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»

на тему: «Обслуживающий аппарат»

Студент ИУ7-72Б _____ Т. М. Сучкова
(Группа) (Подпись, дата) (И. О. Фамилия)

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

2022 г.

1 Задание

Промоделировать систему из генератора, памяти и обслуживающего аппарата следующими принципами.

1. Принцип Δt.
2. Событийный принцип.

Определить минимальный размер буферной памяти (т.е. максимальный размер очереди), при котором не будет потерянных сообщений.

Используются следующие законы распределения.

1. Равномерное распределение — для генерации сообщений.
2. Распределение Пуассона (из ЛР1) — для обслуживающего аппарата.

Предусмотреть возвращение части обработанных сообщений обратно в очередь (указывается в процентах).

2 Теоретическая часть

2.1 Распределения

2.1.1 Равномерное распределение

Случайная величина X имеет равномерное распределение на отрезке $[a; b]$, если ее функция плотности имеет вид

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{иначе} \end{cases} \tag{2.1}$$

Обозначение: $X \sim R[a, b]$.

Функция равномерного распределения имеет следующий вид.

$$F(x) = \begin{cases} 0, & a < x \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ 1, & x > b \end{cases} \tag{2.2}$$

2.1.2 Распределение Пуассона

Случайная дискретная величина X распределена по закону Пуассона с параметром $\lambda > 0$, если она принимает значения 0, 1, 2, ... с вероятностями

$$P\{X = k\} = \frac{\lambda^k}{k!} * e^{-\lambda}, k \in 0, 1, 2, ..., \tag{2.3}$$

где

- k – количество событий,
- λ – математическое ожидание случайной величины.

Обозначение: $X \sim \Pi(\lambda)$.

Функция плотности распределения имеет вид:

$$P\{x = k\} = \frac{\lambda^k}{k!} * e^{-\lambda}, k \in 0, 1, 2, ... \tag{2.4}$$

Тогда соответствующая функция распределения имеет следующий вид.

$$F(x) = P\{X < x\}, X \sim \Pi(\lambda) \tag{2.5}$$

2.2 Принципы

2.2.1 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.

Основной недостаток: значительные затраты вычислительных ресурсов при моделировании системы. А также при недостаточно малом Δt появляется опасность пропуска отдельных событий в системе, исключающая возможность получения правильных результатов.

2.2.2 Событийный принцип

Состояние отдельных устройств изменяется в отдельные моменты времени, совпадающие с моментами времени поступления сообщений в систему, окончания реализации задачи/процесса, возникновения прерываний, возникновения аварийных сигналов. Следовательно, моделирование и продвижение текущего времени в системе удобно проводить, используя событийный принцип.

При использовании данного принципа состояние **всех** блоков иммитационной модели анализируется лишь в момент появления какого-либо события. Момент поступления следующего события определяются минимальным значением из списка будущих событий, представляющего собой совокупность моментов ближайшего изменения состояний каждого из блоков системы.

3 Результат

Следующие параметры были установлены перед запуском тестов.

1. Равномерное распределение:
   - $a = 1$;
   - $b = 10$.
2. Распределение Пуассона:
   - $\lambda = 6$.
3. Количество заявок: 1000.
4. Шаг: 0.01 .

Результаты работы представлены на рисунках 3.1 — 3.5.

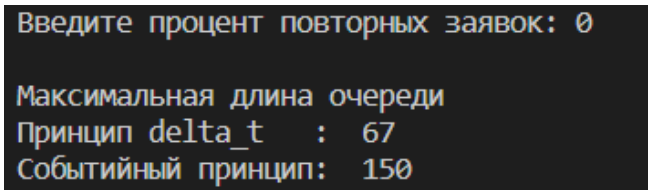

Рисунок 3.1 – Пример работы программы без повторов

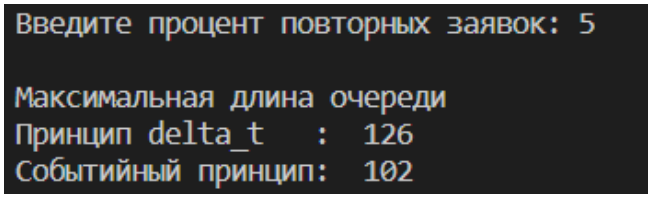

Рисунок 3.2 – Пример работы программы с 5% повторов

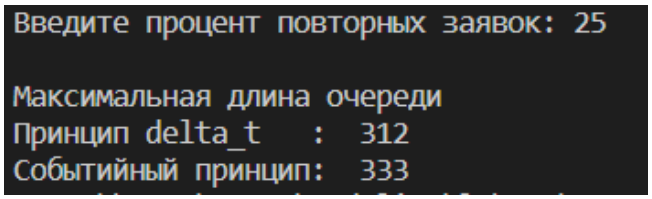

Рисунок 3.3 – Пример работы программы с 25% повторов

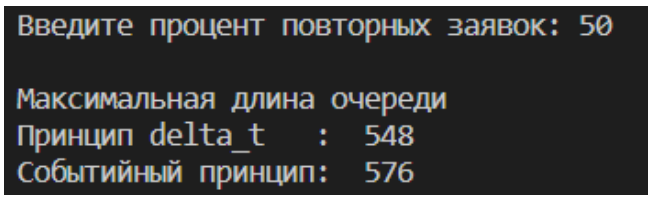

Рисунок 3.4 – Пример работы программы с 50% повторов

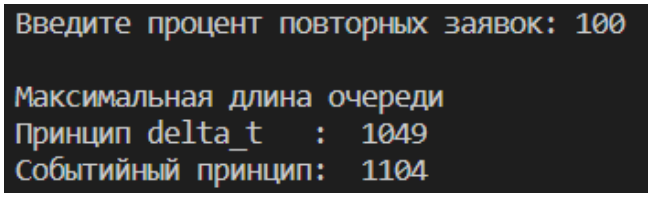

Рисунок 3.5 – Пример работы программы с 100% повторов

4 Код программы

В листингах 4.1–4.3 представлен основной код программы.

Листинг 4.1 – Классы распределений

```
import numpy.random as nprand
import random as rand

class UniformDistribution:
    def __init__(self, a: float, b: float):
        self.a = a
        self.b = b

    def generate(self):
        return self.a + (self.b - self.a) * rand.random()


class PoissonDistribution:
    def __init__(self, lam : float):
        self.lam = lam

    def generate(self):
        return nprand.poisson(self.lam)
```

Листинг 4.2 – Принцип Δt

```
from random import randint

def deltaT_model(generator, processor, total_tasks=0, repeat=0,
   step=0.001):
    done_tasks = 0
    t_curr = step
    t_gen = generator.generate()
    t_gen_prev = t_proc = 0
    cur_qlen = max_qlen = 0
    free = True

    while done_tasks < total_tasks:
        # Генератор
        if t_curr > t_gen:
            cur_qlen += 1
            if cur_qlen > max_qlen:
                max_qlen = cur_qlen
```

```
17
18              t_gen_prev = t_gen
19              t_gen += generator.generate()
20
21          # Обработчик
22          if t_curr > t_proc:
23              if cur_qlen > 0:
24                  was_free = free
25                  if free:
26                      free = False
27                  else:
28                      done_tasks += 1
29                      if randint(1, 100) <= repeat:
30                          cur_qlen += 1
31
32                  cur_qlen -= 1
33                  if was_free:
34                      t_proc = t_gen_prev + processor.generate()
35                  else:
36                      t_proc += processor.generate()
37              else:
38                  free = True
39
40          t_curr += step
41
42      return max_qlen
```

Листинг 4.3 – Событийный принцип

```
1  from random import randint
2
3  def event_model(generator, processor, total_tasks=0, repeat=0):
4      done_tasks = 0
5      cur_qlen = max_qlen = 0
6      events = [[generator.generate(), 'g']]
7      free, process_flag = True, False
8
9      while done_tasks < total_tasks:
10         event = events.pop(0)
11
12         # Генератор
13         if event[1] == 'g':
14             cur_qlen += 1
```

```
15              if cur_qlen > max_qlen:
16                  max_qlen = cur_qlen
17
18              add_event(events, [event[0] + generator.generate(), 'g'
                    ])
19
20              if free:
21                  process_flag = True
22
23          # Обработчик
24          elif event[1] == 'p':
25              done_tasks += 1
26              if randint(1, 100) <= repeat:
27                  cur_qlen += 1
28
29              process_flag = True
30
31          if process_flag:
32              if cur_qlen > 0:
33                  cur_qlen -= 1
34                  add_event(events, [event[0] + processor.generate(),
                        'p'])
35                  free = False
36              else:
37                  free = True
38
39              process_flag = False
40
41      return max_qlen
42
43
44  def add_event(events, event: list):
45      i = 0
46
47      while i < len(events) and events[i][0] < event[0]:
48          i += 1
49
50      if 0 < i < len(events):
51          events.insert(i - 1, event)
52      else:
53          events.insert(i, event)
```