Министерство образования и науки 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

На тему: Обслуживающий аппарат

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-76Б

Преподователь:

Рудаков Игорь Владимирович

Москва, 2021 г.

Содержание

1 Задание

Необходимо промоделировать систему, состоящую из генератора памяти и обслуживающего аппарата. Генератор подаёт сообщения, распределённые по нормальному закону, они проходят в память и выбираются на обработку по закону из лабораторной работы №1. Количество заявок конечно и задано. Предусмотреть случай, когда обработанная заявка возвращается обратно в очередь. Необходимо определить оптимальную длину очереди, при которой не будет потерянных сообщений. Реализовать, используя пошаговый и событийные подходы.

2 Теоритическая часть

2.1 Равномерное распределение

Непрерывное равномерное распределение - распределение случайной вещественной величины, принимающей значения, принадлежащие некоторому промежутку конечной длины, характеризующееся тем, что плотность вероятности на этом промежутке почти всюду постоянна.

Плотность распределения представлена в формуле 1.

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

Функция распределения представлена в формуле 2.

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

2.2 Нормальное распределение

Нормальное распределение — распределение вероятностей, которое в одномерном случае задаётся функцией плотности вероятности, совпадающей с функцией Гаусса.

Плотность распределения представлена в формуле 3.

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

Функция распределения представлена в формуле 4.

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \tag{4}$$

2.3 Пошаговый подход

Пошаговый подход 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 подхода: 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.

2.4 Событийный подход

Характерное свойство моделируемых систем — состояние отдельных устройств изменяется в дискретные моменты времени, которые совпадают с моментами поступления сообщений в систему, моментами окончания решения задач, моментами возникающих аварийных сигналов и т.д. Поэтому, моделирование и продвижение текущего времени в системе удобно проводить используя событийный принцип, при котором состояние всех блоков системы анализируется лишь в момент наступления какого-либо события. Момент наступления следующего события определяется минимальным значением из списка будущих событий, представляющих собой совокупность моментов ближайшего изменения состояний каждого из блоков системы.

3 Результаты

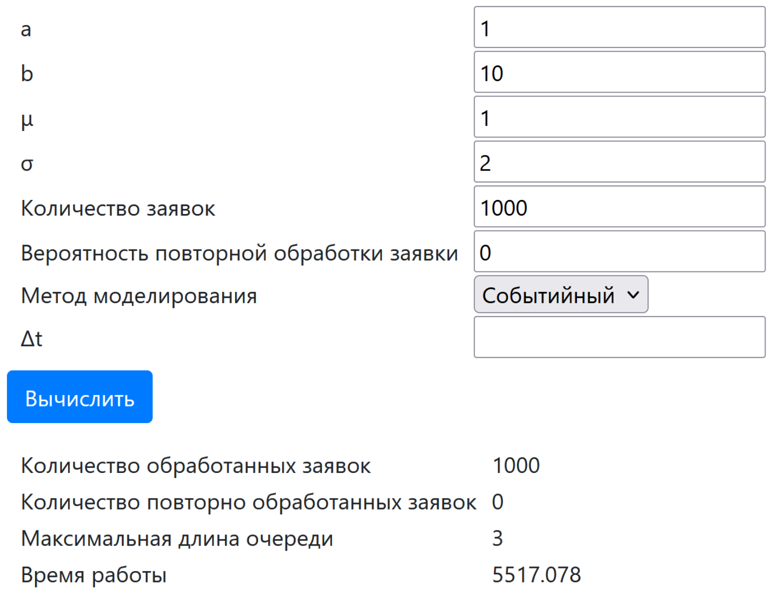

Рис. 1: Событийный подход, вероятность возврата заявки в очередь равна 0

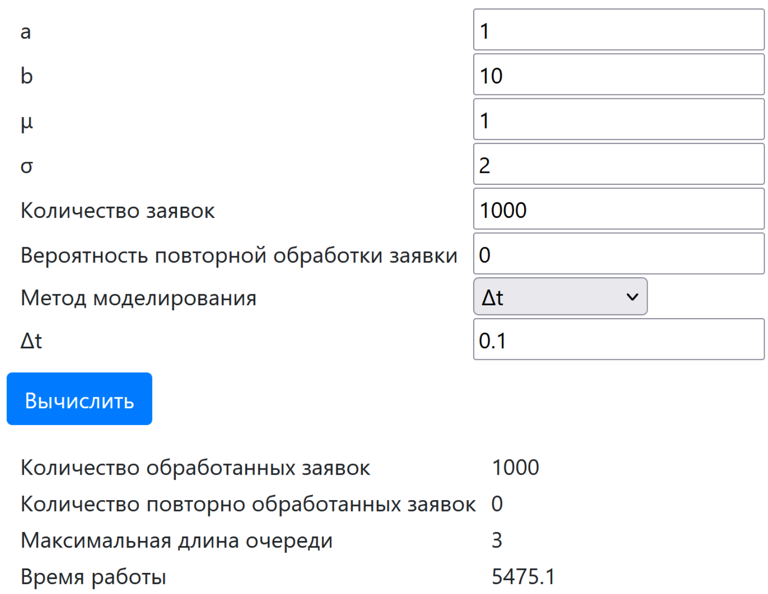

Рис. 2: Пошаговый подход, вероятность возврата заявки в очередь равна 0

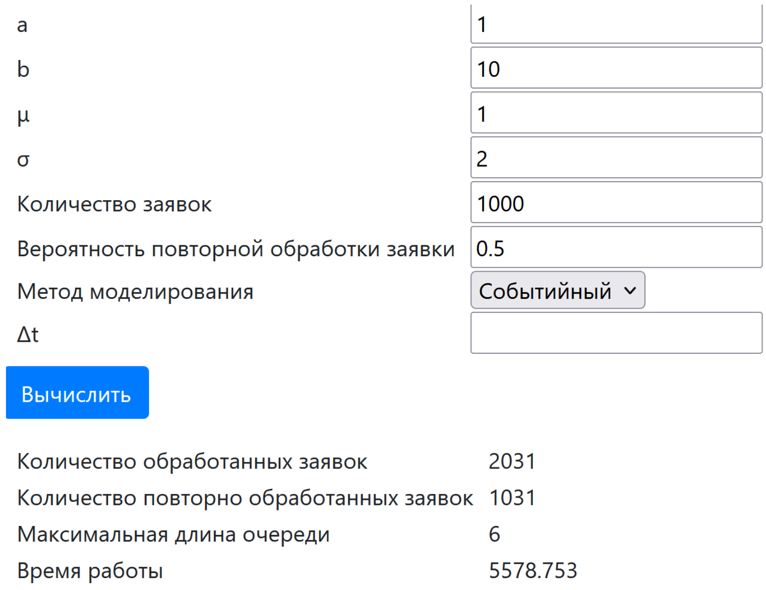

Рис. 3: Событийный подход, вероятность возврата заявки в очередь равна 0.5

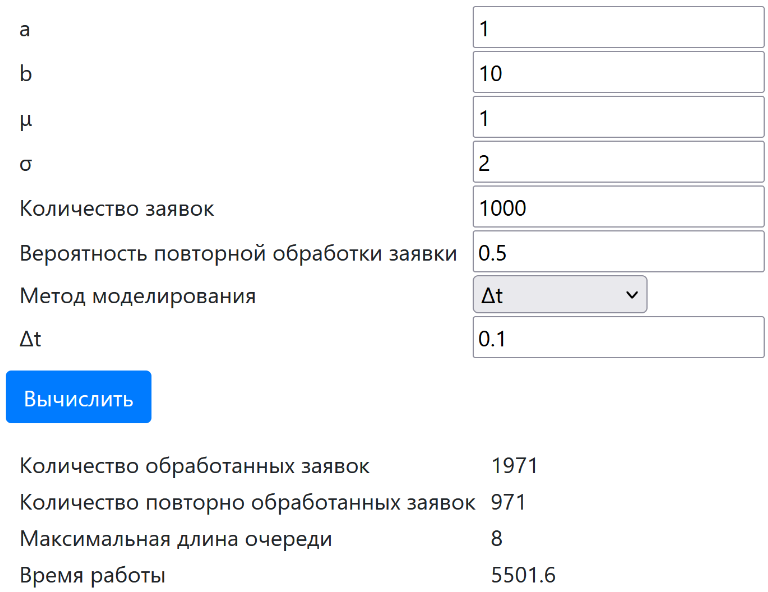

Рис. 4: Пошаговый подход, вероятность возврата заявки в очередь равна 0.5

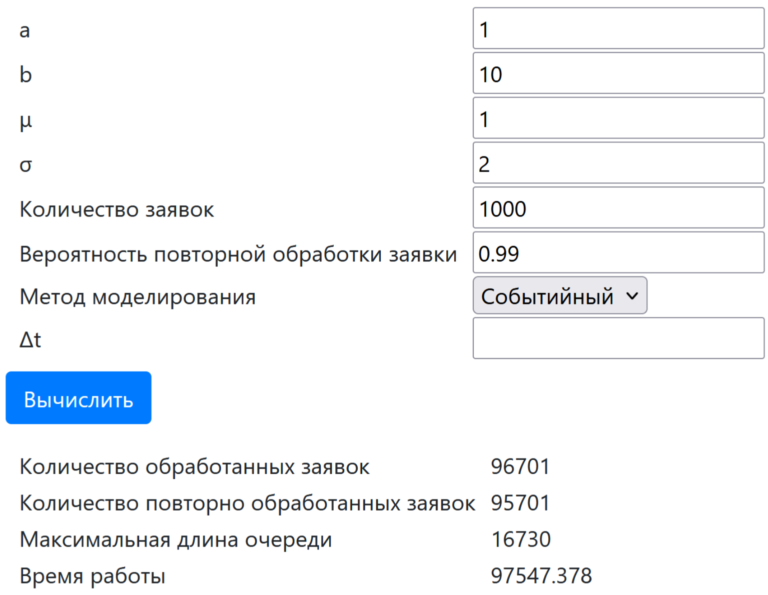

Рис. 5: Событийный подход, вероятность возврата заявки в очередь равна 0.99

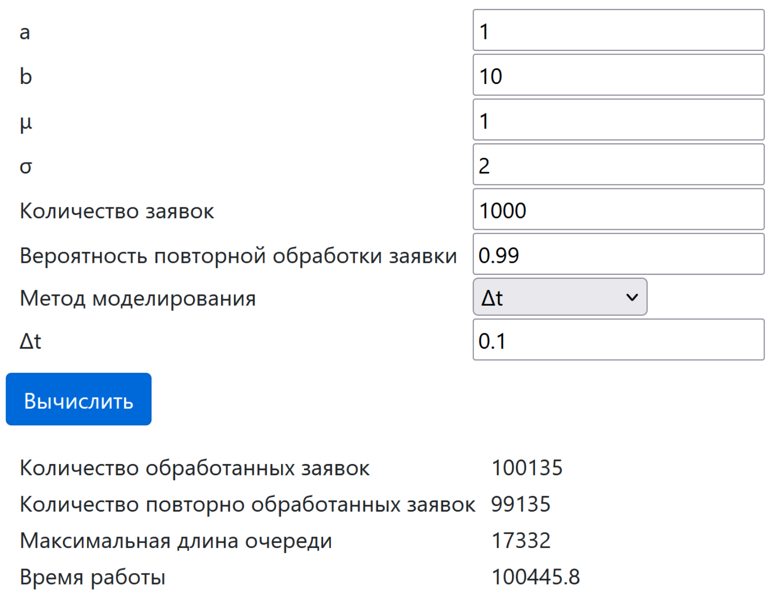

Рис. 6: Пошаговый подход, вероятность возврата заявки в очередь равна 0.99

4 Листинг кода

```
class Uniform:
    def __init__(self, a, b):
        if not 0 <= a <= b:
            raise ex.ParameterError("Parameters must be 0 <= a <= b")
        self._a = a
        self._b = b

    def generate(self):
        return nr.uniform(self._a, self._b)


class Normal:
    def __init__(self, mu, sigma):
        self._mu = mu
        self._sigma = sigma

    def generate(self):
        return nr.normal(self._mu, self._sigma)

class Generator:
    def __init__(self, generator):
        self._generator = generator
        self._receivers = set()

    def add_receiver(self, receiver):
        self._receivers.add(receiver)

    def remove_receiver(self, receiver):
        try:
            self._receivers.remove(receiver)
        except KeyError:
            pass

    def next_time(self):
        return self._generator.generate()

    def emit_request(self):
```

```
        for receiver in self._receivers:
            receiver.receive_request()

class Modeller:
    def __init__(self, uniform_a, uniform_b, n_mu, n_sigma, reenter_prop):
        self._generator = Generator(Uniform(uniform_a, uniform_b))
        self._processor = Processor(Normal(n_mu, n_sigma), reenter_prop)
        self._generator.add_receiver(self._processor)

    def event_based_modelling(self, request_count):
        generator = self._generator
        processor = self._processor

    gen_period = generator.next_time()
    proc_period = gen_period + processor.next_time()
    while processor.processed_requests < request_count:
        if gen_period <= proc_period:
            generator.emit_request()
            gen_period += generator.next_time()
        if gen_period >= proc_period:
            processor.process()
            if processor.current_queue_size > 0:
                proc_period += processor.next_time()
            else:
                proc_period = gen_period + processor.next_time()

    return (processor.processed_requests, processor.reentered_requests,
            processor.max_queue_size, round(proc_period, 3))

    def time_based_modelling(self, request_count, dt):
        generator = self._generator
        processor = self._processor

        gen_period = generator.next_time()
        proc_period = gen_period + processor.next_time()
        current_time = 0
        while processor.processed_requests < request_count:
            if gen_period <= current_time:
                generator.emit_request()
```

```
                gen_period += generator.next_time()
            if current_time >= proc_period:
                processor.process()
                if processor.current_queue_size > 0:
                    proc_period += processor.next_time()
                else:
                    proc_period = gen_period + processor.next_time()
            current_time += dt

        return (processor.processed_requests, processor.reentered_requests,
                processor.max_queue_size, round(current_time, 3))
```

Листинг 1: Программная реализация обслуживающего аппарата