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

ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчёт по лабораторной работе №2.24 по дисциплине:

Основы программной инженерии

Выполнила:
студент группы ПИЖ-б-о-20-1
Лазарева Дарья Олеговна

Проверил:
доцент кафедры инфокоммуникаций
Романкин Р.А.

Ставрополь, 2022 г.

ВЫПОЛНЕНИЕ:

1. Условные переменные

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/2.24/ex1.py
thread 1: order 0
thread 2: order 1
thread 2: order 2
thread 1: order 3
thread 1: order 4
thread 2: order 5
thread 3: order 6
thread 1: order 7
thread 2: order 8
thread 2: order 9
thread 1: stop
thread 3: stop
thread 2: stop
```

2. Семафоры

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/2.24/ex2.py
client 0, service time: 1.0134267807006836client 2, service time: 1.0134918689727783client 1, service time: 1.0134918689727783


client 3, service time: 2.0161404609680176client 4, service time: 2.0161404609680176


Process finished with exit code 0
```

3. События

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/2.24/ex3.py
Main thread
Worker: wrk 0
Worker: wrk 3Worker: wrk 4Worker: wrk 2


Worker: wrk 1

Process finished with exit code 0
```

4. Таймеры

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/2.24/ex4.py
Message from Timer!

Process finished with exit code 0
```

5. Барьеры

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/2.24/ex5.py
Calc part1
Calc part2
Result:  23

Process finished with exit code 0
```

Индивидуальное задание 1: Разработать приложение, в котором выполнить решение вычислительной задачи (например, задачи из области физики, экономики, математики, статистики и т. д.) с помощью паттерна “Производитель-Потребитель”.

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

from threading import Thread
import time
import random
from queue import Queue
import math

queue = Queue()


class Shooter():
    def __init__(self, p, name):
        self.p = p
        self.i = 1
        self.name = name

    def prb(self):
        p = math.pow(self.p, self.i)
        self.i += 1
        return p


class ProducerThread(Thread):
    def run(self):
        count_shoots = 0
        while count_shoots < 20:
```

```
            i = random.randint(0, 5)
            queue.put(shoot[i])
            print(f"Стреляет {shoot[i].name}  -  {shoot[i].i} раз \n")
            time.sleep(0.1)
            count_shoots += 1


class ConsumerThread(Thread):
    def run(self):
        shoot_consumed = 0
        while shoot_consumed < 20:
            num = queue.get()
            queue.task_done()
            print (f"Вероятность попадания {num.i} выстрелов подряд -
{num.prb()} \n")
            time.sleep(0.1)
            shoot_consumed += 1


if __name__ == '__main__':
    str_1 = Shooter(0.5, "Иван ")
    str_2 = Shooter(0.79, "Сергей ")
    str_3 = Shooter(0.57, "Дима ")
    shoot = [str_1, str_2, str_3]
    ProducerThread().start()
    ConsumerThread().start()
```

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/2.24/idz.py
Стреляет Дима   -  1 раз

Вероятность попадания 1 выстрелов подряд -  0.57

Стреляет Иван   -  1 раз
Вероятность попадания 1 выстрелов подряд -  0.5


Стреляет Иван   -  2 раз
Вероятность попадания 2 выстрелов подряд -  0.25
```

Индивидуальное задание 2: 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.

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


from threading import Thread, Lock
from math import factorial, sin
from queue import Queue


eps = .0000001
q = Queue()
lock = Lock()


def inf_sum(x):
    lock.acquire()
    a = 1
    summa = 1
    i = 1
    prev = 0
    while abs(summa - prev) > eps:
        a = ((-1)**i * x**(2*i + 1)) / factorial(2 * i + 1)
        prev = summa
        if i % 2 == 0:
            summa += a
        else:
            summa += -1 * a
        i += 1
    q.put(summa)
    lock.release()


def check_ans(inf_res, d_res):
    print(f"The sum of an infinite series is: {inf_res}")
    print(f"The calculated answer is: {d_res}")


if __name__ == '__main__':
    num = int(input("Enter the number to calculate: "))
    check = sin(num)
    thread1 = Thread(target=inf_sum, args=(num,)).start()
    thread2 = Thread(target=check_ans, args=(q.get(), check)).start()
```

```
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/2.24/idz2.py
Enter the number to calculate: 44
The sum of an infinite series is: 6.425800057179654e+18
The calculated answer is: 0.017701925105413577
```

Ответы на 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().

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

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?

1) 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.

2) 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?

acquire(*args) – захват объекта- блокировки.

release() – освобождение объекта-блокировки.

wait(timeout=None) – блокировка выполнения потока до оповещения о снятии блокировки.

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().

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

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.

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().

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.

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.