

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития  
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №2.24

**Дисциплина: «Анализ данных»**  
**Тема: «Синхронизация потоков в языке  
программирования Python»**

Выполнил:  
Елифанов Алексей Александрович  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной  
техники и автоматизированных систем  
», очная форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Роман Александрович

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

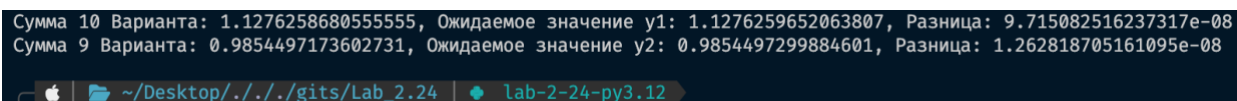
Ставрополь, 2024 г.

Цель: приобретение навыков использования примитивов синхронизации в языке программирования Python версии 3.x.

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.

2. Выполнил индивидуальное задание: Для своего индивидуального задания лабораторной работы 2.23 необходимо организовать конвейер, в котором сначала в отдельном потоке вычисляется значение первой функции, после чего результаты вычисления должны передаваться второй функции, вычисляемой в отдельном потоке. Потоки для вычисления значений двух функций должны запускаться одновременно.



```
Сумма 10 Варианта: 1.1276258680555555, Ожидаемое значение y1: 1.1276259652063807, Разница: 9.715082516237317e-08
Сумма 9 Варианта: 0.9854497173602731, Ожидаемое значение y2: 0.9854497299884601, Разница: 1.262818705161095e-08
```

Рисунок 1. Результат работы

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

```
import math
from threading import Thread, Lock
```

```
lock = Lock()
```

```
# 10 V
```

```
def sum1(x, eps, s_dict):
    s = 0
    n = 0
    while True:
        k = 2 * n
        term = x**k / math.factorial(k)
        if abs(term) < eps:
            break
        else:
            s += term
            n += 1
```

```
with lock:
    s_dict["s1"] = s
```

```
# 9 V
def sum2(x, eps, s_dict):
    s = 0
    n = 0
    while True:
        k = 2 * n + 1
        term = (-1) ** n * x**k / math.factorial(k)
        if abs(term) < eps:
            break
        else:
            s += term
            n += 1
    with lock:
        s_dict["s2"] = s
```

```
def compair(s, y1, y2):
    while True:
        with lock:
            if ("s1" in s and "s2" in s):
                s1 = s["s1"]
                s2 = s["s2"]

                print(
                    f"Сумма 10 Варианта: {s1},"
                    f" Ожидаемое значение y1: {y1}, Разница: {abs(s1 - y1)}"
                )
                print(
                    f"Сумма 9 Варианта: {s2},"
                    f" Ожидаемое значение y2: {y2}, Разница: {abs(s2 - y2)}"
                )
                break
```

```
def main():
    s = {}

    eps = 10**-7
    # 10 V
    x1 = 1 / 2
    y1 = (math.e**x1 + math.e**-x1) / 2
```

```

# 9 V
x2 = 1.4
y2 = math.sin(x2)

thread1 = Thread(target=sum1, args=(x1, eps, s))
thread2 = Thread(target=sum2, args=(x2, eps, s))
thread3 = Thread(target=compair, args=(s, y1, y2))

# Запуск потоков
thread1.start()
thread2.start()
thread3.start()

if __name__ == "__main__":
    main()

```

Контрольные 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 потока до тех пор, пока Lock не будет разблокирован каким-то другим потоком с помощью метода `release()`. Вывоз метода `release()` на свободном Lock-объекте приведет к выбросу исключения `RuntimeError`. Lock-объекты поддерживают протокол менеджера контекста, это позволяет работать с ними через оператор `with`.

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 отличие от него у `RLock` нет метода `locked()`. `RLock`-объекты поддерживают протокол менеджера контекста.

### 3. Как выглядит порядок работы с условными переменными?

Основное назначение условных переменных – это синхронизация работы потоков, которая предполагает ожидание готовности некоторого ресурса и оповещение об этом событии. Наиболее явно такой тип работы выражен в паттерне `Producer-Consumer` (Производитель – Потребитель). Условные переменные для организации работы внутри себя используют `Lock`- или `RLock`-объекты, захватом и освобождением которых управлять не придется, хотя и возможно, если возникнет такая необходимость.

Порядок работы с условными переменными выглядит так:

- На стороне `Consumer`'а: проверить доступен ли ресурс, если нет, то перейти в режим ожидания с помощью метода `wait()`, и ожидать оповещение от `Producer`'а 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`'а: 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 отличие 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

Пример работы с таймером:

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

Он позволяет реализовать алгоритм, когда необходимо дождаться завершения работы группы потоков, прежде чем продолжить выполнение задачи.

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.

Выбор примитива синхронизации зависит от задачи:

`Lock/RLock` для эксклюзивного доступа.

Условные переменные для ожидания изменения состояния.

Семафоры для ограничения числа активных потоков.

События для сигнализации состояний.

Таймеры для выполнения действий по времени.

Барьеры для координации групп потоков.

Вывод: в результате выполнения работы были приобретены навыки использования примитивов синхронизации в языке программирования Python версии 3.x.