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 (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

ОТЧЕТ

Лабораторная работа №4

по курсу «Моделирование»

на тему: «Моделирование работы системы массового обслуживания»

Студент ИУ7-73Б (Группа) ________ (Подпись, дата) К.Э. Ковалец (И. О. Фамилия)

Преподаватель ________ (Подпись, дата) И.В. Рудаков (И. О. Фамилия)

2022 г.

Содержание

1 Задание

Промоделировать систему, состоящую из генератора, памяти и обслуживающего аппарата. Генератор подает сообщения, распределенные по равномерному закону, они приходят в память и выбираются на обработку по закону из ЛР1 (Эрланга). Количество заявок конечно и задано. Предусмотреть случай, когда обработанная заявка возвращается обратно в очередь. Определить оптимальную длину очереди, при которой не будет потерянных сообщений. Реализовать двумя способами: используя пошаговый и событийный подходы.

2 Теоретическая часть

2.1 Равномерное распределение

Функция равномерного распределения:

$$F(x) = \begin{cases} 0, x < a, \\ \dfrac{x-a}{b-a}, x \in [a, b], \\ 0, x > b. \end{cases} \tag{2.1}$$

Функция плотности равномерного распределения:

$$f(x) = \begin{cases} \dfrac{1}{b-a}, x \in [a, b], \\ 0, else. \end{cases} \tag{2.2}$$

2.2 Распределение Эрланга

Функция распределения Эрланга:

$$F_k(x) = 1 - e^{-\lambda \cdot x} \cdot \sum_{i=1}^{k-1} \frac{(\lambda \cdot x)^i}{i!}. \tag{2.3}$$

Функция плотности распределения Эрланга:

$$f_k(x) = \frac{\lambda \cdot (\lambda \cdot x)^{k-1}}{(k-1)!} \cdot e^{-\lambda \cdot x}. \tag{2.4}$$

В данных формулах λ и k — положительные параметры распределения ($\lambda \geqslant 0; k = 1, 2, ...$); $x \geqslant 0$.

2.3 Принципы управляющей программы

2.3.1 Пошаговый подход

Заключается в последовательном анализе состояний всех блоков системы в момент $t + \Delta t$ по заданному состоянию в момент t. При этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием с учетом действующих случайных факторов. В результате этого анализа принимается решение о том, какие системные события должны имитироваться на данный момент времени. Основной недостаток: значительные затраты машинных ресурсов, а при недостаточном малых Δt появляется опасность пропуска события.

2.3.2 Событийный принцип

Характерное свойство модели системы обработки информации: состояние отдельных устройств изменяется в дискретные моменты времени, совпадающие с моментами поступления сообщения, окончания решения задачи, возникновения аварийных сигналов и т. д. При использовании событийного принципа состояния всех боков системы анализируется лишь в момент появления какого-либо события. Момент наступления следующего события определяется минимальным значением из списка будущих событий, представляющий собой совокупность моментов ближайшего изменения состояния каждого из блоков. Момент наступления следующего события определяется минимальным значением из списка событий.

3 Результаты работы

3.1 Листинги программы

В листинге 3.1 представлена функция управления системой массового обслуживания с помощью пошагового принципа.

Листинг 3.1 — Пошаговый принцип

```
def stepModel(generator, processor, countTasks, repeatProbability, step):
    tasksDone = 0
    timeCurrent = step
    timeGenerated = generator.generate()
    timeGeneratedPrev = 0
    timeProcessed = 0

    curQueueLen = 0
    maxQueueLen = 0
    free = True


    while tasksDone < countTasks:
        # Генератор
        if timeCurrent > timeGenerated:
            curQueueLen += 1

            if curQueueLen > maxQueueLen:
                maxQueueLen = curQueueLen

            timeGeneratedPrev = timeGenerated
            timeGenerated += generator.generate()

        # Обработчик
        if timeCurrent > timeProcessed:
            if curQueueLen > 0:
                wasFree = free

                if free:
                    free = False
                else:
                    tasksDone += 1
                    curQueueLen -= 1

                    if randint(1, 100) <= repeatProbability:
                        curQueueLen += 1

                if wasFree:
```

Продолжение листинга 3.1

```
                    timeProcessed = timeGeneratedPrev + processor.generate()
                else:
                    timeProcessed += processor.generate()
            else:
                free = True

        timeCurrent += step

    return maxQueueLen
```

В листинге 3.2 представлена функция управления системой массового обслуживания с помощью событийного принципа.

Листинг 3.2 — Событийный принцип

```
def eventModel(generator, processor, countTasks, repeatProbability):
    tasksDone = 0
    curQueueLen = 0
    maxQueueLen = 0
    free = True
    processFlag = False
    events = [[generator.generate(), "g"]]

    while tasksDone < countTasks:
        event = events.pop(0)

        # Генератор
        if event[1] == "g":
            curQueueLen += 1

            if curQueueLen > maxQueueLen:
                maxQueueLen = curQueueLen

            addEvent(events, [event[0] + generator.generate(), "g"])

            if free:
                processFlag = True

        # Обработчик
        elif event[1] == "p":
            tasksDone += 1

            if randint(1, 100) <= repeatProbability:
                curQueueLen += 1

            processFlag = True
```

Продолжение листинга 3.2

```python

        if processFlag:
            if curQueueLen > 0:
                curQueueLen -= 1
                addEvent(events, [event[0] + processor.generate(), "p"])
                free = False
            else:
                free = True

            processFlag = False

    return maxQueueLen


def addEvent(events: list, event: list):
    i = 0
    while i < len(events) and events[i][0] < event[0]:
        i += 1

    if 0 < i < len(events):
        events.insert(i - 1, event)
    else:
        events.insert(i, event)
```

3.2 Демонстрация работы программы

На рисунках 3.1 - 3.4 представлены примеры работы программы.

Рисунок 3.1 – Результат работы при вероятности возврата заявки 0%

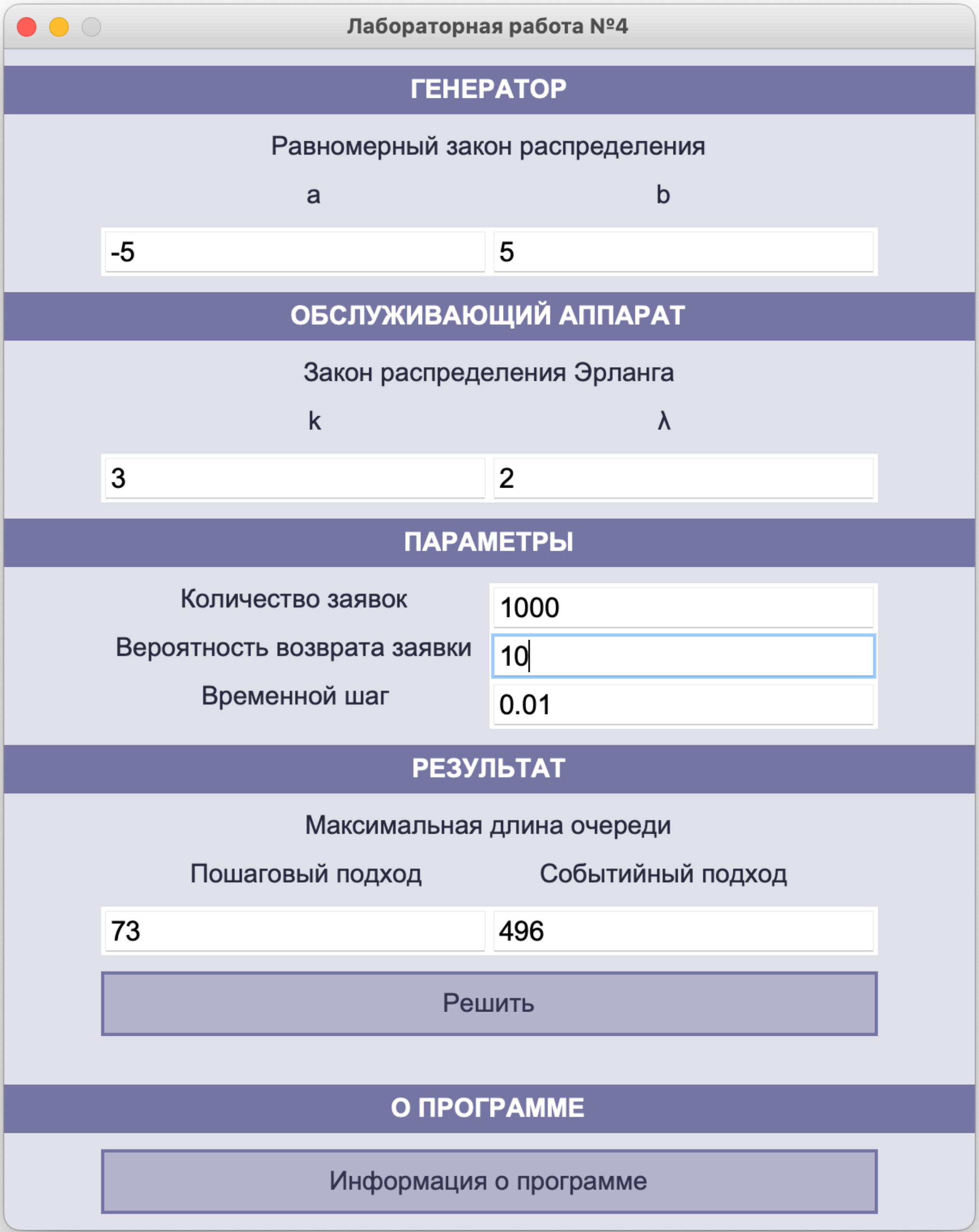

Рисунок 3.2 – Результат работы при вероятности возврата заявки 10%

Рисунок 3.3 – Результат работы при вероятности возврата заявки 50%

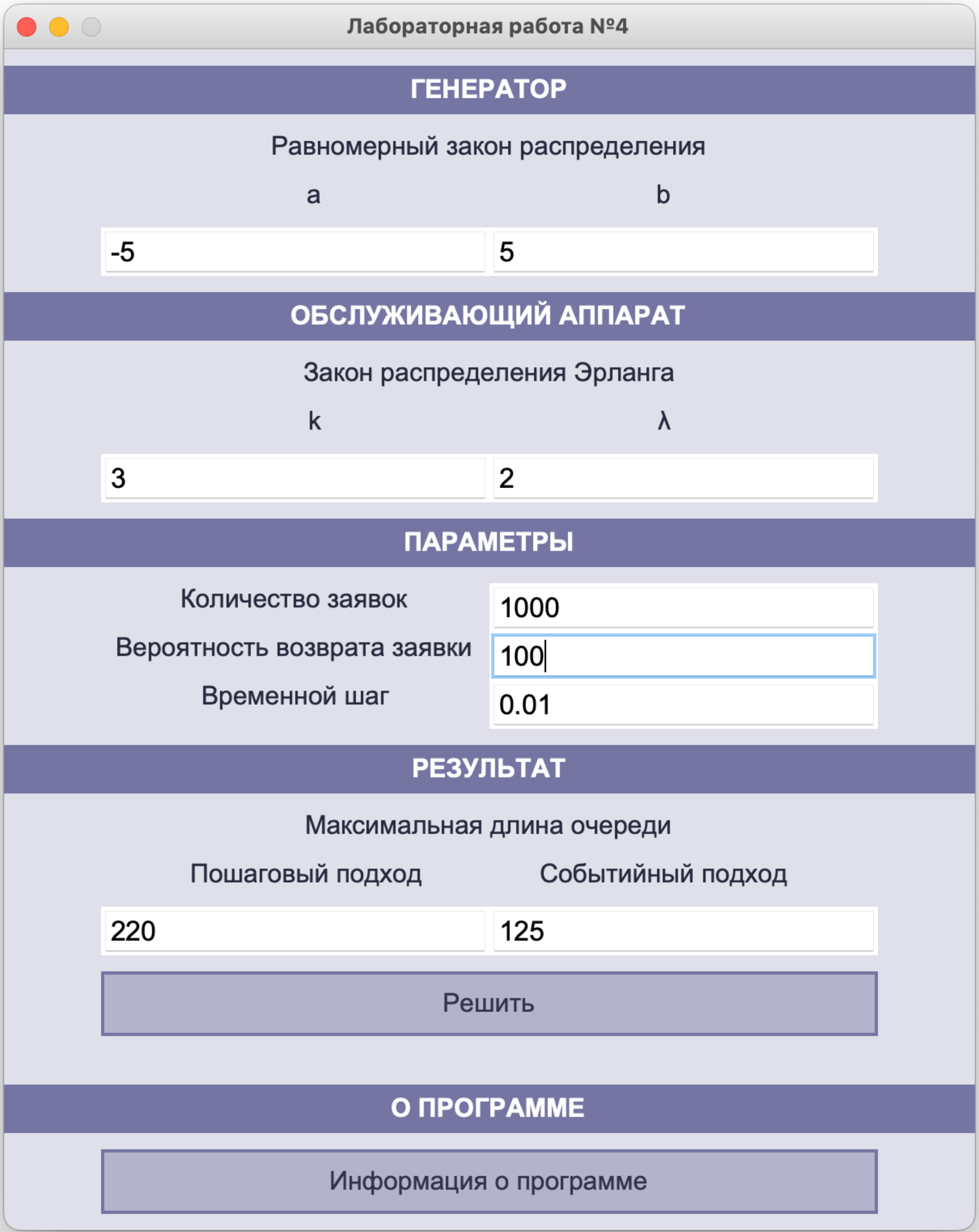

Рисунок 3.4 – Результат работы при вероятности возврата заявки 100%