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 новых 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 $washing\_p = 0.1$;
- клиент, в зависимости от того, требуется ему услуга мойки или нет, встает в очередь либо к оператору автомойки, либо к оператору парковки;
- если клиент встал в очередь к оператору парковки, а 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 длину очереди 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:

$$w = \frac{w_0 + w_1}{p_0 + p_1 + w_0 + w_1} \cdot 100\%; \tag{1}$$

- процент клиентов, которые отказались от услуги парковки:

$$p_p = \frac{p_1}{p_0 + p_1} \cdot 100\%; \tag{2}$$

- процент клиентов, которые отказались от услуги автомойки:

$$p_w = \frac{w_1}{w_0 + w_1} \cdot 100\%; \tag{3}$$

Структурная схема модели приведена на рисунке 1.

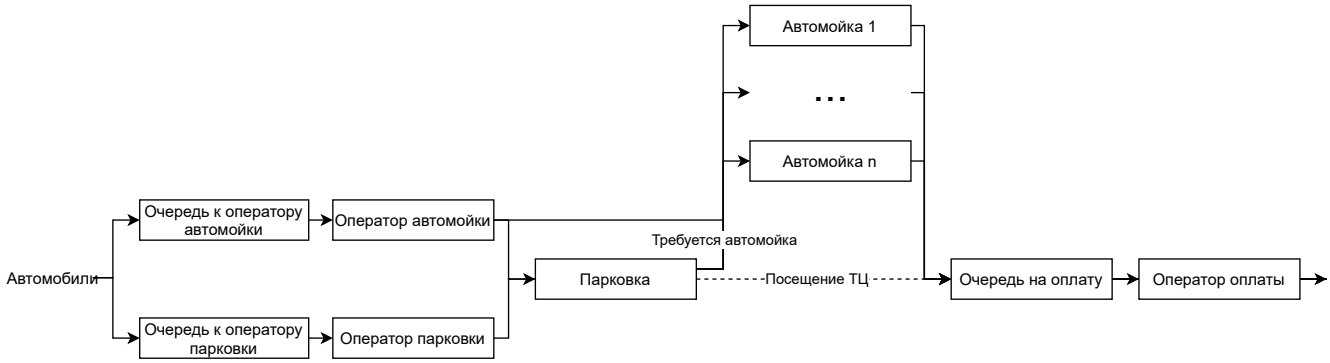

Рисунок 1 – Структурная схема модели

Схема модели в терминах СМО приведена на рисунке 2.

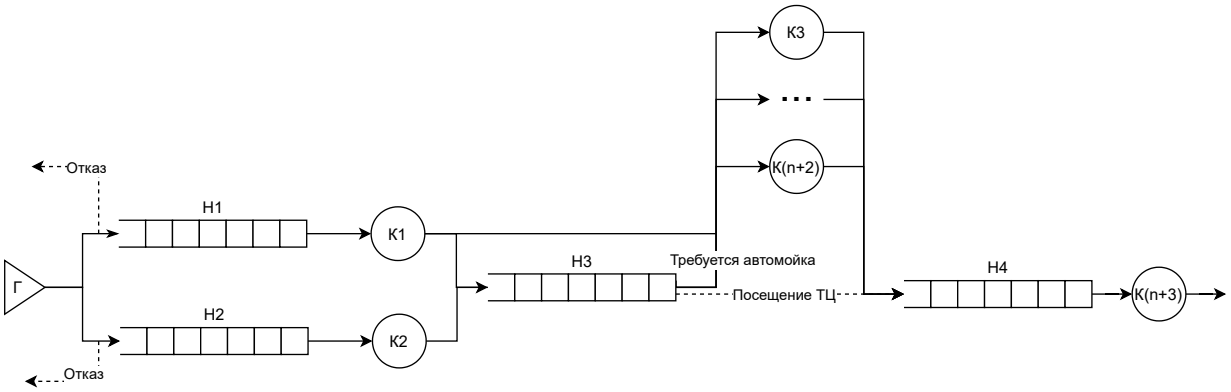

Рисунок 2 – Схема модели в терминах СМО

3 Результаты работы программы

Для исследования разработанная программа была протестирована при различных: количестве генерируемых клиентов; параметре λ распределения Пуассона, по которому распределено количество новых клиентов в минуту; количестве мест для мойки автомобиля; вероятности того, что клиент обратится за услугой автомойки; количестве мест на парковке. 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.

Случай	Им. время модел.	Кол-во клиентов	% обр. за автомойкой	% отказа от парковки	% отказа от автомойки	max очередь на оплату
Исходный	9181	10000	10	47	21	18
requests_lambda=10	1816	10000	11	50	46	11
n_washers=7	9038	10000	10	44	1	132
washing_p=0.3	9123	10000	29	47	42	11
parking_spaces=100	9748	10000	10	45	18	342
parking_spaces=500	16086	10000	10	34	0	2189
requests=50000	43686	50000	10	48	21	17

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
from scipy.stats import poisson
from numpy.random import normal
seed(1)

MOD_TIME_STEP = 0.01


class UniformlyDistributedTimeGenerator:
    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 NormallyDistributedTimeGenerator:
    def __init__(self, m: float, sigma: float):
        self.m = m
        self.sigma = sigma

    def generate(self):
        return normal(self.m, self.sigma)
```

```
26
27
28  class PoissonDistributedTimeGenerator:
29      def __init__(self, lambda_value: int):
30          self.lambda_value = lambda_value
31
32      def generate(self):
33          return poisson.rvs(self.lambda_value, size=1)[0]
34
35
36  class Request:
37      def __init__(self, washing_p):
38          self.need_wash = random() < washing_p
39
40
41  class RequestsGenerator:
42      def __init__(self, time_generator, washing_p):
43          self.remaining_time = 0
44          self.time_generator = time_generator
45          self.washing_p = washing_p
46
47      def update_time_and_check_for_request(self):
48          if self.remaining_time > 0:
49              self.remaining_time -= MOD_TIME_STEP
50              return None
51          else:
52              t = self.time_generator.generate()
53              if t == 0:
54                  self.remaining_time = 1
55              else:
56                  self.remaining_time = 1 / t
57              return Request(self.washing_p)
58
59
60  class OnePlaceChannel:
61      def __init__(self, accum_in: List, accum_out: List, time_generator,
        max_out_accum=None):
62          self.accum_in = accum_in
63          self.accum_out = accum_out
64          self.time_generator = time_generator
```

```
65          self.max_out_accum = max_out_accum
66
67          self.is_busy = False
68          self.remaining_time = 0
69          self.processed_count = 0
70
71      def update_time(self):
72          processed = False
73          if self.max_out_accum is None or len(self.accum_out) < self.
    max_out_accum:
74              if self.is_busy:
75                  self.remaining_time -= MOD_TIME_STEP
76
77                  if self.remaining_time <= 0:
78                      self.processed_count += 1
79                      self.is_busy = False
80                      self.accum_out.append(0)
81                      processed = True
82
83              if not self.is_busy:
84                  if len(self.accum_in) > 0:
85                      self.accum_in.pop(0)
86                      self.is_busy = True
87                      self.remaining_time = self.time_generator.generate()
88          return processed
89
90
91  class SimultaneousChannel:
92      def __init__(self, accum_out, time_generator):
93          self.accum_in = []
94          self.accum_out = accum_out
95          self.time_generator = time_generator
96          self.processed_count = 0
97
98      def start_process_new_request(self):
99          self.accum_in.append(self.time_generator.generate())
100
101     def update_time(self):
102         left_times = []
103         finished_count = 0
```

```
104         for time in self.accum_in:
105             time -= MOD_TIME_STEP
106             if time > 0:
107                 left_times.append(time)
108             else:
109                 finished_count += 1
110                 self.accum_out.append(0)
111         self.accum_in = left_times
112         self.processed_count += finished_count
113 
114         return finished_count
115 
116 
117 def simulate(requests=10000, requests_lambda=1, n_washers=5, washing_p
    =0.1, parking_spaces=50):
118     washing_times = (40, 80)
119     tc_m = 150
120     tc_sigma = 10
121 
122     operator_parking_times = [1, 3]
123     operator_washing_times = [1, 3]
124     operator_paying_times = [2, 4]
125 
126     max_operator_parking_len = 30
127     max_operator_washing_len = 5
128 
129     operator_parking_accum = []
130     operator_washing_accum = []
131     washing_accum = []
132     parking_accum = []
133     paying_accum = []
134     processed_accum = []
135 
136     requests_generator = RequestsGenerator(PoissonDistributedTimeGenerator
    (lambda_value=requests_lambda), washing_p)
137     operator_parking = OnePlaceChannel(operator_parking_accum,
    parking_accum,
138     UniformlyDistributedTimeGenerator(*operator_parking_times),
    max_out_accum=parking_spaces)
139     operator_washing = OnePlaceChannel(operator_washing_accum,
```

```
    parking_accum,
140     UniformlyDistributedTimeGenerator(*operator_washing_times),
    max_out_accum=parking_spaces)
141     tc = SimultaneousChannel(paying_accum,
    NormallyDistributedTimeGenerator(tc_m, tc_sigma))
142     washers = [OnePlaceChannel(washing_accum, paying_accum,
143     UniformlyDistributedTimeGenerator(*washing_times)) for _ in range(
    n_washers)]
144     operator_paying = OnePlaceChannel(paying_accum, processed_accum,
145     UniformlyDistributedTimeGenerator(*operator_paying_times))
146 
147     washing_generated, parking_generated, washing_rejected,
    parking_rejected, modeling_time, max_paying_len = 0, 0, 0, 0, 0, 0
148 
149     while len(processed_accum) < requests:
150         if len(paying_accum) > max_paying_len:
151             max_paying_len = len(paying_accum)
152 
153 
154         modeling_time += MOD_TIME_STEP
155         if (washing_generated + parking_generated) < requests:
156             request = requests_generator.update_time_and_check_for_request
    ()
157             if request is not None:
158                 if request.need_wash:
159                     washing_generated += 1
160                     if len(operator_washing_accum) >=
    max_operator_washing_len:
161                         washing_rejected += 1
162                         processed_accum.append(0)
163                     else:
164                         operator_washing_accum.append(0)
165                 else:
166                     parking_generated += 1
167                     if len(operator_parking_accum) >=
    max_operator_parking_len:
168                         parking_rejected += 1
169                         processed_accum.append(0)
170                     else:
171                         operator_parking_accum.append(0)
```

```
172
173         new_visitor_to_tc = operator_parking.update_time()
174         if new_visitor_to_tc:
175             tc.start_process_new_request()
176         visitors_from_tc = tc.update_time()
177         for _ in range(visitors_from_tc):
178             parking_accum.pop(0)
179
180         washing_request = operator_washing.update_time()
181         if washing_request:
182             washing_accum.append(0)
183
184         for operator in washers:
185             washing_finished = operator.update_time()
186             if washing_finished:
187                 parking_accum.pop(0)
188
189         operator_paying.update_time()
190
191
192     return washing_generated, parking_generated, washing_rejected,
    parking_rejected, modeling_time, max_paying_len
193
194 def add_row(table, name, washing_generated, parking_generated,
    washing_rejected, parking_rejected, modeling_time, max_paying_len):
195     table.add_row([name, round(modeling_time),
196     washing_generated + parking_generated,
197     round(100 * washing_generated / (parking_generated + washing_generated
    )),
198     round(100 * parking_rejected / (parking_generated + parking_rejected))
    ,
199     round(100 * washing_rejected / (washing_generated + washing_rejected))
    ,
200     max_paying_len
201     ])
202
203 def main():
204     res_table = PrettyTable()
205     res_table.field_names = ['Случай', 'Им. время модел.',
206     'Колво- клиентов', '% обр. за автомойкой',
```

```
207     '% отказа от парковки', '% отказа от автомойки', 'max очередь на оплату']
208
209     add_row(res_table, 'Исходный', *simulate())
210     add_row(res_table, 'requests_lambda=10', *simulate(requests_lambda=10)
        )
211     add_row(res_table, 'n_washers=7', *simulate(n_washers=7))
212     add_row(res_table, 'washing_p=0.3', *simulate(washing_p=0.3))
213     add_row(res_table, 'parking_spaces=100', *simulate(parking_spaces=100)
        )
214     add_row(res_table, 'requests=50000', *simulate(requests=50000))
215
216     print(res_table)
217
218
219 if __name__ == '__main__':
220     main()
```