МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет по лабораторной работе № 12
«Синхронизация потоков в языке программирования Python»

по дисциплине «Основы программной инженерии»

Выполнила:
Первых Дарья Александровна,
2 курс, группа ПИЖ-б-о-20-1

Проверил:
Доцент кафедры
инфокоммуникаций, Воронкин Р.А.

Ставрополь, 2022 г

ВЫПОЛНЕНИЕ

```
def order_processor(name):
    while True:
        with cv:
            # Wait while queue is empty
            while q.empty():
                cv.wait()
            try:
                # Get data (order) from queue
```

```
main
C:\Users\podar\study\anaconda\envs\LR12\python.exe
thread 2: order 0
thread 1: order 1
thread 1: order 2
thread 2: order 3
thread 2: order 4
thread 2: order 5
thread 2: order 6
thread 1: order 7
thread 1: order 8
thread 2: order 9
thread 2: stop
thread 1: stop
thread 3: stop
```

Рисунок 1 – Условные переменные

```python
from threading import Thread, BoundedSemaphore
from time import sleep, time


ticket_office = BoundedSemaphore(value=3)


def ticket_buyer(number):
    start_service = time()
    with ticket_office:
        sleep(1)
        print(f"client {number}, service time: {time() - start_service}")


if __name__ == "__main__":
    buyer = [Thread(target=ticket_buyer, args=(i,)) for i in range(5)]
    for b in buyer:
```

if __name__ == "__main__" › for b in buyer

main

```
client 0, service time: 1.0214064121246338
client 2, service time: 1.0214064121246338client 1, service time: 1.0214064121246338

client 3, service time: 2.0332438945770264client 4, service time: 2.0332438945770264
```

Рисунок 2 – Семафоры

```python
def worker(name: str):
    event.wait()
    print(f"Worker: {name}")


if __name__ == "__main__":
    # Clear event
    event.clear()
    # Create and start workers
    workers = [Thread(target=worker, args=(f"wrk {i}",)) for i in range(5)]
    for w in workers:
        w.start()

    print("Main thread")
```

if __name__ == "__main__"

main

```
C:\Users\podar\study\anaconda\envs\LR12\python.exe "C:/Users/podar/study/PyCharm
Main thread
Worker: wrk 2Worker: wrk 1

Worker: wrk 0
Worker: wrk 4
Worker: wrk 3
```

Рисунок 3 – События

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from threading import Timer


if __name__ == "__main__":
    timer = Timer(interval=3, function=lambda: print("Message from Timer!"))
    timer.start()
```

```
main
C:\Users\podar\study\anaconda\envs\LR12\python.exe "C:/Users/podar/study/PyCharm
Message from Timer!
```

Рисунок 4 – Таймеры

```
def f2(x):
    print("Calc part2")
    store.append(x*2)
    sleep(1)
    br.wait()


if __name__ == "__main__":
    Thread(target=f1, args=(3,)).start()
    Thread(target=f2, args=(7,)).start()
    br.wait()
    print("Result: ", sum(store))
```

```
main
C:\Users\podar\study\anaconda\envs\LR12\pytho
Calc part1
Calc part2
Result:  23
```

Рисунок 5 – Барьеры

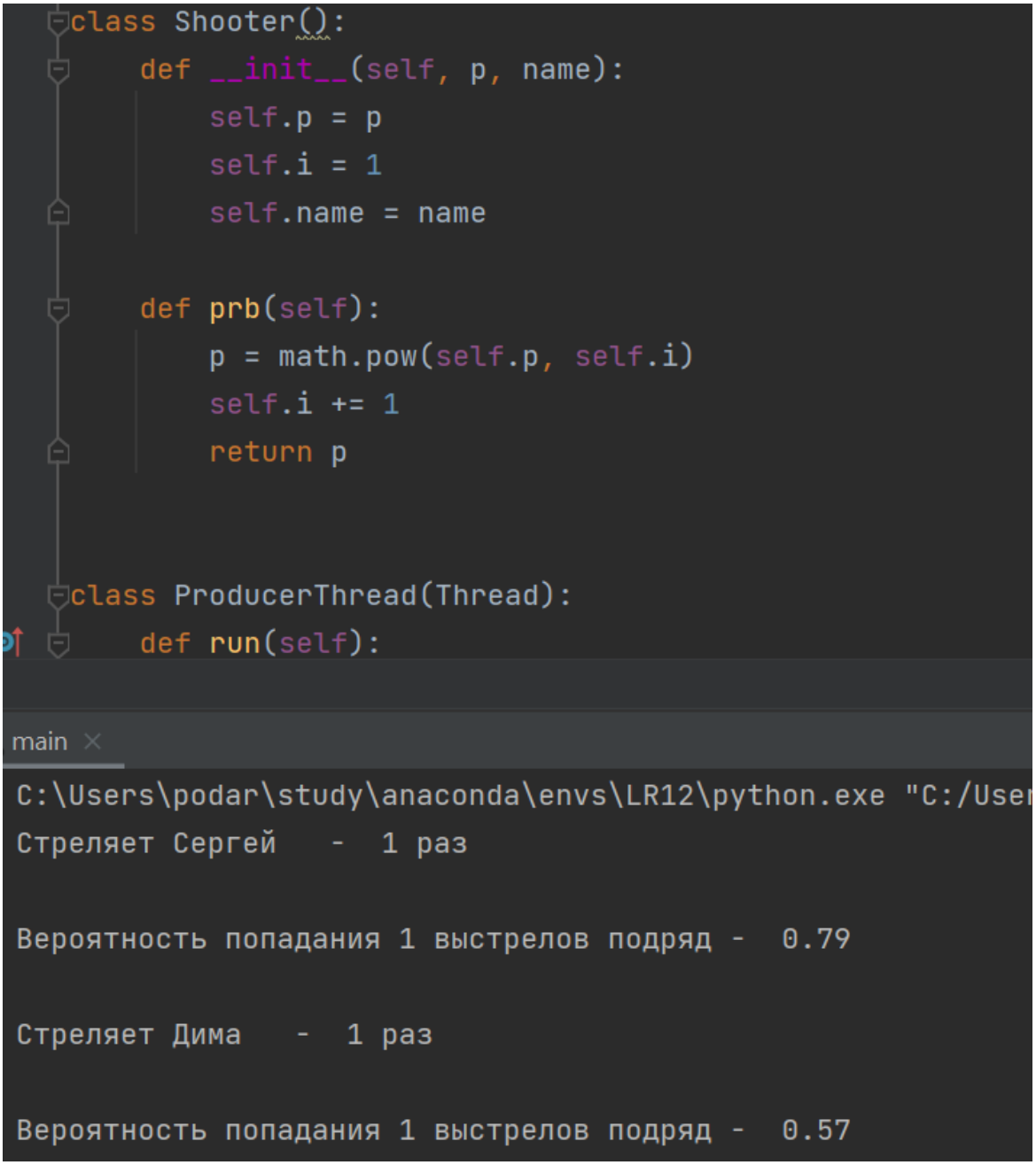

Рисунок 6 – Решение задачи по теории вероятности

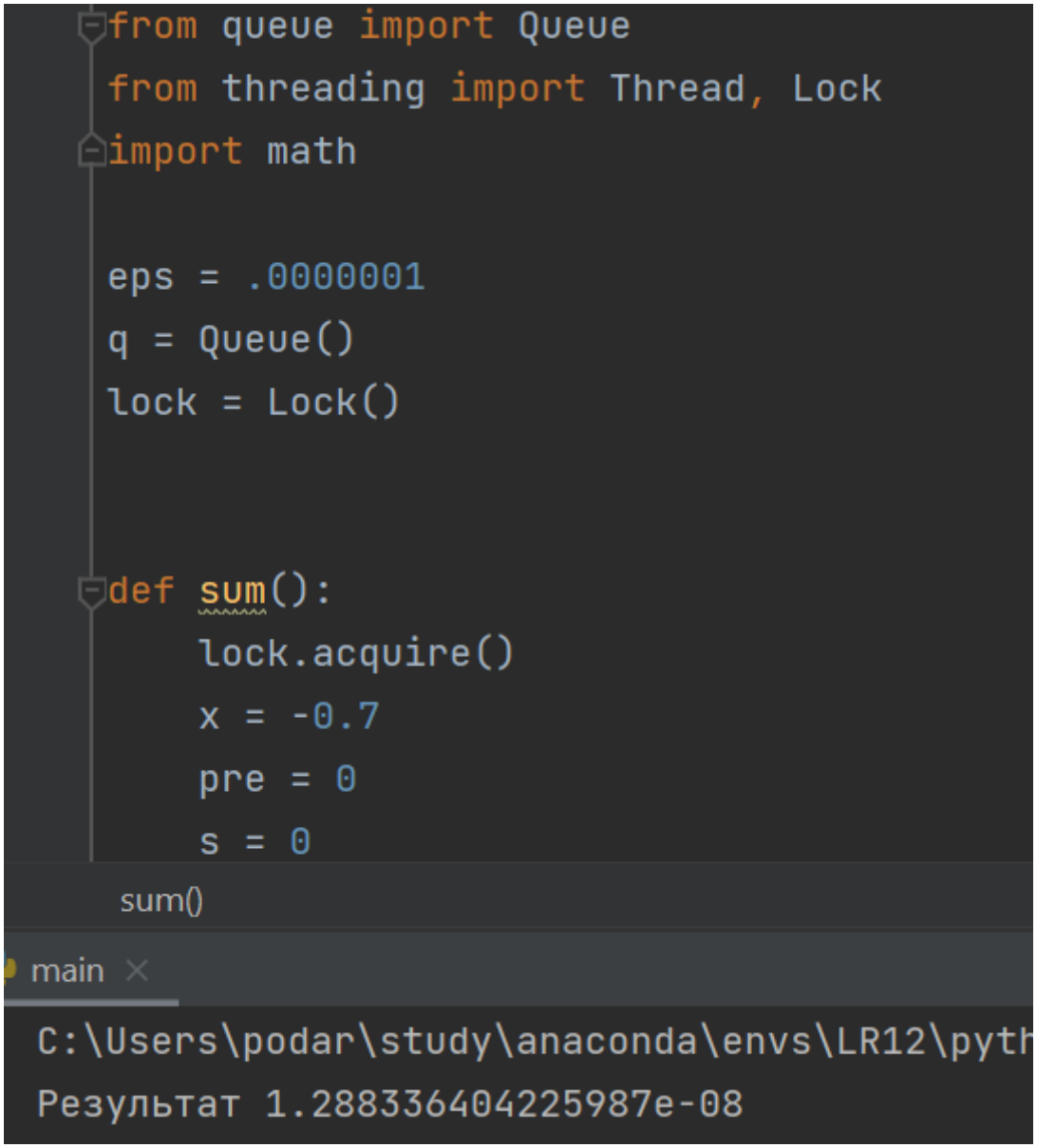

Рисунок 7 – Индивидуальное задание

ВОПРОСЫ

1. Каково назначение и каковы приемы работы с Lock-объектом?

Lock-объект может находиться в двух состояниях: захваченное (заблокированное) и не захваченное (не заблокированное, свободное). После создания он находится в свободном состоянии. Для работы с Lock-объектом используются методы acquire() и release(). Если Lock свободен, то вызов метода acquire() переводит его в заблокированное состояние. Повторный вызов acquire() приведет к блокировке инициировавшего это действие потокадо тех пор, пока Lock не будет разблокирован каким-то другим потоком с помощью метода release(). Вызов метода release() на свободном Lock-объекте приведет к выбросу исключения RuntimeError.

2. В чем отличие работы с RLock-объектом от работы с Lock-объектом?

В отличии от рассмотренного выше Lock-объекта RLock может освободить только тот поток, который его захватил. Повторный захват потоком уже захваченного RLock-объекта не блокирует его. RLock-объекты поддерживают возможность вложенного захвата, при этом освобождение происходит только после того, как был выполнен release() для внешнего acquire(). Сигнатуры и назначение методов release() и acquire() RLock-объектов совпадают с приведенными для Lock, но в отличии от него у RLock нет метода locked(). RLock-объекты поддерживают протокол менеджера контекста.

3. Как выглядит порядок работы с условными переменными?

Порядок работы с условными переменными выглядит так:

- На стороне Consumer'a: проверить доступен ли ресурс, если нет, то перейти в режим ожидания с помощью метода wait(), и ожидать оповещение от Producer'a о том, что ресурс готов и с ним можно работать. Метод wait() может быть вызван с таймаутом, по истечении которого поток выйдет из состояния блокировки и продолжит работу.
- На стороне Producer'a: произвести работы по подготовке ресурса, после того, как ресурс готов оповестить об этом ожидающие потоки с помощью методов notify() или notify_all(). Разница между ними в том, что notify() разблокирует только один поток (если он вызван без параметров), а notify_all() все потоки, которые находятся в режиме ожидания.

4. Какие методы доступны у объектов условных переменных?

При создании объекта Condition вы можете передать в конструктор объект Lock или RLock, с которым хотите работать. Перечислим методы объекта Condition с кратким описанием:

acquire(*args) – захват объекта- блокировки.

release() – освобождение объекта-блокировки.

wait(timeout=None) – блокировка выполнения потока до оповещения о снятии блокировки. Через параметр timeout можно задать время ожидания оповещения о снятии блокировки. Если вызвать wait() на Условной переменной, у которой предварительно не был вызван acquire(), то будет выброшено исключение RuntimeError.

wait_for(predicate, timeout=None) – метод позволяет сократить количество кода, которое нужно написать для контроля готовности ресурса и ожидания оповещения.

notify(n=1) – снимает блокировку с остановленного методом wait() потока. Если необходимо разблокировать несколько потоков, то для этого следует передать их количество через аргумент n.

notify_all() – снимает блокировку со всех остановленных методом wait() потоков.

5. Каково назначение и порядок работы с примитивом синхронизации “семафор”?

Реализация классического семафора, предложенного Дейкстрой. Суть его идеи заключается в том, при каждом вызове метода acquire() происходит уменьшение счетчика семафора на единицу, а при вызове release() – увеличение. Значение счетчика не может быть меньше нуля, если на момент вызова acquire() его значение равно нулю, то происходит блокировка потока до тех пор, пока не будет вызван release().

Семафоры поддерживают протокол менеджера контекста.

Для работы с семафорами в Python есть класс Semaphore, при создании его объекта можно указать начальное значение счетчика через параметр value. Semaphore предоставляет два метода:

- acquire(blocking=True, timeout=None) – если значение внутреннего счетчика больше нуля, то счетчик уменьшается на единицу и метод возвращает True. Если значение счетчика равно нулю, то вызвавший данный метод поток блокируется, до тех пор, пока не будет кем- то вызван метод release(). Дополнительно при вызове метода можно указать параметры blocking и timeout, их назначение совпадает с acquire() для

Lock.

- release() – увеличивает значение внутреннего счетчика на единицу.

  Существует ещё один класс, реализующий алгоритм семафора BoundedSemaphore, в отличии от Semaphore, он проверяет, чтобы значение внутреннего счетчика было не больше того, что передано при создании объекта через аргумент value, если это происходит, то выбрасывается исключение ValueError.

  С помощью семафоров удобно управлять доступом к ресурсу, который имеет ограничение на количество одновременных обращений к нему (например, количество подключений к базе данных и т.п.)

6. Каково назначение и порядок работы с примитивом синхронизации “событие”?

События по своему назначению и алгоритму работы похожи на рассмотренные ранее условные переменные. Основная задача, которую они решают – это взаимодействие между потоками через механизм оповещения. Объект класса Event управляет внутренним флагом, который сбрасывается с помощью метода clear() и устанавливается методом set(). Потоки, которые используют объект Event для синхронизации блокируются при вызове метода wait(), если флаг сброшен.

Методы класса Event:

- is_set() – возвращает True если флаг находится в взведенном состоянии.
- set() – переводит флаг в взведенное состояние.
- clear() – переводит флаг в сброшенное состояние.
- wait(timeout=None) – блокирует вызвавший данный метод поток если флаг соответствующего Event-объекта находится в сброшенном состоянии. Время нахождения в состоянии блокировки можно задать через параметр timeout.

7. Каково назначение и порядок работы с примитивом синхронизации “таймер”?

Модуль threading предоставляет удобный инструмент для запуска задач по таймеру – класс Timer. При создании таймера указывается функция, которая будет выполнена, когда он сработает. Timer реализован как поток, является наследником от Thread, поэтому для его запуска необходимо вызвать start(), если необходимо остановить работу таймера, то вызовите cancel().

Конструктор класса Timer:

Timer(interval, function, args=None, kwargs=None)

Параметры:

- interval – количество секунд, по истечении которых будет вызвана функция function.
- function – функция, вызов которой нужно осуществить по таймеру.
- args, kwargs – аргументы функции function.

Методы класса Timer:

cancel() – останавливает выполнение таймера

8. Каково назначение и порядок работы с примитивом синхронизации “барьер”?

Последний инструмент для синхронизации работы потоков, который мы рассмотрим, является Barrier. Он позволяет реализовать алгоритм, когда необходимо дождаться завершения работы группы потоков, прежде чем продолжить выполнение задачи.

Конструктор класса:

Barrier(parties, action=None, timeout=None)

Параметры:

- parties – количество потоков, которые будут работать в рамках барьера.
- action – определяет функцию, которая будет вызвана, когда потоки будут освобождены (достигнут барьера).
- timeout – таймаут, который будет использовать как значение по умолчанию для методов wait().

Свойства и методы класса:

wait(timeout=None) – блокирует работу потока до тех пор, пока не будет

получено уведомление либо не пройдет время указанное в timeout.

reset() – переводит Barrier в исходное (пустое) состояние. Потокам, ожидающим уведомления, будет передано исключение BrokenBarrierError.

abort() – останавливает работу барьера, переводит его в состояние “разрушен” (broken). Все текущие и последующие вызовы метода wait() будут завершены с ошибкой с выбросом исключения BrokenBarrierError.

parties – количество потоков, которое нужно для достижения барьера.

n_waiting – количество потоков, которое ожидает срабатывания барьера.

broken – значение флага равное True указывает на то, что барьер находится в “разрушенном” состоянии.

9. Сделайте общий вывод о применении тех или иных примитивов синхронизации в зависимости от решаемой задачи.

Блокировка используется для основной защиты совместно используемых ресурсов. Многократные потоки могут попытаться получить блокировку, но только один поток может фактически содержать ее в любой момент времени. В то время как тот поток содержит блокировку, другие потоки должны ожидать. Существует несколько различных типов блокировок, отличаясь в основном по тому, что потоки делают при ожидании для получения их.

Семафор во многом как блокировка, за исключением того, что конечное число потоков может содержать его одновременно. Семафоры могут думаться как являющийся во многом как груды маркеров. Многократные потоки могут взять эти маркеры, но, когда нет ни одного оставленного, поток должен ожидать, пока другой поток не возвращает тот.