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 №5

по курсу «Моделирование»

на тему: «Моделирование работы информационного центра»

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

В информационный центр приходят клиенты через интервал времени 10 ± 2 минуты. Если все три имеющихся оператора заняты, клиенту отказывают в обслуживании. Операторы имеют разную производительность и могут обеспечивать обслуживание среднего запроса пользователя за 20 ± 5; 40 ± 10; 40 ± 20. Клиенты стремятся занять свободного оператора с максимальной производительностью. Полученные запросы сдаются в приемный накопитель. Откуда выбираются на обработку. На первый компьютер запросы от 1 и 2-ого операторов, на второй — запросы от 3-его. Время обработки запросов на 1-ом и 2-ом компьютерах равны соответственно 15 и 30 мин.

Смоделировать процесс обработки 300 запросов. Найти вероятность отказа. За единицу системного времени выбрать 0.01 мин.

Также в отчете необходимо сделать следующее.

1. Построить структурную схему модели (3 оператора, 2 накопителя, 2 компьютера). На входе — клиенты, на выходе — результат.

2. Построить СМО модель.

2 Теоретическая часть

Структурная схема модели приведена на рисунке 2.1.

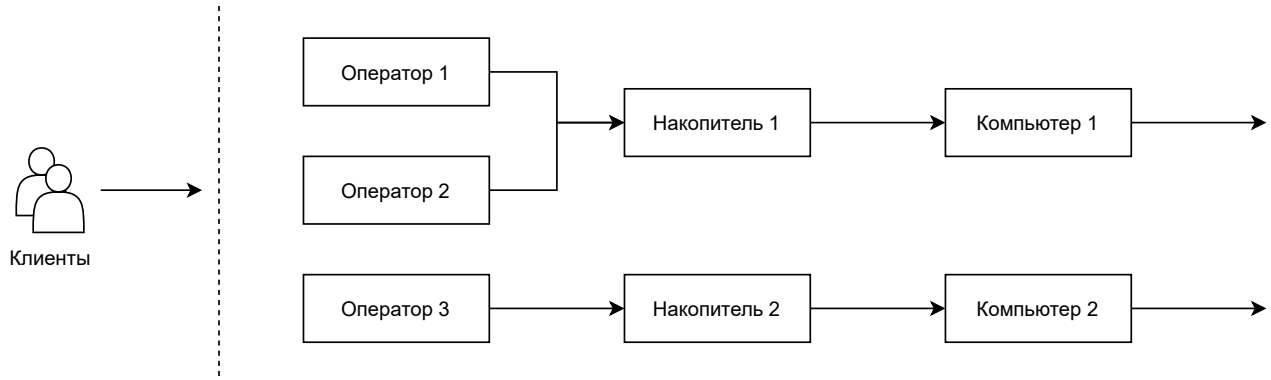


Рисунок 2.1 – труктурная схема модели

Схема модели в терминах СМО приведена на рисунке 2.2.

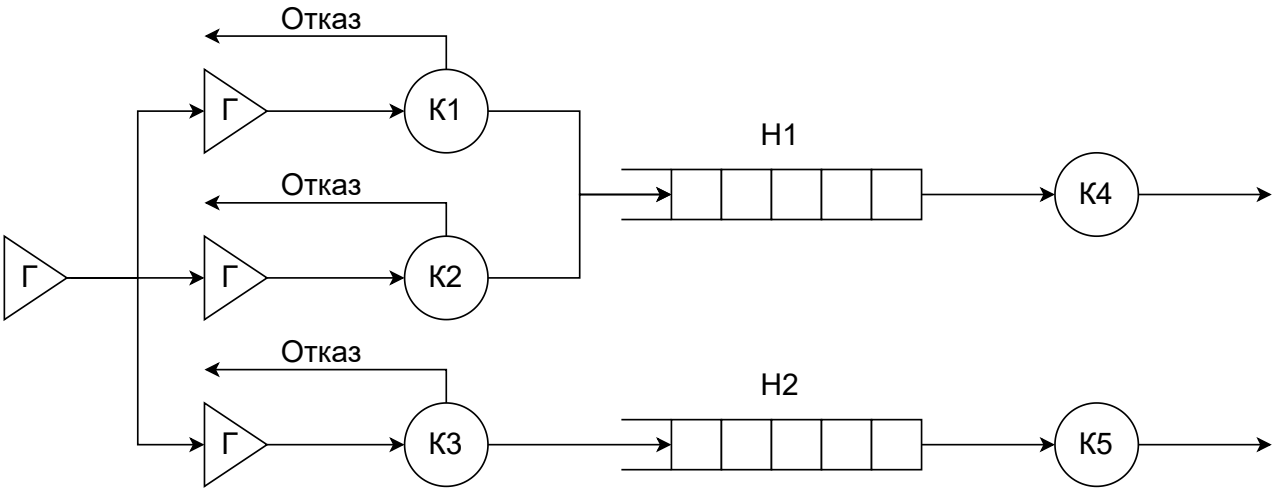


Рисунок 2.2 – Схема модели в терминах СМО

В процессе взаимодействия клиентов с информационным центром возможны следующие режимы.

1. Режим нормального обслуживания, т.е. клиент выбирает одного из свободных операторов, отдавая предпочтение тому, чья скорость обслуживания больше.

2. Режим отказа в обслуживании клиента, когда все операторы заняты.

2.1 Переменные и уравнения имитационной модели

Эндогенные переменные:

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

Экзогенные переменные:

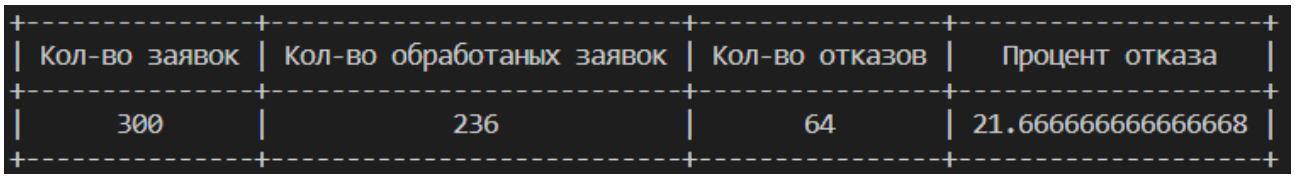
- n_0 — число обслуженных клиентов;
- n_1 — число клиентов получивших отказ.

Уравнение имитационной модели:

$$P = \frac{n_1}{n_1 + n_0} \tag{2.1}$$

3 Результат

Результаты работы программы представлены на рисунке 3.1.



Кол-во заявок	Кол-во обработаных заявок	Кол-во отказов	Процент отказа
300	236	64	21.666666666666668

Рисунок 3.1 – Пример работы программы

4 Код программы

В листингах 4.1–4.5 представлен основной код программы.

Листинг 4.1 – Класс распределения

```
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()
```

Листинг 4.2 – Класс генератора

```
class Generator:
    def __init__(self, generator, count: int):
        self._generator = generator
        self.receivers = []
        self.num_requests = count
        self.next = 0

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

    def generate_request(self):
        self.num_requests -= 1

        for receiver in self.receivers:
            if receiver.receive_request():
                return receiver

        return None
```

Листинг 4.3 – Класс процессора

```
from Generator import Generator


class Processor(Generator):
    def __init__(self, generator, max_queue = -1):
        self._generator = generator
```

```
7           self.curr_queue_size = 0
8           self.max_queue_size = max_queue
9           self.processed_requests = 0
10          self.received_requests  = 0
11          self.next = 0
12
13      # обрабатываем запрос, если они есть
14      def process_request(self):
15          if self.curr_queue_size > 0:
16              self.processed_requests += 1
17              self.curr_queue_size -= 1
18
19      # добавляем реквест в очередь
20      def receive_request(self):
21          if self.max_queue_size == -1 or self.max_queue_size > self.
               curr_queue_size:
22              self.curr_queue_size += 1
23              self.received_requests += 1
24              return True
25
26          return False
27
28      def next_time(self):
29          return self._generator.generate()
```

Листинг 4.4 – Класс модели

```
1  from Generator import Generator
2  from Processor import Processor
3
4  class Model:
5      def __init__(self, generator: Generator, operators: list[
          Processor], computers: list[Processor]):
6          self._generator = generator
7          self._operators = operators
8          self._computers = computers
9
10     # n0 - число обслуженных клиентов
11     # n1 - число клиентов, получивших отказ
12     # n = n0 + n1  - общее число заявок (generated_requests)
13     def start_event(self) -> dict[str, float]:
14         generator = self._generator
15         generated_requests = generator.num_requests
```

```
16          n0 = 0
17          n1 = 0
18
19          generator.receivers = self._operators.copy()
20          self._operators[0].receivers = [self._computers[0]]
21          self._operators[1].receivers = [self._computers[0]]
22          self._operators[2].receivers = [self._computers[1]]
23
24          generator.next = generator.next_time()
25          self._operators[0].next = self._operators[0].next_time()
26
27          blocks = [
28              generator,
29              self._operators[0],
30              self._operators[1],
31              self._operators[2],
32              self._computers[0],
33              self._computers[1],
34          ]
35
36          while generator.num_requests >= 0:
37              # находим наименьшее время
38              curr_t = generator.next
39              for block in blocks:
40                  if 0 < block.next < curr_t:
41                      curr_t = block.next
42
43              # для каждого из блоков
44              for block in blocks:
45                  # если событие наступило для этого блока
46                  if curr_t == block.next:
47                      if not isinstance(block, Processor):
48                          # для генератора
49                          # проверяем, может ли оператор обработать
50                          next_generator = generator.generate_request
                                ()
51                          if next_generator is not None:
52                              next_generator.next = \
53                                  curr_t + next_generator.next_time()
54                              n0 += 1
55                          else:
```

```
                            n1 += 1
                        generator.next = curr_t + generator.
                            next_time()
                    else:
                        block.process_request()
                        if block.curr_queue_size == 0:
                            block.next = 0
                        else:
                            block.next = curr_t + block.next_time()

        return {"refusal_percentage": n1 / generated_requests *
            100,
                "processed": n0,
                "refused": n1,
                }
```

Листинг 4.5 – Основная функция

```
from Model import Model
from Generator import Generator
from Distribute import UniformDistribution
from Processor import Processor

from prettytable import PrettyTable


def create_operators(distribution, t_proc: list[list[int]], n: int,
    max_q: int = 1) -> list[Processor]:
    operators = list()

    for i in range(n):
        operators.append(Processor(
                    distribution(t_proc[i][0] - t_proc[i][1],
                        t_proc[i][0] + t_proc[i][1]),
                    max_queue = max_q)
                    )
    return operators


def create_computers(distribution, t_proc: list[int], n: int) ->
    list[Processor]:
    comps = list()

```

```
23      for i in range(n):
24          comps.append(Processor(
25                      distribution(t_proc[i], t_proc[i]), ) )
26      return comps
27
28
29  def create_table(data: dict[str, float]):
30      table = PrettyTable()
31      table.field_names = ['Кол–во заявок', 'Кол–во обработаных заяво
            к', 'Кол–во отказов', 'Процент отказа']
32      table.add_row([clients_number, data['processed'], data['refused
            '] - 1, data["refusal_percentage"]])
33      print(table)
34
35
36  if __name__ == '__main__':
37      clients_number = 300
38
39      generator = Generator(
40          UniformDistribution(8, 12),
41          clients_number,
42      )
43
44      t_proc_op = list()
45      t_proc_op.append([20, 5]) # t_i, dt_i
46      t_proc_op.append([40, 10])
47      t_proc_op.append([40, 20])
48
49      operators = create_operators(UniformDistribution, t_proc_op,
            len(t_proc_op))
50
51      t_proc_comp = [15, 30]
52
53      computers = create_computers(UniformDistribution, t_proc_comp,
            len(t_proc_comp))
54
55      model = Model(generator, operators, computers)
56      res = model.start_event()
57
58      create_table(res)
```