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 ИУ7-73Б _____ Миронов Г. А.
(Группа) (Подпись, дата) (И. О. Фамилия)

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

2022 г.

1 Задание

Провести моделирование системы, состоящей из генератора, блока памяти, и обслуживающего аппарата.

Генератор подает сообщения, распределенные по равномерному закону, они приходят в память и выбираются на обработку по закону из ЛР1. Количество заявок конечно и задано. Предусмотреть случай, возвращат обработанной заявки обратно в очередь.

Необходимо определить оптимальную длину очереди, при которой в системе не будет потерянных сообщений. Реализовать двумя способами: используя пошаговый и событийный подходы.

2 Теоретическая часть

2.1 Равномерное распределение

Говорят, что случайная величина X имеет равномерное распределение на отрезке $[a; b]$, если её функция плотности имеет вид

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

Обозначается $X \sim R[a, b]$.

Соответствующая функция распределения:

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

2.2 Распределение Пуассона

Говорят, что случайная величина X распределена по закону Пуассона с параметром $\lambda > 0$, если она принимает значения $0, 1, 2, \ldots$ с вероятностями

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

Обозначается $X \sim \Pi(\lambda)$.

Функция плотности распределения имеет вид:

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

Тогда соответствующая функция распределения имеет вид:

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

2.3 Формализация задачи

2.3.1 Δt модель

Данная модельн заключается в последовательном анализе состояний всех блоков системы в момент времени $t + \Delta t$. Новое состояние определяется в соответствии с их алгоритмическим описанием с учетом действия случайных факторов. В результате этого анализа принимается решение о том, какие системные события должны имитироваться на данный момент времени.

Основной недостаток модели: значительные затраты и вероятность пропуска события при больших Δt.

2.3.2 Событийная модель

В данной модели состояния отдельных устройств изменяются в дискретные моменты времени. При использовании событийного принципа, состояния всех блоков системы анализируются лишь в момент возникновения какого либо события. Момент наступления следующего события, определяется минимальным значением из списка событий.

3 Результат работы

3.1 Без повторов, 1000 заявок

Входные данные:

```
a, b = 1, 10

lambda = 4

total_tasks = 1000
repeat_percentage = 0.0
step = 0.01
```

Результаты моделирования:

```
step_model_res = 4
event_model_res = 5
```

На рисурке 3.1 представлена визуализация процесса моделирования системы с заданными параметрами.

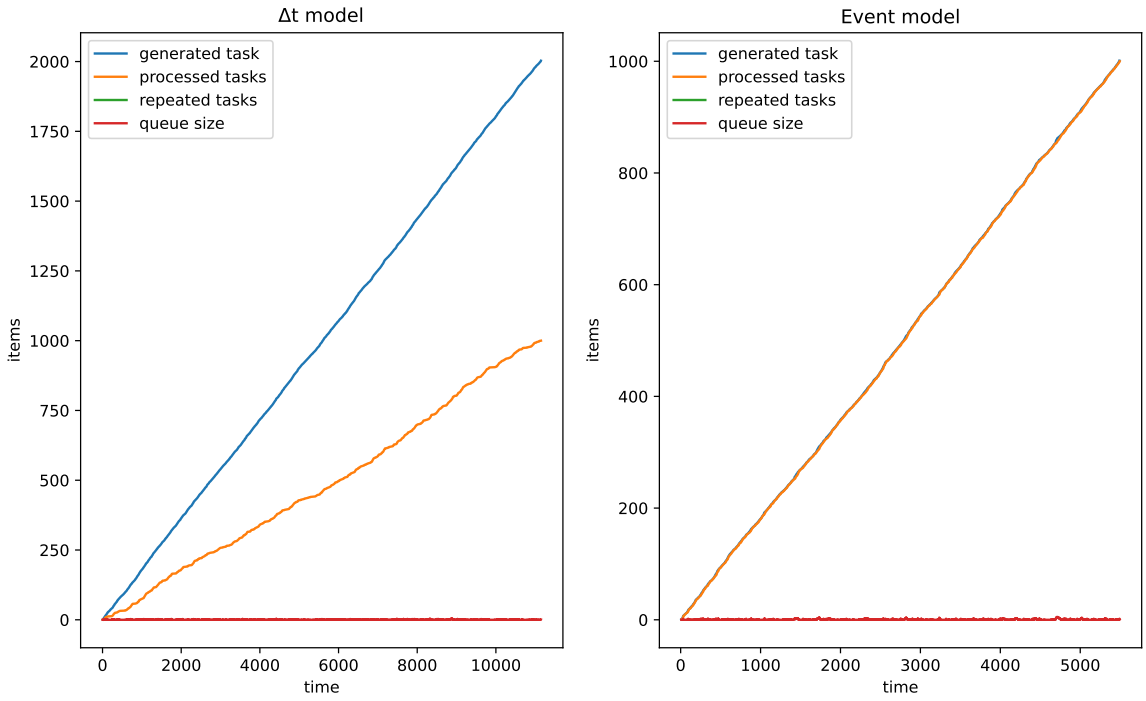

Рисунок 3.1 – Визуализация процесса моделирования системы без повторов, 1000 заявок

3.2 10% повторов, 1000 заявок

Входные данные:

```
a, b = 1, 10

lambda = 4

total_tasks = 1000
repeat_percentage = 0.1
step = 0.01
```

Результаты моделирования:

```
step_model_res = 5
event_model_res = 7
```

На рисурке 3.2 представлена визуализация процесса моделирования системы с заданными параметрами.

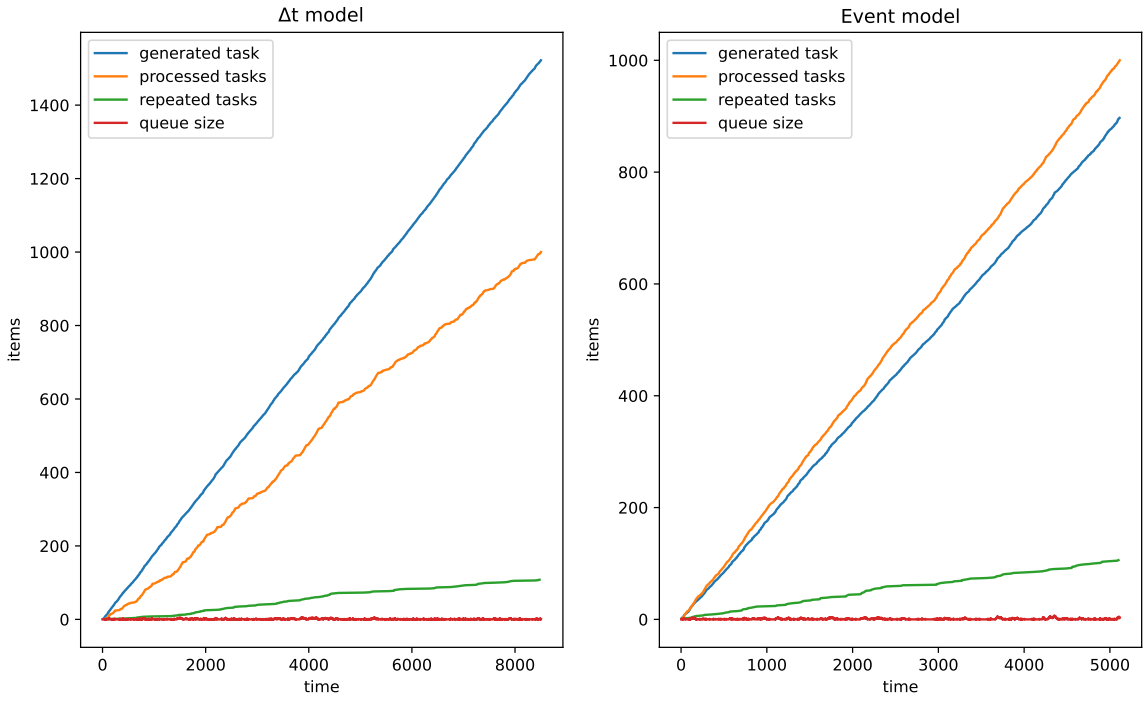

Рисунок 3.2 – Визуализация процесса моделирования системы 10% повторов, 1000 заявок

3.3 25% повторов, 1000 заявок

Входные данные:

```
a, b = 1, 10

lambda = 4

total_tasks = 1000
repeat_percentage = 0.25
step = 0.01
```

Результаты моделирования:

```
step_model_res = 23
event_model_res = 18
```

На рисурке 3.3 представлена визуализация процесса моделирования системы с заданными параметрами.

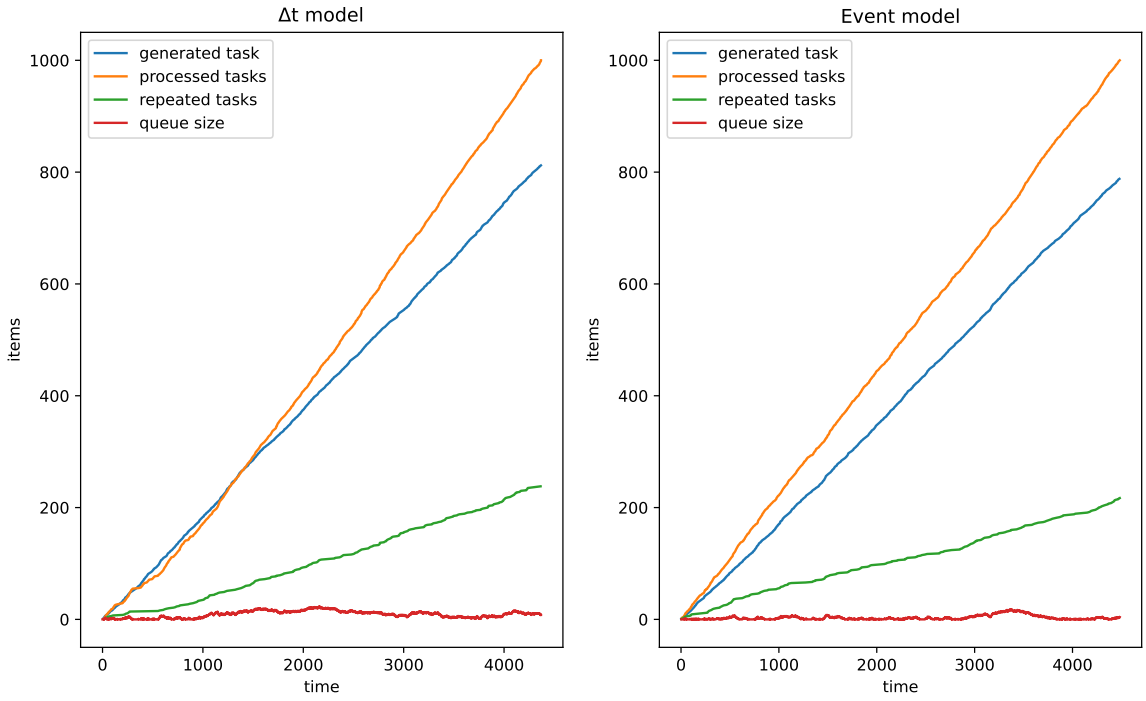

Рисунок 3.3 – Визуализация процесса моделирования системы 25% повторов, 1000 заявок

3.4 25% повторов, 10000 заявок

Входные данные:

```
a, b = 1, 10

lambda = 4

total_tasks = 10000
repeat_percentage = 0.25
step = 0.01
```

Результаты моделирования:

```
step_model_res = 39
event_model_res = 32
```

На рисурке 3.4 представлена визуализация процесса моделирования системы с заданными параметрами.

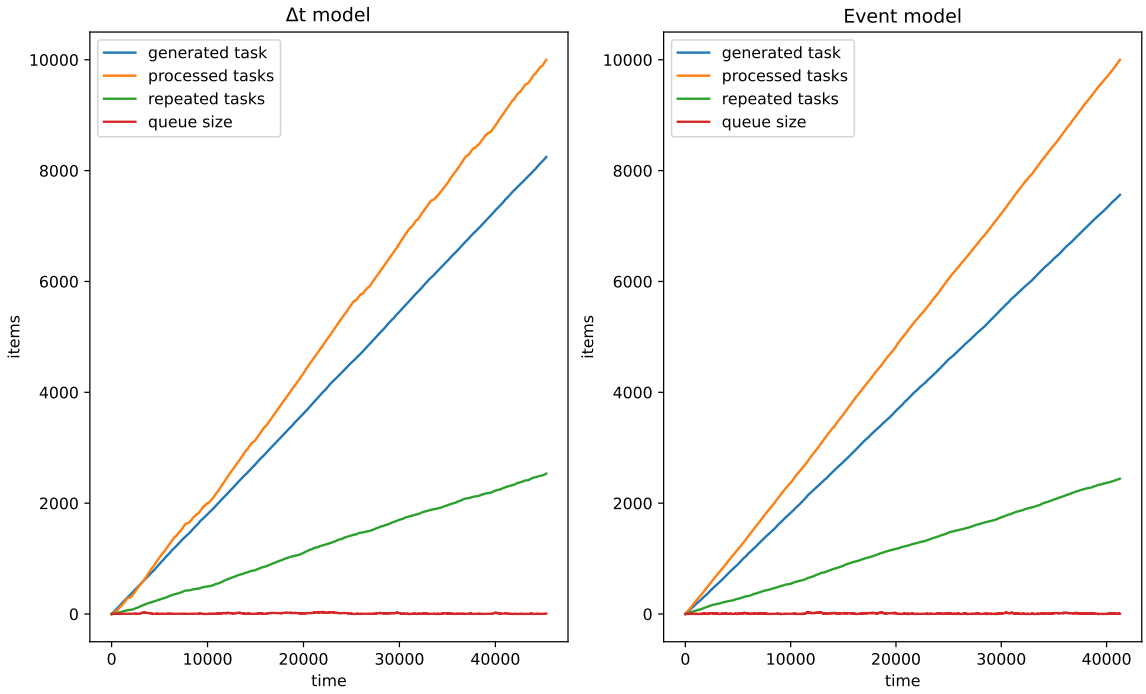

Рисунок 3.4 – Визуализация процесса моделирования системы 25% повторов, 10000 заявок

3.5 50% повторов, 10000 заявок

Входные данные:

```
a, b = 1, 10

lambda = 4

total_tasks = 10000
repeat_percentage = 0.5
step = 0.01
```

Результаты моделирования:

```
step_model_res = 2305
event_model_res = 2211
```

На рисурке 3.5 представлена визуализация процесса моделирования системы с заданными параметрами.

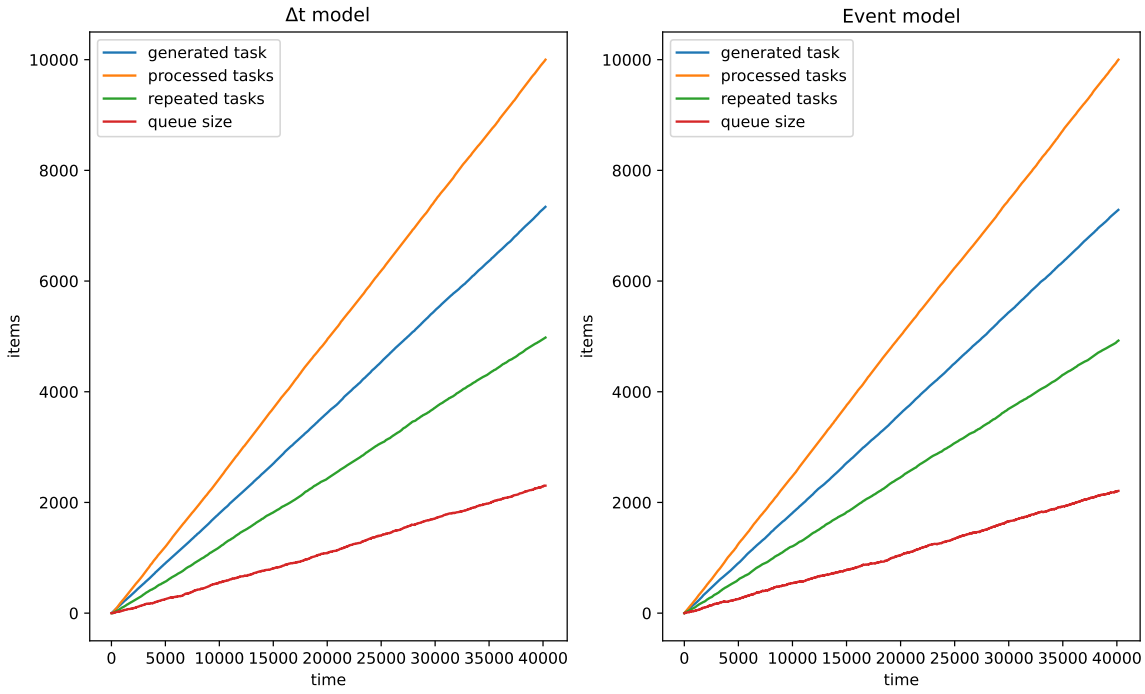

Рисунок 3.5 – Визуализация процесса моделирования системы 50% повторов, 10000 заявок

3.6 100% повторов, 10000 заявок

Входные данные:

```
a, b = 1, 10

lambda = 4

total_tasks = 10000
repeat_percentage = 1.0
step = 0.01
```

Результаты моделирования:

```
step_model_res = 7243
event_model_res = 7217
```

На рисурке 3.6 представлена визуализация процесса моделирования системы с заданными параметрами.

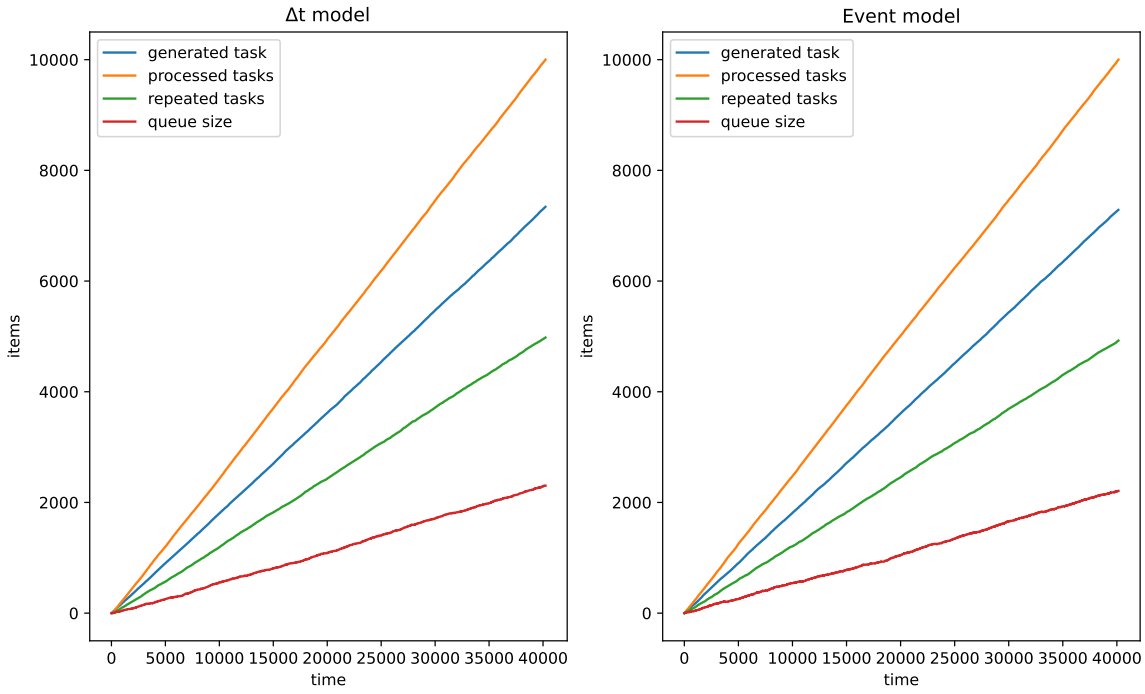

Рисунок 3.6 – Визуализация процесса моделирования системы 100% повторов, 10000 заявок

3.7 Итоговая таблица сравения

Полученные в ходе эксперимента данные, представлены в таблице 3.1.

Таблица 3.1 – Сводные данные по проведенным моделированиям системы и их параметрам

Кол-во сообщений	**Процент повторов**	Шаг Δt	**Равномерное**		**Пуассона**	**Размер очереди**	
			a	**b**	λ	Δt	События
1000	0	0.1	1	10	4	4	5
1000	10	0.1	1	10	4	5	7
1000	25	0.1	1	10	4	23	18
10000	25	0.1	1	10	4	39	32
10000	50	0.1	1	10	4	2305	2211
10000	100	0.1	1	10	4	7243	7217

4 Исходный код программы

Листинг 4.1 – Исходный код программы. Часть 1

```
from typing import Callable, List, Tuple

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

import numba as nb


@nb.experimental.jitclass(
    [
        ("times", nb.types.ListType(nb.types.float64)),
        ("counts", nb.types.ListType(nb.types.int64)),
    ]
)
class TimeSeries:
    def __init__(self):
        self.times = nb.typed.List.empty_list(nb.types.float64)
        self.counts = nb.typed.List.empty_list(nb.types.int64)

    def add(self, at: np.float64) -> None:
        self.put(at, self.counts[-1] + 1 if len(self.counts) > 0
           else 1)

    def put(self, at: np.float64, val: np.int64) -> None:
        self.times.append(at)
        self.counts.append(val)

    def raw(
        self,
    ) -> Tuple[nb.typed.List[nb.types.float64], nb.typed.List[nb.
       types.int64]]:
        return self.times, self.counts


@nb.jit
def step_model(
    generator: Callable[[], np.float64],
    processor: Callable[[], np.float64],
    total_tasks: np.int64 = 0,
    repeat: np.float64 = 0,
    step: np.float64 = 0.001,
):
```

Листинг 4.2 – Исходный код программы. Часть 2

```
43      gen_stats, proc_stats, rep_stats, queue_stats = (
44          TimeSeries(),
45          TimeSeries(),
46          TimeSeries(),
47          TimeSeries(),
48      )
49
50      processed_tasks = np.int64(0)
51      t_curr = np.float64(0)
52      t_gen = generator()
53      t_gen_prev = t_proc = np.int64(0)
54      cur_queue_len = max_queue_len = np.int64(0)
55      free = True
56
57      while processed_tasks < total_tasks:
58          queue_stats.put(t_curr, cur_queue_len)
59
60          # generator
61          if t_curr > t_gen:
62              gen_stats.add(t_gen)
63
64              cur_queue_len += 1
65              max_queue_len = np.maximum(max_queue_len,
                    cur_queue_len)
66              t_gen_prev = t_gen
67              t_gen += generator()
68
69          # processor
70          if t_curr > t_proc:
71              if cur_queue_len <= 0:
72                  free = True
73              else:
74                  was_free = free
75                  if free:
76                      free = False
77                  else:
78                      proc_stats.add(t_proc)
79
80                      processed_tasks += 1
81                      if np.random.random() <= repeat:
82                          rep_stats.add(t_proc)
83                          cur_queue_len += 1
84                  cur_queue_len -= 1
```

Листинг 4.3 – Исходный код программы. Часть 3

```
                t_proc = t_gen_prev + processor() if was_free
                   else t_proc + processor()

        t_curr += step

    return (
        max_queue_len,
        gen_stats.raw(),
        proc_stats.raw(),
        rep_stats.raw(),
        queue_stats.raw(),
    )


@nb.jit
def add_event(events_time, events_type, time, ttype):
    i = 0
    while i < len(events_time) and events_time[i] < time:
        i += 1
    if 0 < i < len(events_time):
        events_time.insert(i - 1, time)
        events_type.insert(i - 1, ttype)
    else:
        events_time.insert(i, time)
        events_type.insert(i, ttype)


@nb.jit
def event_model(generator_rand, processor_rand, total_tasks=0,
   repeat=0):
    gen_stats, proc_stats, rep_stats, queue_stats = (
        TimeSeries(),
        TimeSeries(),
        TimeSeries(),
        TimeSeries(),
    )

    processed_tasks = 0
    cur_queue_len = max_queue_len = 0

    events_time = nb.typed.List.empty_list(nb.types.float64)
    events_type = nb.typed.List.empty_list(nb.types.int64)
    events_time.append(generator_rand())
    events_type.append(0)

    free, process_flag = True, False
```

Листинг 4.4 – Исходный код программы. Часть 4

```
130
131     while processed_tasks < total_tasks:
132         event_time = events_time.pop(0)
133         event_type = events_type.pop(0)
134
135         queue_stats.put(event_time, cur_queue_len)
136
137         # generator
138         if event_type == 0:
139             gen_stats.add(event_time)
140             cur_queue_len += 1
141             if cur_queue_len > max_queue_len:
142                 max_queue_len = cur_queue_len
143
144             add_event(events_time, events_type, event_time +
                    generator_rand(), 0)
145
146             if free:
147                 process_flag = True
148
149         # processor
150         elif event_type == 1:
151             proc_stats.add(event_time)
152             processed_tasks += 1
153             if np.random.random() < repeat:
154                 rep_stats.add(event_time)
155                 cur_queue_len += 1
156
157             process_flag = True
158
159         if process_flag:
160             if cur_queue_len > 0:
161                 cur_queue_len -= 1
162                 add_event(events_time, events_type, event_time +
                        processor_rand(), 1)
163                 free = False
164             else:
165                 free = True
166             process_flag = False
167
168     return (
169         max_queue_len,
170         gen_stats.raw(),
171         proc_stats.raw(),
172         rep_stats.raw(),
173         queue_stats.raw(),
174     )
```

Листинг 4.5 – Исходный код программы. Часть 5

```
175
176
177 g = nb.jit(lambda: np.random.uniform(1, 10))
178
179 p = nb.jit(lambda: np.random.poisson(4))
180
181
182 def main():
183     total_tasks = 1000
184     repeat_percentage = 0.25
185     step = 0.01
186
187     (
188         step_model_res,
189         step_gen_stats,
190         step_proc_stats,
191         step_rep_stats,
192         step_queue_stats,
193     ) = step_model(g, p, total_tasks, repeat_percentage, step)
194
195     print("step_model:", step_model_res)
196
197     (
198         event_model_res,
199         event_gen_stats,
200         event_proc_stats,
201         event_rep_stats,
202         event_queue_stats,
203     ) = event_model(g, p, total_tasks, repeat_percentage)
204
205     print("event_model:", event_model_res)
206
207     _, axis = plt.subplots(1, 2, figsize=(12, 7))
208
209     axis[0].set_title("$\Delta$t␣model")
210     axis[0].plot(*step_gen_stats, label="кол-во␣сгенерированных␣
        заявок")
211     axis[0].plot(*step_proc_stats, label="кол-во␣обработанных␣
        заявок")
212     axis[0].plot(*step_rep_stats, label="кол-во␣повторных␣заявок"
        )
213     axis[0].plot(*step_queue_stats, label="размер␣очереди")
214     axis[0].set_xlabel("время")
215     axis[0].set_ylabel("штук")
216     axis[0].legend()
```

Листинг 4.6 – Исходный код программы. Часть 6

```
218     axis[1].set_title("Event␣model")
219     axis[1].plot(*event_gen_stats, label="кол-во␣сгенерированных␣
           заявок")
220     axis[1].plot(*event_proc_stats, label="кол-во␣обработанных␣
           заявок")
221     axis[1].plot(*event_rep_stats, label="кол-во␣повторных␣заявок
           ")
222     axis[1].plot(*event_queue_stats, label="размер␣очереди")
223     axis[1].set_xlabel("время")
224     axis[1].set_ylabel("штук")
225     axis[1].legend()
226 
227     plt.show()
228 
229 
230 if __name__ == "__main__":
231     main()
```