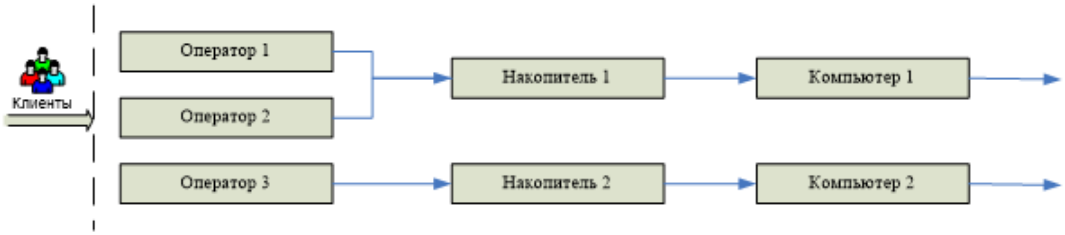

Рисунок 1.1 — Общая схема

2 Теоретическая часть

В процессе взаимодействия клиентов с информационным центром возможно:

1. режим нормального обслуживания, т.е. клиент выбирает одного из свободных операторов, отдавая предпочтение тому у которого меньше номер;

2. режим отказа в обслуживании клиента, когда все операторы заняты.

Переменные и уравнения имитационной модели

Эндогенные переменные: время обработки задания i-ым оператором, время решения этого задания j-ым компьютером.

Экзогенные переменные: число обслуженных клиентов и число клиентов, получивших отказ.

Структурная схема представлена на рисунке 2.2, на нем К1-К3 моделируют работу операторов, К4-К5 – компьютеров.

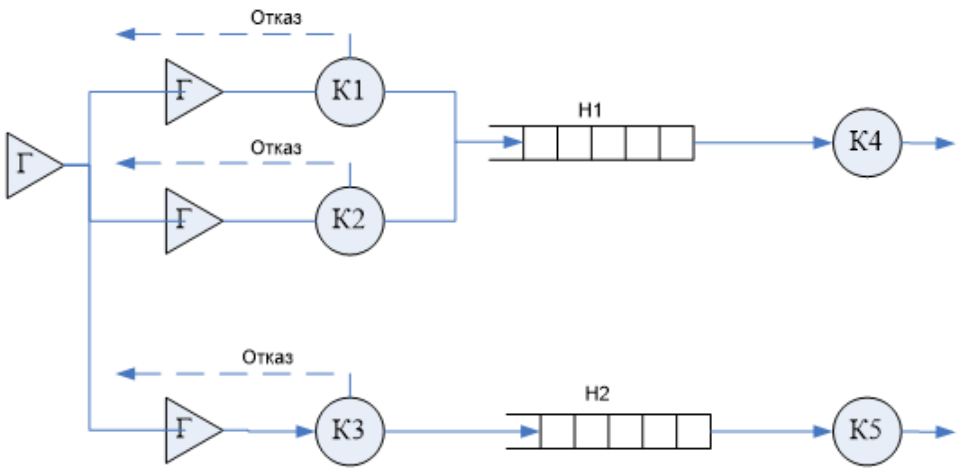

Рисунок 2.2 — Структурная схема

3 Результаты работы программы

Замеры проводились 10 раз, каждый раз моделировался процесс обработки 300 запросов. Проводился подсчёт обслуженных и отказанных заявок, а также времени моделирования, и по результатам вычислялась вероятность, с которой клиенту могут отказать в обработке его заявки. Результаты приведены на рисунке 3.3.

Обслужено	Отказано	Время моделирования	Вероятность отказа
237	63	3036,80544	0,21
238	62	3041,4106	0,207
234	66	3039,56207	0,22
237	63	3058,28853	0,21
236	64	3033,49214	0,213
233	67	3059,58464	0,223
229	71	3043,81769	0,237
232	68	3042,63349	0,227
236	64	3084,91541	0,213
232	68	3048,2092	0,227

Рисунок 3.3 — Результаты замеров

4 Код программы

На Листинге 1 представлены основные методы.

Листинг 1 — Основные методы

```
namespace lab05
{
  class ModelInfo
  {
    public Client clients;
    public Operator[] oprArr;
    public Computer[] comArr;
    public Queue<double> q1;
    public Queue<double> q2;
  }

  class Model : ModelInfo
  {
    private static int _goal;
    public int processed;
    public int refused;
    public double simTime;
    public double pRefuse;
    List<Event> eventArr;

    public Model(int goal)
    {
      processed = 0;
      refused = 0;
      q1 = new Queue<double>();
      q2 = new Queue<double>();
      eventArr = new List<Event>();

      clients = new Client(8, 12);
      comArr = new Computer[2]
      {
        new Computer(15, ref q1),
        new Computer(30, ref q2)
      };
      oprArr = new Operator[3]
      {
        new Operator(15, 25, ref q1),
        new Operator(30, 50, ref q1),
        new Operator(20, 60, ref q2)
      };

      _goal = goal;
    }

    public void Imitation()
```

```
46      {
47        Event curEvent;
48        eventArr.Add(new Event(EventType.IsClient, clients.Next()));
49
50        while (eventArr.Count > 0)
51        {
52          if (refused + processed >= _goal)
53            break;
54
55          curEvent = eventArr[0];
56          eventArr.RemoveAt(0);
57
58          if (curEvent.etype is EventType.IsClient)   ProcessClient(curEvent);
59          else if (curEvent.etype is EventType.IsOperator)    ProcessOperator(curEvent);
60          else    ProcessComputer(curEvent);
61
62          eventArr.Sort((Event x, Event y) =>
63            x.etime > y.etime
64            ? 1 : -1);
65        }
66
67        simTime = eventArr[0].etime;
68        pRefuse = CountPRefuse();
69      }
70
71      private void ProcessClient(Event e)
72      {
73        int refuse = 1;
74        for (int i = 0; i < oprArr.Length; i++)
75          if (oprArr[i].IsFree())
76          {
77            eventArr.Add(new Event(EventType.IsOperator, oprArr[i].Next(e.etime), i));
78            oprArr[i].SetBusy();
79            refuse = 0;
80            break;
81          }
82        refused += refuse;
83        eventArr.Add(new Event(EventType.IsClient, clients.Next(e.etime)));
84      }
85
86      private void ProcessOperator(Event e)
87      {
88        int temp;
89        oprArr[e.ind].AddToQueue(e.etime);
90        temp = (e.ind == 2) ? 1 : 0;
91
92        if (comArr[temp].IsFree())
93        {
94          eventArr.Add(new Event(EventType.IsComputer, e.etime, temp));
95          processed -= 1;
96        }
```

```
97
98          oprArr[e.ind].SetFree();
99          oprArr[e.ind].next = 0;
100       }
101
102       private void ProcessComputer(Event e)
103       {
104         double tempValue;
105         tempValue = comArr[e.ind].GetFromQueue();
106         if (tempValue > 0)
107         {
108           comArr[e.ind].SetBusy();
109           eventArr.Add(new Event(EventType.IsComputer, comArr[e.ind].Next(e.etime), e.ind));
110         }
111         else
112           comArr[e.ind].SetFree();
113         processed += 1;
114       }
115
116       private double CountPRefuse() { return (double)refused / (refused + processed); }
117     }
118
119     class Obj : Generator
120     {
121       public bool _isFree;
122       public double next;
123
124       public Obj(double a, double b) : base(a, b)
125       {
126         SetFree();
127         next = 0;
128       }
129
130       public double Next(double cur_time = 0)
131       {
132         next = cur_time + ProcessTime();
133         return next;
134       }
135
136       public bool IsFree() { return _isFree; }
137       public void SetFree() { _isFree = true; }
138       public void SetBusy() { _isFree = false; }
139       public virtual void AddToQueue(double elem) { throw new Exception(); }
140       public virtual double GetFromQueue() { throw new Exception(); }
141     }
142
143     class Client : Obj  { public Client(double a, double b) : base(a, b) {} }
144
145     class Computer : Obj
146     {
147       private Queue<double> _q;
```

```
148     public Computer(double a, ref Queue<double> q) : base(a, a) { _q = q; }
149     public override double GetFromQueue()
150     {
151       if (_q.Count != 0)
152       return _q.Dequeue();
153       return -1;
154     }
155   }
156
157   class Operator : Obj
158   {
159     private Queue<double> _q;
160     public Operator(double a, double b, ref Queue<double> q) : base(a, b) { _q = q; }
161     public override void AddToQueue(double elem) { _q.Enqueue(elem); }
162   }
163
164   public enum EventType
165   {
166     IsClient,
167     IsOperator,
168     IsComputer
169   }
170
171   class Event
172   {
173     public EventType etype;
174     public double etime;
175     public int ind;
176     public Event(EventType type, double time, int index = 0)
177     {
178       etype = type;
179       etime = time;
180       ind = index;
181     }
182   }
183
184   class Generator
185   {
186     private double _a;
187     private double _b;
188     private Random rnd;
189     public Generator(double a, double b)
190     {
191       _a = a;
192       _b = b;
193       rnd = new Random();
194     }
195
196     public double ProcessTime() { return _a + (_b - _a) * rnd.NextDouble(); }
197   }
198 }
```