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 курсу: «Моделирование»

Тема: «Моделирование аппарата обслуживания»

Студент: Пронин А. С.

Группа: ИУ7-72Б

Преподаватель: Рудаков И. В.

Оценка: _____

Москва

2022

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 (закон Эрланга). Определить длину очереди, при которой не произойдёт потери сообщений.

Промоделировать по следующим принципам:

- принцип Δt;
- событийный принцип.

1 Теория

1.1 Принцип Δt

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

Основной **недостаток** этого принципа: значительные затраты машинного времени на реализацию моделирования системы. А при недостаточно малом Δt появляется опасность пропуска отдельных событий в системе, что исключает возможность получения адекватных результатов при моделировании. **Достоинство**: равномерная протяжка времени.

1.2 Событийный принцип

Характерное свойство моделируемых систем обработки информации то, что состояние отдельных устройств изменяются в дискретные моменты времени, совпадающие с моментами времени поступления сообщений в систему, временем поступления окончания задачи, времени поступления аварийных сигналов и т.д. Следовательно моделирование и продвижение времени в системе удобно проводить, используя событийный принцип. При использовании данного принципа, состояние всех блоков 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 блоков системы.

Недостаток событийного принципа: (самостоятельная обработка)

1.3 Программа

Условием остановки поиска является обслуживание 100 000 сообщений без изменения максимальной длины очереди.

В случае, если такое событие не происходит за 1 000 000 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.

2 Текст программы

В листингах 2.1–2.6 представлен код программы, отвечающий за моделирование.

Листинг 2.1: Код генератора

```
class Generator
{
  public Generator(double _a, double _b)
  {
    a = _a;
    b = _b;
  }

  public bool isReady(double t)
  {
    return (t >= gen_time);
  }

  public Request genRequest()
  {
    Request new_r = new Request(gen_time);
    updateGenTime();
    return new_r;
  }

  private void updateGenTime()
  {
    double res = a + (b - a) * rnd.NextDouble();
    gen_time += res;
  }

  public double a;
  public double b;
  public double gen_time = 0;

  private Random rnd = new Random();
}
```

Листинг 2.2: Код заявки

```
class Request
{
  public Request(double t)
  {
    create_time = t;
  }

  public double create_time = 0;
  public double serve_time = -1;
}
```

Листинг 2.3: Код очереди заявок

```
class ReqQue : Queue<Request>
{
  public bool push(Request r)
  {
    Enqueue(r);
    if (Count > max_size)
    {
      max_size = Count;
      return true;
    }
    return false;
  }
  public Request pop()
  {
    return Dequeue();
  }

  public int max_size = 0;
}
```

Листинг 2.4: Код обслуживающего аппарата

```
class Service
{
  public Service(double l, int a)
  {
    lambda = l;
    alpha = a;
  }

  public bool isFree(double t)
  {
    return (t >= free_time);
  }

  public void serve(Request r, double t)
  {
    updateFreeTime(t);
    r.serve_time = free_time;
  }

  public void updateFreeTime(double t)
  {
    double res = 0;
    for (int i = 0; i < alpha; i++)
      res -= Math.Log(1 - rnd.NextDouble());
    res /= alpha * lambda;

    free_time = t + res;
  }

  public double lambda;
  public int alpha;
  public double free_time = 0;

  private Random rnd = new Random();
}
```

Листинг 2.5: Код модели Δt

```
class DeltaTModel
{
  public DeltaTModel(double _a, double _b, double l, int al,
   double fb = 0)
  {
    generator = new Generator(_a, _b);
    req_que = new ReqQue();
    service = new Service(l, al);
    feedback = fb;
  }

  public int getMaxBufSize(double dT = 0.01, double maxT = 1e6)
  {
    overflow = true;
    while (curT <= maxT)
    {
      if (no_overflow_n >= 1e5)
        overflow = false;
      handleGenerator();
      handleService();
      curT += dT;
    }
    if (overflow)
      return -1;
    else
      return req_que.max_size;
  }

  private void handleGenerator()
  {
    if (generator.isReady(curT))
    {
      Request new_r = generator.genRequest();
      if (req_que.push(new_r))
        no_overflow_n = 0;
    }
  }


```

```
40    private void handleService()
41    {
42      if (req_que.Count != 0 && service.isFree(curT))
43      {
44        Request r = req_que.pop();
45        no_overflow_n++;
46        service.serve(r, curT);
47
48        if (rnd.NextDouble() < feedback)
49        {
50          Request old_r = new Request(curT);
51          req_que.push(old_r);
52        }
53      }
54    }
55
56    public Generator generator;
57    public ReqQue req_que;
58    public Service service;
59    public double feedback;
60    public double curT = 0;
61    public double no_overflow_n = 0;
62    public bool overflow = true;
63
64    private Random rnd = new Random();
65  }
```

Листинг 2.6: Код событийной модели

```
class EventModel
{
  public EventModel(double _a, double _b, double l, int al,
   double fb = 0)
  {
    generator = new Generator(_a, _b);
    req_que = new ReqQue();
    service = new Service(l, al);
    feedback = fb;
  }

  public int getMaxBufSize(double maxT = 1e6)
  {
    overflow = true;
    curT = 0;
    while (curT <= maxT)
    {
      if (no_overflow_n >= 1e5)
        overflow = false;
      handleGenerator();
      handleService();
      curT = times.Min();
      times.Remove(curT);
    }
    if (overflow)
      return -1;
    else
      return req_que.max_size;
  }

  private void handleGenerator()
  {
    if (generator.isReady(curT))
    {
      Request new_r = generator.genRequest();
      if (req_que.push(new_r))
        no_overflow_n = 0;
      times.Add(generator.gen_time);
    }
  }
```

```
40
41    private void handleService()
42    {
43      if (req_que.Count != 0 && service.isFree(curT))
44      {
45        Request r = req_que.pop();
46        no_overflow_n++;
47        service.serve(r, curT);
48        times.Add(service.free_time);
49
50        if (rnd.NextDouble() < feedback)
51        {
52          Request old_r = new Request(curT);
53          req_que.push(old_r);
54        }
55      }
56    }
57
58    public Generator generator;
59    public ReqQue req_que;
60    public Service service;
61    public double feedback;
62    public double curT = 0;
63    public bool overflow = true;
64    public double no_overflow_n = 0;
65
66    private Random rnd = new Random();
67    private List<double> times = new List<double>();
68  }
```

3 Результаты

Примеры работы программы приведены на рисунках 3.1–3.4.

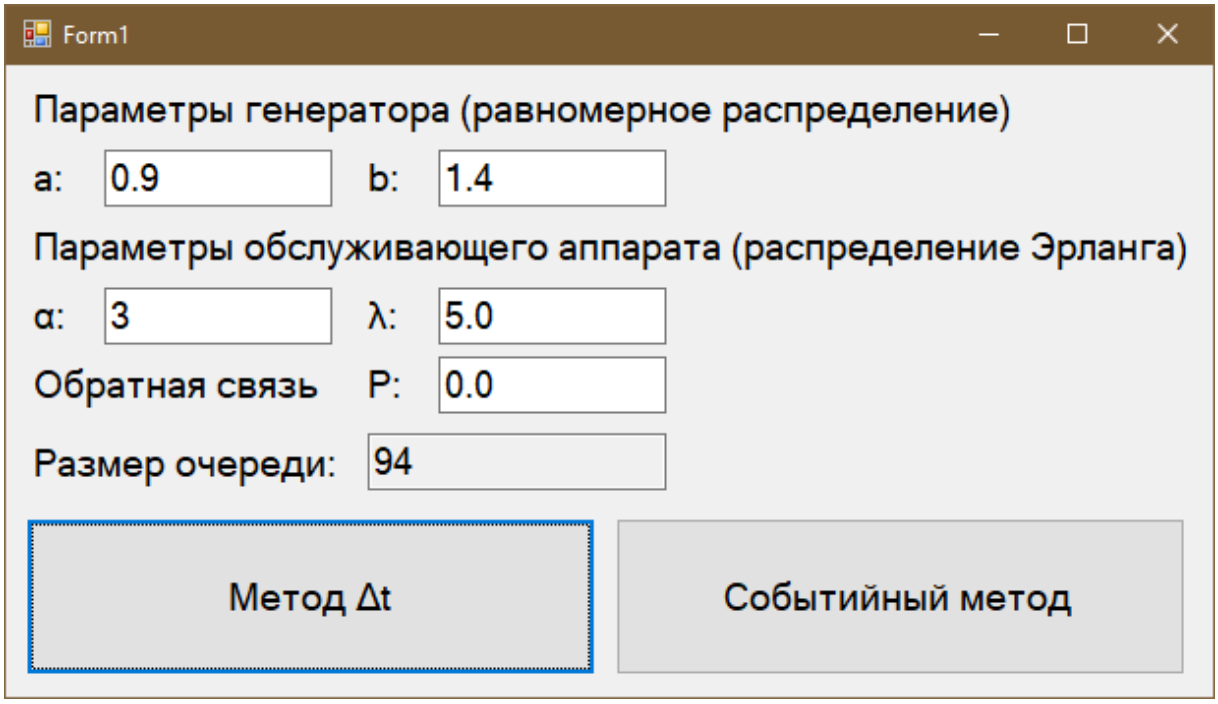

Рис. 3.1: Принцип Δt без обратной связи

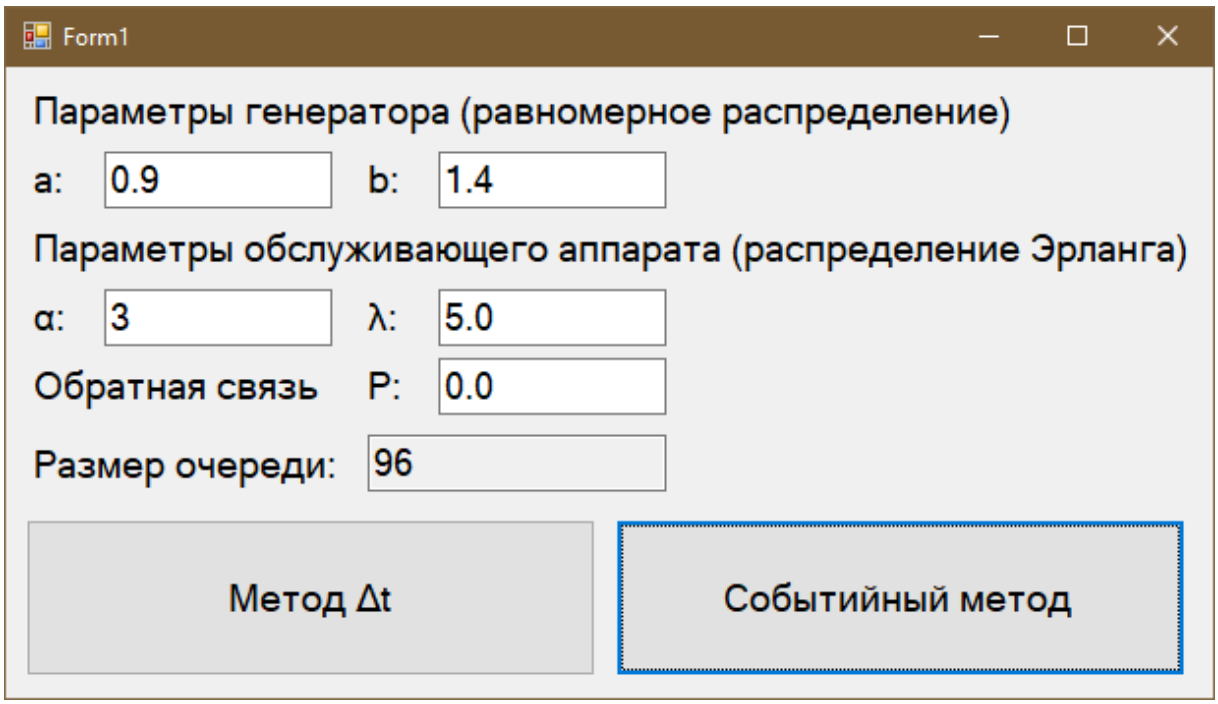

Рис. 3.2: Событийный принцип без обратной связи

По рисункам 3.1 и 3.2 видно, что результаты обоих принципов примерно одинаковы.

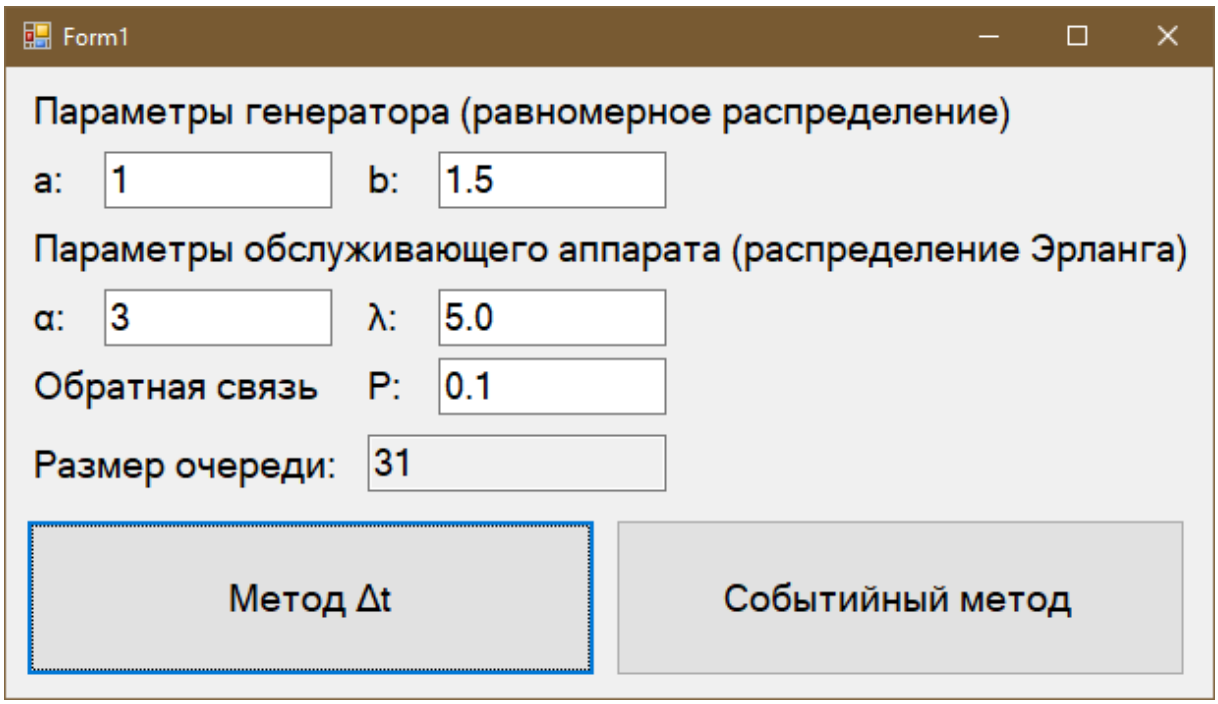

Рис. 3.3: Принцип Δt с обратной связью

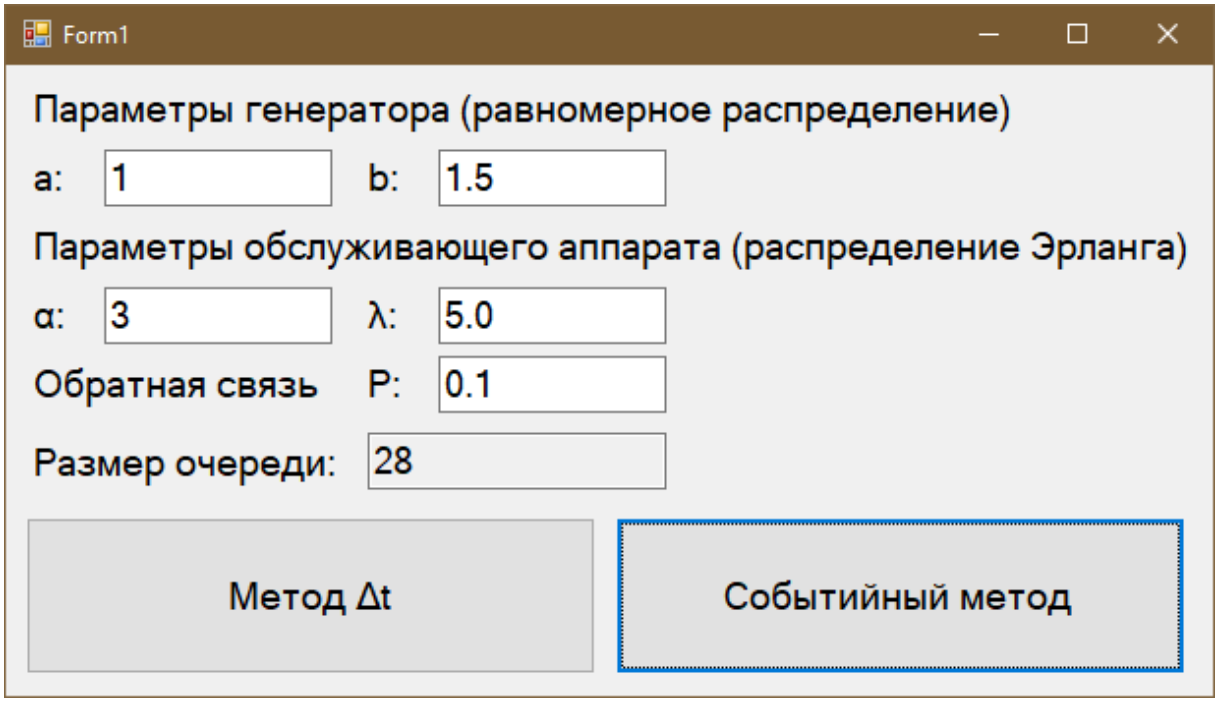

Рис. 3.4: Событийный принцип с обратной связью = 1

Как видно на рисунках 3.3 и 3.4 при обратной связи результаты разных методов также похожи.

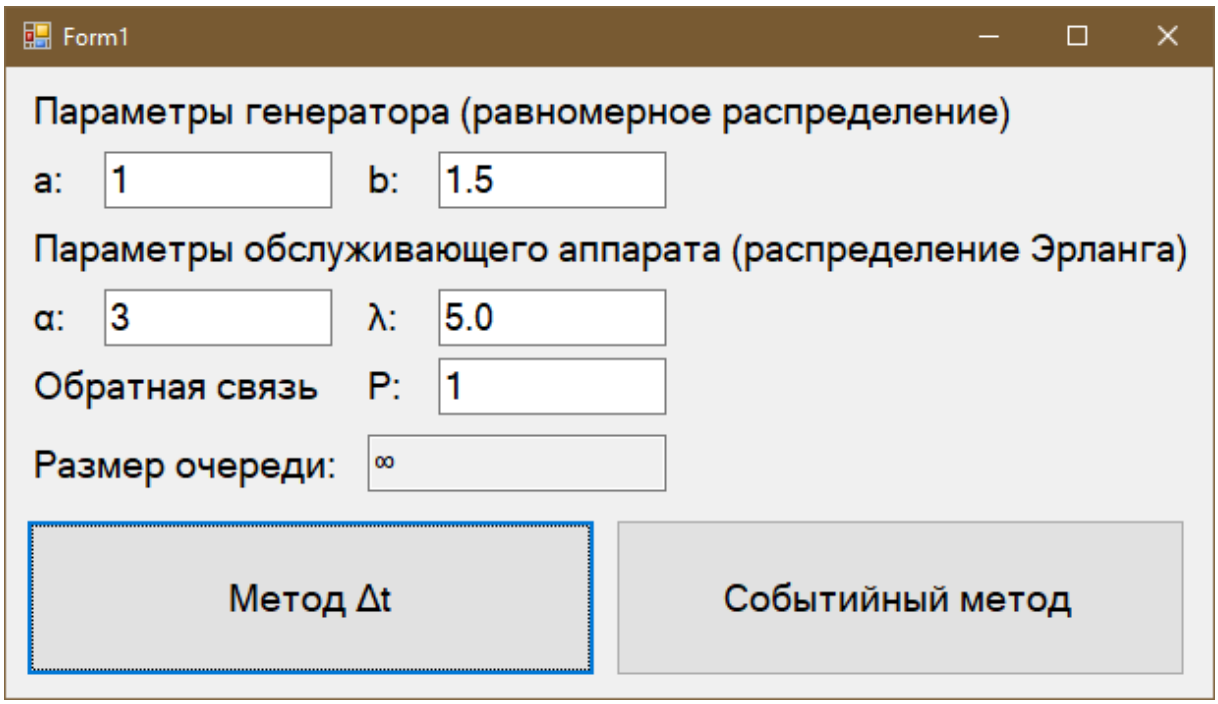

Рис. 3.5: Принцип Δt с обратной связью

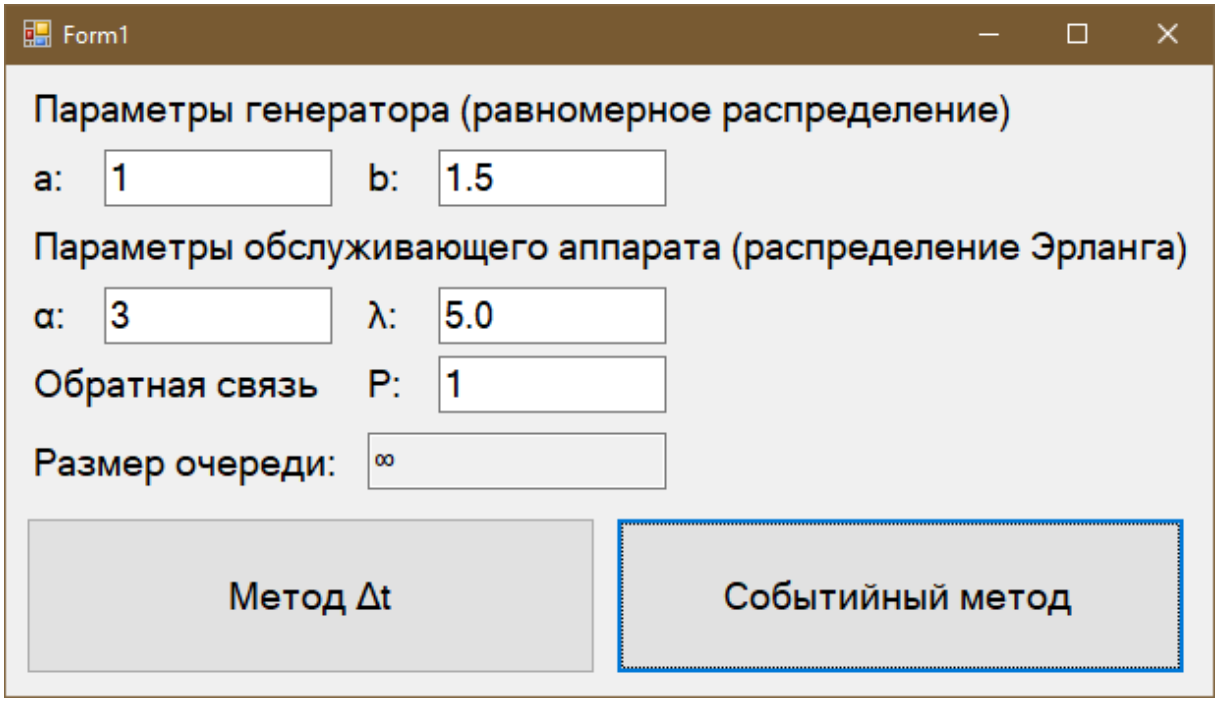

Рис. 3.6: Событийный принцип с обратной связью = 1

Очевидно что при обратной связи равной 1, потери сообщений не избежать вне зависимости от используемого метода, что видно на рисунках 3.5 и 3.6.