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 курсу: «Моделирование»

Тема: «Моделирование работы информационного центра»

Студент: Ле Ни Куанг

Группа: ИУ7И-76Б

Преподаватель: Рудаков И. В.

Москва

2021

Содержание

1 Задание **3**

2 Теоритическая часть **3**

3 Результаты **4**

4 Листинг кода **5**

1 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.

2 Теоритическая 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:

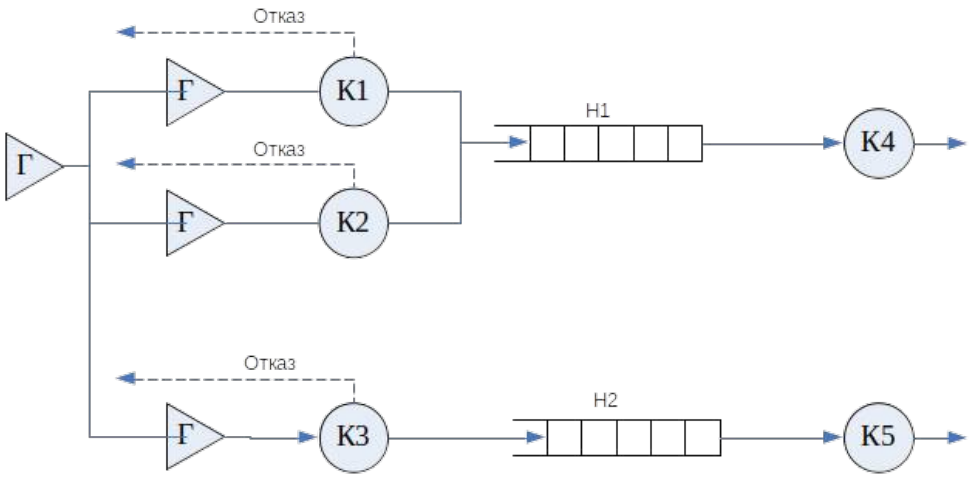
- Режим нормального обслуживания, т.е. клиент выбирает одного из свободных операторов, отдавая предпочтение тому у которого меньше номер.
- Режим отказа в обслуживании клиента, когда все операторы заняты

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.

Вероятность отказа:

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



3 Результаты

```
Task 100:   t=984.30    Генератор, generated
            t=984.30    Оператор 1, rejected
            t=984.30    Оператор 2, rejected
            t=984.30    Оператор 3, rejected
            t=984.30    Failed,

Task 200:   t=1996.90   Генератор, generated
            t=1996.90   Оператор 1, accepted
            t=2023.90   Компьютер 1, accepted
            t=2038.90   Succeed,

Task 300:   t=3003.20   Генератор, generated
            t=3003.20   Оператор 1, rejected
            t=3003.20   Оператор 2, accepted
            t=3039.30   Компьютер 1, accepted
            t=3054.30   Succeed,

Генератор  :    обработал 300 запросов
Оператор 1 :    обработал 120 запросов, отклонил 180 запросов
Оператор 2 :    обработал  61 запросов, отклонил 119 запросов
Оператор 3 :    обработал  55 запросов, отклонил  64 запросов
Компьютер 1:    обработал 181 запросов
Компьютер 2:    обработал  55 запросов

Число обслуженных клиентов: 236
Число клиентов получивших отказ: 64
Вероятность отказа: 0.2133
```

4 Листинг кода

Листинг 1 – Программная реализация информационного центра

```
if __name__ == '__main__':
    from system.generator import *
    from system.model import Endpoint, simulate
    from system.service import Service, QueueService

    success = Endpoint('Succeed')
    fail = Endpoint('Failed')

    comp1 = QueueService(Const(15), success, name='Компьютер 1')
    comp2 = QueueService(Const(30), success, name='Компьютер 2')

    op3 = Service(Uniform(20, 60), comp2, fail, 'Оператор 3')
    op2 = Service(Uniform(30, 50), comp1, op3, 'Оператор 2')
    op1 = Service(Uniform(15, 25), comp1, op2, 'Оператор 1')

    n_tasks = 300
    requests = RequestGenerator(Uniform(8, 12), n_tasks, op1,
       name='Генератор')

    nodes = [requests, op1, op2, op3, comp1, comp2]
    end_condition = lambda: success.count + fail.count == n_tasks
    simulate(nodes, end_condition)

    for node in nodes:
        print(node)
    print('\nЧисло обслуженных клиентов:', success.count)
    print('Число клиентов получивших отказ:', fail.count)
    print(f'Вероятность отказа: {fail.count / n_tasks:.4f}')
```

Листинг 2 – Программный интерфейс

```
from __future__ import annotations

from numpy.random import default_rng


class IGenerator:
    generator = default_rng()

    def generate(self) -> float:
        raise NotImplementedError


class Task:
    def __init__(self, task_id):
        self.task_id = task_id
        self.log = []

    def add_log(self, node_name, time, event):
        self.log.append([node_name, time, event])

    def __str__(self):
        res = f'Task {self.task_id}:\t'
        for e in self.log:
            res += f't={e[1]:.2f}\t{e[0]}, {e[2]}\n\t\t\t'
        return res


class Node:
    timer = 0

    def __init__(self, next_node: Node = None, fail_node: Node = None,
       name: str = ''):
        self.next_node = next_node
        self.fail_node = fail_node
        self.n_succeed = 0
        self.n_failed = 0
        self.name = name

    def handle(self, task: Task):
        raise NotImplementedError

    def elapse(self, time):
        pass

    def next(self, task: Task):
        if self.next_node: self.next_node.handle(task)
        self.n_succeed += 1

    def fail(self, task: Task):
        task.add_log(self.name, self.timer, 'rejected')
        if self.fail_node: self.fail_node.handle(task)
```

```
        self.n_failed += 1

    def __str__(self):
        res = f'{self.name:11}:\tобработал {self.n_succeed:3d} запросов'
        if self.n_failed:
            res += f',\tотклонил {self.n_failed:3d} запросов'
        return res


class Endpoint(Node):
    def __init__(self, name=''):
        super().__init__()
        self.name = name
        self.count = 0

    def handle(self, task):
        if task.task_id % 100 == 0:
            task.add_log(self.name, Node.timer, '')
            print(task)
        self.count += 1


class GNode(Node):
    """
    Node whose processing time depends on generator
    """
    EPS = 10e-4

    def __init__(self, generator: IGenerator, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._generator = generator
        self._remaining_time = 0
        self._is_ready = True

    def start_process(self):
        r = -1
        while r <= 0:
            r = self._generator.generate()
        self._remaining_time = r
        self._is_ready = False

    def is_ready(self) -> bool:
        return self._is_ready


def simulate(nodes: [Node], func_end_condition, dt=10e-2):
    Node.timer = 0
    while not func_end_condition():
        for node in nodes:
            node.elapse(dt)
        Node.timer += dt
```