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 №6

По курсу: «Моделирование»

Тема: «Система параллельно обслуживает старый и новый версии»

Студент: Ле Ни Куанг

Группа: ИУ7И-76Б

Преподаватель: Рудаков И. В.

Москва

2021

Содержание

1 Задание

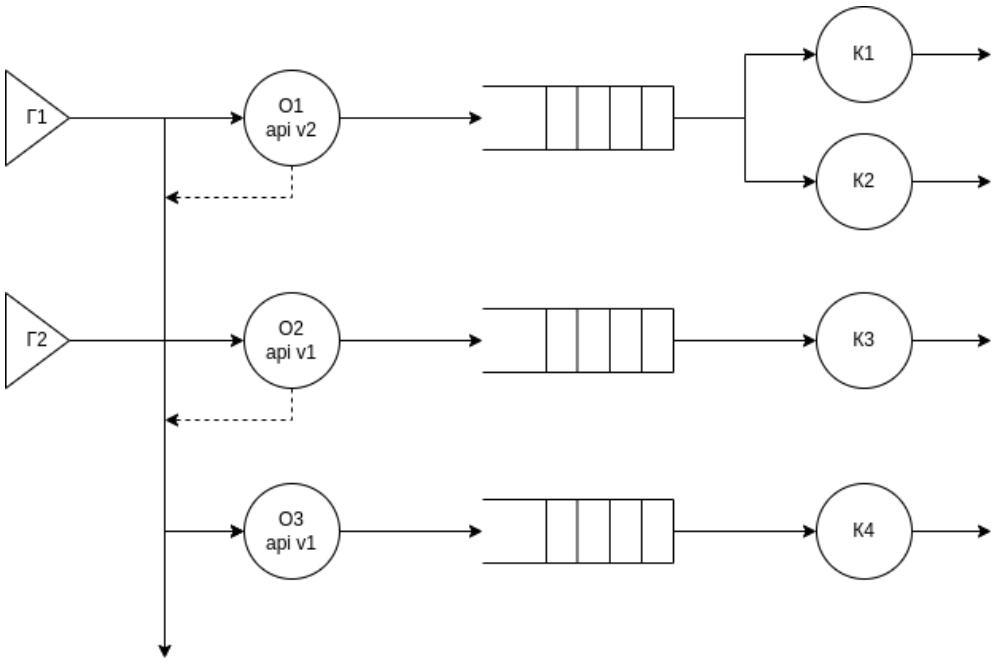
Запросы к серверу бывают двух типов: запросы к любому АПИ или запросы, которые определяют использование старого АПИ. Новый АПИ вернется к старому АПИ, если новый сервер занят. Если все три имеющихся оператора заняты, запросу отказывают в обслуживании.

Генератор, который генерирует запросы любого типа, и тот, который генерирует только запросы к старому АПИ, имеет время для обработки запроса, следуя распределению $U(10, 15)$ и $U(10, 18)$ соответственно. Три оператора имеют разную производительность, время обработки запроса которых соответствует правилам $U(10, 20), U(20, 25), U(15, 30)$. Четыре компьютера, время обработки запроса которых соответствует правилам $N(20, 4^2), N(30, 5^2), N(20, 4^2), N(30, 5^2)$.

Промоделировать процесс обработки 500 запросов.

2 Теоритическая часть

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

```
Запрос 400: t=2621.80   Генератор 1, сгенерирован
            t=2621.80   Оператор1 (v2), отказ
            t=2621.80   Оператор2 (v1), отказ
            t=2621.80   Оператор3 (v1), отказ
            t=2621.80   отклонен,

Запрос 500: t=3278.80   Генератор 1, сгенерирован
            t=3278.80   Оператор1 (v2), отказ
            t=3278.80   Оператор2 (v1), принят
            t=3299.90   Компьютер 3, принят
            t=3315.30   обработан,

Генератор 1:    обработал 266 запросов
Генератор 2:    обработал 238 запросов
Оператор1 (v2): обработал 148 запросов, отклонил 117 запросов
Оператор2 (v1): обработал 118 запросов, отклонил 236 запросов
Оператор3 (v1): обработал 109 запросов, отклонил 127 запросов
Компьютер 1:    обработал 109 запросов
Компьютер 2:    обработал  38 запросов
Компьютер 3:    обработал 118 запросов
Компьютер 4:    обработал 108 запросов

Число обслуженных клиентов: 373
Число клиентов получивших отказ: 127
Вероятность отказа: 0.2540
```

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
    from system.queue import Queue

    success = Endpoint('обработан')
    failure = Endpoint('отклонен')

    q1 = Queue()
    q2 = Queue()
    q3 = Queue()

    comp1 = QueueService(Normal(20, 4), q1, success, name='Компьютер 1')
    comp2 = QueueService(Normal(30, 5), q1, success, name='Компьютер 2')
    comp3 = QueueService(Normal(20, 4), q2, success, name='Компьютер 3')
    comp4 = QueueService(Normal(30, 5), q3, success, name='Компьютер 4')

    op3 = Service(Uniform(15, 30), q3, failure, 'Оператор3 (v1)')
    op2 = Service(Uniform(20, 25), q2, op3, 'Оператор2 (v1)')
    op1 = Service(Uniform(10, 20), q1, op2, 'Оператор1 (v2)')

    n_tasks = 500
    anyApiRequests = RequestGenerator(Uniform(10, 15), n_tasks, op1,
        name='Генератор 1')
    oldApiRequests = RequestGenerator(Uniform(10, 18), n_tasks, op2,
        name='Генератор 2')

    nodes = [anyApiRequests, oldApiRequests, op1, op2, op3, comp1, comp2,
        comp3, comp4]
    end_condition = lambda: success.count + failure.count == n_tasks
    simulate(nodes, end_condition)

    for node in nodes:
        print(node)
    print('\nЧисло обслуженных клиентов:', success.count)
    print('Число клиентов получивших отказ:', failure.count)
    print(f'Вероятность отказа: {failure.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'Запрос {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, 'отказ')
        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
```