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»

Отчет по лабораторной работе №4 по дисциплине "Моделирование"

Тема Моделирование работы системы массового обслуживания

Вариант 15 (3)

Студент Малышев И. А.

Группа ИУ7-71Б

Оценка (баллы) _____

Преподаватель: Рудаков И. В.

Москва — 2022 г.

1 | Задание

Смоделировать работу системы, состоящей из генератора, очереди и обслуживающего аппарата. Генерация заявок происходит по закону равномерного распределения с заданными параметрами a, b. Обработка заявок происходит по закону распределения Пуассона с заданным параметром λ.

Требуется определить длину очереди, при которой не будет потери сообщений.

Также смоделировать работу системы с построенной обратной связью, в качестве параметра используется процент обработанных заявок, вновь поступивших на обработку.

Протяжка модельного времени должна осуществляться по Δt и по событийному принципу. Обозначить, есть ли разница в результатах.

2 | Решение

2.1 Теоретическая часть

2.1.1 Система массового обслуживания

СМО – это система, которая производит обслуживание поступающих в неё требований. Обслуживание требований в СМО осуществляется обслуживающими аппаратами. Классическая СМО содержит в себе от одного до бесконечного числа подобных аппаратов.

2.1.2 Протяжка модельного времени по Δt

Принцип Δt заключается в последовательном анализе состояний всех элементов системы в некоторый момент времени $t + \Delta t$ по заданному состоянию этих элементов в момент времени t. При этом новое состояние элементов определяется в соответствии с их алгоритмическим описанием с учётом действующих случайных факторов, задаваемых распределениями вероятности. В результате такого анализа принимается решение о том, какие общесистемные события должны имитироваться программной моделью на текущий момент времени.

2.1.3 Протяжка модельного времени по событийному принципу

Характерным свойством систем обработки информации является тот факт, что состояние отдельных элементов изменяется в некоторые дискретные моменты времени, совпадающие с моментами времени поступления сообщений в систему и так далее. Моделирование и продвижение времени в системе посему так же удобно проводить, используя событийный принцип, при котором состояние всех элементов имитационной модели анализируется лишь в момент появления какого-либо события. Момент поступления следующего события определяется минимальным значением из списка будущих событий, представляющего собой совокупность моментов ближайшего изменения состояния каждого из элементов системы.

2.2 Листинг

Далее представлен фрагмент программы, выполняющий поставленное задание.

```
internal class Processor
{
  double lambda;

  public Processor(double lambda, double probabilityOfReturnToQueue)
  {
    this.lambda = lambda;
    ProbabilityOfReturnToQueue = probabilityOfReturnToQueue;
  }

  public void GetRequest()
  {
    CurrentNumberOfRequestsInQueue++;

    if (CurrentNumberOfRequestsInQueue > DetectedMaxOfRequestsInQueue)
      DetectedMaxOfRequestsInQueue = CurrentNumberOfRequestsInQueue;
  }

  public void ProcessRequest()
  {
    if (CurrentNumberOfRequestsInQueue == 0)
      return;

    CurrentNumberOfRequestsInQueue--;
    if (ContinuousUniform.Sample(0, 1) < ProbabilityOfReturnToQueue)
    {
      NumberOfReturnedRequests++;
      GetRequest();
    }
  }

  public double GetNextTimeOfRequestProcessed()
  {
    return Poisson.Sample(lambda);
  }

  public int DetectedMaxOfRequestsInQueue { get; set; }
  public int CurrentNumberOfRequestsInQueue { get; set; }
  public int NumberOfReturnedRequests { get; set; }
  public double ProbabilityOfReturnToQueue { get; set; }
}

internal class RequestsGenerator
{
  double a, b;

```

```
47    public RequestsGenerator(double a, double b)
48    {
49      this.a = a;
50      this.b = b;
51    }
52
53    public double GetNextTimeOfRequestGenerated()
54    {
55      return ContinuousUniform.Sample(a, b);
56    }
57  }
58
59  internal class Simulator
60  {
61    Processor processor;
62    RequestsGenerator requestGenerator;
63
64    public Simulator(Processor processor, RequestsGenerator requestGenerator)
65    {
66      this.processor = processor;
67      this.requestGenerator = requestGenerator;
68    }
69
70    public (double, double) SimulateUsingDeltaTMethod(int requestsCount)
71    {
72      double timeOfGeneration =
          requestGenerator.GetNextTimeOfRequestGenerated();
73      double timeOfProcessing = timeOfGeneration +
          processor.GetNextTimeOfRequestProcessed();
74
75      int numberOfSentRequests = 0;
76      for (double currentTime = 0; numberOfSentRequests < requestsCount;
          currentTime += 1e-3)
77      {
78        while (timeOfGeneration <= currentTime)
79        {
80          numberOfSentRequests++;
81          processor.GetRequest();
82
83          timeOfGeneration +=
              requestGenerator.GetNextTimeOfRequestGenerated();
84        }
85
86        while (timeOfProcessing <= currentTime)
87        {
88          processor.ProcessRequest();
89
90          if (processor.CurrentNumberOfRequestsInQueue > 0)
91          timeOfProcessing += processor.GetNextTimeOfRequestProcessed();
92          else
```

```
93          timeOfProcessing = timeOfGeneration +
               processor.GetNextTimeOfRequestProcessed();
94        }
95      }
96
97      while (processor.CurrentNumberOfRequestsInQueue > 0)
98      processor.ProcessRequest();
99
100     return (processor.NumberOfReturnedRequests,
           processor.DetectedMaxOfRequestsInQueue);
101   }
102
103   public (double, double) SimulateUsingEventMethod(int requestsCount)
104   {
105     double timeOfGeneration =
           requestGenerator.GetNextTimeOfRequestGenerated();
106     double timeOfProcessing = timeOfGeneration +
           processor.GetNextTimeOfRequestProcessed();
107
108     int numberOfSentRequests = 0;
109     while (numberOfSentRequests < requestsCount)
110     {
111       while (timeOfGeneration <= timeOfProcessing)
112       {
113         numberOfSentRequests++;
114         processor.GetRequest();
115
116         timeOfGeneration +=
               requestGenerator.GetNextTimeOfRequestGenerated();
117       }
118
119       while (timeOfGeneration >= timeOfProcessing)
120       {
121         processor.ProcessRequest();
122
123         if (processor.CurrentNumberOfRequestsInQueue > 0)
124         timeOfProcessing += processor.GetNextTimeOfRequestProcessed();
125         else
126         timeOfProcessing = timeOfGeneration +
               processor.GetNextTimeOfRequestProcessed();
127       }
128     }
129
130     while (processor.CurrentNumberOfRequestsInQueue > 0)
131     processor.ProcessRequest();
132
133     return (processor.NumberOfReturnedRequests,
           processor.DetectedMaxOfRequestsInQueue);
134   }
135 }
```

2.3 Результаты работы

На рисунке 2.1 представлен пользовательский интерфейс программы в исходном состоянии.

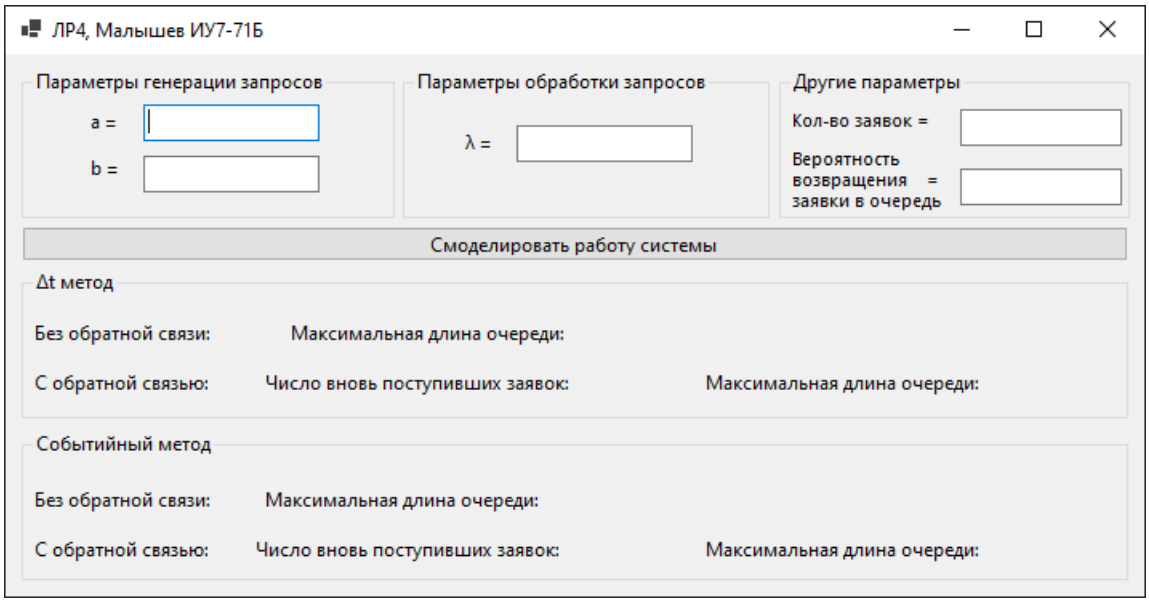

Рис. 2.1: Пользовательский интерфейс программы в исходном состоянии.

На рисунках 2.2-2.5 представлены примеры результатов работы программы с указанными данными.

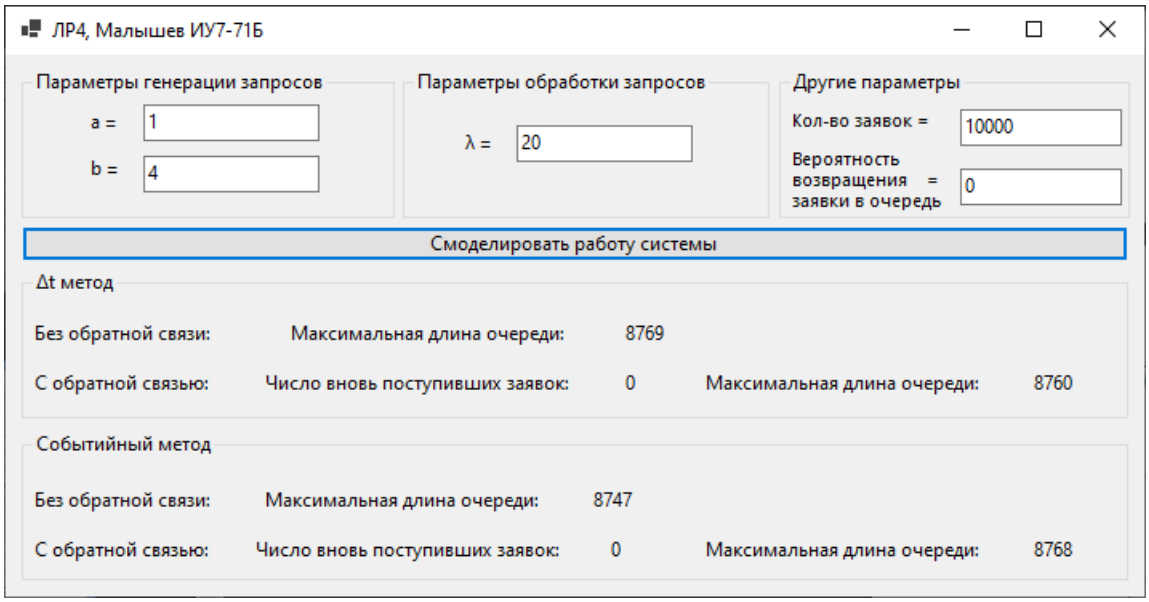

Рис. 2.2: Пример работы для системы без обратной связи с ОА, обрабатывающим запросы медленнее их генерации.

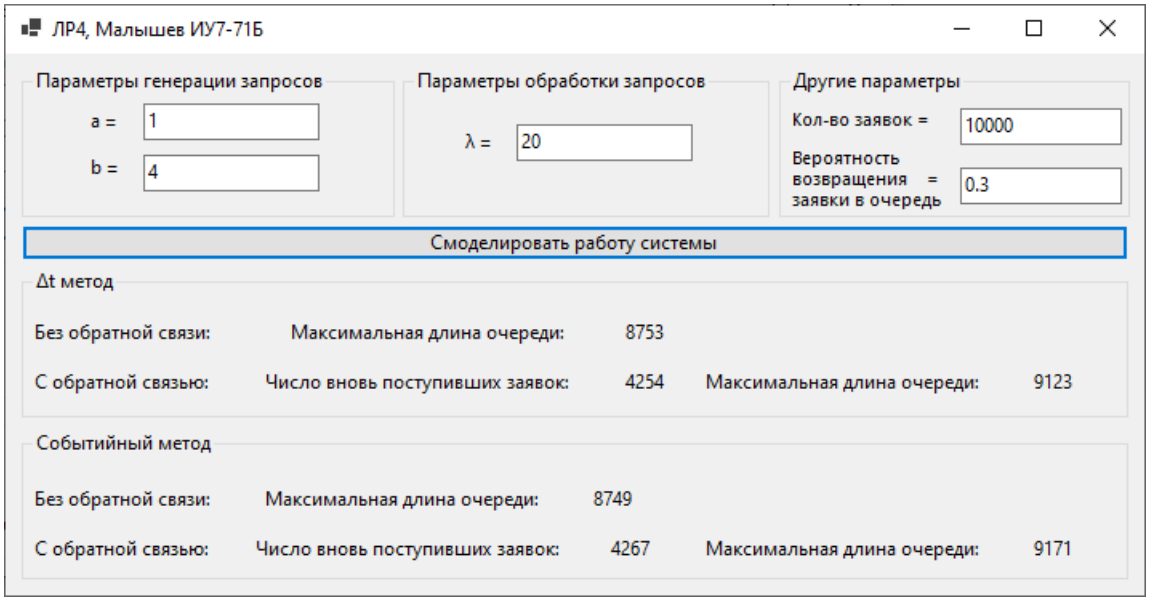

Рис. 2.3: Пример работы для системы без обратной связи с ОА, обрабатывающим запросы медленнее их генерации.

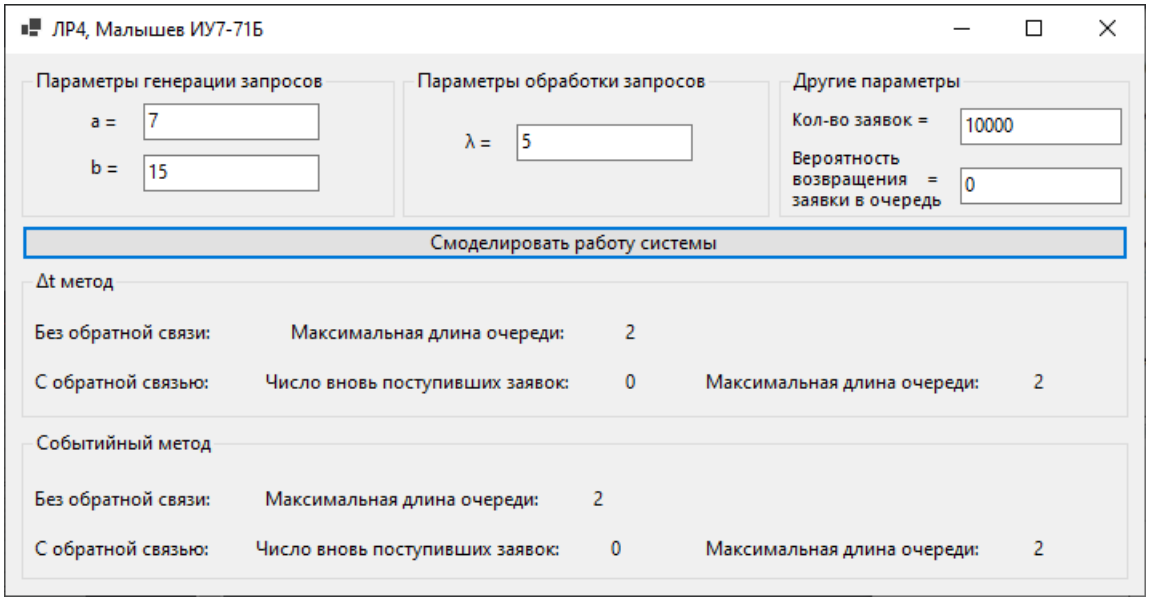

Рис. 2.4: Пример работы для системы без обратной связи с ОА, обрабатывающим запросы быстрее их генерации.

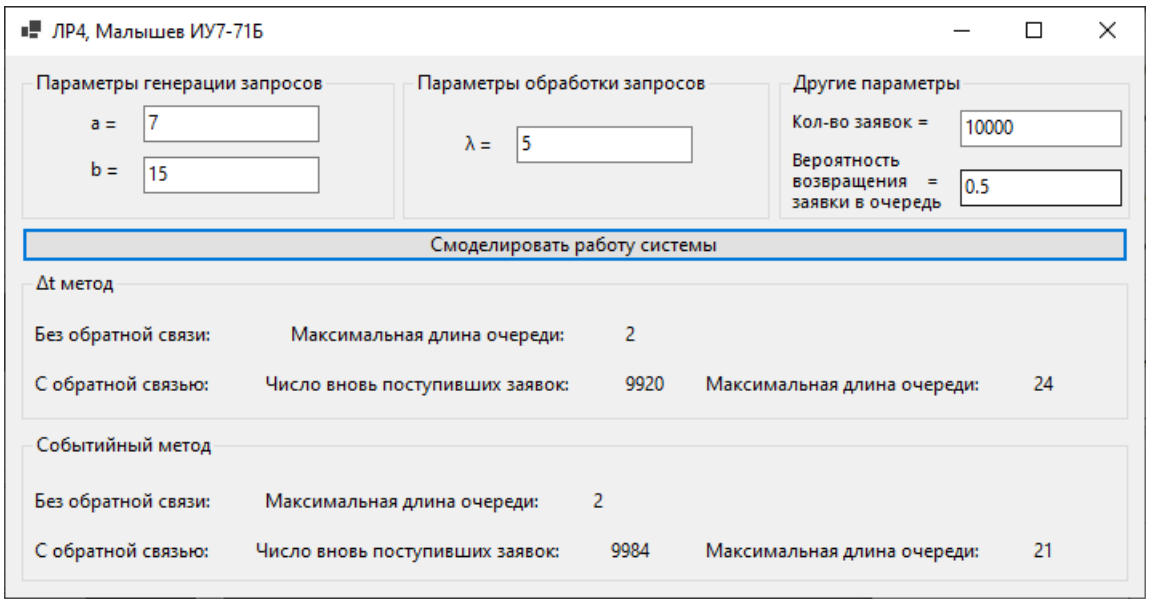

Рис. 2.5: Пример работы для системы без обратной связи с ОА, обрабатывающим запросы быстрее их генерации.

Выводы

На рисунке 2.6 предоставлен один из результатов моделирования для системы, в которую поступает 1000 заявок, каждая из которых имеет 50% шанс вернуться в очередь.

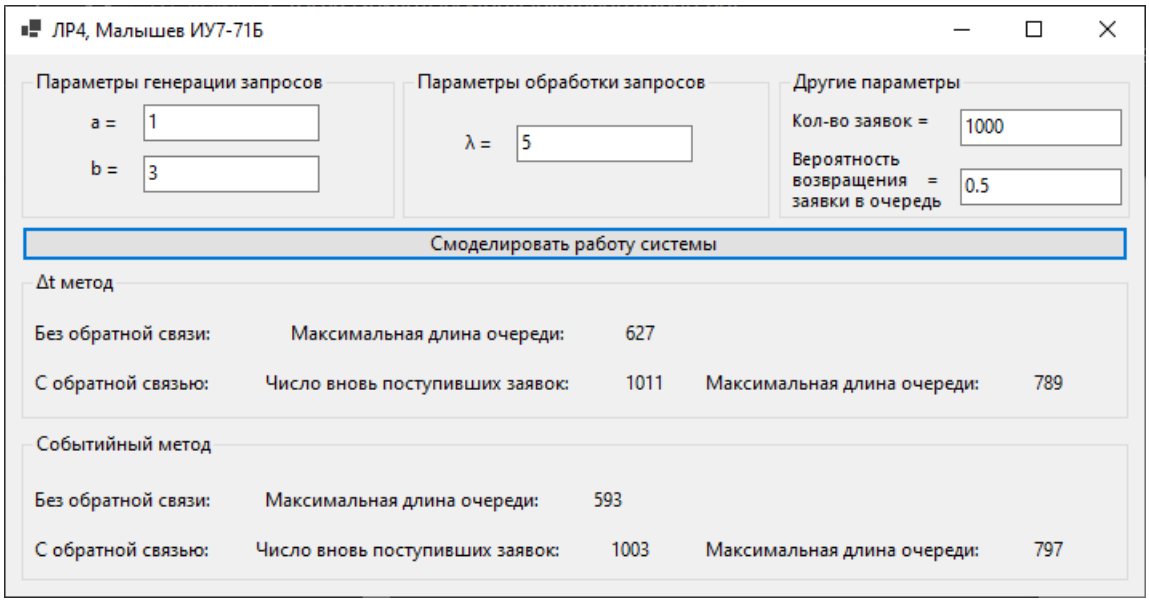

Рис. 2.6: Пример работы программы.

Как видно из результатов: при моделировании вновь поступило около 1000 заявок. Это связано с фактом того, что каждая из вернувшихся в очередь заявок может вернуться в данную очередь вновь с той же заданной вероятностью в 0.5. Таким образом:

$$\begin{aligned} a_0 &= 1000 \\ a_{i+1} &= \frac{a_i}{2} \\ \sum_{i=1}^{a_i>0} a_i &= 1001, \end{aligned} \tag{2.1}$$

что и является ожидаемым значением количества вновь поступивших заявок.

Из рисунков 2.2-2.5 заметно, что результаты моделирования с использованием двух отличных способов протяжки модельного времени отличаются несущественно.

Основным недостатком протяжки времени по Δt является значительные затраты машинного времени на реализацию моделирования системы. При этом недостаточное малое Δt появляется опасность пропуска отдельных событий в системе, что исключает возможность получения адекватных результатов. В данной лабораторной работе $\Delta t = 10^{-3}$.

Основным недостатком протяжки времени по событийному принципу является надобность в постоянном анализе списка будущих событий, что с большим количеством событий в системе может привести к большим затратам памяти и машинного времени.