МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1**

**по дисциплине**  
 **«МНОГОАГЕНТНОЕ МОДЕЛИРОВАНИЕ»**

Выполнил студент группы 45/2                                       Т. Э. Айрапетов

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Курс    4

Отчет принял доктор физико-математических наук, профессор                                                                                       А.И. Миков

Краснодар

2024 г.

**Задание**: Имеется n агентов и окружающая их среда. В окружающей среде в случайные моменты времени появляются клиенты, требующие обслуживания. Каждый клиент делает публичную заявку на обслуживание с указанием его сложности. Агенты могут предоставить такое обслуживание клиентам. Работают они индивидуально, но координируют между собой действия.

Каждый агент создает очередь — места для клиентов, ожидающих обслуживания. На обслуживание клиенты выбираются из очереди в соответствии с дисциплиной FIFO. Агенты наблюдают за окружающей средой: как только появляется новый клиент, они принимают совместное решение о том, кто его возьмет в свою очередь. Работа всей системы заканчивается по окончании обслуживания m-го клиента (клиент с номером m + 1 уже не принимается ни в какую из очередей).

По окончании работы агенты составляют совместный отчет, в котором каждый указывает свой id, сколько клиентов он обслужил п какое время на это затратил. Записи в отчете упорядочены по убыванию количества клиентов; при одинаковом количестве клиентов — по возрастанию времени.

Правила для окружающей среды:

Время начинается с t = 0.

Первый клиент объявляется через случайный интервал времени с равномерным распределением на [a, b]. a и b — вещественные числа. Интервал между i-м и (i +1)-м клиентом также случайный с равномерным распределением на [a, b]. Все интервалы взаимно независимы.

Для каждого клиента задается сложность его обслуживания — целочисленная случайная величина в пределах от 1 до 10 единиц времени.

Правила для совещания агентов:

Задачу берет тот агент, у которого наименьшая текущая загрузка (суммарная сложность обслуживания клиентов, находящихся в очереди, плюс время дообслуживания клиента, находящегося на обслуживании у агента). При равной загрузке агентов задачу берет первый в списке id.

**Решение**.

Для решения задания опишем класс Agent, представляющий собой агента, содержащий поля id (идентификатор), queue (очередь клиентов), count (кол-во обработанных клиентов), tek\_time (время обработки клиентов в очереди), time (общее затраченное время). Также в классе для простоты определим метод add\_client, добавляющий клиента в очередь и обновляющий значения соответствующих полей и переопределим методы для сравнения агентов между собой. Класс для клиента описывать не будем, так как он представляет собой лишь значение сложности обработки.

В основном цикле программы генерируется интервал, через который появится очередной клиент. За этот интервал из очередей агентов могли уйти клиенты, поэтому в цикле обрабатываются очереди всех клиентов. Затем находится наименее загруженный агент на основании поля tek\_time и к нему в очередь добавляется клиент через функцию add\_client.

Текст программы на языке Python:

*from* random *import* randint, random

*import* numpy *as* np

t = 0

a, b = 1, 10

complexity = (1, 100)

n, m = 4, 40

class Agent:

def \_\_init\_\_(*self*, *id*):

*self*.id = *id*

*self*.queue = []

*self*.tek\_time = 0

*self*.count = 0

*self*.time = 0

def add\_client(*self*, *k*):

*self*.tek\_time += *k*

*self*.queue.append(*k*)

*self*.count += 1

*self*.time += *k*

def \_\_str\_\_(*self*):

*return* *self*.time

def \_\_lt\_\_(*self*, *other*):

*return* *self*.tek\_time < *other*.tek\_time

def \_\_eq\_\_(*self*, *other*):

*return* *self*.tek\_time == *other*.tek\_time

def \_\_gt\_\_(*self*, *other*):

*return* *self*.tek\_time > *other*.tek\_time

agents = np.array([Agent(i) *for* i *in* range(n)])

*for* i *in* range(m):

dt = a + random()\*b

*for* ag *in* agents:

ag.tek\_time = max(ag.tek\_time-dt, 0)

dt1 = dt

*while* ag.queue and dt1 >= ag.queue[0]:

dt1 -= ag.queue[0]

ag.queue.pop(0)

*if* ag.queue:

ag.queue[0] -= dt1

agents[agents.argmin()].add\_client(randint(\*complexity))

res = sorted(agents, *key*=lambda *x*:[m-*x*.count, *x*.time])

*for* i *in* range(n):

print(f"Агент {res[i].id} клиентов {res[i].count}, время {res[i].time}")

Результаты тестов:



Рисунок 1 - Пример работы программы (n=4, m=10, сложность от 1 до 10)

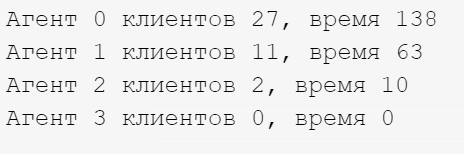


Рисунок 2 - Пример работы программы (n=4, m=40, сложность от 1 до 10)

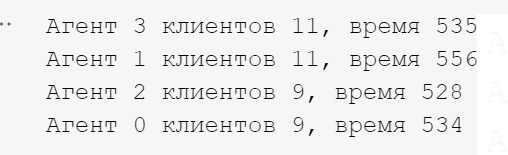


Рисунок 3 - Пример работы программы (n=4, m=40, сложность от 1 до 100)

**Вывод**:

На тестовых запусках программы видно насколько важна правильная балансировка нагрузки в зависимости от исходных данных, то есть, при увеличении средней сложности обработки клиента при относительно малых промежутках между заявками, стоит запастись достаточным количеством агентов. В обратной же ситуации, часть агентов будет простаивать без дела.