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 №6

по курсу «Моделирование»

на тему: «Определение вероятности отказа»

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

В пункт выдачи заказов приходят клиенты через промежутке времени $[4; 20]$ минут. Если все три имеющихся оператора заняты, клиенту отказывают в обслуживании. Операторы имеют разную производительность и могут обеспечивать обслуживание среднего запроса пользователя за 3 ± 2, 5 ± 1 и 10 ± 3 минут, соответственно.

Клиенты стремятся занять свободного оператора с максимальной производительностью. Полученные запросы сдаются в накопитель, откуда выбираются на обработку. На первый компьютер запросы от 1-ого операторов, на второй – запросы от 2-ого, на третий – запросы от 3-его. Так же есть четвертый компьютер, на который оператор передает запросы, если тот менее загружен. Время обработки запросов первыми треям компьютерами равны 20 ± 2 мин., время работы четвертого компьютера равно 25 ± 1 мин. Промоделировать процесс обработки 300 запросов.

2 Теоретическая часть

2.1 Общий вид системы

На рисунке 2.1 представлена концептуальная модель моделируемой системы в общем виде.

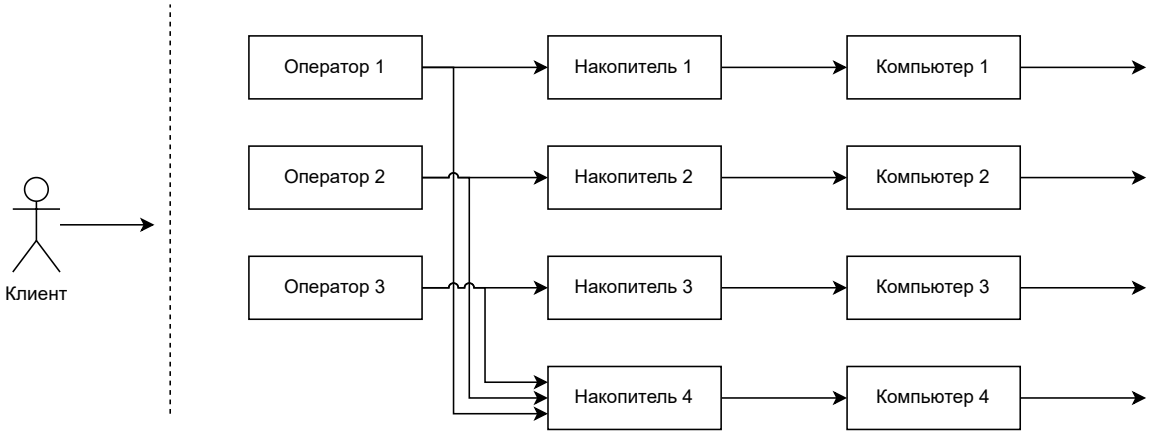

Рисунок 2.1 – Концептуальная модель в общем виде

2.2 Концептуальная модель системы в терминах СМО

На рисунке 2.2 представлена концептуальная модель моделируемой системы в терминах СМО.

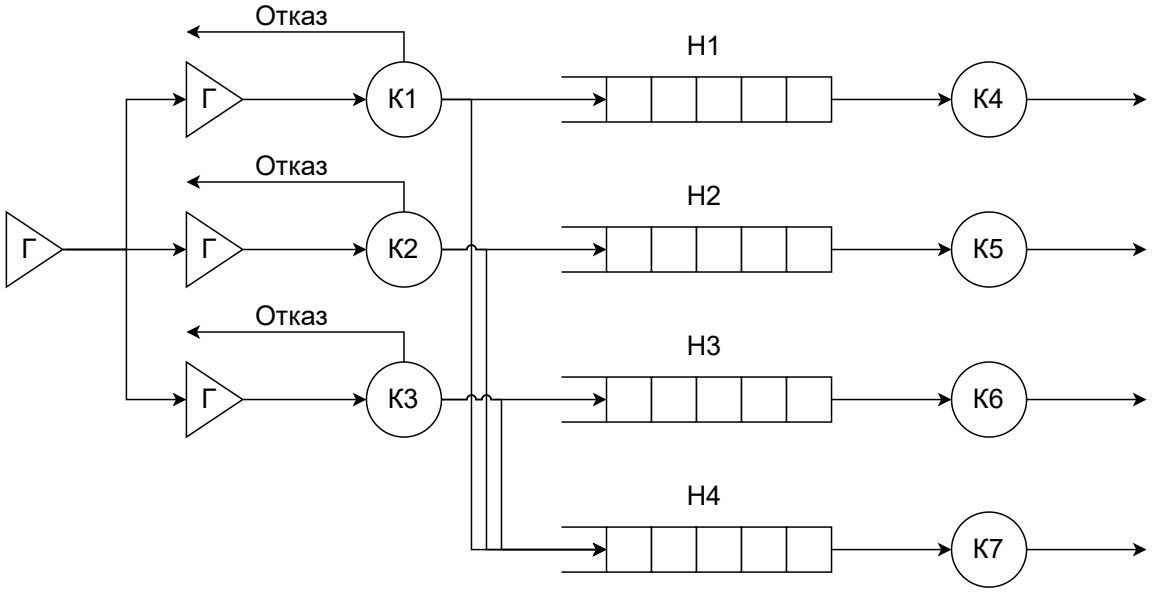

Рисунок 2.2 – Концептуальная модель в терминах СМО

В процессе взаимодействия клиентов с информационным центром возможны:

- режим нормального обслуживания, т.е. клиент выбирает одного из свободных операторов, отдавая предпочтение тому у которого меньше номер;
- режим отказа в обслуживании клиента, когда все операторы заняты.

2.3 Переменные и уравнения имитационной модели

Эндогенные переменные:

- время обработки задания i-ым оператором;
- время решения этого задания j-ым компьютером.

Экзогенные переменные:

- число обслуженных клиентов;
- число клиентов, получивших отказ.

Вероятность отказа в обслуживании клиента:

$$\frac{C_{\text{отк}}}{C_{\text{отк}} + C_{\text{обсл}}}$$

где $C_{\text{отк}}$ – количество заявок которым отказали в обслуживании, а $C_{\text{обсл}}$ – количество обслуженных заявок.

3 Результат работы

3.1 300 заявок

```
Processing time (seconds):  0.9656
Total requests:  300
Processed requests:  293
Canceled requests:  7
Cancelation percent:  0.0233
```

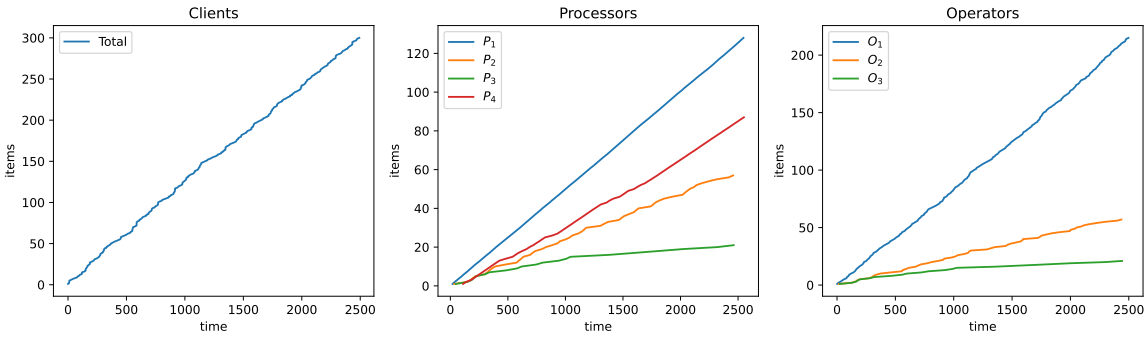

Рисунок 3.1 – Результаты моделирования для 300 клиентов

3.2 1000 заявок

```
Processing time (seconds):  3.249
Total requests:  1000
Processed requests:  977
Canceled requests:  23
Cancelation percent:  0.023
```

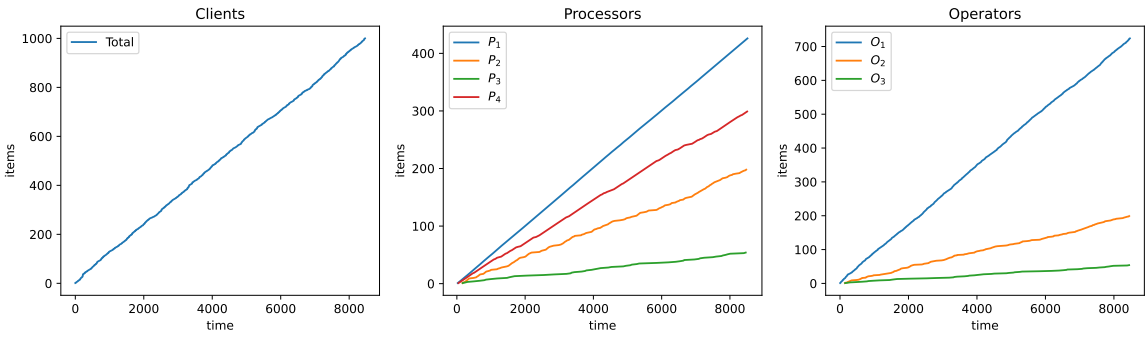

Рисунок 3.2 – Результаты моделирования для 1000 клиентов

3.3 3000 заявок

```
Processing time (seconds):  9.8286
Total requests:  3000
Processed requests:  2932
Canceled requests:  68
Cancelation percent:  0.0227
```

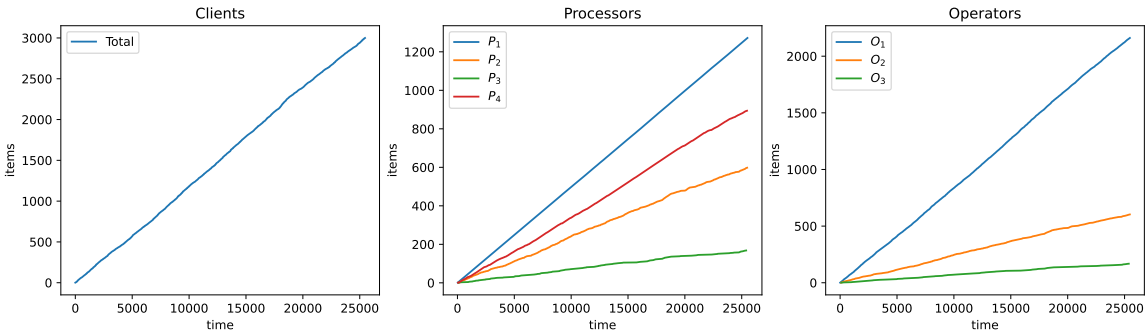

Рисунок 3.3 – Результаты моделирования для 3000 клиентов

4 Исходный код программы

Листинг 4.1 – Исходный код программы. Часть 1

```
from time import time

import numpy as np

import matplotlib.pyplot as plt

TIME_DELTA = 0.01
FINISH_PROCESS_REQUEST = 1
CURRENT_REQUEST = 0
DONT_HAVE_FREE_OPERATOR = -1


class TimeSeries:
    def __init__(self, step: np.float64 = TIME_DELTA):
        self.time = 0
        self.step = step
        self.times = []
        self.counts = []

    def update_time(self):
        self.time += self.step

    def add(self) -> None:
        self.put(self.time, 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):
        return self.times, self.counts


def distribution(a, b):
    return lambda: np.random.uniform(a - b, a + b)


class ClientGenerator:
    def __init__(self, time_distribution):
        self.time_distribution = time_distribution
        self.remaining_time = 0
        self.stat = TimeSeries()
```

Листинг 4.2 – Исходный код программы. Часть 2

```
43
44      def update_time(self):
45          self.stat.update_time()
46          if self.remaining_time > 0:
47              self.remaining_time -= TIME_DELTA
48
49          if self.remaining_time <= 1e-5:
50              self.remaining_time = self.time_distribution()
51              self.stat.add()
52              return Request()
53
54          return None
55
56
57  class Operator:
58      def __init__(self, recipients, time_distribution):
59          self.time_distribution = time_distribution
60          self.recipients = recipients
61          self.remaining_time = 0
62          self.is_busy = False
63          self.processing_request = None
64          self.stat = TimeSeries()
65
66      def update_time(self):
67          self.stat.update_time()
68          self.remaining_time -= TIME_DELTA
69          if self.is_busy and self.remaining_time <= 1e-5:
70              self.finish_process_request()
71
72      def start_process_new_request(self, request):
73          self.is_busy = True
74          self.processing_request = request
75          self.remaining_time = self.time_distribution()
76
77      def finish_process_request(self):
78          min(self.recipients, key=len).append(self.
               processing_request)
79          self.is_busy = False
80          self.processing_request = None
81          self.stat.add()
82
83
84  class Request:
85      request_id = 0
86
87      def __init__(self):
88          global CURRENT_REQUEST
89          self.request_id = CURRENT_REQUEST
90          CURRENT_REQUEST += 1
```

Листинг 4.3 – Исходный код программы. Часть 3

```
91
92
93  class Processor:
94      def __init__(self, requests_storage, time_distribution):
95          self.requests_storage = requests_storage
96          self.time_distribution = time_distribution
97          self.is_busy = False
98          self.processing_request = None
99          self.remaining_time = 0
100         self.stat = TimeSeries()
101
102     def update_time(self):
103         self.stat.update_time()
104         if self.remaining_time != 0:
105             self.remaining_time -= TIME_DELTA
106
107         if self.is_busy and self.remaining_time <= 1e-5:
108             self.is_busy = False
109             self.processing_request = None
110             self.stat.add()
111             return FINISH_PROCESS_REQUEST
112
113         if not self.is_busy and len(self.requests_storage) != 0:
114             self.processing_request = self.requests_storage.pop
                    (0)
115             self.remaining_time = self.time_distribution()
116             self.is_busy = True
117
118
119 def find_free_operator(operators):
120     return next(
121         (i for i, o in enumerate(operators) if not o.is_busy),
                DONT_HAVE_FREE_OPERATOR
122     )
123
124
125 def iteration(clients, operators, processors, request_info,
        is_new=True):
126     if is_new:
127         request = clients.update_time()
128         if request:
129             request_info["generated_count"] += 1
130             free_operator_number = find_free_operator(operators)
131             if free_operator_number == DONT_HAVE_FREE_OPERATOR:
132                 request_info["lost_count"] += 1
133             else:
134                 operators[free_operator_number].
                        start_process_new_request(request)
```

Листинг 4.4 – Исходный код программы. Часть 4

```
136     for operator in operators:
137         operator.update_time()
138 
139     for processor in processors:
140         result = processor.update_time()
141         if result == FINISH_PROCESS_REQUEST:
142             request_info["processed_count"] += 1
143 
144 
145 def modeling(clients, operators, processors, requests_count):
146     statistics_info = {"generated_count": 0, "processed_count":
          0, "lost_count": 0}
147 
148     while statistics_info["generated_count"] < requests_count:
149         iteration(clients, operators, processors, statistics_info
            )
150 
151     while (
152         statistics_info["lost_count"] + statistics_info["
            processed_count"]
153         < requests_count
154     ):
155         iteration(clients, operators, processors, statistics_info
            , False)
156 
157     return statistics_info
158 
159 
160 if __name__ == "__main__":
161     requests_count = 300
162 
163     clients = ClientGenerator(distribution(8, 12))
164 
165     storage_1, storage_2, storage_3, storage_4 = [], [], [], []
166 
167     processors = [
168         Processor(storage_1, distribution(20, 2)),
169         Processor(storage_2, distribution(20, 2)),
170         Processor(storage_3, distribution(20, 2)),
171         Processor(storage_4, distribution(25, 1)),
172     ]
173 
174     operators = [
175         Operator([storage_1, storage_4], distribution(3, 2)),
176         Operator([storage_2, storage_4], distribution(5, 1)),
177         Operator([storage_3, storage_4], distribution(10, 4)),
```

Листинг 4.5 – Исходный код программы. Часть 5

```
178     ]
179
180     start_time = time()
181
182     result = modeling(clients, operators, processors,
            requests_count)
183
184     print("Processing␣time␣(seconds):␣", round((time() -
            start_time), 4))
185     print("Total␣requests␣time:␣", result["generated_count"])
186     print("Processed␣requests␣time:␣", result["processed_count"])
187     print("Canceled␣requests␣time:␣", result["lost_count"])
188     print("Cancelation␣percent:␣", round((result["lost_count"] /
            requests_count), 4))
189
190     fig, axis = plt.subplots(1, 3, figsize=(16, 4))
191
192     axis[0].set_title("Clients")
193     axis[0].plot(*clients.stat.raw(), label="A")
194     axis[0].set_xlabel("time")
195     axis[0].set_ylabel("items")
196     axis[0].legend()
197
198     axis[1].set_title("Processors")
199     axis[1].set_xlabel("time")
200     axis[1].set_ylabel("items")
201     for i, p in enumerate(processors):
202         axis[1].plot(*p.stat.raw(), label=f"$P_{i+1}$")
203     axis[1].legend()
204
205     axis[2].set_title("Operators")
206     axis[2].set_xlabel("time")
207     axis[2].set_ylabel("items")
208     for i, o in enumerate(operators):
209         axis[2].plot(*o.stat.raw(), label=f"$O_{i+1}$")
210     axis[2].legend()
211
212     plt.show()
```