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 №11
дисциплины «Анализ данных»

Вариант №2

Выполнила:
Беседина Инга Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических наук, доцент, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Управление процессами в Python

Цель: Приобретение навыков написания многозадачных приложений на языке программирования Python версии 3.x

Ход работы

Для своего индивидуального задания лабораторной работы 2.23 необходимо реализовать вычисление значений в двух функций в отдельных процессах.

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


"""
Вариант 2.
С использованием многопоточности для заданного значения x
найти сумму ряда S с точностью члена ряда по абсолютному значению e=10^-7
и произвести сравнение полученной суммы с контрольным значением функции
для двух бесконечных рядов. Необходимо реализовать вычисление значений
функций в отдельных процессах.
"""


from multiprocessing import Barrier, Manager, Process


def calc_sum1(x, eps, s, lock, br):
    sum1 = 0
    n = 0
    while True:
        el = x**n
        if abs(el) < eps:
            break
        sum1 += el
        n += 1
    with lock:
        s["s1"] = sum1
    br.wait()


def calc_sum2(x, eps, s, lock, br):
    sum2 = 0
    n = 0
    while True:
        t = n + 1
        el = (-1) ** n * x**n / 2**t
        if abs(el) < eps:
```

```
            break
        else:
            sum2 += el
            n += 1
    with lock:
        s["s2"] = sum2
    br.wait()


def res(s, y1, y2, br):
    br.wait()
    s1 = s["s1"]
    s2 = s["s2"]

    print(f"Вариант №2. Сумма ряда S: {s1}")
    print(
        f"Контрольное значение функции для бесконечного ряда: {y1}"
    )

    print(f"Вариант №3. Сумма ряда S: {s2}")
    print(
        f"Контрольное значение функции для бесконечного ряда: {y2}"
    )


def main(m):
    br = Barrier(3)
    lock = m.Lock()

    s = m.dict()
    e = 1e-7

    x1 = 0.7
    y1 = 1 / (1 - x1)

    x2 = 1.2
    y2 = 1 / (2 + x2)

    process1 = Process(target=calc_sum1, args=(x1, e, s, lock, br))
    process2 = Process(target=calc_sum2, args=(x2, e, s, lock, br))
    process3 = Process(target=res, args=(s, y1, y2, br))

    process1.start()
    process2.start()
    process3.start()

    process1.join()
```

```
    process2.join()
    process3.join()


if __name__ == "__main__":
    with Manager() as manager:
        main(manager)
```

```
(venv) PS C:\Rep\DA_11\Project> python indv.py
Вариант №2. Сумма ряда S: 3.333333083650555
Контрольное значение функции для бесконечного ряда: 3.3333333333333333
Вариант №3. Сумма ряда S: 0.31250004145136007
Контрольное значение функции для бесконечного ряда: 0.3125
```

Рисунок 1. Результат работы программы

Контрольные вопросы:

1. Создание и завершение процессов в Python: - Создание процесса: В Python для создания процессов можно использовать модуль multiprocessing. Процесс создается путем создания экземпляра класса Process и вызова метода start().

- Завершение процесса: Процесс завершается автоматически, когда выполнение функции внутри процесса завершается. Также можно использовать метод join(), чтобы дождаться завершения процесса.

2. Особенность создания классов-наследников от Process:

- При создании класса-наследника от multiprocessing.Process необходимо определить метод run(), который будет содержать код, выполняемый в новом процессе. При создании экземпляра класса-наследника и вызове метода start(), будет выполнен метод run() в новом процессе.

3. Принудительное завершение процесса:

- В Python нет прямой возможности принудительно завершить процесс, как, например, в операционной системе. Однако можно отправить сигнал SIGKILL, используя модуль os или signal, чтобы принудительно завершить процесс. Это может быть нежелательным, поскольку это может привести к непредсказуемому поведению программы.

4. Процессы-демоны: - Процессы-демоны (daemon processes) - это процессы, которые работают в фоновом режиме и обычно не имеют прямого взаимодействия с пользователем. Они могут выполнять различные служебные задачи, такие как обслуживание серверов или периодические задачи.

- Запуск процесса-демона: Для запуска процесса в режиме демона можно использовать модуль daemon из стандартной библиотеки Python. Этот модуль предоставляет удобные инструменты для создания процессов-демонов.

Вывод: В ходе выполнения лабораторной работы были приобретены навыки написания многозадачных приложений на языке программирования Python версии 3.x