Министерство образования и науки Российской Федерации

федеральное государственное бюджетное образовательное

учреждение высшего образования

«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра информатики, вычислительной техники и информационной безопасности

(наименование кафедры)

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель Перепелкин Е.А.

(подпись) (и.о., фамилия)

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г.

(дата)

Отчет

по лабораторной работе №2

«Моделирование систем массового обслуживания (СМО)»

по дисциплине Моделирование информационных процессов

(наименование дисциплины)

ЛР 09.04.01.12.000 О

(обозначение документа)

Студенты группы 8ИВТ-71 Уваров К.А.

(фамилия и.о.)

Преподаватель  Перепелкин Е.А**.**

(должность, ученое звание) (фамилия и.о.)

Барнаул 2018

**1 Задание**.

Написать программу моделирования одноканальной СМО с пуассоновским потоком заявок с интенсивностью λ и экспоненциальным законом распределения времени обслуживания заявок с интенсивностью µ. По результатам моделирования оценить среднюю длину очереди заявок, среднее время пребывания заявки в очереди, среднее время простоя обслуживающего устройства. Сравнить с теоретическими значениями.

**2 Теория**

Одноканальная система массового обслуживания состоит из потока поступающих на обслуживание заявок (требований), очереди заявок, обслуживающего устройства (канала, прибора, сервера).

Заявки поступают в случайные моменты времени. Простейший однородный пуассоновский поток заявок описывается функцией распределения и плотностью распределения.

Система уравнений Колмогорова

Предельные вероятности находятся из системы уравнений

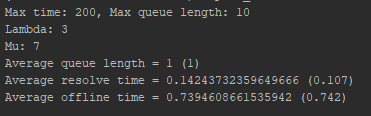
Эти вероятности равны

Средняя длина очереди заявок

По формуле Литтла среднее время пребывания заявки в очереди

3 **Ход работы**

**Тест 1.**



Гистограмма результата:

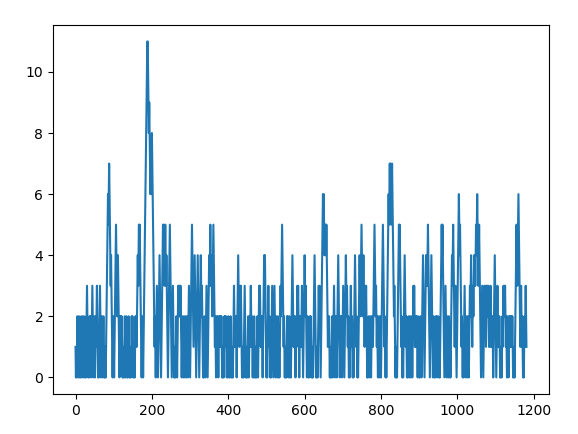
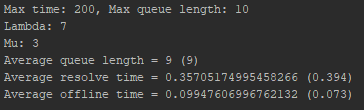
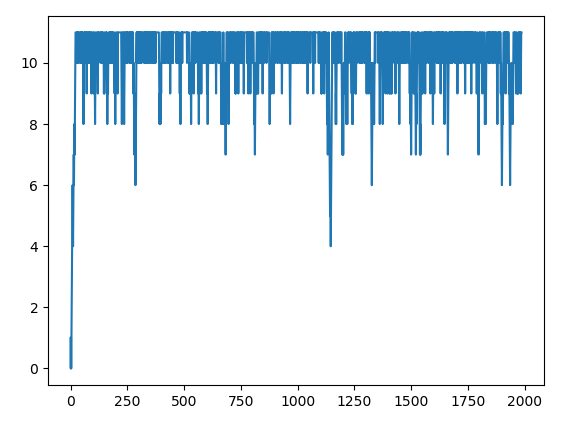


Рисунок 1 ‑ Гистограмма первого испытания

**Тест 2**



Гистограмма результата:



**Вывод**: как видно из проведенных тестов, вероятность попадания в одно из состояний, при большом числе экспериментов, не зависит от начального состояния. Однако при малом числе переходов эти вероятности изменяются.

**Приложение А. Код программы:**

import random

import matplotlib.pyplot as plt

MAX\_TIME\_LENGTH = 200

MAX\_QUEUE\_SIZE = 10

class Status:

def \_\_init\_\_(self):

self.status\_hist = []

self.queue\_length = -1

def \_\_save\_current\_status(self, time, spend\_time, action):

self.status\_hist.append({'time': time,

'spend\_time': spend\_time,

'action': action,

'queue\_length': self.queue\_length,

'working\_status': self.queue\_length != -1})

# self.status\_hist.sort(key=lambda sh: sh['time'])

def add\_task(self, time, spend\_time):

if self.queue\_length < MAX\_QUEUE\_SIZE:

self.queue\_length += 1

self.\_\_save\_current\_status(time, spend\_time, 'add')

def remove\_task(self, time, spend\_time):

self.queue\_length -= 1

self.\_\_save\_current\_status(time, spend\_time, 'remove')

def get\_avg\_queue\_length(self):

sum\_queue = 0

for sh in self.status\_hist:

if sh['queue\_length'] != -1:

sum\_queue += sh['queue\_length']

return round(sum\_queue / len(self.status\_hist)) if len(self.status\_hist) != 0 else 0

def get\_avg\_resolve\_time(self):

solve\_time\_list = [sl['spend\_time'] for sl in self.status\_hist if sl['action'] == "remove"]

return sum(solve\_time\_list) / len(solve\_time\_list) if len(solve\_time\_list) != 0 else 0

def get\_avg\_resolve\_offline\_time(self):

list\_remove = [sl['time'] for sl in self.status\_hist if sl['action'] == "remove" and sl['working\_status'] == False]

list\_add = [sl['time'] for sl in self.status\_hist if sl['action'] == "add" and sl['queue\_length'] == 0]

return (sum(list\_remove) - sum(list\_add[:len(list\_remove )])) / len(list\_remove) if len(list\_remove) != 0 else 0

class TaskGenerator:

def \_\_init\_\_(self, \_lambda):

self.\_lambda = \_lambda

def get\_next\_tick(self):

return random.expovariate(self.\_lambda)

class TaskResolver:

def \_\_init\_\_(self, \_mu):

self.\_mu = \_mu

def get\_next\_tick(self):

return random.expovariate(self.\_mu) # if queue\_length > 0 else 0

def main():

print("Enter lambda")

\_lambda = int(input())

# \_lambda = 7

# \_lambda = 3

print("Enter mu")

\_mu = int(input())

# \_mu = 4

# \_mu = 7

tg = TaskGenerator(\_lambda)

tr = TaskResolver(\_mu)

status = Status()

max\_array\_time = MAX\_TIME\_LENGTH

current\_time = 0

t\_tick = tg.get\_next\_tick()

r\_tick = tr.get\_next\_tick()

t\_current\_time = t\_tick

r\_current\_time = r\_tick

while current\_time <= max\_array\_time:

if t\_current\_time < r\_current\_time or status.queue\_length == -1:

current\_time = t\_current\_time

if status.queue\_length == -1:

r\_current\_time += t\_tick

status.add\_task(current\_time, t\_tick)

t\_tick = tg.get\_next\_tick()

t\_current\_time += t\_tick

else:

current\_time = r\_current\_time

status.remove\_task(current\_time, r\_tick)

r\_tick = tr.get\_next\_tick()

r\_current\_time += r\_tick

for \_ in status.status\_hist:

print(\_)

print(f"Max time: {MAX\_TIME\_LENGTH}, Max queue length: {MAX\_QUEUE\_SIZE}")

print(f"Lambda: {\_lambda}")

print(f"Mu: {\_mu}")

avg\_queue\_length = status.get\_avg\_queue\_length()

avg\_resolve\_time = status.get\_avg\_resolve\_time()

avg\_offline\_time = status.get\_avg\_resolve\_offline\_time()

print(f"Average queue length = {avg\_queue\_length} ({avg\_queue\_length})")

print(f"Average resolve time = {avg\_resolve\_time} ({get\_abs\_resolve\_value(avg\_resolve\_time)})")

print(f"Average offline time = {avg\_offline\_time} ({get\_abs\_resolve\_value(avg\_offline\_time)})")

plt.plot([\*range(len(status.status\_hist))], [sh['queue\_length'] + 1 for sh in status.status\_hist])

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

main()