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

к лабораторной работе №5

По курсу: «Моделирование»

Студент ***ИУ7И-76Б*** _____ **Нгуен Ф. С.**
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ **Рудаков И.В.**
(Подпись, дата) (И.О. Фамилия)

Москва, 2021 г.

Оглавление

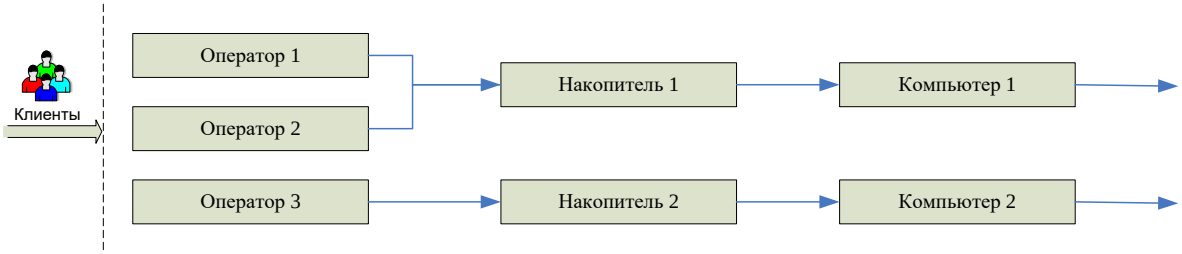
I. Задача

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 накопитель. Откуда выбираются на обработку. На первый компьютер запросы от 1 и 2-ого операторов, на второй – запросы от 3-его. Время обработки запросов первым и 2-м компьютером равны соответственно 15 и 30 мин. Промоделировать процесс обработки 300 запросов.

II. Теоретическая часть

Необходимо создать концептуальную модель в терминах СМО, определить эндогенные и экзогенные переменные и уравнения модели.

За единицу системного времени выбрать 0,01 минуты.



В процессе взаимодействия клиентов с информационным центром возможно:

1) Режим нормального обслуживания, т.е. клиент выбирает одного из свободных операторов, отдавая предпочтение тому у которого меньше номер.

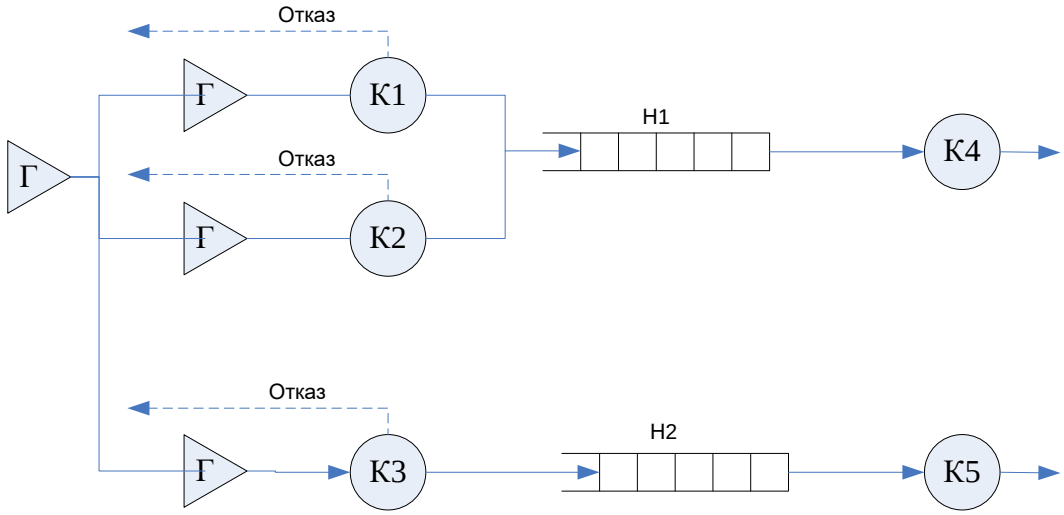
2) Режим отказа в обслуживании клиента, когда все операторы заняты

Переменные и уравнения имитационной модели.

Эндогенные переменные: время обработки задания i-ым оператором, время решения этого задания j-ым компьютером.

Экзогенные переменные: число обслуженных клиентов и число клиентов получивших отказ.

На рисунке представлена концептуальная модель в терминах СМО.



Вероятность отказа:

$$P = \frac{C_{\text{откл}}}{C_{\text{откл}} + C_{\text{обсл}}} \quad (1)$$

III. Экспериментальная часть

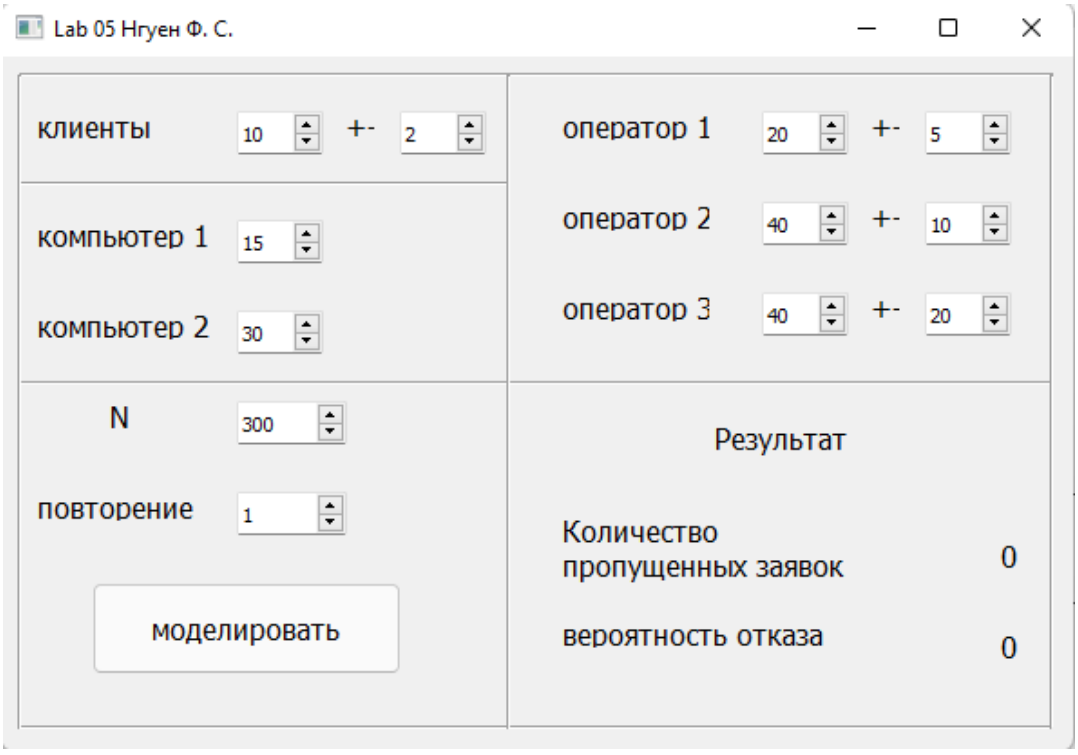


Рисунок 1. Интерфейс программы

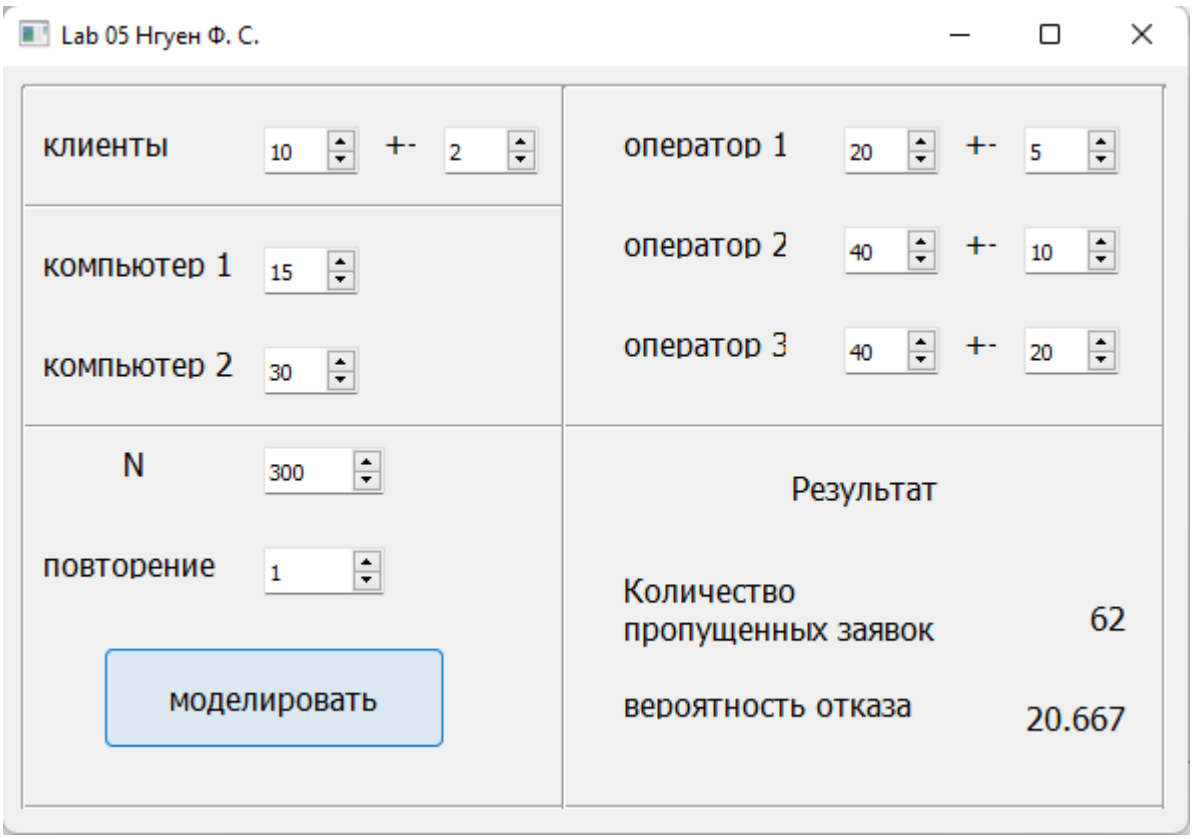


Рисунок 2. пример работы программы

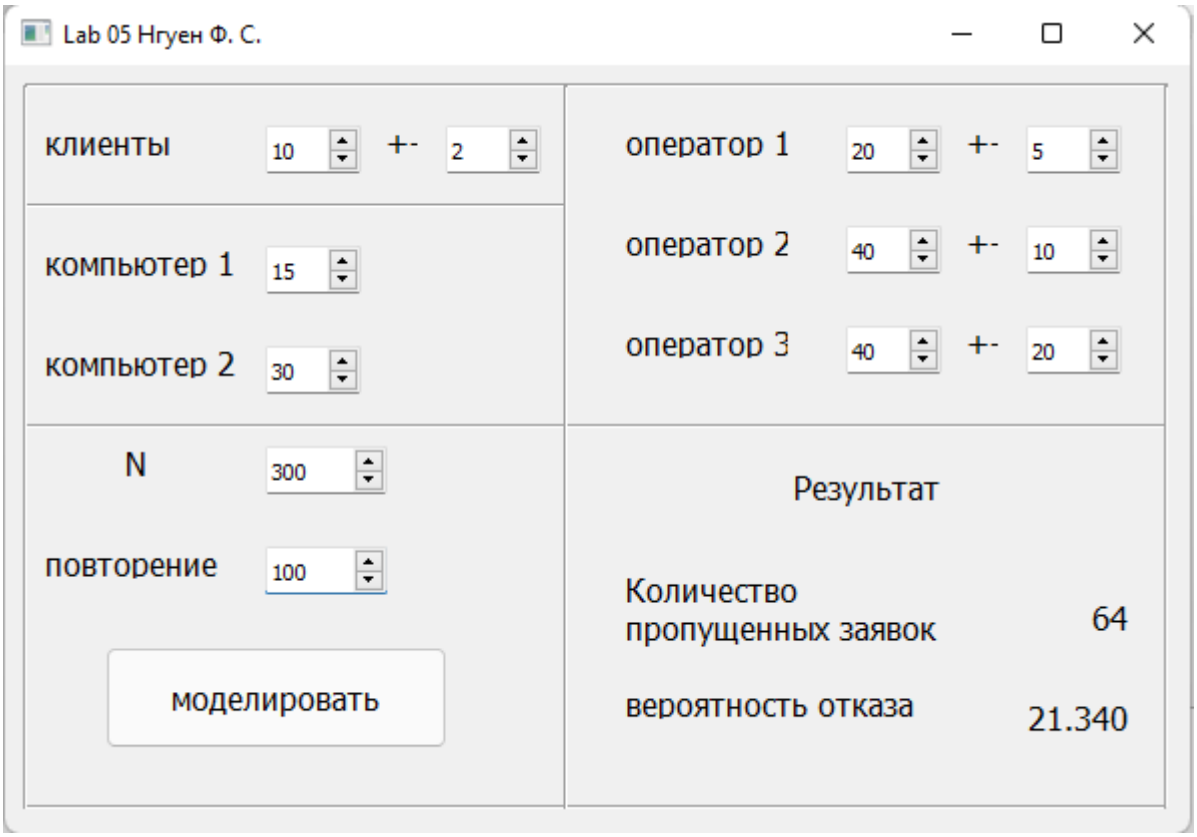


Рисунок 3. пример работы программы с повторением 100 раз

IV. Код программы:

```
import numpy.random as rn
from PyQt5 import QtWidgets, uic
from PyQt5.QtWidgets import QTableWidgetItem
from PyQt5.QtGui import QPen, QColor, QImage, QPixmap, QPainter
from PyQt5.QtCore import Qt, QTime, QCoreApplication, QEventLoop, QPointF


class RandomGenerator:
    def __init__(self, begin, delta=0):
        self.begin = begin
        self.d = delta

    def new_random(self):
        if (self.d == 0):
            return self.begin
        return rn.uniform(self.begin - self.d, self.begin + self.d)


class GenerateRequest:
    def __init__(self, generator, count):
        self.random_generator = generator
        self.num_requests = count
        self.receivers = []
        self.next = 0

    def generate_request(self):
        self.num_requests -= 1
        for receiver in self.receivers:
            if receiver.receive_request():
                return receiver
        return None

    def delay(self):
        return self.random_generator.new_random()


class ProcessRequest:
    def __init__(self, generator, max_queue_size=-1):
        self.random_generator = generator
        self.queue, self.received, = 0, 0
  self.max_queue, self.processed = max_queue_size, 0
        self.next = 0

    def receive_request(self):
        if self.max_queue == -1 or self.max_queue > self.queue:
            self.queue += 1
            self.received += 1
            return True
        return False

    def process_request(self):
        if self.queue > 0:
            self.queue -= 1
            self.processed += 1

    def delay(self):
        return self.random_generator.new_random()
```

```
class Model:
    def __init__(self, generator, operators, computers):
        self.generator = generator
        self.operators = operators
        self.computers = computers

    def event_mode(self):
        refusals = 0
        generated_requests = self.generator.num_requests
        generator = self.generator

         generator.receivers = [self.operators[0], self.operators[1], self.operators[2]]
        self.operators[0].receivers = [self.computers[0]]
        self.operators[1].receivers = [self.computers[0]]
        self.operators[2].receivers = [self.computers[1]]

        generator.next = generator.delay()
        self.operators[0].next = self.operators[0].delay()

        blocks = [generator,
                  self.operators[0],
                  self.operators[1],
                  self.operators[2],
                  self.computers[0],
                  self.computers[1]]

        while generator.num_requests >= 0:
            current_time = generator.next
            for block in blocks:
                if 0 < block.next < current_time:
                    current_time = block.next

            for block in blocks:
                if current_time == block.next:
                    if not isinstance(block, ProcessRequest):
                        next_generator = generator.generate_request()
                        if next_generator is not None:
                            next_generator.next = current_time + next_generator.delay()
                        else:
                            refusals += 1
                        generator.next = current_time + generator.delay()
                    else:
                        block.process_request()
                        if block.queue == 0:
                            block.next = 0
                        else:
                            block.next = current_time + block.delay()

        return {"refusal_percentage": refusals / generated_requests * 100,
                "refusals": refusals}


class Window(QtWidgets.QMainWindow):
    def __init__(self):
        QtWidgets.QWidget.  init  (self)
        uic.loadUi("window.ui", self)
        self.BtnModeling.clicked.connect(lambda: startModeling(self))

        self.count.setText("0")
        self.percent.setText("0")
        self.resLabel.setText("Результат")
```

```
def startModeling(win):

    win.resLabel.setText("Processing....")
    print("Procssing ....")
    #Get Value From Inteface
    try:
        time_clients = win.time_client.value()
        delta_time_clients = win.d_time_client.value()
        #print("Get Data Form Client OK")

        first_operator = win.time_op_1.value()
        second_operator = win.time_op_2.value()
        third_operator = win.time_op_3.value()

        delta_first_operators = win.d_time_op_1.value()
        delta_second_operators = win.d_time_op_2.value()
        delta_third_operators = win.d_time_op_3.value()

        #print("Get Data From Operators OK")

        first_computer = win.time_comp_1.value()
        second_computer = win.time_comp_2.value()
        #print("Get Data From Computers OK")

        clients_number = win.n.value()

        repeat_time = win.repeat.value()
    except:
        print("Error While Getting Data From Interface")
    res = {"refusal_percentage": 0, "refusals": 0}
    #print("Start Processing")
    #Processing
    for i in range(repeat_time):
        #Generators
        #print("Get Generator");
        generator = GenerateRequest(RandomGenerator(time_clients, delta_time_clients),
clients_number)

        #Operators
        #print("Get Operator")
        operators = [ProcessRequest(RandomGenerator(first_operator, delta_first_operators),
max_queue_size=1),
                     ProcessRequest(RandomGenerator(second_operator,
delta_second_operators), max_queue_size=1),
                     ProcessRequest(RandomGenerator(third_operator, delta_third_operators),
max_queue_size=1)]

        #Computers
        #print("Get Computer")
        computers = [ProcessRequest(RandomGenerator(first_computer)),
                     ProcessRequest(RandomGenerator(second_computer))]

        #Start Modeling
        #print("Start Modeling")
        model = Model(generator, operators, computers)
        #print("Get Result")
        result = model.event_mode()
        #print(result)
        res['refusals'] += result['refusals']
        res['refusal_percentage'] += result['refusal_percentage']

    res['refusal_percentage'] /= repeat_time
    res['refusals'] = int(res['refusals'] / repeat_time)
```

```
    win.count.setText("{}".format(res['refusals']))
    win.percent.setText("{:7.3f}".format(res['refusal_percentage']))
    win.resLabel.setText("Результат")
    print("Result Update OK")


if __name__ == '__main__':
    import sys
    app = QtWidgets.QApplication(sys.argv)
    w = Window()
    w.show()
    sys.exit(app.exec_())
```