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 № 6

Название: Моделирование работы центра вакцинации

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

1.1 Задание

Центр вакцинации

В центре вакцинации от коронавируса представлен только Спутник V (2 этапа).

Люди приходят каждые 5 ± 2 минут за 1ым компонентном и 6 ± 2 за 2ым.

Те, кто только собираются сделать 1ый компонент, должны подтвердить свою запись на стойке регистрации, где на человека уходит 5.5 ± 3 минут, оператор только один, поэтому к нему выстраивается очередь, максимальный размер которой по новым нормам – 7 человек, остальные получают отказ в обслуживании.

После подтверждения регистрации клиент встает в общую очередь к двум медсёстрам, которые делают только первый компонент. Первая обслуживает за 8 ± 2 минут, вторая – 12 ± 4 минут.

Те, кто уже сделал 1ый компонент ранее и пришёл за 2ым, сразу встают в бесконечную очередь к медсестре, которая его делает. На одного человека у неё уходит примерно 5 ± 2 минут.

После укола все пациенты встают в единую очередь за справкой и печатью, которую обслуживают два врача, один из которых тратит 7 ± 2 минут на человека, другой 5±1.

Промоделировать процесс для 1000 клиентов.

2 Теоретическая часть

2.1 Описание модели

Структурная схема представлена на рисунке 2.1.

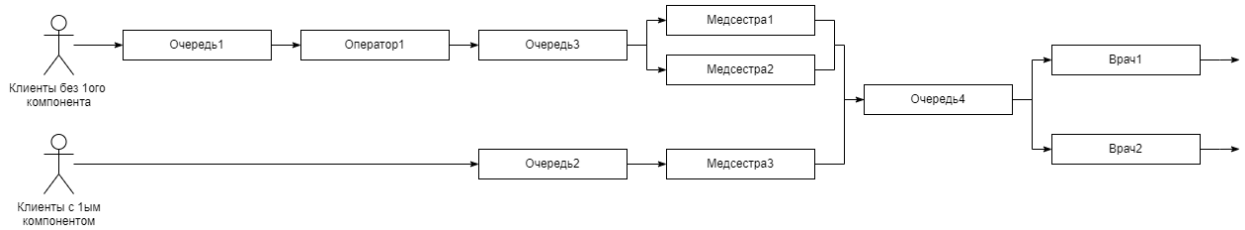

Рисунок 2.1 — Общая схема

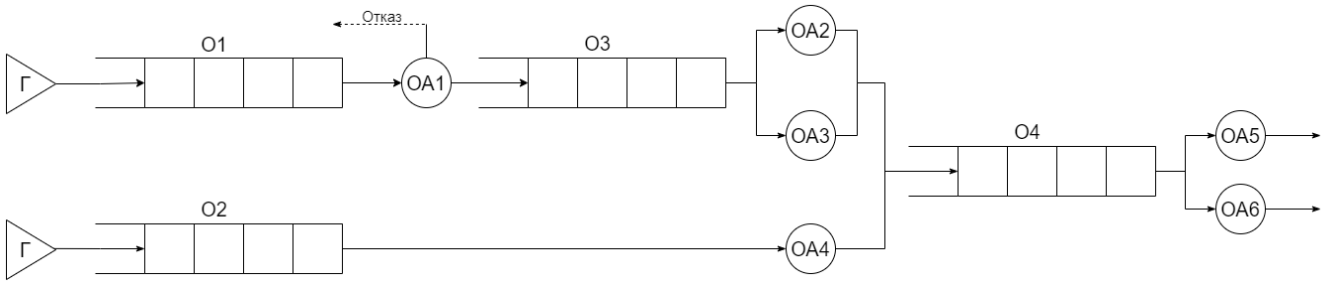

Рисунок 2.2 — Структурная схема

Время обработки клиентов подчиняется закону равномерного распределения. В состав модели входят:

- ОА1 – оператор;
- ОА2, ОА3, ОА4 – медсёстры, первые две отвечают за первый компонент, третья – за второй.
- ОА5, ОА6 – врачи.

Переменные и уравнения имитационной модели

Эндогенные переменные: время обработки клиентов оператором, медсёстрами и врачами.

Экзогенные переменные: число обслуженных клиентов и число клиентов, получивших отказ, максимальные длины очередей, время моделирования.

3 Результаты работы программы

Замеры проводились 10 раз, каждый раз моделировался процесс обработки 1000 клиентов. Проводился подсчёт обслуженный и отказанных заявок, времени моделирования и максимального размера очередей. Результаты приведены на рисунке 3.3.

Обслужено	Отказано	Время мод-ния	Вероятность отказа	Макс размер очереди 1	2	3	4
940	60	2768,014	0,06	7	3	3	7
971	29	2845,01215	0,029	7	2	4	33
946	54	2783,78278	0,054	7	2	3	17
957	43	2813,94855	0,043	7	2	3	20
949	51	2812,22349	0,051	7	2	2	11
946	54	2796,08242	0,054	7	1	4	23
958	42	2799,751	0,042	7	2	5	23
948	52	2778,62566	0,052	7	1	3	12
945	55	2764,64587	0,055	7	2	2	17
954	46	2794,94875	0,046	7	1	3	29

Рисунок 3.3 — Результаты моделирования

4 Код программы

На Листинге 1 представлены основные методы.

Листинг 1 — Основные методы

```
namespace lab06
{
  public enum EventType
  {
    IsClient1,
    IsClient2,
    IsOperator,
    IsNurse,
    IsDoctor
  }

  class Event
  {
    public EventType etype;
    public double etime;
    public int ind;

    public Event(EventType type, double time, int index = 0)
    {
      etype = type;
      etime = time;
      ind = index;
    }
  }

  class Generator
  {
    private double _a;
    private double _b;
    private static Random rnd = new Random();

    public Generator(double m, double d)
    {
      _a = m - d;
      _b = m + d;
    }

    public double ProcessTime(){ return _a + (_b - _a) * rnd.NextDouble(); }
  }

  public enum EventType
  {
    IsClient1,
    IsClient2,
    IsOperator,
```

```
46      IsNurse,
47      IsDoctor
48    }
49
50    class Event
51    {
52      public EventType etype;
53      public double etime;
54      public int ind;
55
56      public Event(EventType type, double time, int index = 0)
57      {
58        etype = type;
59        etime = time;
60        ind = index;
61      }
62    }
63
64    class Obj : Generator
65    {
66      public bool _isFree;
67      public double next;
68
69      public Obj(double a, double b) : base(a, b)
70      {
71        SetFree();
72        next = 0;
73      }
74
75      public double Next(double cur_time = 0)
76      {
77        next = cur_time + ProcessTime();
78        return next;
79      }
80
81      public bool IsFree() { return _isFree; }
82      public void SetFree() { _isFree = true; }
83      public void SetBusy() { _isFree = false; }
84      public virtual void AddToQueue(double elem) { throw new Exception(); }
85      public virtual double GetFromQueue() { throw new Exception(); }
86    }
87
88    class Operator : Obj
89    {
90      private Queue<double> _inQ;
91      private Queue<double> _outQ;
92
93      public Operator(double a, double b, ref Queue<double> inQ, ref Queue<double> outQ) :
        base(a, b)
94      {
95        _inQ = inQ;
```

```
96          _outQ = outQ;
97        }
98
99        public override void AddToQueue(double elem) { _outQ.Enqueue(elem); }
100
101       public override double GetFromQueue()
102       {
103         if (_inQ.Count != 0)
104         return _inQ.Dequeue();
105         return -1;
106       }
107     }
108
109     class Client : Obj
110     {
111       private Queue<double> _q;
112
113       public Client(double a, double b, ref Queue<double> q) : base(a, b) { _q = q; }
114
115       public override void AddToQueue(double elem) { _q.Enqueue(elem); }
116     }
117
118     class Nurse : Operator
119     {
120       public Nurse(double a, double b, ref Queue<double> inQ, ref Queue<double> outQ) : base(a
          , b, ref inQ, ref outQ) {}
121     }
122
123     class Doctor : Obj
124     {
125       private Queue<double> _inQ;
126
127       public Doctor(double a, double b, ref Queue<double> inQ) : base(a, b) { _inQ = inQ; }
128
129       public override double GetFromQueue()
130       {
131         if (_inQ.Count != 0)
132         return _inQ.Dequeue();
133         return -1;
134       }
135     }
136
137     class Model
138     {
139       public Queue<double>[] qArr;
140       public Client[] clientArr;
141       public Nurse[] nrsArr;
142       public Doctor[] dctArr;
143       public int[] maxSizeQ;
144
145       public Operator opr;
```

```
146
147     List<Event> eventArr;
148
149     public int processed;
150     public int refused;
151     public double simTime;
152     public double pRefuse;
153
154     private static int _limit;
155     private static int _maxQueue;
156
157     public Model(int limit)
158     {
159       processed = 0;
160       refused = 0;
161       _limit = limit;
162       _maxQueue = 7;
163
164       eventArr = new List<Event>();
165
166       qArr = new Queue<double>[]
167       {
168         new Queue<double>(),
169         new Queue<double>(),
170         new Queue<double>(),
171         new Queue<double>()
172       };
173
174       clientArr = new Client[2]
175       {
176         new Client(5, 2, ref qArr[0]),
177         new Client(6, 2, ref qArr[1])
178       };
179
180       opr = new Operator(5.5, 3, ref qArr[0], ref qArr[2]);
181
182       nrsArr = new Nurse[3]
183       {
184         new Nurse(8, 2, ref qArr[2], ref qArr[3]),
185         new Nurse(12, 4, ref qArr[2], ref qArr[3]),
186         new Nurse(5, 2, ref qArr[1], ref qArr[3])
187       };
188
189       dctArr = new Doctor[2]
190       {
191         new Doctor(7, 2, ref qArr[3]),
192         new Doctor(5, 1, ref qArr[3])
193       };
194
195       maxSizeQ = new int[4] { 0, 0, 0, 0 };
196     }
```

```
197
198     public void Imitation()
199     {
200       Event curEvent;
201
202       eventArr.Add(new Event(EventType.IsClient1, clientArr[0].Next(), 0));
203       eventArr.Add(new Event(EventType.IsClient2, clientArr[1].Next(), 1));
204
205       while (eventArr.Count > 0)
206       {
207         eventArr.Sort((Event x, Event y) =>
208           x.etime > y.etime
209           ? 1 : -1);
210
211         if (processed + refused >= _limit)
212           break;
213
214         curEvent = eventArr[0];
215         eventArr.RemoveAt(0);
216
217         switch (curEvent.etype)
218         {
219           case EventType.IsClient1:
220           case EventType.IsClient2:
221           ProcessClient(curEvent);
222           break;
223
224           case EventType.IsOperator:
225           ProcessOperator(curEvent);
226           break;
227
228           case EventType.IsNurse:
229           ProcessNurse(curEvent);
230           break;
231
232           case EventType.IsDoctor:
233           ProcessDoctor(curEvent);
234           break;
235         }
236
237         CountMaxQ();
238       }
239
240       simTime = eventArr[0].etime;
241       pRefuse = CountPRefuse();
242     }
243
244     private void ProcessClient(Event e)
245     {
246       if (e.etype is EventType.IsClient1)
247       {
```

```
248         if (qArr[0].Count >= _maxQueue)
249         {
250           refused += 1;
251           eventArr.Add(new Event(e.etype, clientArr[e.ind].Next(e.etime), e.ind));
252           return;
253         }
254       }
255
256       if (e.etype is EventType.IsClient1)
257       {
258         if (opr.IsFree())
259           eventArr.Add(new Event(EventType.IsOperator, e.etime));
260
261         qArr[0].Enqueue(e.etime);
262       }
263       else
264       {
265         if (nrsArr[2].IsFree())
266           eventArr.Add(new Event(EventType.IsNurse, e.etime, 2));
267
268         qArr[1].Enqueue(e.etime);
269       }
270
271       eventArr.Add(new Event(e.etype, clientArr[e.ind].Next(e.etime), e.ind));
272     }
273
274     private void ProcessOperator(Event e)
275     {
276       double tempValue;
277
278       if (!opr.IsFree())
279       {
280         opr.AddToQueue(e.etime);
281
282         if (nrsArr[0].IsFree())
283           eventArr.Add(new Event(EventType.IsNurse, e.etime, 0));
284         else if (nrsArr[1].IsFree())
285           eventArr.Add(new Event(EventType.IsNurse, e.etime, 1));
286       }
287
288       tempValue = opr.GetFromQueue();
289       if (tempValue > 0)
290       {
291         opr.SetBusy();
292         eventArr.Add(new Event(EventType.IsOperator, opr.Next(e.etime)));
293       }
294       else
295         opr.SetFree();
296     }
297
298     private void ProcessNurse(Event e)
```

```
299     {
300       double tempValue;
301 
302       if (!nrsArr[e.ind].IsFree())
303       {
304         nrsArr[e.ind].AddToQueue(e.etime);
305 
306         if (dctArr[0].IsFree())
307           eventArr.Add(new Event(EventType.IsDoctor, e.etime, 0));
308         else if (dctArr[1].IsFree())
309           eventArr.Add(new Event(EventType.IsDoctor, e.etime, 1));
310       }
311 
312       tempValue = nrsArr[e.ind].GetFromQueue();
313       if (tempValue > 0)
314       {
315         nrsArr[e.ind].SetBusy();
316         eventArr.Add(new Event(EventType.IsNurse, nrsArr[e.ind].Next(e.etime), e.ind));
317       }
318       else
319         nrsArr[e.ind].SetFree();
320     }
321 
322     private void ProcessDoctor(Event e)
323     {
324       double tempValue;
325 
326       if (!dctArr[e.ind].IsFree())
327       {
328         processed += 1;
329         dctArr[e.ind].SetFree();
330       }
331 
332       tempValue = dctArr[e.ind].GetFromQueue();
333       if (tempValue > 0)
334       {
335         dctArr[e.ind].SetBusy();
336         eventArr.Add(new Event(EventType.IsDoctor, dctArr[e.ind].Next(e.etime), e.ind));
337       }
338     }
339 
340     private double CountPRefuse(){ return (double)refused / (refused + processed); }
341 
342     private void CountMaxQ()
343     {
344       for (int i = 0; i < qArr.Count(); i++)
345         if (qArr[i].Count() > maxSizeQ[i])
346           maxSizeQ[i] = qArr[i].Count();
347     }
348   }
349 }
```