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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №10
дисциплины «анализ данных»

Выполнил:
Середа Кирилл Витальевич
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Синхронизация потоков в языке программирования Python

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

Ход работы:

Создал файл (ind.py) в котором при помощи Lock и barier синхронизировал работу потоков рис.2 и организовал конвейер, в котором от результата выполнения работы двух других потоков приходят данные и сравниваются они с контрольным значением.

На рис. 1 представлена функция, реализующая конвейер:

```
def show_results(results):
    barrier.wait()
    with lock:
        sum1, sum2 = results[0], results[1]
        control_value1 = pi / 4
        control_value2 = - log(2 * sin(pi / 2))

        print(f"x1 = {pi/2}")
        print(f"Сумма ряда 1: {round(sum1, 4)}")
        print(f"Контрольное значение ряда 1: {round(control_value1, 4)}")
        print(f"Проверка 1: {round(sum1, 4) == round(control_value1, 4)}")

        print(f"x2 = {pi}")
        print(f"Сумма ряда 2: {round(sum2, 4)}")
        print(f"Контрольное значение ряда 2: {round(control_value2, 4)}")
        print(f"Проверка: {round(sum2, 4) == round(control_value2, 4)}")
```

Рисунок 1 – Функция check_result

```
def sum_1(results, x):

    a = sin(x)
    S, k = a, 2
    # Найти сумму членов ряда.
    while fabs(a) > EPS:
        coef = 2 * k - 1
        a = sin(coef * x) / coef
        S += a
        k += 1

    with lock:
        results[0] = S
    barrier.wait()


# 12 Вариант
def sum_2(results, x):

    a = cos(x)
    S, k = a, 2
    # Найти сумму членов ряда.
    while fabs(a) > EPS:
        a = cos(k * x) / k
        S += a
        k += 1

    with lock:
        results[1] = S
    barrier.wait()
```

Рисунок 2 – Функции расчета рядов

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

Контрольные вопросы:

1. Назначение и приемы работы с Lock-объектом

Назначение: Lock-объект используется для обеспечения синхронизации доступа к общим ресурсам в многопоточной среде, позволяя только одному потоку за раз получить доступ к критической секции кода.

Приемы работы: Для работы с Lock-объектом необходимо сначала захватить блокировку с помощью метода acquire(), выполнить необходимые операции с общими ресурсами, а затем освободить блокировку с помощью метода release().

2. Отличие работы с RLock-объектом от работы с Lock-объектом

Основное отличие RLock-объекта от Lock-объекта заключается в том, что RLock позволяет потоку многократно захватывать блокировку, которая должна быть освобождена столько же раз, сколько была захвачена.

3. Порядок работы с условными переменными

Для работы с условными переменными необходимо сначала создать объект условной переменной с помощью метода threading.Condition(). Затем, при помощи методов wait(), notify() и notify_all() можно организовать ожидание и уведомление потоков о состоянии условной переменной.

4. Методы доступные у объектов условных переменных

Основные методы доступные у объектов условных переменных в Python включают wait(), notify() и notify_all().

5. Назначение и порядок работы с примитивом синхронизации "семафор"

Семафор используется для управления доступом к общим ресурсам в многопоточной среде. Для работы с семафором необходимо создать объект семафора с помощью метода threading.Semaphore() и использовать методы acquire() и release() для захвата и освобождения семафора соответственно.

6. Назначение и порядок работы с примитивом синхронизации "событие"

Событие используется для уведомления одного или нескольких потоков о том, что произошло определенное событие. Для работы с событием необходимо создать объект события с помощью метода threading.Event() и использовать методы set() и clear() для установки и сброса события соответственно.

7. Назначение и порядок работы с примитивом синхронизации "таймер"

Таймер используется для запуска выполнения определенной функции через определенный промежуток времени. Для работы с таймером необходимо создать объект таймера с помощью класса threading.Timer() и запустить его с помощью метода start().

8. Назначение и порядок работы с примитивом синхронизации "барьер"

Барьер используется для синхронизации нескольких потоков, ожидающих достижения определенной точки в программе. Для работы с барьером необходимо создать объект барьера с помощью класса threading.Barrier() и использовать метод wait() для ожидания достижения барьера.

9. Применение тех или иных примитивов синхронизации зависит от конкретной задачи. Например, если требуется управлять доступом к общим ресурсам, то Lock-объекты могут быть полезны для блокировки доступа к критическим секциям кода. В случае необходимости уведомления потоков о возникновении определенного события, применение условных переменных или событий может быть эффективным. Семафоры могут быть использованы для управления доступом к ресурсам с ограниченной пропускной способностью. Таймеры могут быть полезны для запуска выполнения определенной функции через определенный промежуток времени. Барьеры могут быть применены для синхронизации нескольких потоков, ожидающих

достижения определенной точки в программе. Таким образом, выбор примитива синхронизации зависит от конкретной задачи и требований к управлению параллельными процессами.