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 Обслуживающий аппарат

Студент Егорова П.А.

Группа ИУ7-74Б

Преподаватели Рудаков И.В.

Москва — 2023 г.

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

Функция распределения имеет вид:

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

2.2 Нормальное распределение

Функция распределения имеет вид:

$$F_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \int\limits_{-\infty}^{x} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \tag{2}$$

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 Код программы

В листинге 1 представлены основные методы.

Листинг 1 — Основные методы

```
func eventModel(_ generator: EvenDistribution, _ processor: NormalDistribution, _ totalTasks
    : Int = 0, _ repeat_percentage: Int = 0) -> Int {
    var processedTasks = 0
    var curQueueLen = 0
    var maxQueueLen = 0
    var events: [(Double, String)] = [(generator.generate(), "g")]
    var free = true
    var processFlag = false

    while processedTasks < totalTasks {
        let event = events.removeFirst()
        if event.1 == "g" {
            curQueueLen += 1
            if curQueueLen > maxQueueLen {
                maxQueueLen = curQueueLen
            }
            addEvent(&events, (event.0 + generator.generate(), "g"))

            if free {
                processFlag = true
            }
        } else if event.1 == "p" {
            processedTasks += 1

            if Int.random(in: 1...100) <= repeat_percentage {
                curQueueLen += 1
            }
            processFlag = true
        }

        if processFlag {
            if curQueueLen > 0 {
                curQueueLen -= 1
                addEvent(&events, (event.0 + processor.generate(), "p"))
                free = false
            } else {
                free = true
            }
            processFlag = false
        }
    }

    return maxQueueLen
}

```

```
45 func addEvent(_ events: inout [(Double, String)], _ newEvent: (Double, String)) {
46     var i = 0
47 
48     while i < events.count && events[i].0 < newEvent.0 {
49         i += 1
50     }
51 
52     0 < i && i < events.count ? events.insert(newEvent, at: i - 1) : events.insert(newEvent,
       at: i)
53 }
54 
55 func stepModel(_ generator: EvenDistribution, _ processor: NormalDistribution, _ totalTasks:
       Int, _ repeat_percentage: Int, _ step: Double) -> Int {
56     var processedTasks = 0
57     var tCurr: Double = step
58     var tGen = generator.generate()
59     var tGenPrev: Double = 0
60     var tProc: Double = 0
61     var curQueueLen = 0
62     var maxQueueLen = 0
63     var free = true
64 
65     while processedTasks < totalTasks {
66         if tCurr > tGen {
67             curQueueLen += 1
68 
69             if curQueueLen > maxQueueLen {
70                 maxQueueLen = curQueueLen
71             }
72 
73             tGenPrev = tGen
74             tGen += generator.generate()
75         }
76 
77         if tCurr > tProc {
78             if curQueueLen > 0 {
79                 let wasFree = free
80                 if free {
81                     free = false
82                 } else {
83                     processedTasks += 1
84                     if Int.random(in: 1...100) <= repeat_percentage {
85                         curQueueLen += 1
86                     }
87                 }
88 
89                 curQueueLen -= 1
90 
91                 if wasFree {
92                     tProc = tGenPrev + processor.generate()
93                 } else {
```

```
94                      tProc += processor.generate()
95                  }
96
97              } else {
98                  free = true
99              }
100         }
101         tCurr += step
102     }
103
104     return maxQueueLen
105 }
```

4 Результаты работы программы

Интерфейс предполагает ввод всех необходимых параметров модели. Моделирование происходит до тех пор, пока не будет обработано 1 000 сообщений.

Также предоставляется возможность указания в процентах объема сообщений, которые возвращаются обратно в очередь.

На рисунках 4.1 – 4.5 демонстрируются результаты работы.

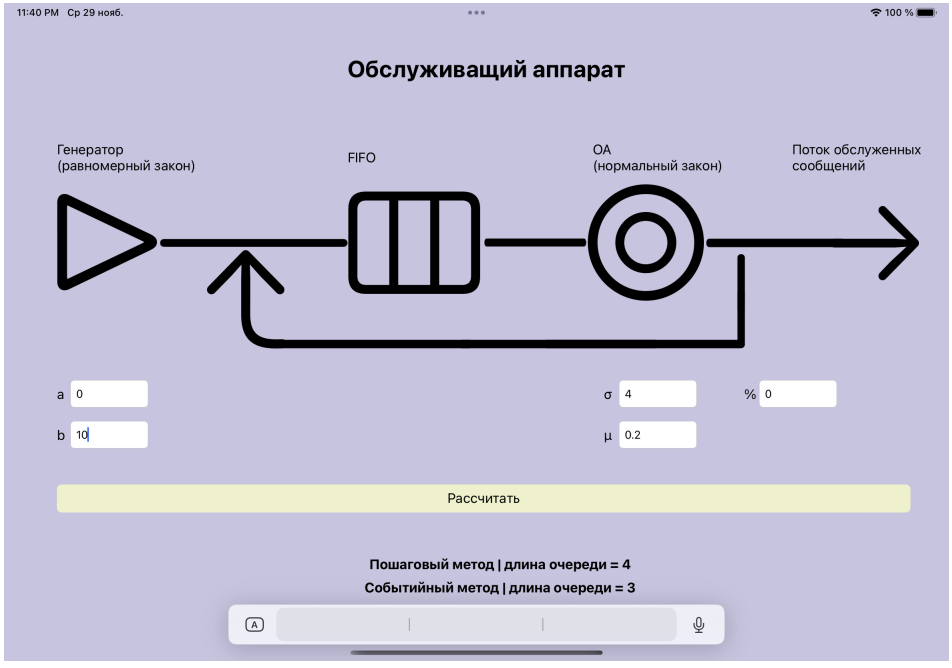

Рисунок 4.1 — Пример 1. Генератор и обслуживающий автомат работают с примерно одинаковой производительностью

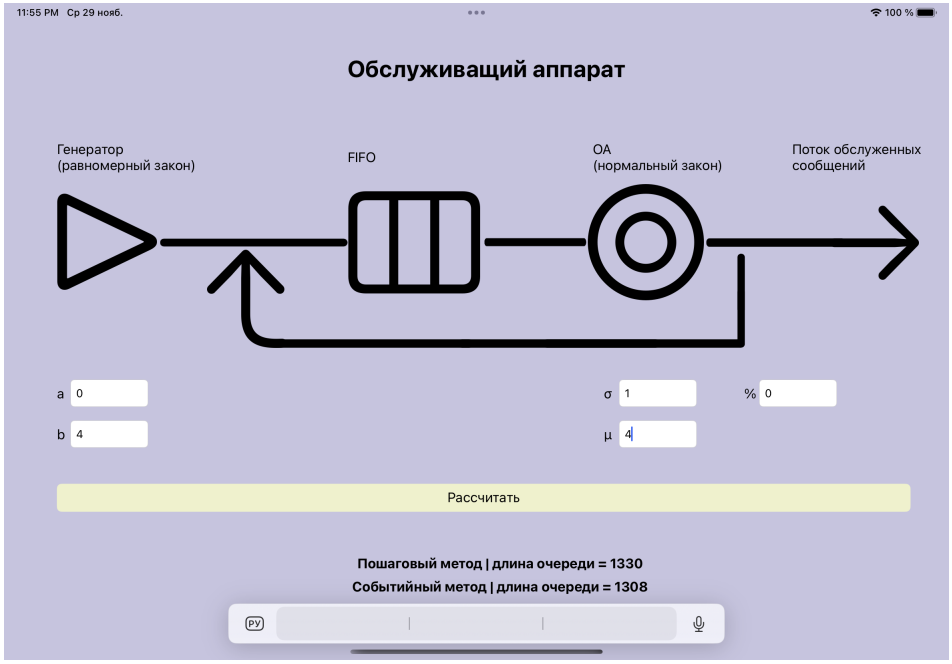

Рисунок 4.2 — Пример 2. Генератор создает сообщения интенсивнее, чем их обрабатывает автомат (при таких параметрах возникает переполнение)

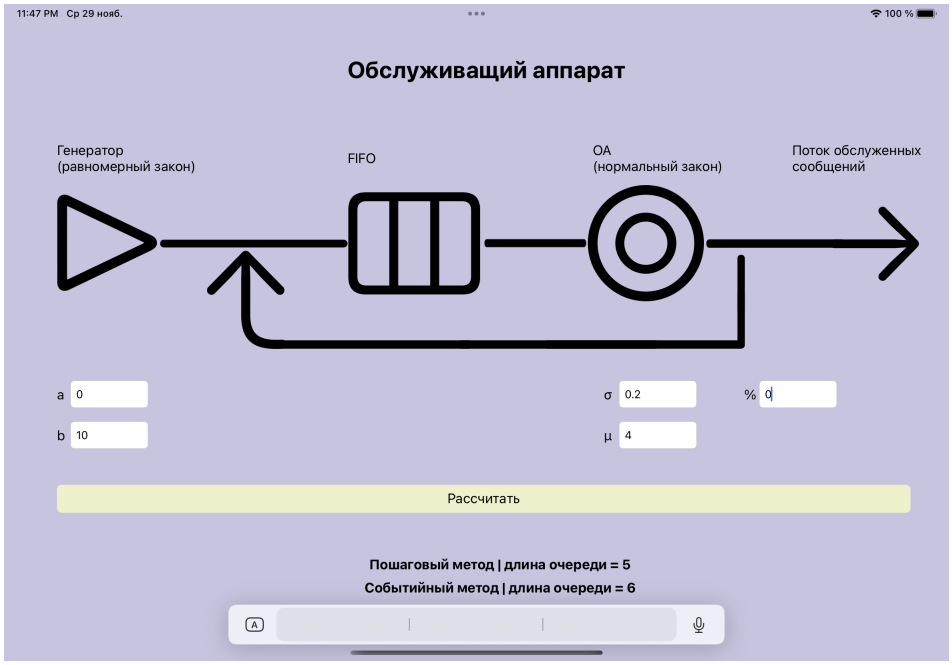

Рисунок 4.3 — Пример 3. Автомат обрабатывает сообщения интенсивнее, чем их создаёт генератор

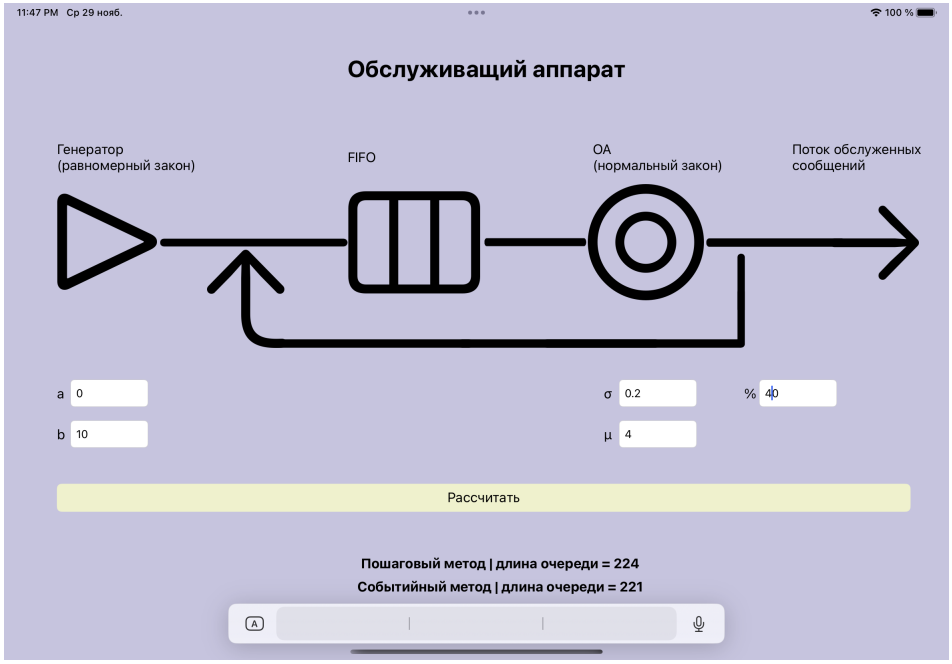

Рисунок 4.4 — Пример 4. Параметры такие же, как в примере 3, но есть процент возврата – 40%

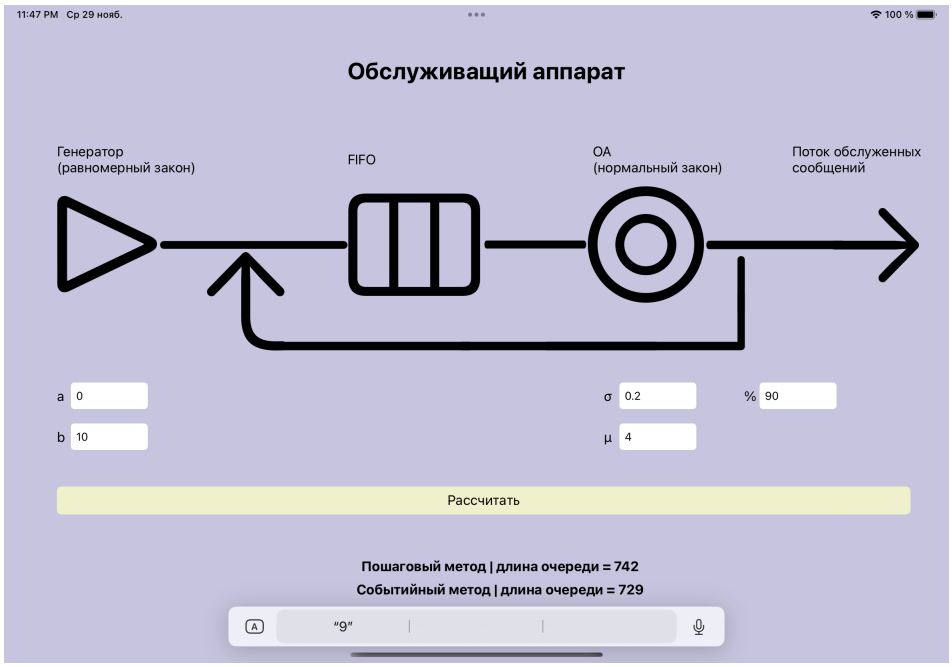

Рисунок 4.5 — Пример 5. Генератор и обслуживающий автомат работают с примерно одинаковой производительностью, но есть процент возврата – 90%