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 по дисциплине «Моделирование»

Тема Моделирование информационного центра

Студент Слепокурова М.Ф.

Группа ИУ7-76Б

Оценка (баллы) _____

Преподаватель Рудаков И.В.

Москва — 2023 г.

Постановка задачи

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 сдаются в приемный накопитель, откуда они выбираются для обработки. На первый компьютер — запросы от первого и второго оператора, на второй компьютер — от третьего оператора. Время обработки на первом и втором компьютере равны соответственно 15 и 30 минутам. За единицу системного времени выбрать 0.01 минуты.

В процессе взаимодействия клиентов и центра возможны: режим нормального обслуживания (клиент выбирает одного из свободных операторов, но предпочитает того, у которого номер меньше) и режим отказа.

В рамках поставленной задачи необходимо:

- смоделировать процесс обработки 300 запросов;
- определить вероятность отказа;
- построить структурную схему модели;
- нарисовать модель в терминах СМО.

Теория

Анализ задачи

Эндогенные переменные: время обработки задания i-м оператором и время решения задания на j-м компьютере.

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

Вероятность отказа может быть рассчитана по следующей формуле: $\frac{n_1}{n_0+n_1}$.

На рисунке 1 изображена структурная схема реализуемой модели.

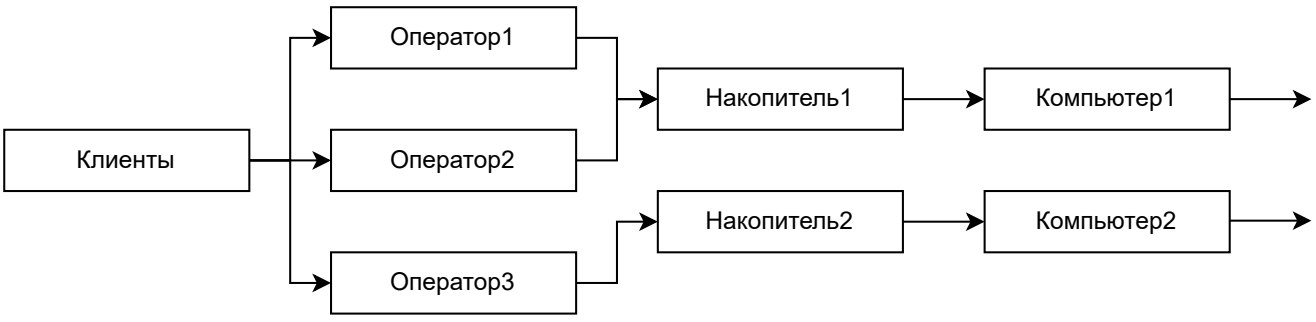

Рисунок 1 – Структурная схема модели

На рисунке 2 изображена схема модели в терминах СМО, где Г — генератор заявок, $К_i$ — канал обработки, $Н_i$ — накопитель.

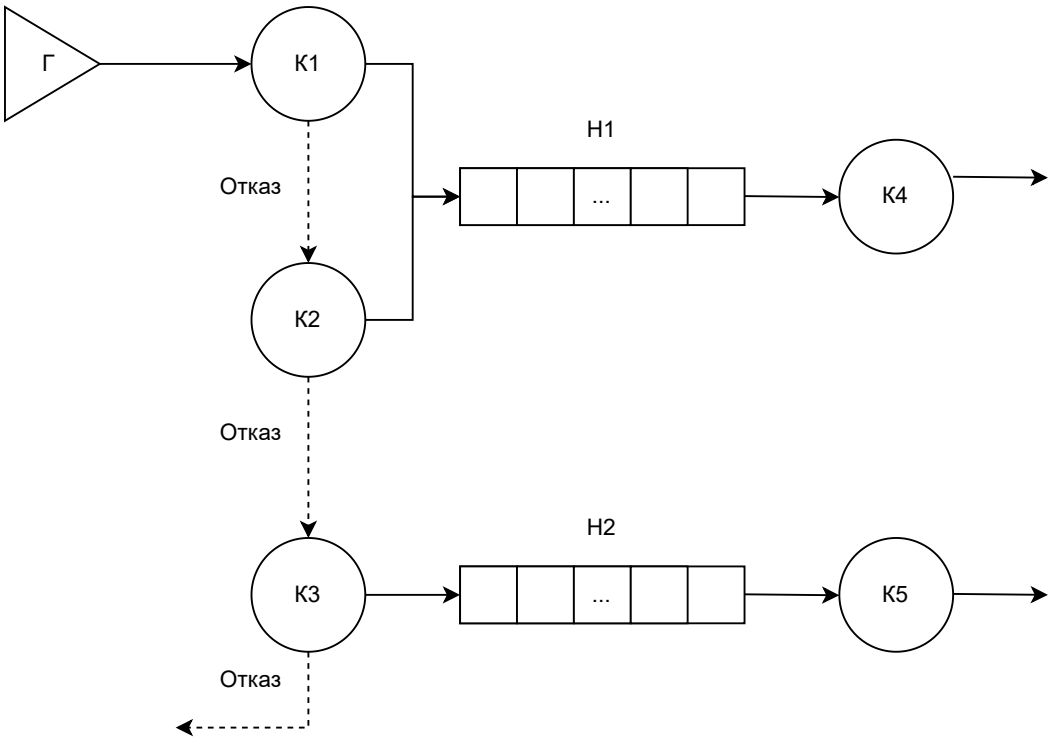

Рисунок 2 – Схема модели в терминах СМО

Принцип Δt

Данный принцип заключается в последовательном анализе состояний всех блоков в момент $t + \Delta t$ по заданному состоянию блоков в момент времени t, при этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием с учетом действующих случайных факторов, задаваемых распределениями вероятности. В результате такого анализа принимается решение о том, какие общесистемные события должны имитироваться программной моделью на данный момент времени.

Средства реализации

Для реализации приложения был выбран язык программирования Python.

Листинг кода

```
from numpy.random import uniform

class TimeGenerator:
  def __init__(self, time, delta):
    self.time = time
    self.delta = delta

  def randomTime(self):
    return uniform(self.time - self.delta, self.time + self.delta)

class RequestGenerator:
  def __init__(self, timeGenerator, count, recievers = []):
    self.timeGenerator = timeGenerator
    self.requestCount = count
    self.receivers = recievers
    self.next = 0

  def generateRequest(self, currTime):
    self.requestCount -= 1
    self.next = currTime + self.generateDuration()
    for receiver in self.receivers:
      if not receiver.busy: return receiver
    return None

  def generateDuration(self):
    return self.timeGenerator.randomTime()

class RequestOperator:
  def __init__(self, timeGenerator, processor):
    self.timeGenerator = timeGenerator
    self.next = 0
    self.processor = processor
    self.busy = False

  def receiveRequest(self, currTime):
    self.busy = True
    self.next = currTime + self.generateDuration()

  def processRequest(self):
    if self.busy:
      self.next = 0
      self.busy = False

  def generateDuration(self):
    return self.timeGenerator.randomTime()
```

```
class RequestProcessor:
  def __init__(self, timeGenerator):
    self.timeGenerator = timeGenerator
    self.queueSize = 0
    self.next = 0

  def pushRequest(self):
    self.queueSize += 1

  def popRequest(self, currTime):
    if self.queueSize > 0:
      self.queueSize -= 1
      self.next = currTime + self.generateDuration()
    else:
      self.next = 0

  def generateDuration(self):
    return self.timeGenerator.randomTime()

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

  def simulate(self, delta):
    refusalCount = 0
    generatedRequests = self.generator.requestCount
    blocks = [self.generator, *self.operators, *self.processors]

    currTime = 0
    while self.generator.requestCount > 0:
      for block in blocks:
        if block.next <= currTime:
          if isinstance(block, RequestGenerator):
            receiver = self.generator.generateRequest(currTime)
            if not receiver: refusalCount += 1
            else: receiver.receiveRequest(currTime)
          elif isinstance(block, RequestOperator):
            block.processRequest()
            block.processor.pushRequest()
          elif isinstance(block, RequestProcessor):
            block.popRequest(currTime)
      currTime += delta

    return { "refusalProbability": refusalCount / generatedRequests, "
       refusalCount": refusalCount }

requestCount = 300
```

```
computer1 = RequestProcessor(TimeGenerator(15, 0))
computer2 = RequestProcessor(TimeGenerator(30, 0))

operator1 = RequestOperator(TimeGenerator(20, 5), computer1)
operator2 = RequestOperator(TimeGenerator(40, 10), computer1)
operator3 = RequestOperator(TimeGenerator(40, 20), computer2)

requestGenerator = RequestGenerator(TimeGenerator(10, 2), requestCount, [
    operator1, operator2, operator3])

model = Model(requestGenerator, [operator1, operator2, operator3], [
    computer1, computer2])
result = model.simulate(0.01)

refusalCount, refusalProbability = result["refusalCount"], result["
    refusalProbability"]

print("Requests: ", requestCount)
print("Refusals: ", refusalCount)
print("Refusal probability: ", round(refusalProbability, 2))
```

Демонстрация работы программы

На рисунке 3 изображен пример работы программы для 300 заявок.

```
Requests:  300
Refusals:  62
Refusal probability:  0.21
```

Рисунок 3 – Пример работы программы — 1

На рисунке 4 изображен пример работы программы для 500 заявок.

```
Requests:  500
Refusals:  106
Refusal probability:  0.21
```

Рисунок 4 – Пример работы программы — 2

На рисунке 5 изображен пример работы программы для 1000 заявок.

```
Requests:  1000
Refusals:  222
Refusal probability:  0.22
```

Рисунок 5 – Пример работы программы — 3