**Дипломная работа на тему «Сравнение методов асинхронного программирования»  
ФИО: Зайцев Андрей Андреевич**

**Введение**

**Обоснование выбора темы**

Асинхронное программирование играет важную роль в разработке современных приложений, где важно эффективно использовать ресурсы процессора и избегать блокировки операций. Разработка приложений с использованием Python предоставляет несколько подходов к организации асинхронности: библиотека asyncio, потоковая многозадачность через модуль threading и многопроцессорность с помощью multiprocessing. Каждый из этих подходов имеет свои сильные и слабые стороны, что делает их подходящими для разных типов задач.

**Цели и задачи исследования**

Цель исследования: сравнить три подхода к реализации асинхронного программирования в Python — asyncio, threading и multiprocessing — для выполнения различных типов задач и их оценки по производительности и применимости.

Задачи исследования:

1. Провести обзор библиотек asyncio, threading и multiprocessing.
2. Реализовать тестовые задачи с использованием каждого подхода.
3. Провести измерение производительности в различных сценариях: обработка I/O, интенсивные вычисления и смешанные задачи.
4. Проанализировать результаты и дать рекомендации по применению каждого подхода.

**Основные понятия и определения**

Для лучшего понимания темы работы рассмотрим основные понятия и термины, связанные с асинхронным программированием.

1. **Асинхронное программирование (Asynchronous Programming):** метод написания кода, позволяющий выполнять операции без ожидания завершения других операций. Это важно для задач с интенсивным вводом-выводом (I/O).
2. **I/O-операции:** действия, связанные с вводом и выводом данных, такие как чтение или запись файлов, сетевые запросы и взаимодействие с базами данных. Эти операции часто занимают много времени, что делает их подходящими для асинхронного выполнения.
3. **asyncio:** стандартная библиотека Python, предоставляющая возможности для создания событийного цикла, задач и управления асинхронными операциями. Использует ключевые слова async и await для упрощения работы с асинхронными функциями.
4. **Threading (Потоковая многозадачность):** подход к параллельному выполнению задач в одном процессе путем создания потоков. Потоки разделяют общую память, что делает их подходящими для задач с интенсивным обменом данными, но может приводить к состояниям гонки (race conditions).
5. **Multiprocessing (Многопроцессорность):** метод параллельного выполнения задач с использованием нескольких процессов. Каждый процесс имеет свою выделенную память, что позволяет избегать проблем синхронизации, но увеличивает накладные расходы на создание и коммуникацию между процессами.
6. **Эвент-луп (Event Loop):** центральный элемент asyncio, отвечающий за управление выполнением задач, таймеров и обработкой событий. Эвент-луп запускает и координирует выполнение асинхронных операций.
7. **Параллелизм:** выполнение нескольких задач одновременно с использованием нескольких процессоров или ядер. Пример — multiprocessing.
8. **Конкурентность (Concurrency):** управление несколькими задачами одновременно, но не обязательно выполнение их в одно и то же время. Пример — asyncio и threading.

Эти определения служат основой для дальнейшего анализа и разработки решений в рамках работы.

**Методы и подходы к разработке**

**Подходы к реализации асинхронного программирования**

**asyncio**

Библиотека asyncio позволяет организовывать асинхронные операции, используя механизм событийного цикла. Основные преимущества asyncio:

* Низкие накладные расходы на переключение задач.
* Простая работа с сетевыми операциями и вводом-выводом.
* Поддержка современного синтаксиса Python (async/await).

Использование asyncio подходит для задач, где важна работа с I/O-операциями, такими как обработка сетевых запросов, взаимодействие с базами данных и файловыми системами.

**threading**

Модуль threading позволяет создавать потоки для выполнения задач параллельно. Основные преимущества:

* Подходит для задач, требующих разделения задач между потоками.
* Легкость реализации для задач, требующих одновременной обработки нескольких операций.

Однако threading может сталкиваться с ограничением GIL (Global Interpreter Lock), что снижает эффективность для вычислительных задач.

**multiprocessing**

Модуль multiprocessing предоставляет возможность выполнять задачи в отдельных процессах. Основные преимущества:

* Полный доступ ко всем ядрам процессора.
* Отсутствие ограничений GIL, что делает его подходящим для вычислительно-емких задач.

Однако использование multiprocessing связано с большими накладными расходами на создание процессов и обмен данными между ними.

**Этапы разработки тестовых задач**

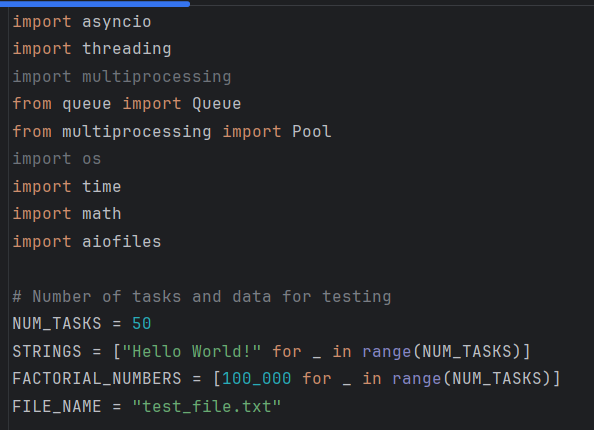
1. **Определение типов задач:**
   * I/O-интенсивные задачи (например, чтение и запись файлов, выполнение сетевых запросов).
   * Вычислительные задачи (например, вычисление чисел Фибоначчи или факториала).
2. **Реализация каждой задачи с использованием asyncio, threading и multiprocessing:**
   * Создание асинхронных функций для asyncio.
   * Реализация многопоточности с использованием threading.
   * Использование процессов для параллельной обработки в multiprocessing.
3. **Измерение производительности:**
   * Время выполнения каждой задачи.
   * Использование ресурсов процессора и памяти.
4. **Анализ полученных результатов:**
   * Сравнение эффективности подходов для каждого типа задачи.
   * Определение наиболее подходящего подхода для различных сценариев.

**Разработка тестовых задач**

**Реализация задач с использованием подходов**

Для проведения сравнения asyncio, threading и multiprocessing были выбраны следующие тестовые задачи:

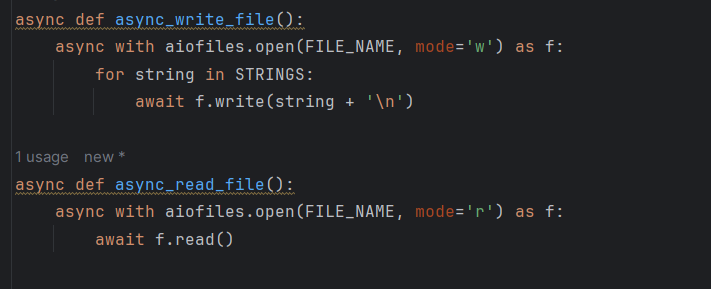
1. **I/O-интенсивная задача:** чтение и запись большого количества файлов.
2. **Вычислительная задача:** нахождение факториала большого числа для набора входных данных.

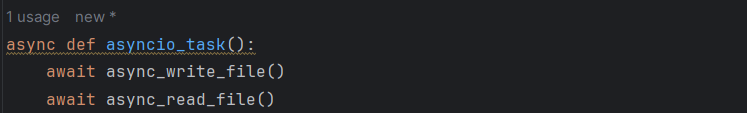


**Реализация I/O-интенсивной задачи**

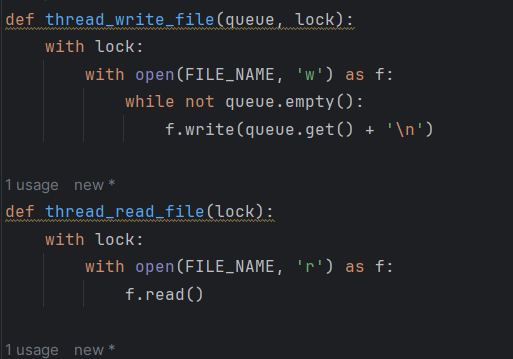
**asyncio:**

* Использован модуль aiofiles для асинхронного чтения и записи файлов.
* Асинхронные функции позволяют выполнять операции с минимальной задержкой при переключении контекста.





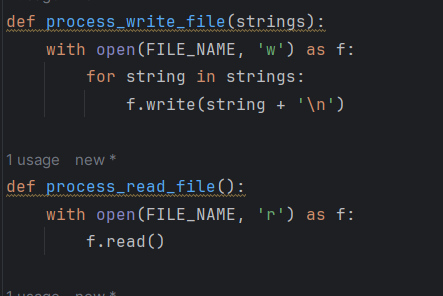
**threading:**

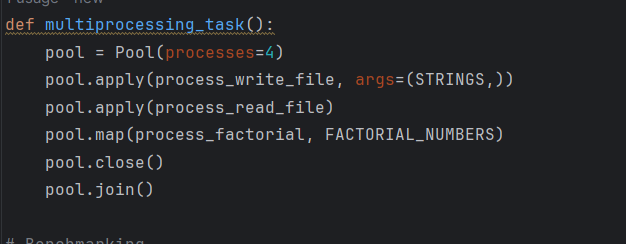
* Реализованы потоки, каждый из которых выполняет операции с файлами.
* Синхронизация потоков обеспечивается с помощью объекта Lock, чтобы избежать коллизий при доступе к ресурсам.



**multiprocessing:**

* Созданы процессы для выполнения операций чтения и записи.
* Используется очередь для передачи задач между процессами.



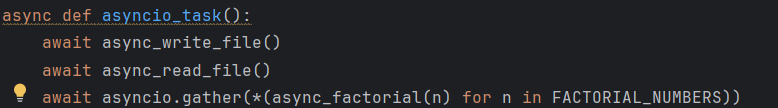


**Реализация вычислительной задачи**

**asyncio:**

* Получен результат с использованием метода gather: сборки всех результатов из асинхронного кода.

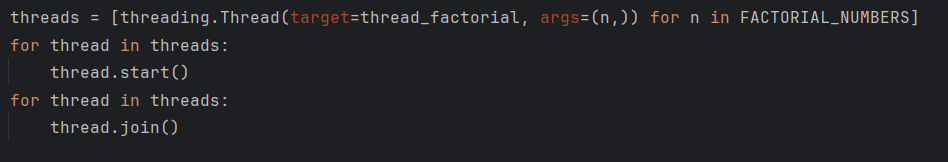




**threading:**

* Каждое вычисление выполняется в отдельном потоке. Однако эффективность ограничена GIL.

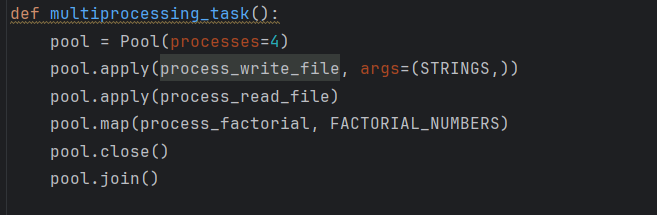




**multiprocessing:**

* Используется пул процессов для разделения вычислений между ядрами процессора. Это позволяет избежать ограничений GIL.

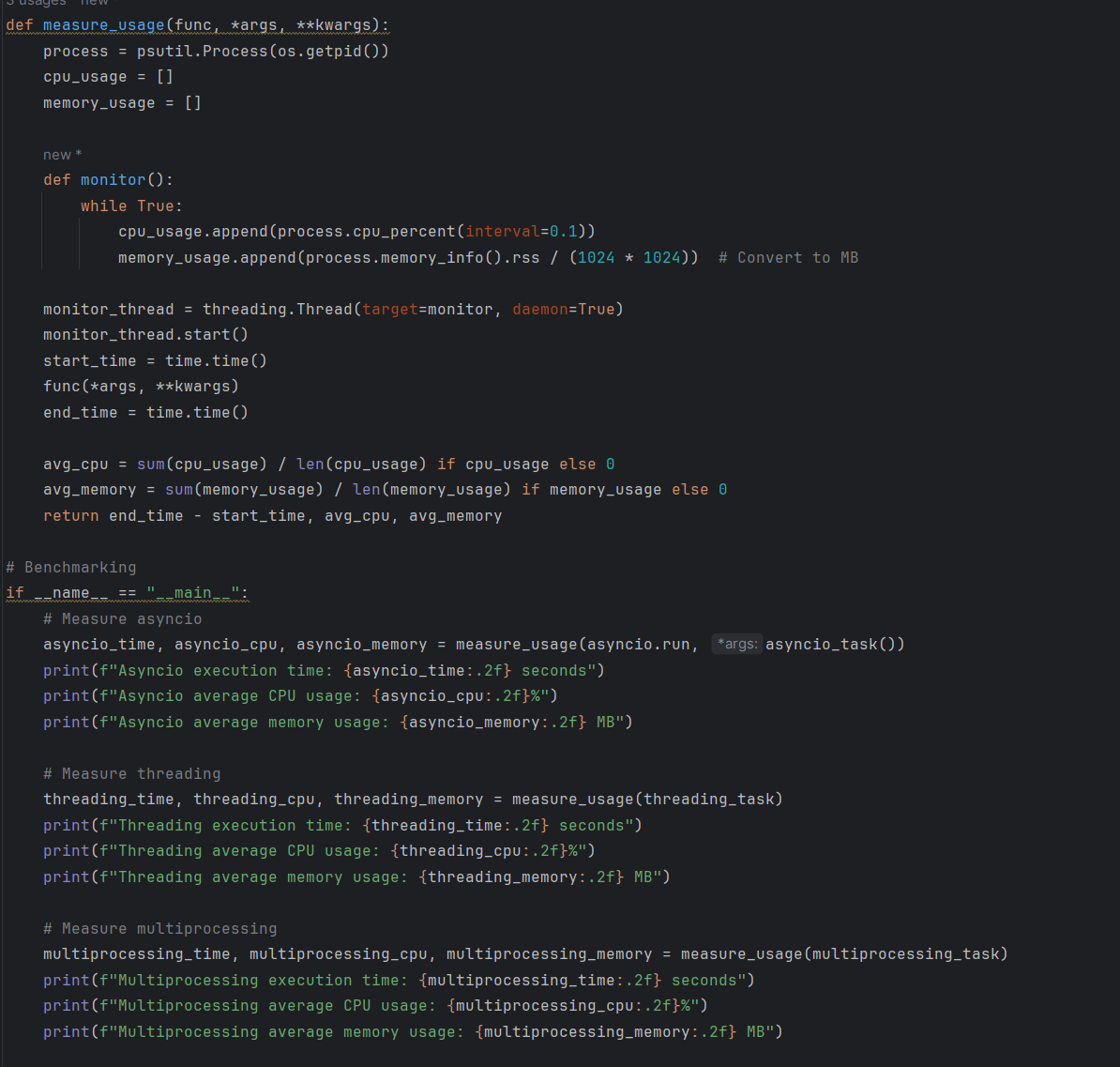




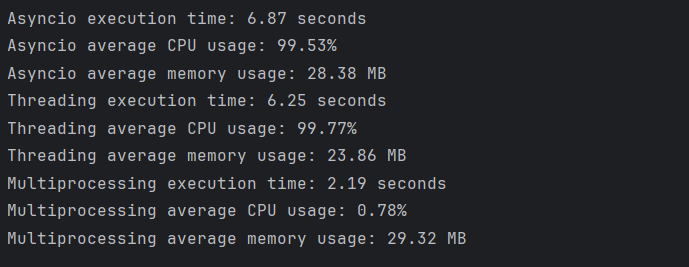
**Методы измерения производительности**

Для объективного анализа были использованы следующие метрики:

1. **Время выполнения:** общее время завершения задачи для каждого подхода.
2. **Использование CPU:** степень загрузки процессора.
3. **Использование памяти:** объём оперативной памяти, необходимый для выполнения задач.



Тесты проводились в одинаковых условиях, включая использование одинакового оборудования, размеров данных и числа задач.



По результату видим, что скорость выполнения программы с multiprocessing гораздо быстрее, но и нагрузка на оперативную память там больше. С нагрузкой на CPU у меня вышли проблемы, но multiprocessing также оказывает большую нагрузку и на него.  
  
**Заключение и дальнейшие цели**

Мною были созданы и протестированы три модели асинхронного программирования, которые я смогу свободно имплементировать в свои будущие работы. В будущем планируется работа с ботами на основе этих методов.