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

Институт цифрового развития
Кафедра инфокоммуникаций

«Лабораторная работа 2.23 Управление потоками в Python»

ОТЧЕТ
по лабораторной работе №27
дисциплины
«Основы программной инженерии»

Выполнил:
Луценко Дмитрий Андреевич
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Лабораторная работа Лабораторная работа 2.24 Синхронизация потоков в языке программирования Python

Цель работы: приобретение навыков написания многопоточных приложений на языке программирования Python версии 3.x.

Ход работы:

. Разработать приложение, в котором выполнить решение вычислительной задачи (например, задачи из области физики, экономики, математики, статистики и т. д.) с помощью паттерна “Производитель-Потребитель”, условие которой предварительно необходимо согласовать с преподавателем.

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

from threading import Thread, Condition
from queue import Queue
import time


G = 6.6743e-11 #Гравитационная постоянная
MASS_EARTH = 5.97e24 #Масса земли
RADIUS_EARTH = 6.37e6 #Радиус земли

q_pos = Queue()
q_vel = Queue()
q_time = Queue()
cv = Condition()


def satellite_motion():
    """
    Функция для вычисления координат и скоростей спутника.
    """
    x0 = RADIUS_EARTH + 500e3
    v0 = ((G * MASS_EARTH) / x0) ** 0.5
    dt = 1
    t = 0
    x, y, z = x0, 0, 0
    vx, vy, vz = 0, v0, 0

    while t < 10:
        with cv:
            t += dt
            r = (x ** 2 + y ** 2 + z ** 2) ** 0.5
            Fg = -G * MASS_EARTH / r ** 2

            ax = Fg * x / r
            ay = Fg * y / r
            az = Fg * z / r

            vx += ax * dt
            vy += ay * dt
            vz += az * dt

            x += vx * dt
            y += vy * dt
            z += vz * dt
```

```
            q_pos.put((x, y, z))
            q_vel.put((vx, vy, vz))
            q_time.put(t)

            time.sleep(0.5)

def print_motion():
    """
    Функция для вывода текущих координат и скоростей спутника
    """
    while True:
        with cv:
            try:
                x, y, z = q_pos.get_nowait()
                vx, vy, vz = q_vel.get_nowait()
                t = q_time.get_nowait()
                print(f"Координаты({t}): ({x}, {y}, {z})")
                print(f"Скорости({t}): ({vx}, {vy}, {vz})")
            except:
                pass
            time.sleep(0.5)


if __name__ == '__main__':
    th_motion = Thread(target=satellite_motion)
    th_print = Thread(target=print_motion)
    th_motion.start()
    th_print.start()
```

Листинг 1 – Индивидуальное задание 1

```
Координаты(1): (6869991.557587257, 7615.732108330199, 0.0)
Скорости(1): (-8.442412743209829, 7615.732108330199, 0.0)
Координаты(2): (6869974.6727565825, 15231.454857814355, 0.0)
Скорости(2): (-16.88483067376579, 7615.722749484155, 0.0)
Координаты(3): (6869949.345513166, 22847.15888958342, 0.0)
Скорости(3): (-25.32724341687363, 7615.704031769067, 0.0)
Координаты(4): (6869915.575872568, 30462.834844756853, 0.0)
Скорости(4): (-33.76964059766895, 7615.675955173433, 0.0)
Координаты(5): (6869873.363860727, 38078.473364454105, 0.0)
Скорости(5): (-42.21201184123002, 7615.638519697254, 0.0)
Координаты(6): (6869822.709513955, 45694.065089806136, 0.0)
Скорости(6): (-50.65434677259046, 7615.591725352033, 0.0)
Координаты(7): (6869763.612878938, 53309.600661966906, 0.0)
Скорости(7): (-59.09663501675202, 7615.535572160772, 0.0)
Координаты(8): (6869696.07401274, 60925.07072212488, 0.0)
Скорости(8): (-67.53886619869733, 7615.470060157977, 0.0)
Координаты(9): (6869620.092982796, 68540.46591151453, 0.0)
Скорости(9): (-75.98102994340269, 7615.395189389656, 0.0)
Координаты(10): (6869535.66986692, 76155.77687142785, 0.0)
Скорости(10): (-84.42311587585073, 7615.310959913318, 0.0)
```

Рисунок 1 – Результат выполнения программы для индивидуального задания

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, Condition
from math import log, sqrt
from queue import Queue


EPS = .0000001

q = Queue()
cv = Condition()

def sum_func(x):
```

```
    with cv:
        summ = x
        prev = 0
        i = 1
        while abs(summ - prev) > EPS:
            prev = summ
            summ += x ** (i * 2 + 1) / (i * 2 + 1)
            i += 1

        q.put(summ)


def check_func(x):
    with cv:
        checking = q.get()
        res = log(sqrt((1 + x) / (1 - x)))
        print(f"Sum is: {checking}")
        print(f"Check: {res}")


if __name__ == '__main__':
    x = 0.35
    th1 = Thread(target=sum_func, args=(x,))
    th2 = Thread(target=check_func, args=(x,))
    th1.start()
    th2.start()
```

Листинг 2 – Код индивидуального задания 2

```
Sum is: 0.3654437434397286
Check: 0.3654437542713962
```

Рисунок 2 – Результат выполнения программы

Вывод: в ходе выполнения лабораторной работы приобретены навыки написания многопоточных приложений на языке программирования Python версии 3.x.

Ответы на контрольные вопросы:

1. Каково назначение и каковы приемы работы с Lock-объектом. Lock-объект используется для синхронизации доступа к общим ресурсам из нескольких потоков. Приемы работы с Lock-объектом включают вызов метода acquire() для получения блокировки и release() для освобождения блокировки.

2. В чем отличие работы с RLock-объектом от работы с Lock-объектом. RLock-объект представляет собой рекурсивную блокировку и может быть захвачен несколько раз одним и тем же потоком. В отличие от

Lock объекта, приемы работы с RLock-объектом включают вызов метода acquire() и release() в тех же потоках в любом количестве.

3. Как выглядит порядок работы с условными переменными? Для работы с условными переменными необходимо создать объект класса Condition. Затем потоки могут вызывать методы wait(), notify() и notify_all(), чтобы ожидать определенного условия и уведомлять другие потоки о его выполнении.

4. Какие методы доступны у объектов условных переменных? У объектов условных переменных доступны методы wait(), notify() и notify_all().

5. Каково назначение и порядок работы с примитивом синхронизации "семафор"? Семафор используется для синхронизации доступа к ограниченному количеству ресурсов. Для создания семафора в Python используется класс Semaphore. При получении блокировки при помощи метода acquire() счетчик семафора уменьшается, а при освобождении блокировки методом release() увеличивается.

6. Каково назначение и порядок работы с примитивом синхронизации "событие"? Событие используется для синхронизации потоков на выполнении какого-то события, например, завершении задания. В Python для этого используется класс Event. Потоки могут ждать на выполнение события при помощи метода wait(), а событие можно установить методом set() и снять методом clear().

7. Каково назначение и порядок работы с примитивом синхронизации "таймер"? Таймер используется для выполнения задач через определенный промежуток времени. В Python для этого можно использовать класс Timer. При создании объекта таймера необходимо указать интервал времени, после которого выполнится задача. Путем вызова метода start() таймер запускается, а метод cancel() останавливает выполнение задачи.

8. Каково назначение и порядок работы с примитивом синхронизации "барьер"? Барьер используется для ожидания завершения выполнения задач несколькими потоками. Для этого в Python используется

класс Barrier. Создается объект барьера с указанием количества потоков и методами wait() для ожидания завершения задач и reset() для возврата барьера в исходное состояние.

9. Сделайте общий вывод о применении тех или иных примитивов синхронизации в зависимости от решаемой задачи. Выбор конкретного примитива синхронизации зависит от решаемой задачи. Lock-объекты и RLock-объекты используются для синхронизации доступа к общим ресурсам, семафоры - для синхронизации ограниченного количества доступа к ресурсам, события - для уведомления потоков об определенных событиях, таймеры - для отложенного выполнения задач, а барьеры - для синхронизации одновременного завершения работы нескольких потоков.