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

Название Моделирование работы информационного центра

Дисциплина Моделирование

Студент Зайцева А. А.

Группа ИУ7-72Б

Оценка (баллы) _____

Преподаватель Рудаков И. В.

Москва — 2022 г.

1 Задание

Дана концептуальная модель информационного центра:

- в информационный центр приходят клиенты через интервалы 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 минут; клиенты стремятся занять свободного оператора с максимальной производительностью;
- полученные запросы попадают в приемный накопитель, откуда они выбираются на обработку;
- на первый компьютер поступают запросы от 1-ого и 2-ого операторов, на второй – от 3-его; время обработки запросов на первом и втором компьютерах равно, соответственно, 15 и 30 минутам.

За единицу имитационного времени принять 0.01 минуты.

Промоделировать процесс обработки 300 запросов. Определить вероятность отказа. Построить структурную схему модели, а также схему модели в терминах систем массового обслуживания (СМО).

2 Теоретическая часть

В процессе взаимодействия клиентов с информационным центром возможны два режима работы.

1. Режим нормального обслуживания: клиент выбирает одного из свободных операторов (с максимальной производительностью).

2. Режим отказа в обслуживании: если все операторы заняты.

Эндогенные переменные: время обработки задания i-ым оператором, время решения задания на j-ом компьютере.

Экзогенные переменные: n_0 – число обслуженных клиентов, n_1 – число клиентов, получивших отказ.

Уравнение модели (вероятность отказа в обслуживании клиента):

$$p_{отк} = \frac{n_1}{n_0 + n_1}. \tag{1}$$

Структурная схема модели приведена на рисунке 1.

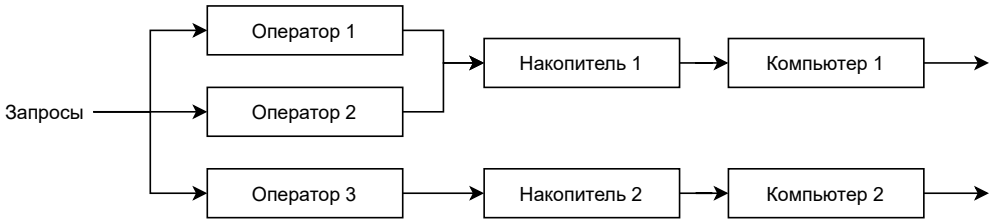

Рисунок 1 – Структурная схема модели

Схема модели в терминах СМО приведена на рисунке 2.

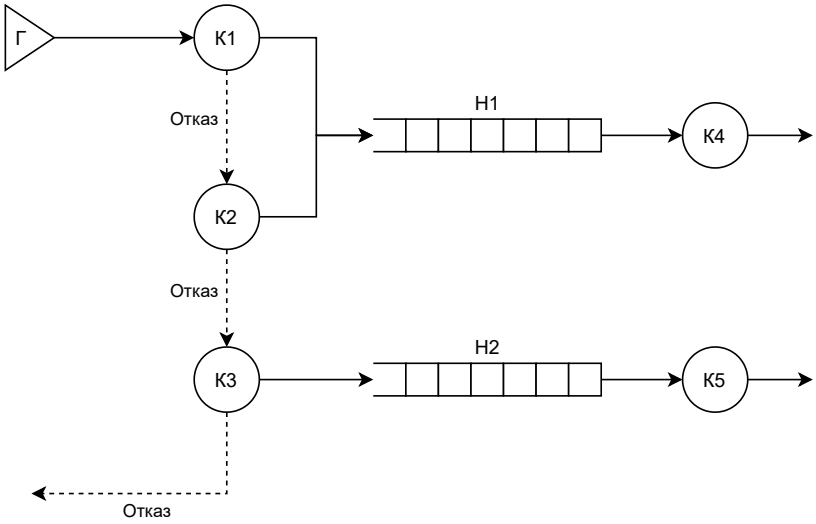

Рисунок 2 – Схема модели в терминах СМО

3 Результаты работы программы

Для исследования разработанная программа была протестирована при различном числе генерируемых заявок, различных временах генерации заявок, различных временах работы операторов и компьютеров. На рисунке 3 приведена таблица, где описаны параметры и результаты моделирования. В каждом случае изменялось не более одного (указанного в таблице) параметра, остальные сохранялись из условия. Сгенерированные псевдослучайные числа во всех случаях одинаковы.

Случай	Имитационное время моделирования	Вероятность отказа
Исходные настройки	3041.390000019966	0.23
Количество заявок увеличено в 10 раза	30170.419998104397	0.21
Время 2 компьютера увеличено в 3 раза	5007.830000062889	0.23
Время 3 оператора увеличено в 3 раза	3156.3000000224743	0.31
Время генерации заявок уменьшено в 2 раза	1588.0899999987007	0.55

Рисунок 3 – Таблица с результатами исследования программы

4 Код программы

В листинге 1 приведен код разработанной программы (используемый язык – Python).

Листинг 1 – Код разработанной программы

```
from typing import *
from prettytable import PrettyTable
from random import random, seed
seed(0)

REQUESTS_TO_GENERATE = 300
MOD_TIME_STEP = 0.01

CLIENT_TIMES = [8, 12]
O1_TIMES = [15, 25]
O2_TIMES = [30, 50]
```

```
O3_TIMES = [20, 60]
C1_TIME = 15
C2_TIME = 30

ACCUMULATORS = [0, 0]
O1_ACCUM_INDEX = 0
O2_ACCUM_INDEX = 0
O3_ACCUM_INDEX = 1
C1_ACCUM_INDEX = 0
C2_ACCUM_INDEX = 1


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

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


class RequestsGenerator:
    def __init__(self, time_generator: DistributedTimeGenerator):
        self.time_generator = time_generator
        self.remaining_time = 0

    def update_time_and_check_for_request(self):
        if self.remaining_time > 0:
            self.remaining_time -= MOD_TIME_STEP
            return False
        else:
            self.remaining_time = self.time_generator.generate()
            return True


class Operator:
    def __init__(self, accum_index: int, time_generator:
    DistributedTimeGenerator):
        self.accum_index = accum_index
        self.time_generator = time_generator
```

```
50
51          self.is_busy = False
52          self.remaining_time = 0
53
54      def update_time(self):
55          self.remaining_time -= MOD_TIME_STEP
56
57          if self.is_busy and self.remaining_time <= 0:
58              self.is_busy = False
59              ACCUMULATORS[self.accum_index] += 1
60
61      def start_process_new_request(self):
62          self.is_busy = True
63          self.remaining_time = self.time_generator.generate()
64
65
66  class Computer:
67      def __init__(self, accum_index: int, processing_time: int):
68          self.accum_index = accum_index
69          self.processing_time = processing_time
70          self.is_busy = False
71          self.remaining_time = 0
72
73      def update_time_and_check_for_finished_processing(self):
74          self.remaining_time -= MOD_TIME_STEP
75
76          if self.is_busy:
77              if self.remaining_time <= 0:
78                  self.is_busy = False
79                  return True
80          else:
81              if ACCUMULATORS[self.accum_index] > 0:
82                  ACCUMULATORS[self.accum_index] -= 1
83                  self.is_busy = True
84                  self.remaining_time = self.processing_time
85
86          return False
87
88
89  def find_free_operator(operators):
```

```
90      for i in range(len(operators)):
91          if not operators[i].is_busy:
92              return i
93
94
95  def simulate():
96      requests_generator = RequestsGenerator(DistributedTimeGenerator(*
        CLIENT_TIMES))
97
98      operators = [
99      Operator(O1_ACCUM_INDEX, DistributedTimeGenerator(*O1_TIMES)),
100     Operator(O2_ACCUM_INDEX, DistributedTimeGenerator(*O2_TIMES)),
101     Operator(O3_ACCUM_INDEX, DistributedTimeGenerator(*O3_TIMES))
102     ]
103
104     computers = [
105     Computer(C1_ACCUM_INDEX, C1_TIME),
106     Computer(C2_ACCUM_INDEX, C2_TIME)
107     ]
108
109     generated, processed, rejected, modeling_time = 0, 0, 0, 0
110     while processed + rejected < REQUESTS_TO_GENERATE:
111         modeling_time += MOD_TIME_STEP
112         if generated < REQUESTS_TO_GENERATE:
113             request = requests_generator.update_time_and_check_for_request
        ()
114             if request:
115                 generated += 1
116                 free_operator_index = find_free_operator(operators)
117                 if free_operator_index is None:
118                     rejected += 1
119                 else:
120                     operators[free_operator_index].
        start_process_new_request()
121
122         for operator in operators:
123             operator.update_time()
124
125         for computer in computers:
126             if computer.update_time_and_check_for_finished_processing():
```

```
                processed += 1

    return generated, processed, rejected, modeling_time


def main():
    global REQUESTS_TO_GENERATE
    global C2_TIME
    global O1_TIMES
    global CLIENT_TIMES

    res_table = PrettyTable()
    res_table.field_names = ['Случай', 'Имитационное время моделирования', '
    Вероятность отказа']

    seed(0)
    generated, processed, rejected, modeling_time = simulate()
    print(generated, processed, rejected, modeling_time)
    res_table.add_row(['Исходные настройки', modeling_time, round(rejected /
    generated, 2)])

    seed(0)
    mn = 10
    tmp = REQUESTS_TO_GENERATE
    REQUESTS_TO_GENERATE = REQUESTS_TO_GENERATE * mn
    generated, processed, rejected, modeling_time = simulate()
    print(generated, processed, rejected, modeling_time)
    res_table.add_row([f'Количество заявок увеличено в {mn} раза',
    modeling_time, round(rejected / generated, 2)])
    REQUESTS_TO_GENERATE = tmp

    seed(0)
    mn = 3
    tmp = C2_TIME
    C2_TIME = C2_TIME * mn
    generated, processed, rejected, modeling_time = simulate()
    print(generated, processed, rejected, modeling_time)
    res_table.add_row([f'Время 2 компьютера увеличено в {mn} раза',
    modeling_time, round(rejected / generated, 2)])
    C2_TIME = tmp
```

```
163
164     seed(0)
165     mn = 3
166     tmp = O1_TIMES
167     O1_TIMES = list(map(lambda time: time * mn, O1_TIMES))
168     generated, processed, rejected, modeling_time = simulate()
169     print(generated, processed, rejected, modeling_time)
170     res_table.add_row([f'Время 1 оператора увеличено в {mn} раза',
    modeling_time, round(rejected / generated, 2)])
171     O1_TIMES = tmp
172
173     seed(0)
174     mn = 2
175     tmp = CLIENT_TIMES
176     CLIENT_TIMES = list(map(lambda time: time // mn, CLIENT_TIMES))
177     generated, processed, rejected, modeling_time = simulate()
178     print(generated, processed, rejected, modeling_time)
179     res_table.add_row([f'Время генерации заявок уменьшено в {mn} раза',
    modeling_time, round(rejected / generated, 2)])
180     CLIENT_TIMES = tmp
181
182     print(res_table)
183
184 if __name__ == '__main__':
185     main()
```