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:
Данилецкий Дмитрий Витальевич
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

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

_____
(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

1. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python. Выполнил клонирование созданного репозитория.

2. Дополнил файл .gitignore необходимыми правилами.

3. Организовал созданный репозиторий в соответствие с необходимыми требованиями.

4. Добавил в файл README.md информацию о группе и ФИО студента, выполняющего лабораторную работу.

5. Выполнил индивидуальное задание. Привел в отчете скриншоты работы программы решения 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.. Вариант 7-8

$$S = \sum_{n=1}^{\infty} \frac{(-1)^{n+1} \sin nx}{n} = \sin x - \frac{\sin 2x}{2} + \ldots; \; x = -\frac{\pi}{2};$$

Рисунок 1. Функция варианта 7

$$S = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots; \; x = 2; \; y = \frac{e^x - e^{-x}}{2}.$$

Рисунок 2. Функция варианта 8

Рисунок 3. Результат работы программы индивидуального задания

Контрольные вопросы

1. Назначение и приемы работы с Lock-объектом

Назначение: Обеспечить эксклюзивный доступ к ресурсу в многопоточной среде.

Создание: lock = threading.Lock()

Захват: lock.acquire()

Освобождение: lock.release()

Использование с with: with lock:

2. Отличие работы с RLock-объектом от работы с Lock-объектом RLock (рекурсивная блокировка):

Позволяет одному потоку захватывать блокировку несколько раз. Требует столько же вызовов release, сколько было вызовов acquire.

Lock: Блокирует поток, если он пытается захватить блокировку повторно.

3. Порядок работы с условными переменными

Порядок работы:

Создание: condition = threading.Condition()

Ожидание события: with condition:

condition.wait()

Уведомление об событии: with condition:

condition.notify() # или condition.notify_all()

4. Методы, доступные у объектов условных переменных

wait(): Ожидание уведомления.

notify(): Уведомление одного ожидающего потока.

notify_all(): Уведомление всех ожидающих потоков.

acquire(): Захват внутренней блокировки.

release(): Освобождение внутренней блокировки.

5. Назначение и порядок работы с примитивом синхронизации “семафор”

Назначение: Управление доступом к ресурсу с ограниченной емкостью.

Порядок работы:

Создание: semaphore = threading.Semaphore(value)

Захват (уменьшение счётчика): semaphore.acquire()

Освобождение (увеличение счётчика): semaphore.release()

6. Назначение и порядок работы с примитивом синхронизации “событие”

Назначение: Синхронизация потоков через ожидание наступления события.

Порядок работы:

Создание: event = threading.Event()

Установка события: event.set()

Сброс события: event.clear()

Ожидание события: event.wait()

7. Назначение и порядок работы с примитивом синхронизации “таймер”

Назначение: Выполнение функции по истечении заданного времени.

Порядок работы:

Создание и запуск:

timer = threading.Timer(interval, function, args=None, kwargs=None)

timer.start()

Отмена: timer.cancel()

8. Назначение и порядок работы с примитивом синхронизации “барьер”

Назначение: Синхронизация группы потоков, чтобы они могли продолжить выполнение только после того, как все достигнут определенной точки.

Порядок работы:

Создание: barrier = threading.Barrier(parties)

Ожидание: barrier.wait()

9. Общий вывод о применении примитивов синхронизации

Lock: Для простого эксклюзивного доступа к ресурсу.

RLock: Для рекурсивного захвата блокировки одним потоком.

Condition: Для ожидания и уведомления о событиях.

Semaphore: Для ограничения доступа к ресурсу с несколькими экземплярами.

Event: Для ожидания и установки/сброса событий.

Timer: Для выполнения задачи через определенное время.

Barrier: Для синхронизации группы потоков до определенной точки.

Вывод: в результате выполнения работы были получены навыки по использованию примитивов синхронизации в языке программирования Python версии 3.x.