

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:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и
вычислительная техника (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных
систем», очная форма обучения

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических
наук, доцент, доцент кафедры
инфокоммуникаций

(подпись)

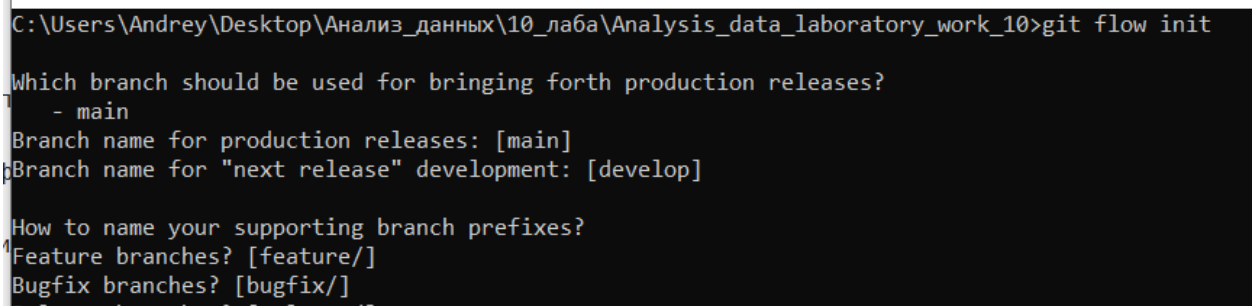
Ставрополь, 2024 г.

Тема: Синхронизация потоков в языке программирования Python.

Цель: приобрести навыки использования примитивов синхронизации в языке программирования Python версии 3.x.

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git-flow» (рис. 1).



```
C:\Users\Andrey\Desktop\Анализ_данных\10_лаба\Analysis_data_laboratory_work_10>git flow init
Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Hotfix branches? [hotfix/]
```

Рисунок 1 – Организация модели ветвления «git-flow»

Выполнение индивидуальных заданий:

Задание 1.

Необходимо с использованием многопоточности для заданного значения x найти сумму ряда S с точностью члена ряда по абсолютному значению и произвести сравнение полученной суммы с контрольным значением функции $y(x)$ для двух бесконечных рядов.

Необходимо доработать программу лабораторной работы 2.23, 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 ряда (Вариант 26 (1)):

1.
$$S = \sum_{n=0}^{\infty} \frac{x^n \ln^n 3}{n!} = 1 + \frac{x \ln 3}{1!} + \frac{x^2 \ln^2 3}{2!} + \dots; \quad x = 1; \quad y = 3^x.$$

Сумма ряда (Вариант 26 (2)):

2.
$$S = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \dots; x = 0,7; y = \frac{1}{1-x}.$$

Код программы данной задачи:

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

import math
from threading import Thread
from queue import Queue
import sympy as sp

"""
Необходимо с использованием многопоточности для заданного значения x найти
сумму ряда S
с точностью члена ряда по абсолютному значению и произвести сравнение
полученной суммы с
контрольным значением функции y(x) для двух бесконечных рядов.
Необходимо доработать программу лабораторной работы 2.23, 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.
(Вариант 26 (1 и 2)).
"""
E = 1e-7 # Точность

def series_1(x, eps, queue):
    """
    Функция вычисления суммы ряда задачи №1 (x = 1).
    """
    s = 0
    n = 0
    while True:
        term = x ** n * sp.log(3) ** n / math.factorial(n)
        if abs(term) < eps:
            break
        s += term
        n += 1
    queue.put(s)

def series_2(x, eps, queue):
    """
    Функция вычисления суммы ряда задачи №2 (x = 0,7).
    """
    s = 0
    n = 0
    while True:
        term = x ** n
        if abs(term) < eps:
            break
        s += term
        n += 1
    queue.put(s)
```

```

def main():
    """
    Главная функция программы.
    """
    # Определение символа n
    n = sp.symbols('n')

    x1 = 1
    control1 = sp.Sum(x1 ** n * sp.log(3) ** n / sp.factorial(n), (n, 0,
sp.oo)).evalf()

    x2 = 0.7
    control2 = sp.Sum(x2 ** n, (n, 0, sp.oo)).evalf()

    queue_1 = Queue()
    queue_2 = Queue()

    # Создание потоков для вычисления сумм.
    thread_1 = Thread(target=series_1, args=(x1, E, queue_1))
    thread_2 = Thread(target=series_2, args=(x2, E, queue_2))

    # Запуск созданных потоков.
    thread_1.start()
    thread_2.start()

    sum_1 = queue_1.get()
    sum_2 = queue_2.get()

    # Блокировка основного потока, пока эти два потока не завершатся.
    thread_1.join()
    thread_2.join()

    print(f"x1 = {x1}")
    print(f"Sum of series 1: {sum_1:.7f}")
    print(f"Control value 1: {control1:.7f}")
    print(f"Match 1: {round(sum_1, 7) == round(control1, 7)}")

    print(f"x2 = {x2}")
    print(f"Sum of series 2: {sum_2:.7f}")
    print(f"Control value 2: {control2:.7f}")
    print(f"Match 2: {round(sum_2, 7) == round(control2, 7)}")

if __name__ == '__main__':
    main()

```

Результаты работы данной программы при заданных значениях x (рис. 2).

```
C:\Users\Andrey\anaconda3\envs\lab_9\python.exe C:\Users\An
x1 = 1
Sum of series 1: 2.9999999
Control value 1: 2.7182818
Match 1: False
x2 = 0.7
Sum of series 2: 3.3333331
Control value 2: 1.0000000
Match 2: False
```

Рисунок 2 – Результаты работы программы

Ответы на контрольные вопросы:

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

Назначение: Lock (или блокировка) – это механизм синхронизации, предназначенный для предотвращения конфликтов доступа к общим ресурсам из нескольких потоков. Приемы работы: `acquire()`: Захватывает блокировку. Если блокировка уже захвачена другим потоком, текущий поток блокируется до ее освобождения. `release()` освобождает блокировку. Если есть ожидающие потоки, один из них получит блокировку.

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

RLock (Reentrant Lock) – это вариант Lock, который может быть захвачен несколько раз одним и тем же потоком. Однако, чтобы успешно освободить RLock, его также необходимо освободить столько раз, сколько было сделано захватов.

3. Как выглядит порядок работы с условными переменными?

Создать объект `Condition`, связанный с блокировкой. Использовать методы `wait()`, `notify()`, и `notify_all()` для организации ожидания и оповещения.

4. Какие методы доступны у объектов условных переменных?

`acquire()` захватывает связанную блокировку, `release()` освобождает связанную блокировку, `wait(timeout=None)` ожидает оповещения, освобождая блокировку. Может быть прерван методом `notify()` или по истечении времени. `notify(n=1)` будит один поток, ожидающий условную переменную. `notify_all()` будит все потоки, ожидающие условную переменную.

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

Семафор - это объект синхронизации, ограничивающий доступ к общему ресурсу. Он имеет счетчик, который уменьшается при захвате и увеличивается при освобождении. Порядок работы: создать семафор с начальным значением, использовать `acquire()` для захвата и `release()` для освобождения.

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

Событие – это сигнальный механизм, позволяющий одному потоку оповещать другие о том, что что-то произошло. Порядок работы: создать событие использовать `set()` для установки события и `clear()` для сброса. Другие потоки могут использовать `wait()` для блокировки до установки события.

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

Таймер – это событие, которое срабатывает через определенный интервал времени. Порядок работы: создать таймер с указанием интервала и функции, которая будет выполнена по истечении времени, запустить таймер.

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

Барьер – это точка синхронизации, где несколько потоков могут встречаться и дожидаться друг друга. Порядок работы: создать барьер с

указанием количества потоков, которые должны встретиться. Каждый поток использует `wait()` для ожидания других потоков. После того как все потоки встретились, они могут продолжить выполнение.

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

Выбор примитива синхронизации зависит от конкретной задачи. Например, `Lock` подходит для простых критических секций, `Condition` для ожидания определенного состояния, `Semaphore` для управления ресурсами, `Event` для сигнализации, `Timer` для отложенных действий, и `Barrier` для синхронизации нескольких потоков в одной точке.

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