Сравнение стратегий обработки "медленных" задач

Основная цель

Сравнить четыре различных подхода к выполнению набора из **10-20 однотипных, но медленных задач** с точки зрения общего времени выполнения и эффективности использования ресурсов.

Требования к реализации

Создайте структуру, которая позволяет выполнять одинаковый набор задач, используя четыре разные стратегии:

1. Имитация "Медленной" Задачи (Функция-обработчик)

- **Задача:** Имитировать внешнюю операцию, которая тратит время на ожидание (например, сетевой запрос, чтение с диска).
- **Реализация:** Создайте функцию, которая принимает идентификатор задачи (ID) и выполняет **ожидание** в течение случайного времени (например, от 0.5 до 2.0 секунд), используя time.sleep() или asyncio.sleep().
- **Вывод:** Функция должна распечатать, что она завершила работу, и вернуть результат (например, строку с обработанным ID).

2. Четыре Стратегии Запуска

Примените функцию-обработчик для выполнения всех 10-20 задач, используя следующие механизмы:

- **Синхронный подход (База):** Запускать задачи строго **последовательно**, ожидая завершения каждой перед началом следующей.
- **Многопоточный подход (threading):** Запускать задачи параллельно, используя **потоки**.
- **Многопроцессорный подход (multiprocessing):** Запускать задачи параллельно, используя **процессы**.

- **Асинхронный подход (asyncio):** Запускать задачи параллельно, используя **асинхронное программирование** (при этом важно, чтобы функция-обработчик использовала await для имитации ожидания).

3. Измерение и Сравнение

- **Измерение:** В начале и конце каждой стратегии используйте time.perf_counter() для измерения **общего времени выполнения** всего набора задач.
- **Сравнение:** Соберите результаты измерений и выведите их в виде итоговой сводки, чтобы наглядно показать разницу во времени.

Углубление понимания (Опционально)

Для максимальной пользы от задания, вы можете провести дополнительный эксперимент, изменив тип нагрузки:

1. **Текущая нагрузка (I/O-bound):** Это то, что мы описали выше (time.sleep / asyncio.sleep). Она имитирует операции, ограниченные **вводом/выводом** (I/O), где большая часть времени тратится на ожидание, а не на вычисления.
    - **Ожидаемый результат:** Потоки (threading) и асинхронность (asyncio) должны быть самыми быстрыми.
2. **Измененная нагрузка (CPU-bound):** Замените **ожидание** на функцию, которая выполняет **интенсивные математические вычисления** (например, вычисление множества чисел Фибоначчи или перемножение больших матриц) без какого-либо sleep. Это имитирует операции, ограниченные **процессором** (CPU).
    - **Ожидаемый результат:** Процессы (multiprocessing) должны быть самыми быстрыми, так как они обходят **Global Interpreter Lock (GIL)** и могут использовать несколько ядер CPU.