



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

О Т Ч Е Т

по лабораторной работе № 4

Тема: Моделирование аппарата обслуживания

Дисциплина: Моделирование

Студент

ИУ7-72Б

(Группа)

(Подпись, дата)

В.А. Иванов

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

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.

The screenshot shows a window titled 'Лабораторная работа 4'. It contains several input fields and two buttons. The 'Событийный метод' button is highlighted with a blue border.

Генератор	a:	b:
	3.0	5.0

ОА	μ :	σ :
	3.0	1.0

Обратная связь	R:
	0.0

Очередь
2

Buttons: Событийный метод, Метод Δt

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

The screenshot shows the same window as Figure 2.1, but the 'Метод Δt ' button is now highlighted with a blue border.

Генератор	a:	b:
	3.0	5.0

ОА	μ :	σ :
	3.0	1.0

Обратная связь	R:
	0.0

Очередь
2

Buttons: Событийный метод, Метод Δt

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

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

Лабораторная работа 4

Генератор a: 3.0 b: 5.0

ОА μ : 2.0 σ : 1.0

Обратная связь P: 0.5

Очередь 94

Событийный метод Метод Δt

Рис. 2.3 — Обратная связь 50%

Лабораторная работа 4

Генератор a: 0.0 b: 10.0

ОА μ : 6.0 σ : 3.0

Обратная связь P: 0.0

Очередь ∞

Событийный метод Метод Δt

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

Лабораторная работа 4

Генератор	a:	0.0	b:	10.0
ОА	μ :	1.0	σ :	0.1
Обратная связь	P:	0.95		
Очередь		∞		

Событийный метод Метод Δt

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

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

```

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

```



```

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

```

```

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

```

```

134
135 class DTModel
136 {
137     public double CurT;
138     private Generator Gen;
139     private ReqQueue Que;
140     private Service Serv;
141
142     public int createdN = 0;
143     public int servedN = 0;
144     public int returnedN = 0;
145     public bool IsOverflowed = false;
146
147     public DTModel(Generator g, Service s)
148     {
149         CurT = 0;
150         Gen = g;
151         Que = new ReqQueue();
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
158         < endT)
159         {
160             HandleGenerator();
161             HandleService();
162             CurT += dT;
163         }
164         Console.WriteLine("{0} {1}", CurT, Que.PeakTime);
165         if (CurT > 1e5)
166         {
167             IsOverflowed = true;

```

```

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

```

```

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

```

```

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

```

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