



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 № 4

Название: Обслуживающий аппарат

Дисциплина: Моделирование

Студент

ИУ7-72Б

(Группа)

(Подпись, дата)

Е.В. Брянская

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

И.В. Рудаков

(И.О. Фамилия)

Москва, 2021

1 Задание

Необходимо промоделировать работу прибора обслуживания, определить длину очереди, при которой не будет потерянных сообщений и вывести результат на экран.

Закон генерации сообщений равномерный, обслуживание происходит согласно закону в соответствии с вариантом: **нормальный**.

Следует предусмотреть возможность построения обратной связи, указывая в процентах долю заявок, которые возвращаются назад в очередь.

Реализовать двумя способами: событийный и пошаговый.

2 Теоретическая часть

2.1 Равномерное распределение

Плотность выражается формулой:

$$f_X(x) = \begin{cases} \frac{1}{b-a}, x \in [a, b], \\ 0, x \notin [a, b]. \end{cases} \quad (1)$$

Функция распределения имеет вид:

$$F_X(x) = \begin{cases} 0, x < a, \\ \frac{x-a}{b-a}, a \leq x < b, \\ 1, x \geq b. \end{cases} \quad (2)$$

2.2 Нормальное распределение

Плотность выражается формулой:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (3)$$

Функция распределения имеет вид:

$$F_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \quad (4)$$

Стандартным нормальным распределением называется нормальное распределение с математическим ожиданием $\mu = 0$ и стандартным отклонением $\sigma = 1$.

2.3 Пошаговый принцип (Δt)

Этот принцип заключается в последовательном анализе состояний всех блоков системы в момент $t + \Delta t$. При этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием.

Недостаток: значительные временные затраты на реализацию моделирования системы. А также при недостаточно малом Δt отдельные события в системе могут быть пропущены, что может повлиять на адекватность результатов.

2.4 Событийный принцип

Состояние отдельных устройств изменяются в дискретные моменты времени, совпадающие с моментами времени поступления сообщений в систему, временем окончания обработки задачи и т.д.

При использовании событийного принципа состояние всех блоков системы анализируется лишь в момент проявления какого-либо события. Моменты наступления следующего события определяются минимальным значением из списка событий.

3 Результаты работы программы

Интерфейс предполагает ввод всех необходимых параметров модели. Моделирование происходит до тех пор, пока не будет обработано 10 000 сообщений. В случае, если длина очереди превышает 10% от этого числа, считается, что дальше она будет только расти, поэтому принимается, что определить искомую длину очереди нельзя.

Также предоставляется возможность построения обратной связи, указывается необходимый процент.

На рисунках 3.1 – 3.5 демонстрируются результаты работы.

Form1

Лабораторная работа № 4

Генератор

a = 0

b = 4

a < b

Обслуживающий автомат

μ = 2

σ = 1

σ > 0

Обратная связь = 0 %

Готово

Максимальная длина очереди при пошаговом подходе: 412

Максимальная длина очереди при событийном подходе: 411

* — означает, что возникает переполнение

Рисунок 3.1 — Пример 1. Генератор и обслуживающий автомат работают с примерно одинаковой производительностью

Form1

Лабораторная работа № 4

Генератор
 $a = 0$
 $b = 4$
 $a < b$

Обслуживающий автомат
 $\mu = 4$
 $\sigma = 1$ $\sigma > 0$
 Обратная связь = 0 %

Готово

Максимальная длина очереди при пошаговом подходе: $-$

Максимальная длина очереди при событийном подходе: $-$

* — означает, что возникает переполнение

Рисунок 3.2 — Пример 2. Генератор создает сообщения интенсивнее, чем их обрабатывает автомат (при таких параметрах возникает переполнение)

Form1

Лабораторная работа № 4

Генератор
 $a = 0$
 $b = 6$
 $a < b$

Обслуживающий автомат
 $\mu = 2$
 $\sigma = 1$ $\sigma > 0$
 Обратная связь = 0 %

Готово

Максимальная длина очереди при пошаговом подходе: 7

Максимальная длина очереди при событийном подходе: 7

* — означает, что возникает переполнение

Рисунок 3.3 — Пример 3. Автомат обрабатывает сообщения интенсивнее, чем их создаёт генератор

Form1

Лабораторная работа № 4

Генератор
 $a = 0$
 $b = 6$
 $a < b$

Обслуживающий автомат
 $\mu = 2$
 $\sigma = 1$ $\sigma > 0$
 Обратная связь = 20 %

Готово

Максимальная длина очереди при пошаговом подходе: 14

Максимальная длина очереди при событийном подходе: 12

* — означает, что возникает переполнение

Рисунок 3.4 — Пример 4. Параметры такие же, как в примере 3, но есть обратная связь – 20%

Form1

Лабораторная работа № 4

Генератор
 $a = 0$
 $b = 4$
 $a < b$

Обслуживающий автомат
 $\mu = 2$
 $\sigma = 1$ $\sigma > 0$
 Обратная связь = 10 %

Готово

Максимальная длина очереди при пошаговом подходе: —

Максимальная длина очереди при событийном подходе: —

* — означает, что возникает переполнение

Рисунок 3.5 — Пример 5. Генератор и обслуживающий автомат работают с примерно одинаковой производительностью, но есть обратная связь – 10%

4 Код программы

На Листинге 1 представлены основные методы.

Листинг 1 — Основные методы

```
1 namespace ComputingDevice{
2     class Device {
3         private static Random rnd;
4         private static int doneTasks;
5
6         public Device() {
7             rnd = new Random();
8             doneTasks = 10000;
9         }
10
11         public void ProcessByTime(double a, double b, double m, double s, double rvr, ref int
maxQueue, out bool flag) {
12             Generator gnr = new Generator(a, b);
13             ServiceMachine mch = new ServiceMachine(m, s);
14             Queue<double> q = new Queue<double>();
15
16             int numTasks = 0;
17             double curTime = 0, gnrTime = gnr.Get(), srvTime = 0, step = 1;
18             double srvTask;
19
20             do {
21                 while (gnrTime <= curTime) {
22                     q.Enqueue(gnrTime);
23                     gnrTime += gnr.Get();
24                 }
25
26                 if (q.Count > maxQueue) maxQueue = q.Count;
27
28                 while (srvTime <= curTime && q.Count != 0) {
29                     srvTask = q.Dequeue();
30                     if (srvTime < srvTask)
31                         srvTime = srvTask;
32
33                     srvTime += mch.Get();
34                     numTasks += 1;
35
36                     if (isReverse(rvr)){
37                         srvTask = srvTime;
38                         q.Enqueue(srvTask);
39                     }
40                 }
41
42                 curTime += step;
43             } while (numTasks < doneTasks);
44 }
```

```

45     flag = maxQueue > 0.1 * doneTasks ? true : false;
46 }
47
48 private static bool isReverse(double rvr) {
49     double p = rnd.NextDouble();
50     if (p < rvr)
51         return true;
52     return false;
53 }
54
55 public void ProcessByEvents(double a, double b, double m, double s, double rvr, ref int
maxQueue, out bool flag) {
56     Generator gnr = new Generator(a, b);
57     ServiceMachine mch = new ServiceMachine(m, s);
58     Queue<double> q = new Queue<double>();
59
60     int numTasks = 0;
61     double gnrTime = gnr.Get(), curTime = gnrTime, srvTime = 0;
62     double srvTask;
63
64     maxQueue = 0;
65
66     do{
67         if (gnrTime <= curTime) {
68             q.Enqueue(gnrTime);
69             gnrTime += gnr.Get();
70         }
71
72         if (q.Count > maxQueue) maxQueue = q.Count;
73
74         if (srvTime <= curTime) {
75             if (q.Count == 0) {
76                 curTime = gnrTime;
77                 continue;
78             }
79
80             srvTask = q.Dequeue();
81             if (srvTime < srvTask)
82                 srvTime = srvTask;
83
84             srvTime += mch.Get();
85             numTasks += 1;
86
87             if (isReverse(rvr)) {
88                 srvTask = srvTime;
89                 q.Enqueue(srvTask);
90             }
91         }
92
93         curTime = gnrTime < srvTime ? gnrTime : srvTime;
94     } while (numTasks < doneTasks);

```



```

95
96     flag = maxQueue > 0.1 * doneTasks ? true : false;
97 }
98
99 class Generator {
100     private static double _a;
101     private static double _b;
102     private static Random rnd;
103
104     public Generator(double a, double b) {
105         _a = a;
106         _b = b;
107         rnd = new Random();
108     }
109
110     public double Get() {
111         return _a + (_b - _a) * rnd.NextDouble();
112     }
113 }
114
115 class ServiceMachine {
116     private static double _m;
117     private static double _s;
118     private static Random rnd;
119
120     private MathNet.Numerics.Distributions.Normal n;
121
122     public ServiceMachine(double m, double s) {
123         _m = m;
124         _s = s;
125         rnd = new Random();
126     }
127
128     public double Get() {
129         double s = 0, res;
130
131         for (int i = 0; i < 12; i++)
132             s += rnd.NextDouble();
133
134         res = _m + (s - 6) * _s;
135         if (res < 0)
136             res = Get();
137         return res;
138     }
139 }
140 }
141 }

```