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»

Отчёт

по лабораторной работе № 5

Название: Определение вероятности отказа

Дисциплина: Моделирование

Студент ИУ7-75Б _____ Д.В. Сусликов

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ И.В. Рудаков

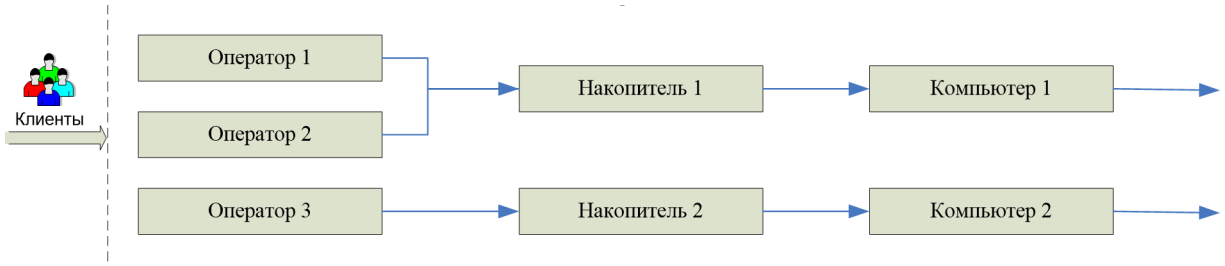
(Подпись, дата) (И.О. Фамилия)

Москва, 2021

Содержание

Задание

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. Для выполнения поставленного задания необходимо создать концептуальную модель в терминах СМО, определить эндогенные и экзогенные переменные и уравнения модели. За единицу системного времени выбрать 0,01 минуты.



Аналитическая часть

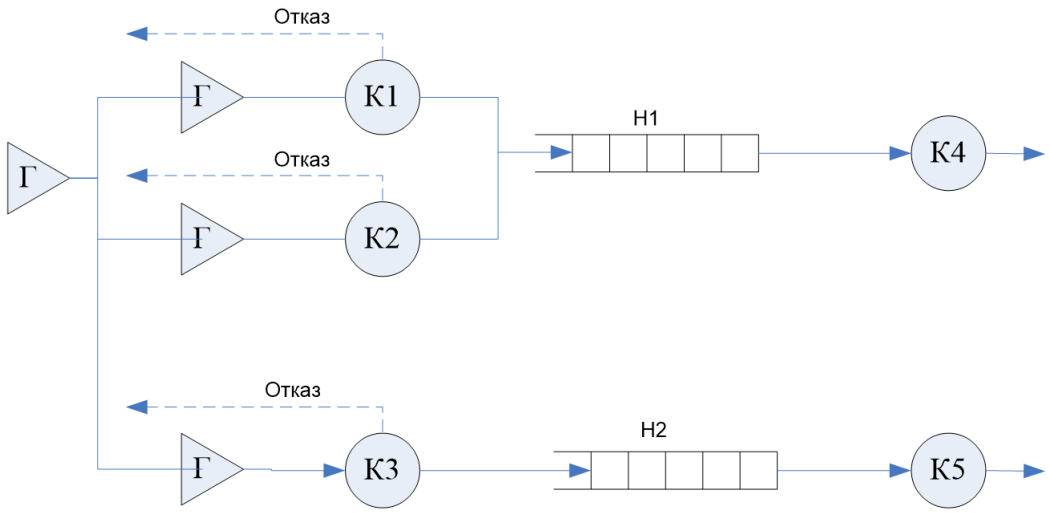
В процессе взаимодействия клиентов с информационным центром возможно:

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.



$$P_{\text{отк}} = \frac{C_{\text{отк}}}{C_{\text{отк}} + C_{\text{обсл}}}$$

Примеры

```
time (secs) 0.3257584571838379
generated 300
lost 63
processed 237
lost 0.21
```

Рисунок 1 – 300 заявок

```
time (secs) 1.0843586921691895
generated 1000
lost 214
processed 786
lost 0.214
```

Рисунок 2 – 1000 заявок

```
time (secs) 3.370170831680298
generated 3000
lost 633
processed 2367
lost 0.211
```

Рисунок 3 – 3000 заявок

Вывод

Основываясь на примерах работы системы при 300, 1000 и 3000 заявок, можно сделать вывод, что процент потерь примерно равен 21%.

Листинги

Листинг 1 – main.py

```
from time import time
from UniformDistribution import UniformDistribution
from Generator import Generator
from Operator import Operator
from Processor import Processor

timeStep = 0.01



def ChooseOperator(operators):
  for i in range(len(operators)):
    if not operators[i].busy:
      return i
  return -1

def DoStep(generator, operators, processors, requestInfo,
   newGenerate=True):
  if newGenerate:
    request = generator.UpdateTime(timeStep)
  if request:
    requestInfo['generated'] += 1
    operatorIndex = ChooseOperator(operators)
    if operatorIndex == -1:
      requestInfo['lost'] += 1
    else:
      operators[operatorIndex].acceptRequest(request)

  for curOperator in operators:
    curOperator.UpdateTime(timeStep)

  for curProcessor in processors:
```

```
31          res = curProcessor.UpdateTime(timeStep)
32          if res == 0:
33            requestInfo['processed'] += 1
34
35
36      def modeling(generator, operators, processors,
            incomingRequestAmount):
37        requestInfo = {'generated': 0, 'lost': 0, 'processed': 0}
38
39        while requestInfo['generated'] < incomingRequestAmount:
40          DoStep(generator, operators, processors, requestInfo)
41
42        while requestInfo['lost'] + requestInfo['processed'] <
              incomingRequestAmount:
43          DoStep(generator, operators, processors, requestInfo, False)
44
45        return requestInfo
46
47
48      def main():
49        clientGenerator = Generator(UniformDistribution(8, 12))
50
51        firstQueue = []
52        secondQueue = []
53
54        operators = [
55          Operator(firstQueue, UniformDistribution(15, 25)),
56          Operator(firstQueue, UniformDistribution(30, 50)),
57          Operator(secondQueue, UniformDistribution(20, 60))
58        ]
59
60        processors = [
61          Processor(firstQueue, UniformDistribution(15, 15)),
62          Processor(secondQueue, UniformDistribution(30, 30))
63        ]
```

```
64
65      totalRequests = 3000
66
67      tStart = time()
68      res = modeling(clientGenerator, operators, processors,
            totalRequests)
69
70      print('time (secs)', time() - tStart)
71      for key in res.keys():
72      print(key, res[key])
73
74    print('lost', res['lost'] / totalRequests)
75
76
77    if __name__ == '__main__':
78      main()
```

Листинг 2 – Generator.py

```
1   class Generator:
2     def __init__(self, distribution):
3       self.timeDistribution = distribution
4       self.finishTime = 0
5       self.requestId = -1
6
7     def UpdateTime(self, dt):
8       self.finishTime -= dt
9
10      if self.finishTime <= 1e-5:
11        self.finishTime = self.timeDistribution.generate()
12        self.requestId += 1
13        return self.requestId
14
15      return None
```

Листинг 3 – Processor.py

```
class Processor:
  def __init__(self, requestsQueue, distribution):
    self.timeDistribution = distribution
    self.busy = False
    self.requestsQueue = requestsQueue
    self.finishTime = 0

  def UpdateTime(self, dt):
    self.finishTime -= dt

    if self.busy and self.finishTime <= 1e-5:
      self.busy = False
      self.curRequest = None
      return 0

    if not self.busy and len(self.requestsQueue) != 0:
      self.requestsQueue.pop(0)
      self.finishTime = self.timeDistribution.generate()
      self.busy = True
      return 1

    return 2
```

Листинг 4 – Operator.py

```
class Operator:
  def __init__(self, queue, distribution):
    self.timeDistribution = distribution
    self.busy = False
    self.queue = queue
    self.curRequest = None
    self.finishTime = 0

  def acceptRequest(self, request):
```

```
10          self.busy = True
11          self.curRequest = request
12          self.finishTime = self.timeDistribution.generate()
13
14      def finishCurRequest(self):
15          self.queue.append(self.curRequest)
16          self.busy = False
17          self.curRequest = None
18
19      def UpdateTime(self, dt):
20          self.finishTime -= dt
21          if self.busy and self.finishTime <= 1e-5:
22              self.finishCurRequest()
23              return 0
24          return 2
```