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

Название Моделирование системы массового обслуживания

Дисциплина Моделирование

Студент Зайцева А. А.

Группа ИУ7-72Б

Оценка (баллы) _____

Преподаватель Рудаков И. В.

Москва — 2022 г.

1 Задание

Дана концептуальная модель парковки торгового центра, где также предоставляется услуга автомойки:

- количество мест на парковке $parking\_spaces = 50$;
- количество мест для автомойки $n\_washers = 5$;
- количество клиентов, которые приезжают на парковку каждую минуту, распределенно по закону Пуассона с параметром $requests\_lambda = 3$;
- вероятность того, что владелец обратится за услугой мойки автомобиля $washing\_py = 0.1$;
- клиент встает в очередь либо к оператору автомойки, либо к оператору парковки, в зависимости от того, требуется ему услуга мойки или нет;
- если клиент встал в очередь к оператору парковки, и длина этой очереди больше $max\_operator\_parking\_len = 30$, то клиент расстраивается и уезжает, иначе ожидает в очереди;
- оператор парковки может начать печатать талон для текущего клиента, только если на парковке есть свободное место, оператор парковки обрабатывает одного клиента за 2 ± 1 минуту;
- если клиент встал в очередь к оператору мойки, и длина этой очереди больше $max\_operator\_washing\_len = 5$, то клиент расстраивается и уезжает, иначе ожидает в очереди;

- оператор мойки может начать печатать талон для текущего клиента, если есть свободное место на автомойке (тогда клиент сразу отправляется на автомойку) или если есть свободное место на парковке (тогда клиент занимает место на парковке и ставится в очередь на автомойку), оператор автомойки обрабатывает одного клиента за 2 ± 1 минуту;

- мойка одного автомобиля занимает 60 ± 20 минут;

- если клиенту не требуется автомойка, значит он идет в торговый центр и проводит там время, распределенное по нормальному закону с параметрами $m = 150$ минут и $\sigma = 10$;

- когда услуга автомойки завершена или клиент вернулся из торгового центра, клиент освобождает парковочное место и встает в очередь к оператору оплаты, оператор оплаты обрабатывает одного клиента за 3 ± 1 минуту;

За единицу имитационного времени принять 0.01 минуты.

Промоделировать процесс обработки 10000 клиентов. Определить имитационное время моделирования; процент клиентов, которым потребовалась услуга автомойки; проценты клиентов, которые отказались от услуг парковки и автомойки; максимальную очередь на оплату.

Также провести моделирование при изменении отдельных параметров модели.

Построить структурную схему модели, а также схему модели в терминах систем массового обслуживания (СМО).

2 Теоретическая часть

В процессе взаимодействия клиентов с парковкой с услугой автомойки возможны два режима работы.

1. Режим нормального обслуживания: клиент получает требуемую услугу.

2. Режим отказа от услуги: если очередь превышает предельно допустимую.

Эндогенные переменные: количество парковочных мест, время обработки клиентов операторами; длины очередей, при которых клиенты отказываются от услуг; время, которое клиенты проводят в торговом центре; время мойки автомобиля.

Экзогенные переменные: p_0 – число клиентов, которые припарковались и сходили в торговый центр, p_1 – число клиентов, которые решили не дожидаться освобождения мест для парковки; w_0 – число клиентов, которые получили услугу автомойки, w_1 – число клиентов, которые решили не дожидаться освобождения мест для автомойки.

Уравнения модели:

- процент клиентов, которым потребовалась услуга автомойки:

$$w = \frac{w_0 + w_1}{p_0 + p_1 + w_0 + w_1}; \tag{1}$$

- процент клиентов, которые отказались от услуг парковки:

$$p_p = \frac{p_1}{p_0 + p_1}; \tag{2}$$

- процент клиентов, которые отказались от услуг автомойки:

$$p_w = \frac{w_1}{w_0 + w_1}; \tag{3}$$

Структурная схема модели приведена на рисунке 1.

Схема модели в терминах СМО приведена на рисунке 2.

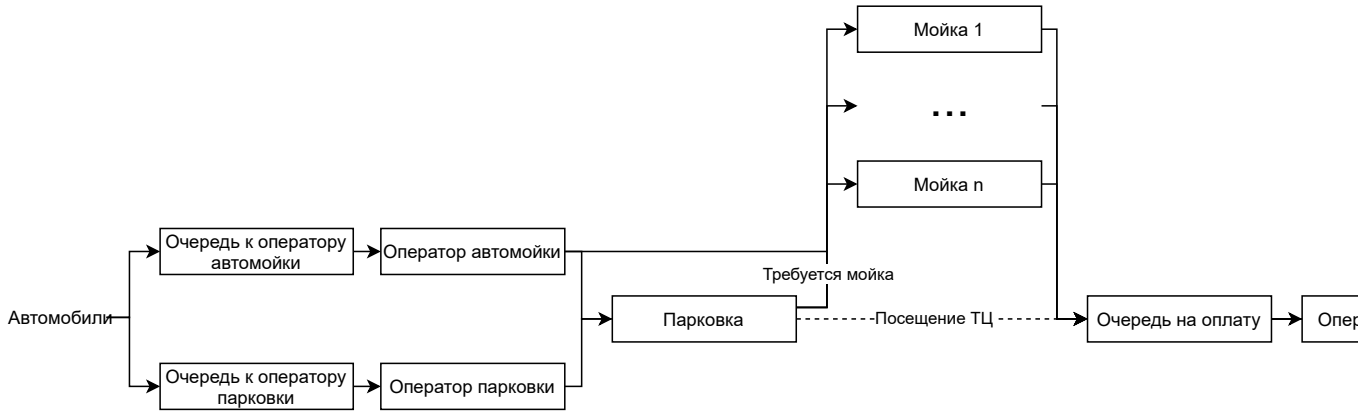

Рисунок 1 – Структурная схема модели

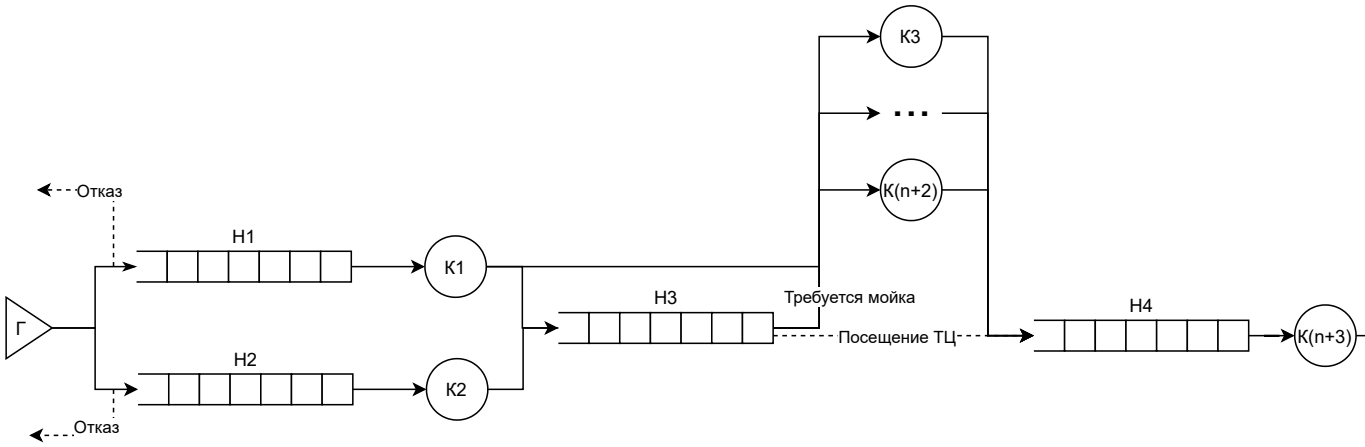

Рисунок 2 – Схема модели в терминах СМО

3 Результаты работы программы

TODO 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	Им. время модел.	Количество клиентов	% обратившихся за автомойкой	% отказа от парковки	% отказа от автомойки	max очередь
Исходный	9077	10000	10	47	20	21
requests_lambda=10	1832	10000	10	49	46	15
n_washers=7	9505	10000	10	43	1	219
washing_p=0.3	9150	10000	29	47	43	15
parking_spaces=100	9755	10000	10	45	17	357
requests=50000	43549	50000	10	48	21	21

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):
```

```
25          self.a = a
26          self.b = b
27
28      def generate(self):
29          return self.a + (self.b - self.a) * random()
30
31
32  class RequestsGenerator:
33      def __init__(self, time_generator: DistributedTimeGenerator):
34          self.time_generator = time_generator
35          self.remaining_time = 0
36
37      def update_time_and_check_for_request(self):
38          if self.remaining_time > 0:
39              self.remaining_time -= MOD_TIME_STEP
40              return False
41          else:
42              self.remaining_time = self.time_generator.generate()
43              return True
44
45
46  class Operator:
47      def __init__(self, accum_index: int, time_generator:
        DistributedTimeGenerator):
48          self.accum_index = accum_index
49          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():
127                 processed += 1
128
129     return generated, processed, rejected, modeling_time
130
131
132 def main():
133     global REQUESTS_TO_GENERATE
134     global C2_TIME
135     global O1_TIMES
136     global CLIENT_TIMES
137
138     res_table = PrettyTable()
139     res_table.field_names = ['Случай', 'Имитационное время моделирования', '
    Вероятность отказа']
```

```
140
141     seed(0)
142     generated, processed, rejected, modeling_time = simulate()
143     print(generated, processed, rejected, modeling_time)
144     res_table.add_row(['Исходные настройки', modeling_time, round(rejected /
    generated, 2)])
145
146     seed(0)
147     mn = 10
148     tmp = REQUESTS_TO_GENERATE
149     REQUESTS_TO_GENERATE = REQUESTS_TO_GENERATE * mn
150     generated, processed, rejected, modeling_time = simulate()
151     print(generated, processed, rejected, modeling_time)
152     res_table.add_row([f'Количество заявок увеличено в {mn} раза',
    modeling_time, round(rejected / generated, 2)])
153     REQUESTS_TO_GENERATE = tmp
154
155     seed(0)
156     mn = 3
157     tmp = C2_TIME
158     C2_TIME = C2_TIME * mn
159     generated, processed, rejected, modeling_time = simulate()
160     print(generated, processed, rejected, modeling_time)
161     res_table.add_row([f'Время 2 компьютера увеличено в {mn} раза',
    modeling_time, round(rejected / generated, 2)])
162     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
```

```
    CLIENT_TIMES = list(map(lambda time: time // mn, CLIENT_TIMES))
    generated, processed, rejected, modeling_time = simulate()
    print(generated, processed, rejected, modeling_time)
    res_table.add_row([f'Время генерации заявок уменьшено в {mn} раза',
    modeling_time, round(rejected / generated, 2)])
    CLIENT_TIMES = tmp

    print(res_table)

if __name__ == '__main__':
    main()
```