

Тема:

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 09.03.01 Информатика и вычислительная техника

ОТЧЕТ

Моделирование аппарата обслуживания

по лабораторной работе № 4____

| Дисциплина: | <u>Моделирование</u> | | |
|---------------|----------------------|------------------------------------|--|
| Студент | <u>ИУ7-72Б</u> | | В.А. Иванов |
| Преподаватель | (Группа) | (Подпись, дата) (Подпись, дата) | (И.О. Фамилия) - И.В. Рудаков (И.О. Фамилия) |

Москва, 2021

1. Задание

Промоделировать систему, состоящую из генератора, очереди и обслуживающего автомата. Генератор создаёт сообщения по равномерному закону, откуда они поступают в очередь. Из очереди сообщения получает обслуживающий автомат, работающий по закону из первой лабораторной работы (нормальный закон). Определить длину очереди, при которой не произойдёт потери сообщений. Промоделировать двумя методами: пошаговый и событийный.

2. Результаты

2.1. Теория

2.1.1. Пошаговый метод

Система рассматривается в моменты времени, с Δt разницей $-t_i = t_{i-1} + \Delta t$. В момент t_i моделируются все события, которые должны были произойти с момента t_{i-1} , таким образом система приходит в состояние, соответствующее текущему времени. Преимуществом метода является простота его реализации. Очевидный минус необходимость в анализе моментов времени без изменения в системе.

2.1.2. Событийный метод

Система также рассматривается дискретно, в моменты времени t_i , однако отличием от предыдущего метода является их выбор. Анализ системы производится только в моменты происхождения одного или нескольких событий. После выполнения соответствующих преобразований производится выбор t_{i+1} как ближайшего из всех ожидаемых событий.

2.2. Работа программы

Условием остановки поиска является обслуживание 1000 сообщений без изменения максимальной длины очереди.

В случае, если такое событие не происходит за 100 000 заявок, принимается, что генерация вместе с обратной связью помещают сообщения с большей интенсивностью, чем успевает обрабатывать их ОА. Следовательно, со временем длина очереди будет в среднем только расти, поэтому для любой выбранной очереди в определенный момент произойдут потери.

Примеры работы программы приведены на рисунках 2.1, 2.2 и 2.3.

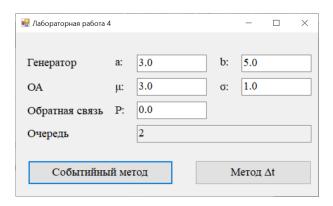


Рис. 2.1 — Обслуживание в среднем быстрее генерации (Событийный метод)

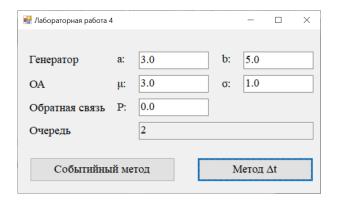


Рис. 2.2 — Обслуживание в среднем быстрее генерации (Пошаговый метод)

В ходе экспериментов было установлено, что оба метода выдают одинаковый результат.

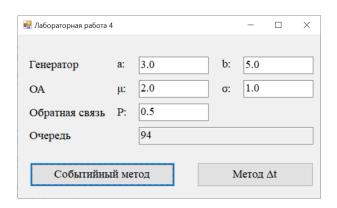


Рис. 2.3 — Обратная связь 50%

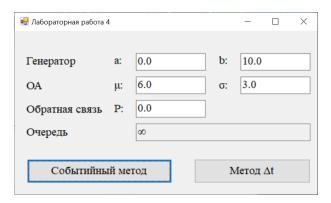


Рис. 2.4 — Генерация интенсивнее обработки (размер очереди неопределён)

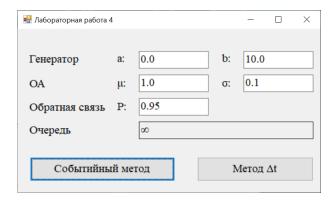


Рис. 2.5 — Генерация и обратная связь интенсивнее обработки (размер очереди неопределён)

3. Текст программы

В листинге 3.1 представлен фрагмент кода программы, отвечающий за моделирование.

Листинг 3.1 — Реализация модели

```
class Request
  {
|2|
3
     public double CreationT;
     public double ServeT;
4
     public Request(double t)
5
6
7
       CreationT = t;
8
       ServeT = -1;
9
     }
10
     public bool IsServed()
11
12
       return ServeT >= 0;
13
14
15|}
16
  class Generator
17
18
     public double A { get; }
19
     public double B { get; }
20
21
     public Request r;
     private Random rnd = new Random();
22
     public Generator (double a, double b, double current T = 0.0)
23
24
     {
      A = a; B = b;
25
       New(currentT);
26
27
     }
28
29
     public double Aver()
30
     {
       return (A + B) / 2;
31
```

```
32
    }
33
     public double WhenReady()
34
35
       return r.CreationT;
36
37
    public bool IsReady(double curT)
38
39
       return r = null \&\& curT >= r.CreationT - 1e-5;
40
     }
41
42
    public Request New(double curT)
43
44
       Request oldR = r;
45
       r = new Request(curT + A + (B - A) * rnd.NextDouble());
46
47
       return oldR;
48
49 }
50
51 class Service
52 {
53
     public double Mu { get; }
     public double Sig { get; }
54
     public double FeedBackP { get; }
55
56
     public Request r;
     private Random rnd;
57
     public Service (double mu, double sig, double fbP = 0.0)
58
59
60
      Mu = mu; Sig = sig;
       FeedBackP = fbP;
61
62
       rnd = new Random();
63
       r = null;
    }
64
65
```

```
public double WhenReady()
66
67
       return r ServeT;
68
69
     public bool IsEmpty()
70
71
72
       return r == null;
73
     public bool IsReady(double curT)
74
75
       return r = null \&\& curT >= r.ServeT - 1e-5;
76
     }
77
78
     public void Put(Request newR, double curT)
79
     {
80
81
       r = newR:
       double roll = Delay();
82
       r.ServeT = curT + roll;
83
84
     }
85
     public Request Get(double curT)
86
87
       Request oldR = r;
88
       r = null;
89
90
       if (rnd.NextDouble() < FeedBackP)</pre>
91
         oldR = new Request(curT);
92
93
       return oldR;
    }
94
95
     private double Delay()
96
97
98
       const int n = 12;
99
       double acc = 0;
```

```
100
        for (int i = 0; i < n; i++)
101
        acc += rnd.NextDouble();
102
103
104
        acc = n / 2.0;
        acc *= Sig * Math.Sqrt (12.0 / n);
105
        acc += Mu;
106
107
        if (acc <= 0)
108
109
          acc = Delay();
110
111
        return acc;
112
     }
113 }
114
115 class ReqQueue: Queue<Request>
116 | {
117
      public int PeakLen = 0;
      public double PeakTime = -1;
118
      public ReqQueue() {}
119
120
      public void Push(Request r, double curT)
121
122
        Enqueue(r);
123
        if (Count > PeakLen)
124
          PeakLen = Count;
125
          PeakTime = curT;
126
        }
127
128
     }
129
      public Request Pop()
130
        return Dequeue();
131
132
     }
133 }
```

```
134
135 class DTModel
136 {
137
      public double CurT;
      private Generator Gen;
138
      private ReqQueue Que;
139
      private Service Serv;
140
141
      public int created N = 0;
142
143
      public int servedN = 0;
      public int returned N = 0;
144
      public bool IsOverflowed = false;
145
146
      public DTModel(Generator g, Service s)
147
148
      {
        CurT = 0:
149
        Gen = g;
150
        Que = new ReqQueue();
151
152
        Serv = s;
     }
153
154
155
      public int Run(double dT=0.01, double endT= 1e5)
156
      {
157
        while (CurT - Que.PeakTime < 1000.0 * Gen.Aver() && CurT
      < endT)
158
          HandleGenerator();
159
          HandleService();
160
          CurT += dT;
161
162
        Console. WriteLine ("\{0\} \{1\}", CurT, Que. PeakTime);
163
        if (CurT > 1e5)
164
165
        {
          IsOverflowed = true;
166
```

```
167
          return -1;
168
        } else
          return Que.PeakLen;
169
170
      }
171
      private void HandleGenerator()
172
173
        if (Gen.IsReady(CurT))
174
175
          Request r = Gen.New(CurT);
176
          Que. Push (r, CurT);
177
178
179
          createdN++;
180
        }
     }
181
182
      private void HandleService()
183
184
        if (Serv.lsReady(CurT))
185
186
        {
          Request r = Serv.Get(CurT);
187
          if (!r.lsServed())
188
          {
189
            Que.Push(r, CurT);
190
             returned N ++;
191
          } else
192
             servedN++;
193
        }
194
195
        if (Serv.lsEmpty() && Que.Count > 0)
196
          Serv.Put(Que.Pop(), CurT);
197
198
      }
199 }
200
```

```
201 class EventModel
202 {
      public double CurT;
203
204
      private Generator Gen;
      private ReqQueue Que;
205
      private Service Serv;
206
207
      public int created N = 0;
208
      public int servedN = 0;
209
210
      public int returned N = 0;
      public bool IsOverflowed = false;
211
212
213
      private List < double > Events;
214
      public EventModel(Generator g, Service s)
215
216
      {
        CurT = 0;
217
218
        Gen = g;
        Que = new ReqQueue();
219
        Serv = s;
220
221
222
        Events = new List < double > \{ g. WhenReady() \};
223
     }
224
225
      public int Run(double endT = 1e5)
226
        while (CurT - Que.PeakTime < 1000.0 * Gen.Aver() && CurT
227
      < endT)
228
        {
229
          HandleGenerator();
          HandleService();
230
231
232
          CurT = Events.Min();
233
          Events . Remove (CurT);
```

```
234
        }
235
236
         if (CurT > 1e5)
237
           IsOverflowed = true;
238
239
           return -1;
240
        }
         else
241
242
           return Que PeakLen;
      }
243
244
      private void HandleGenerator()
245
246
        if (Gen.IsReady(CurT))
247
        {
248
           Request r = Gen.New(CurT);
249
           Que. Push (r, CurT);
250
251
252
           created N++;
           {\sf Events.Add} \, (\, {\sf Gen.WhenReady} \, (\, ) \, ) \, ;
253
254
        }
      }
255
256
257
      private void HandleService()
258
         if (Serv.lsReady(CurT))
259
260
           Request r = Serv.Get(CurT);
261
           if (!r.lsServed())
262
263
             Que.Push(r, CurT);
264
265
              returned N ++;
           }
266
           else
267
```

```
servedN++;
268
        }
269
270
        if (Serv.IsEmpty() && Que.Count > 0)
271
        {
272
          Serv.Put(Que.Pop(), CurT);
273
           Events . Add ( Serv . When Ready ( ) );
274
275
        }
276
      }
277 }
```