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:
Андреев Александр Алексеевич
Группа: ИУ7-74Б

Преподователь:
Рудаков Игорь Владимирович

Москва, 2022 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 Распредление Эрланга

Распределение Эрланга – это гамма-распределение Γ(x | a, b) с параметрома, принимающим лишь целые значения. Здесь оно приводится лишь из-за того, что часто встречается в инженерных приложениях, особенно телефонии.

Плотность распределения n-го порядка представлена в формуле 3.

$$p(x) = \lambda \frac{(\lambda x)^{n-1}}{!(n-1)} e^{-\lambda x}, x >= 0 \quad (3)$$

Функция распределения представлена в формуле 4.

$$F(x) = \int_0^x \lambda \frac{(\lambda t)^{n-1}}{!(n-1)} e^{-\lambda t} dt \quad (4)$$

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 использую 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

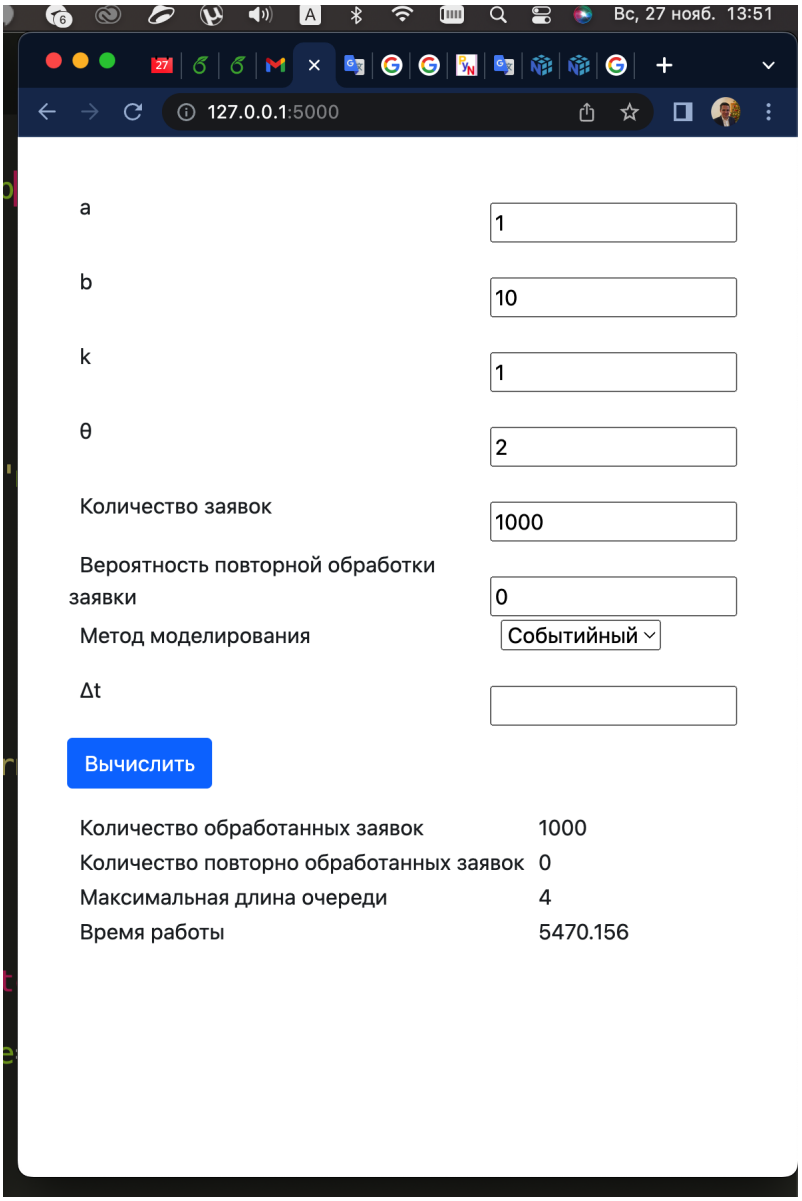

Рис. 1: Событийный подход, вероятность возврата заявки в очередь равна 0

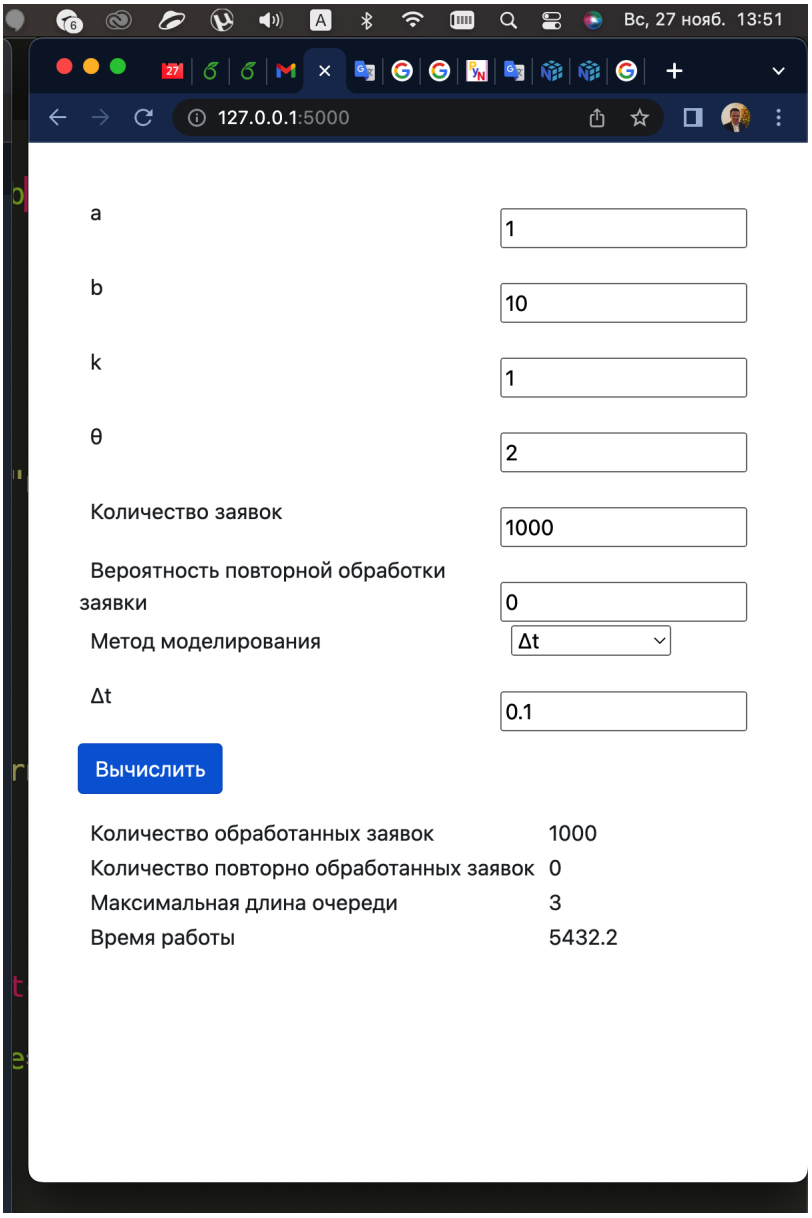

Рис. 2: Пошаговый подход, вероятность возврата заявки в очередь равна 0

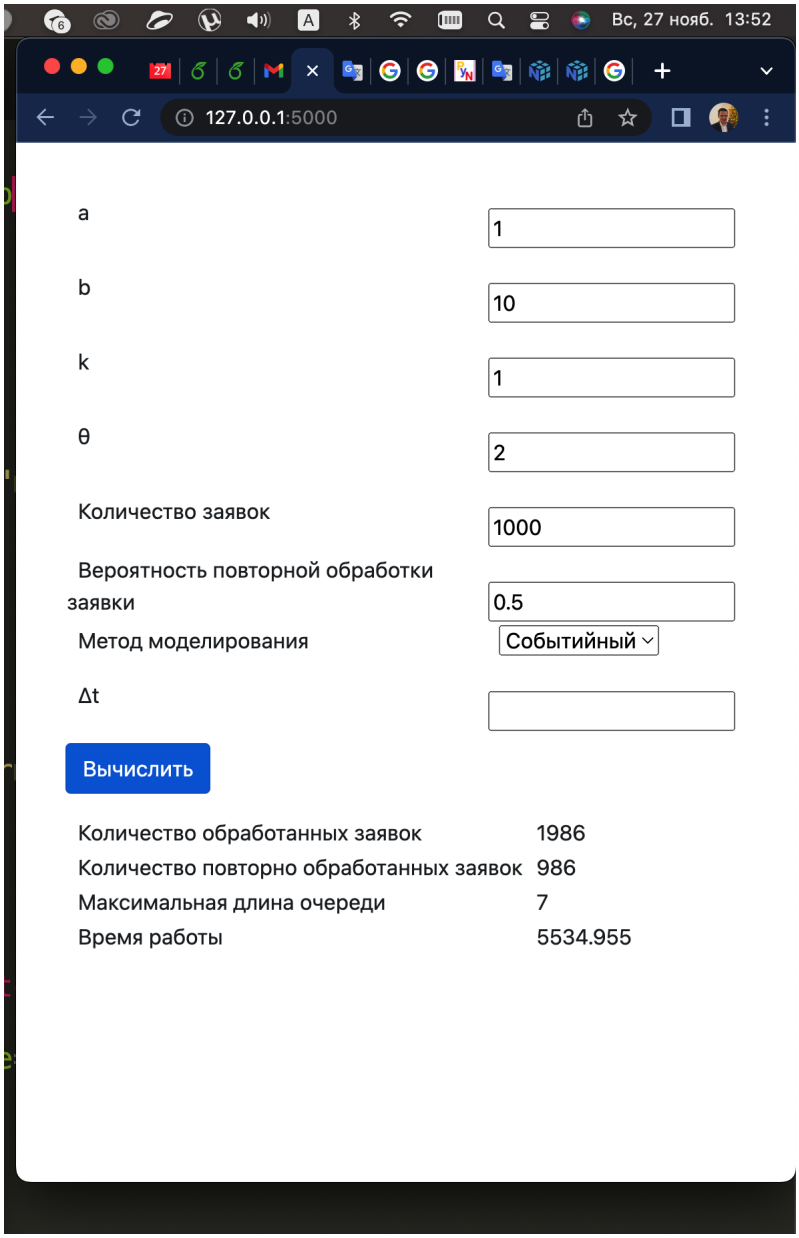

Рис. 3: Событийный подход, вероятность возврата заявки в очередь равна 0.5

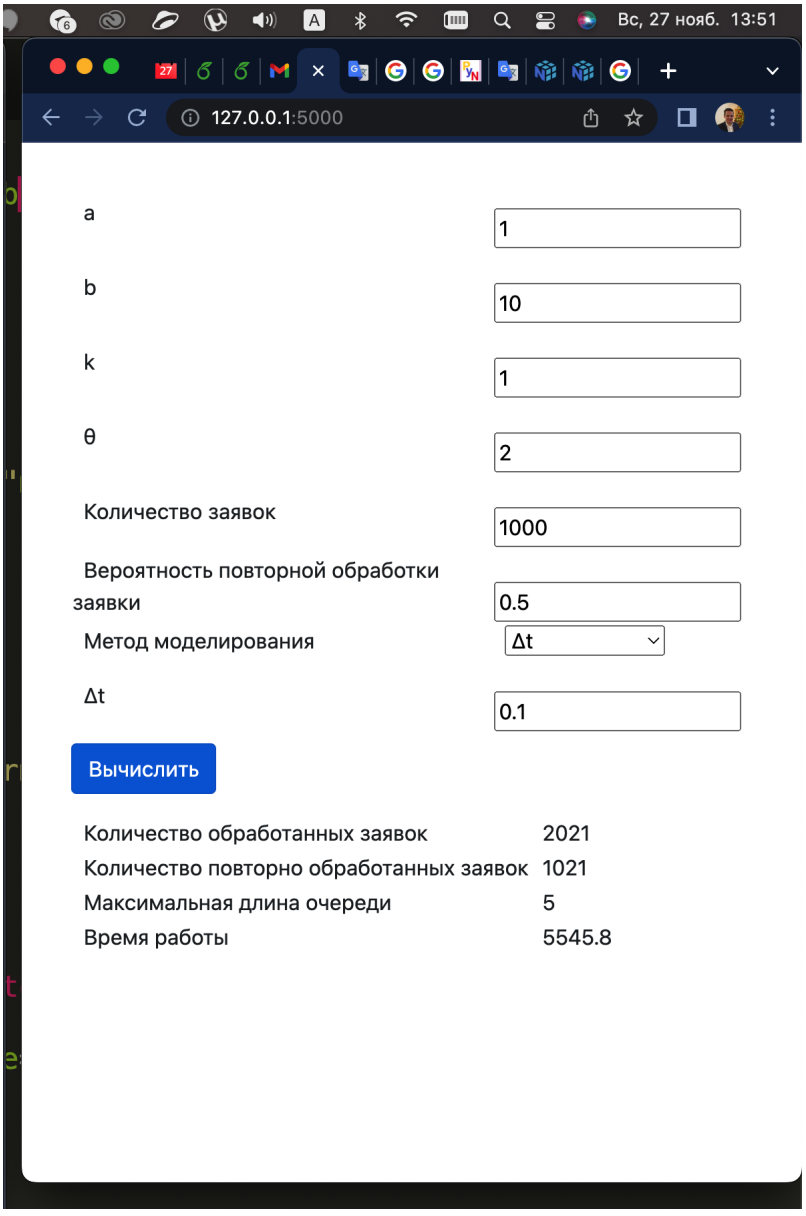

Рис. 4: Пошаговый подход, вероятность возврата заявки в очередь равна 0.5

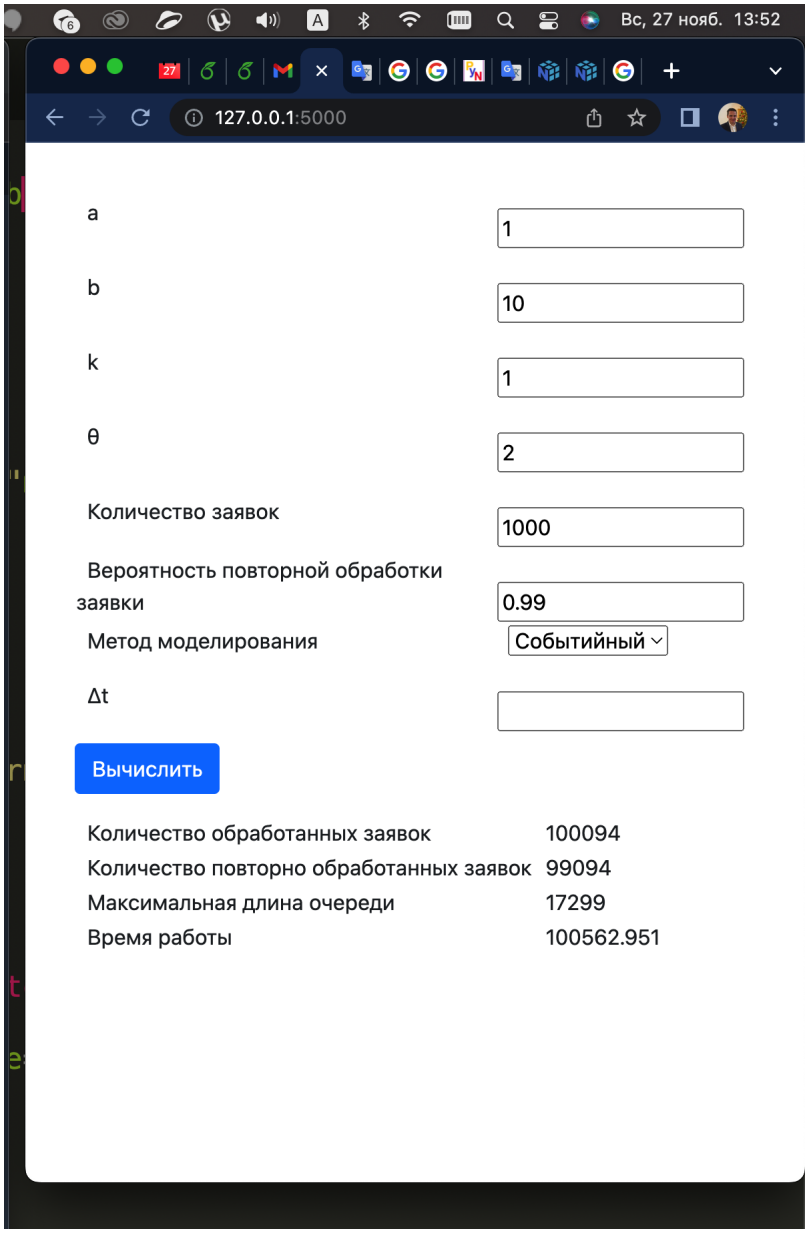

Рис. 5: Событийный подход, вероятность возврата заявки в очередь равна 0.99

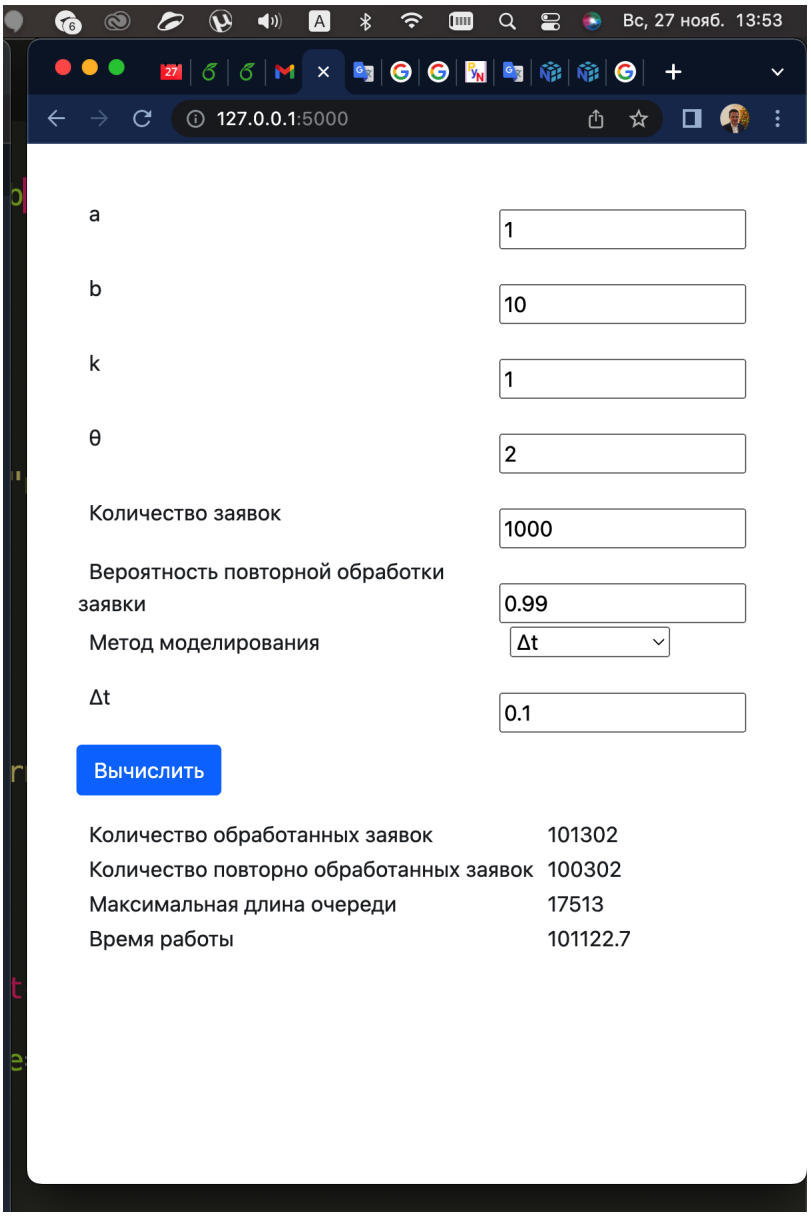

Рис. 6: Пошаговый подход, вероятность возврата заявки в очередь равна 0.99

4 Листинг кода

```
from numpy import random as nr
from . import exceptions as ex

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 Erlang:
    def __init__(self, shape, scale, reenter_prop):
        self._shape = int(shape)
        self._scale = int(scale)
        self._reenter_prop = int(reenter_prop)

    def generate(self):
        print(nr.gamma(self._shape, self._scale, self._reenter_prop))
        return nr.gamma(self._shape, self._scale, self._reenter_prop)
```

Листинг 1: Реализация сущностей равномерного распределения и Эрланга

```
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
```

```
14
15      def next_time(self):
16          return self._generator.generate()
17
18      def emit_request(self):
19          for receiver in self._receivers:
20              receiver.receive_request()
```

Листинг 2: Реализация генератора

```
1  from .distribution import Uniform, Erlang
2  from .generator import Generator
3  from .processor import Processor
4
5  class Modeller:
6      def __init__(self, uniform_a, uniform_b, g_shape, g_scale, reenter_prop):
7          self._generator = Generator(Uniform(uniform_a, uniform_b))
8          self._processor = Processor(Erlang(g_shape, g_scale, reenter_prop), reent
9          self._generator.add_receiver(self._processor)
10
11     def event_based_modelling(self, request_count):
12         generator = self._generator
13         processor = self._processor
14
15         gen_period = generator.next_time()
16         proc_period = gen_period + processor.next_time()
17         while processor.processed_requests < request_count:
18             if gen_period <= proc_period:
19                 generator.emit_request()
20                 gen_period += generator.next_time()
21             if gen_period >= proc_period:
22                 processor.process()
23                 if processor.current_queue_size > 0:
24                     proc_period += processor.next_time()
25                 else:
26                     proc_period = gen_period + processor.next_time()
27
28         return (processor.processed_requests, processor.reentered_requests,
29                 processor.max_queue_size, round(proc_period, 3))
30
```

```
    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 3: Реализация сущности моделирования обслуживающего аппарата

```
from numpy import random as nr
from .generator import Generator


class Processor(Generator):
    def __init__(self, generator, reenter_probability=0):
        super().__init__(generator)
        self._current_queue_size = 0
        self._max_queue_size = 0
        self._processed_requests = 0
        self._reenter_probability = reenter_probability
        self._reentered_requests = 0

    def process(self):
        if self._current_queue_size > 0:
            self._processed_requests += 1
```

```
            self._current_queue_size -= 1
            self.emit_request()
            if nr.random_sample() <= self._reenter_probability:
                self._reentered_requests += 1
                self._processed_requests -= 1
                self.receive_request()

    def receive_request(self):
        self._current_queue_size += 1
        if self._current_queue_size > self._max_queue_size:
            self._max_queue_size = self._current_queue_size

    @property
    def processed_requests(self):
        return self._processed_requests

    @property
    def max_queue_size(self):
        return self._max_queue_size

    @property
    def current_queue_size(self):
        return self._current_queue_size

    @property
    def reentered_requests(self):
        return self._reentered_requests
```

Листинг 4: Реализация Processor