**ООО «ЭДЕКС»**

**Онлайн-университет Urban University**

**ДИПЛОМНАЯ РАБОТА**

на тему: Сравнение различных подходов к реализации асинхронного программирования: asyncio, threading и multiprocessing

Выполнил: Макеев Алексей Игоревич, курс «Python-разработчик, 2024г.

Руководитель:

ДАТА ЗАЩИТЫ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ОЦЕНКА\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Краснодар 2024

**СОДЕРЖАНИЕ**

[1. Введение 4](#_Toc181966334)

[1.1 Обоснование выбора темы 4](#_Toc181966335)

[1.2 Определение цели и задач исследования 5](#_Toc181966336)

[2. Основные понятия и определения 7](#_Toc181966337)

[3. Методы и подходы к разработке 9](#_Toc181966338)

[3.1 Метод №1. Определение лиц на фотографии с использованием компьютерного зрения 9](#_Toc181966339)

[3.2 Метод №2. HTTP запросы на состояние погоды 10](#_Toc181966340)

[3.3 Метод №3. Голосование 11](#_Toc181966341)

[4. Обзор инструментов асинхронного программирования 12](#_Toc181966342)

[4.1 Asyncio 12](#_Toc181966343)

[4.2 Threading – мультипоточность 13](#_Toc181966344)

[4.3 Multiprocessing 14](#_Toc181966345)

[5. Сравнение различных подходов к реализации асинхронного программирования 16](#_Toc181966346)

[5.1 Метод №1. Поиск лиц 16](#_Toc181966347)

[5.1.1 Asyncio 16](#_Toc181966348)

[5.1.2 Threading 17](#_Toc181966349)

[5.1.3 Multiprocessing 17](#_Toc181966350)

[5.2 Метод №2. Состояние погоды 18](#_Toc181966351)

[5.2.1 Asyncio 18](#_Toc181966352)

[5.2.2 Threading 19](#_Toc181966353)

[5.2.3 Multiprocessing 20](#_Toc181966354)

[5.3 Метод №3. Голосование 21](#_Toc181966355)

[5.3.1 Asyncio 21](#_Toc181966356)

[5.3.2 Threading 23](#_Toc181966357)

[5.3.3 Multiprocessing 23](#_Toc181966358)

[6. Анализ и интерпретация результатов 25](#_Toc181966359)

[6.1 Анализ полученных результатов 25](#_Toc181966360)

[6.2 Интерпретация результатов 26](#_Toc181966361)

[6.3 Выводы 27](#_Toc181966362)

[7. Заключение 27](#_Toc181966363)

[7.1 Обзор выполненной работы 27](#_Toc181966364)

[7.2 Дальнейшие планы 28](#_Toc181966365)

[8. Список используемой литературы 29](#_Toc181966366)

**Список принятых сокращений**

GIL – (англ. Global Interpreter Lock, GIL) – глобальная блокировка интерпретатора.

HTTP – (англ. HyperText Transfer Protocol) - протокол передачи гипертекста.

WEB – (англ. World Wide Web) – всемирная паутина

# **Введение**

Асинхронное программирование — это парадигма программирования, которая позволяет одновременное и неблокирующее выполнение задач внутри приложения.

Основная идея асинхронного программирования заключается в том, что вместо того, чтобы ждать завершения одной задачи перед началом другой, приложение может продолжать выполнять другие операции (рисунок 1.1).

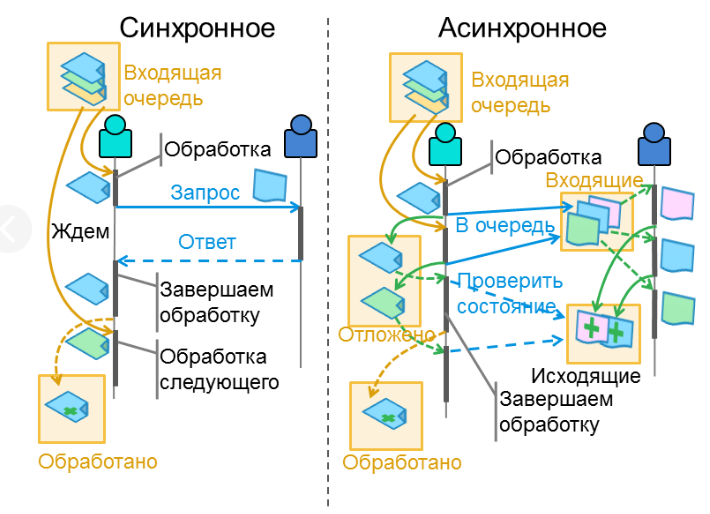


рис. 1.1 Синхронное и асинхронное взаимодействие

* 1. **Обоснование выбора темы**

Использование кода асинхронного программирования позволяет освободить поток выполнения, из которого он был запущен, что приводит к экономии ресурсов, а также предоставляет возможность параллельных вычислений. Асинхронное программирование используется в тех местах. в которых требуется оптимизация программ, испытывающих повышенную нагрузку с частым ожиданием ответа системы. Пользователям программ с графическим интерфейсом пользователя асинхронное программирование обеспечивает быстрый отклик. Серверным приложениям асинхронное программирование предоставляет больше возможностей для масштабируемости, по сравнению с синхронным. Вместе с тем, использование различных подходов к асинхронному программированию может показывать различную эффективность в зависимости от поставленной задачи, затрат на её выполнение и требуемого результата.

Асинхронное программирование используется во многих сферах:

* В веб-разработке. Асинхронность помогает делать сайты и веб-приложения быстрыми и «отзывчивыми». Например, ленты новостей в соцсетях загружаются асинхронно: страница отображается сразу, а данные подтягиваются по мере готовности.
* При работе с файлами. Асинхронность помогает избежать заморозки интерфейса при выполнении длительных операций, таких как чтение или запись больших файлов.
* В играх. Асинхронные методы позволяют игре продолжать работу и реагировать на действия игрока без задержек.
* При проектировании высоконагруженных систем позволяет обрабатывать большую нагрузку и эффективнее распределять доступное процессорное время.
  1. **Определение цели и задач исследования**

**Предмет исследования** – изучение основных принципов асинхронного программирования.

**Объект исследования** – решения задач различными методами используя разные подходы асинхронного программирования: Asyncio, Threading, Multiprocessing (далее – подходы).

**Целью работы** провести анализ преимуществ и недостатков каждого из подходов и определить оптимальные подходы для решения каждой из задач, рассмотренных в проекте.

**Задачи исследования:**

1. Изучение принципов работы каждого из подходов
2. Определение методов для тестирования подходов.
3. Анализ полученных результатов тестирования подходов.
4. Сравнение результатов и определение оптимального решения.

**Научная и практическая значимость** основывается на выборе оптимального подхода, улучшении производительности и отзывчивости программы, так как время ожидания длительных операций сокращается, а основной поток программы может продолжать выполнение других задач.

1. **Основные понятия и определения**

**Асинхронное программирование** — концепция программирования, которая заключается в том, что результат выполнения функции доступен не сразу, а через некоторое время в виде некоторого асинхронного (нарушающего обычный порядок выполнения) вызова. В отличие от синхронного программирования, где компьютер выполняет инструкции последовательно и ожидает завершения системных операций (обращение к устройствам ввода-вывода, жесткому диску, сетевой запрос) блокируя следующие операции в потоке выполнения, в асинхронном программировании длительные операции запускаются без ожидания их завершения и не блокируя дальнейшее выполнение программы [1].

**GIL (глобальная блокировка интерпретатора)** — способ синхронизации потоков, который используется в некоторых интерпретируемых языках программирования. GIL является самым простым способом избежать конфликтов при одновременном обращении разных потоков к одним и тем же участкам памяти. Когда один поток захватывает его, GIL, работая по принципу мьютекса, блокирует остальные. Очерёдность выполнения потоков определяет интерпретатор в зависимости от реализации, переключение между потоками может происходить: когда активный поток пытается осуществить ввод-вывод, по исчерпании лимита выполненных инструкций, либо по таймеру.

**Корутины (coroutines), или сопрограммы** — это блоки кода, которые работают асинхронно, то есть по очереди. В нужный момент исполнение такого блока приостанавливается с сохранением всех его свойств, чтобы запустился другой код. Когда управление возвращается к первому блоку, он продолжает работу.

**Фьючерсы (Futures)** — Объекты Future используются для объединения низкоуровневого кода на основе обратных вызовов с высокоуровневым кодом async/await. Future представляет собой возможный результат асинхронной операции. Не является потокобезопасным. [2].

**OpenAPI** — это формализованная спецификация и экосистема множества инструментов, предоставляющая интерфейс между front-end системами, кодом библиотек низкого уровня и коммерческими решениями в виде API.

1. **Методы и подходы к разработке**

Для того, чтобы определить методы для анализа инструментов асинхронного программирования, нужно определиться в реальных примерах, когда асинхронность показывает свою эффективность по сравнению с синхронным выполнением кода.

* 1. **Метод №1. Определение лиц на фотографии с использованием компьютерного зрения**

В первом методе рассмотрим какую-то функцию, требующую больших вычислительных мощностей, например обработка фото -видео фрагментов. В качестве примера используем библиотеку OpenCV (далее – cv2). Данная библиотека содержит обширную коллекцию оптимизированных алгоритмов — как классических, так и современных для компьютерного зрения и машинного обучения. Компьютерное зрение поможет нам определить, что находится на фотографии и использовать полученные данные для дальнейшей работы.

Например, достав свой современный смартфон, мы можем навести камеру на человека, и алгоритмы камеры автоматически определяют лицо, автоматически применяют к нему фильтры улучшение, регулируют освещение и применяют фокусировку относительно определённого алгоритмом лица.

В исполняемом коде достаточно будет привести пример обработки фотографии в целях определения лица. Ввиду того, что обработка каждой фотографии занимает определённое время, а выборка из нескольких фотографий увеличит время работы в n-ое количество раз, соответствующее количеству фотографий, рационально эти фотографии обрабатывать не по очереди, а параллельно.

Рассмотрим функцию, определяющую лица на фотографии, выводящую количество определённых лиц и рисующую вокруг них прямоугольники. Для усложнения операции прокрутим фотографию 4 раза на 90 градусов, это сделано для того, чтобы алгоритм определил не только лица в горизонте, но и под углом к нему. К тому же человеку не придётся вручную поворачивать фотографию в нужную ориентацию, если она (фотография) получилась «вверх-ногами», функция сделает это за него. Выполним параллельно такую функцию для 10 фотографий.

* 1. **Метод №2. HTTP запросы на состояние погоды**

В следующем методе рассмотрим web запросы. Отправка/получение данных, один из самых распространённых примером использования асинхронного программирования. При выполнении всего одной операции разницы во времени не будет, но если эта операция будет повторятся несколько раз с разными аргументами, то пока не выполнится первый запрос и не будет получен ответ, ко второму запросу программа не приступит. А что будет если запросов не 2-3, а сто или даже тысяча? Нам поможет асинхронность.

В качестве примера, напишем несложной код, который используя спецификацию OpenApi будет отправлять HTTP запрос на получение погоды с сервиса <http://api.weatherapi.com/>.

Определим, что нам нужно получить текущее состояние погоды, реальную температуру, ощущаемую температуру для всех столиц нашей планеты. Разумеется, можно включить в подборку все города, но в данном примере достаточно будет и текущего списка. К тому же сервис WeatherApi позволяет выполнять ограниченное число запросов бесплатно, в реальной жизни мы конечно можем приобрести подписку и использовать вышеупомянутую функцию в полную силу.

* 1. **Метод №3. Голосование**

Одна из самых больших головных болей асинхронных функция, не считая, конечно, обмен данными между параллельными потоками – это блокирующие функции. Блокирующие функции останавливают (блокируют) выполнение процесса до тех пор, пока производимая ими операция не будет выполнена. Например, для функции посылки сообщения это означает, что все пересылаемые данные помещены в буфер, а для функции приёма сообщения блокируется выполнение других операций, пока все данные из буфера не будут помещены в адресное пространство принимающего процесса.

В этом методе рассмотрим блокирующую функцию голосования. В качестве примера рассмотрим ситуацию, когда нам нужно провести голосование у выборки пользователей. Но чтобы привлекать 100 человек для анализа приведённого метода сымитируем деятельность человека простой функцией sleep. Она поможет нам сделать паузу для выбора конкретного пункта голосования, после чего выбранный голос будет передан основной функции.

1. **Обзор инструментов асинхронного программирования**
   1. **Asyncio**

Asyncio относительно недавний инструмент асинхронного программирования. Название модуля Asyncio — это сокращение от asynchronous I/O (асинхронный ввод/вывод). Это — Python-библиотека, которая позволяет нам выполнять код, используя модель асинхронного программирования. Такой подход даёт возможность одновременно обрабатывать несколько операций ввода/вывода, а приложение при этом не теряет возможности реагировать на внешние воздействия.

Достигается такая возможность путём введения так называемых корутин или сопрограмм. Корутина — это функция, выполнение которой можно приостановить и возобновить. Корутина: корутины — это более общая форма подпрограмм. Подпрограммы имеют одну точку входа и одну точку выхода. А корутины поддерживают множество точек входа, выхода и возобновления их выполнения. При этом корутина может запустить другую корутину, и программа приостановит выполнение первой корутины и переключится на вторую, после выполнения который вернется назад и продолжит свою работу.

Центральной концепцией модуля asyncio является цикл событий (Event Loop). Это механизм, который выполняет программы, основанные на корутинах, и реализует кооперативную многозадачность корутин.

Модуль Asyncio предоставляет и высокоуровневый, и низкоуровневый API.

Высокоуровневый API предназначен для разработчиков Python-приложений, а низкоуровневый API — преимущественно для разработчиков фреймворков, но, в большинстве случаев, не для разработчиков приложений.

Большинство вариантов использования asyncio реализуются с использование высокоуровневого API, который даёт в распоряжение программистов инструменты для работы с корутинами, потоками, примитивами синхронизации, подпроцессами и очередями, предназначенными для разделения данных между корутинами.

Низкоуровневый API — это основа высокоуровневого API, в его состав входят внутренние механизмы цикла событий, транспортные протоколы, политики и другие механизмы.

* 1. **Threading – мультипоточность**

Мультипоточность в Python — это метод в программировании, позволяющий одновременно запускать несколько потоков за счёт быстрого переключения между ними с помощью процессора.

В упрощённом виде потоки — это параллельно-выполняемые задачи. По умолчанию используется один поток — это значит, что программа делает всё по очереди, линейно, без возможности делать несколько дел одновременно.

Но если мы сделаем в программе два потока задач, то они будут работать параллельно и независимо друг от друга. Одному потоку не нужно будет становиться на паузу, когда в другом что-то происходит.

Главным недостатком мультипоточности является то, что потоки выполняются в рамках одного процесса. Во-первых, в задачах, требующих больших вычислений, скорость выполнения будет ограничена мощностью ядра и использование нескольких потоков не даст прироста результата. Во-вторых, появляется главная проблема, называемая гонками данных. Это возникает в связи с тем, что несколько потоков перехватывают данные и пытаются их изменить, что приводит к нарушению работы программы. Самым ярким примером из нашей памяти можно привести знаменитое письмо Дяди Фёдора из Простоквашино своим родителям. (рисунок 2).



рис. 4.1. Пример гонки данных

* 1. **Multiprocessing**

Многопроцессорность в Python позволяет использовать несколько процессов для параллельного выполнения кода, что обеспечивает более эффективное использование многоядерных процессоров.

Основное его отличие от мультипоточности в том, что несмотря на то, что в мультипоточности несколько поток идут параллельно, все они выполняются в рамках одного процесса. Ввиду этого, задачи, связанные с повышенными вычислительными процессами, не приведут к ускорению работы программы, так как истинного параллелизма мультипоточностью достигнуть невозможно. В то время как в мультипроцессорности процессы могут выполняться параллельно на нескольких ядрах процессора, обойдя ограничения Global Interpreter Lock (GIL).

Большим недостатком мультипроцессорности является в сложности налаживания взаимосвязи между процессами, в следствие чего данные могут накладываться друг на друга, что приводит к неправильной работе программы.

У процессов в отличие от потоков собственная память, вследствие чего у них нет главного недостатка мультипоточности – гонки данных.

Процессы могут проходить как независимо друг от друга, так и быть связаны между собой при помощи метода Pipe, который обеспечивает канал связи между процессами, что помогает при создании двунаправленной связи между двумя процессами и помогает в синхронизации данных для устранения возможных ошибок. В случаях, когда невозможно установить синхронизацию между процессами, используется метод Lock, который блокирует выполнение других процессов, пока не выполнит свою работу основной процесс.

1. **Сравнение различных подходов к реализации асинхронного программирования**
   1. **Метод №1. Поиск лиц**

В данном методе используем функцию find\_faces, которая будем принимать в себя название изображений, которые требуется обработать.

Сама функция представляет собой библиотеку Cv2, использовать которую мы будем как инструмент компьютерного зрения. Предварительно загрузим предобученный шаблон компьютерного зрения «haarcascade\_frontalface\_default.xml», который предлагают разработчики библиотеки. В данной функции распознавание будет повторятся 4 раза для того, чтобы искать лица с каждой из четырёх сторон по кругу, библиотека Cv2 предлагает большой инструментарий для работы с изображениями.

Запустим для начала без использования асинхронности, с использованием цикла for (рисунок 5.1).

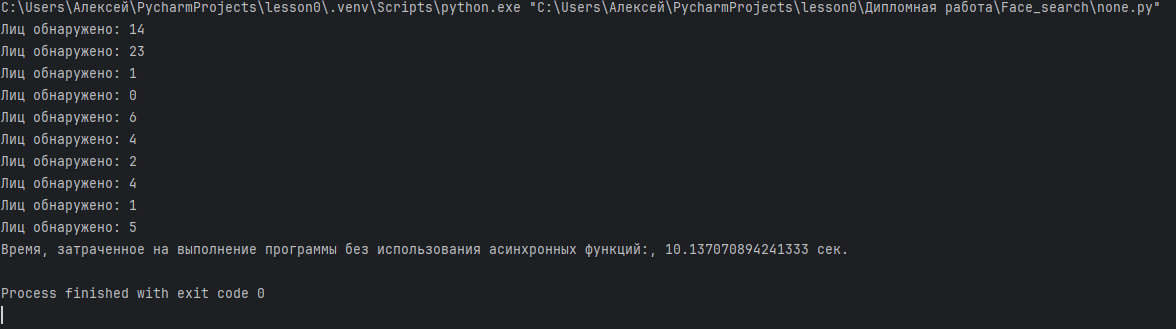
****

рис. 5.1. Поиск лиц без асинхронных функций

* + 1. **Asyncio**

Сложности запуска асинхронной функции возникнуть не может быть. Однако ввиду того, что, как и мультипоточность, Asyncio выполняется в рамках одного процесса, прироста производительности мы не получим (рисунок 5.2).

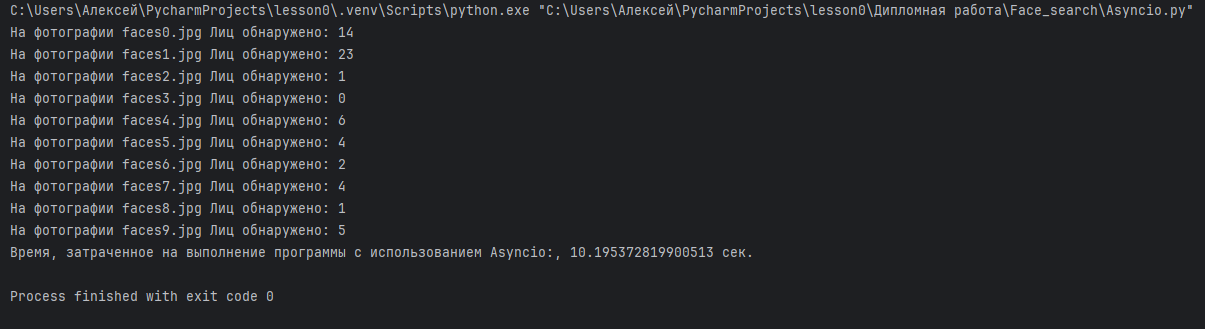
****

рис. 5.2. Поиск лиц с использованием Asyncio

* + 1. **Threading**

Использование подхода мультипоточности практически не отличается от предыдущего подхода, так как также используется одно ядро (рисунок 5.3).

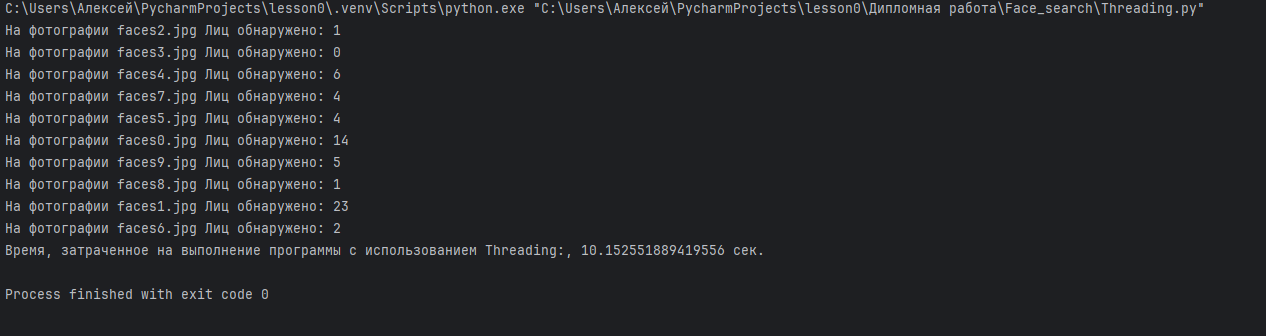


рис. 5.3. Поиск лиц с использованием Threading

* + 1. **Multiprocessing**

Функция поиска лиц – трудозатратная для процесса функция, поэтому распределение её между процессами ускоряет выполнение программы (рисунок 5.4).

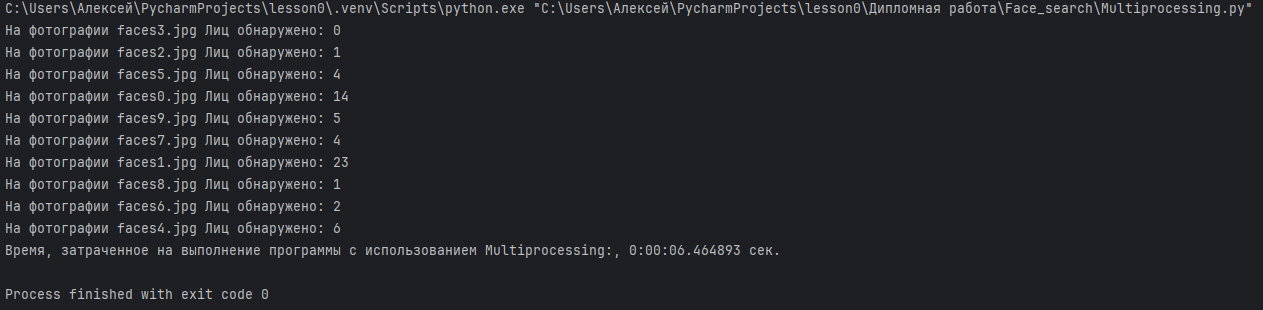
****

рис. 5.4. Поиск лиц с использованием Multiprocessing

* 1. **Метод №2. Состояние погоды**

В данном методе используем функцию get\_weather, которая обращается к url: <http://api.weatherapi.com/>, формирует запрос, обрабатывает результат и выводит нам текущее состояние погоды в выбранном городе. Для удобства список городов выведем в отдельный файл, который в будущем мы сможем редактировать по своему усмотрению. Попробуем запустить функцию get\_weather напрямую (рисунок 5.5).

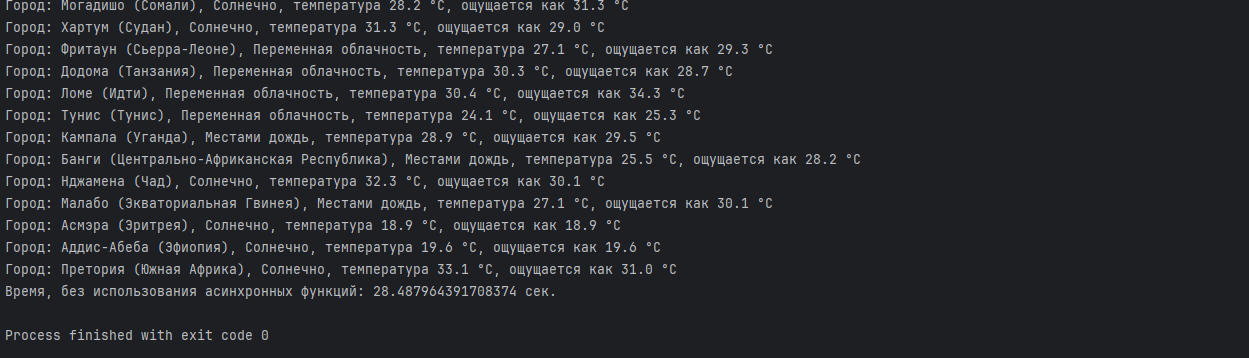


рис. 5.5. Получение погоды без асинхронных функций

Как видим, каждый запрос выполнялся по порядку, и заняло это достаточно много времени.

* + 1. **Asyncio**

Добавим в исходный код бибилиотеку aiohttp и asyncio. Функцию get\_weather преобразуем в виде корутины, представив её в виде async def get\_weather, и вместо request будем использовать сессии ClientSession из библиотеки aiohttp. Добавим список задач (tasks) и внесём в этот список наши задачи, запустив параллельно все запросы по всем городам (рисунок 5.6).

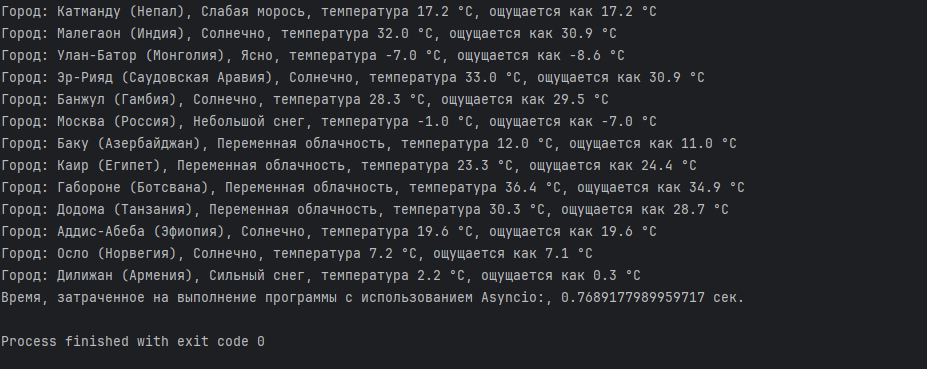


рис. 5.6. Получение погоды с использованием Asyncio

Используя данный подход, мы получили существенный прирост в скорости выполнения программы, так как все запросы происходили параллельно, время было ограничено лишь откликом самого долгого запроса. Кстати, в Москве похолодало.

* + 1. **Threading**

Кода с использованием мультипоточности представляет собой основную функцию get\_weather запущенную в разных потоках (рисунок 5.7). Так как потоки представляют собой независимую функцию от переменных, гонки данных здесь быть не может.

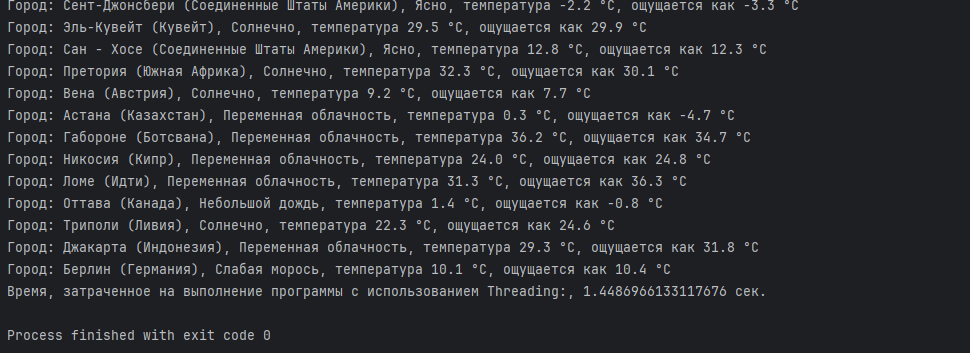


рис. 5.7. Получение погоды с использованием Threading

В целом, мультипоточность показала неплохой результат, немногим уступая Asyncio, однако в более сложных операциях ввода-вывода, мультипоточность будет существенно отставать.

* + 1. **Multiprocessing**

Исходный код будет немногим отличаться от мультипоточности, только управлять будем не потоками, а процессами (рисунок 5.8).

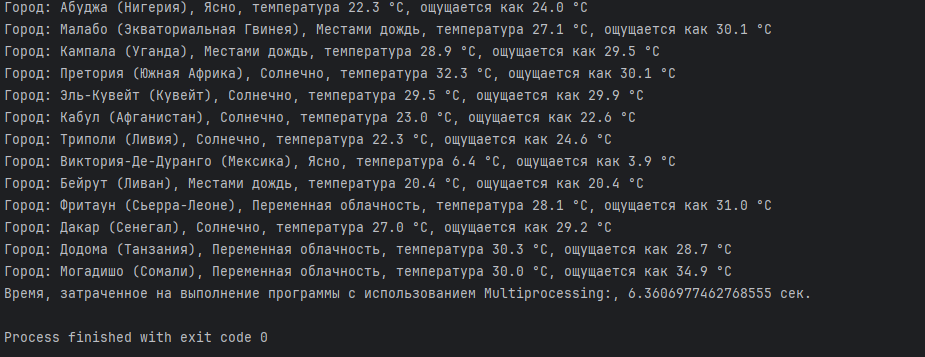


рис. 5.8 Получение погоды с использованием Multiprocessing

Сложность заключается в подборе оптимального количества процессов, если процессов будет слишком мало, каждый процесс получит большой объем работы, если слишком много процессов, то получим такое положение, при котором запуск процесса будет более затратным по ресурсам и времени, чем работа в рамках меньшего количества процессов.

В тесте использовалась машина с центральным процессором 13th Gen Intel(R) Core(TM) i5-13500, имеющая 14 физических ядер и 20 логических процессов, подобрав подходящее значение, оптимальным числом процессов оказалось 16, но даже это показало результат недостаточно эффективным, чем предыдущие подходы.

* 1. **Метод №3. Голосование**

В данном методе используем простейшую функцию take\_poll, которая принимает в себя число пунктов голосования (в тесте не имеет значение, но примем число 3), и пользователя, который получит голосование. Примем число пользователей 200 и запустим синхронное голосование, результат будет ожидаемым, каждая новая функция не запустится пока не будет полностью отработана предыдущая (рисунок 5.9).

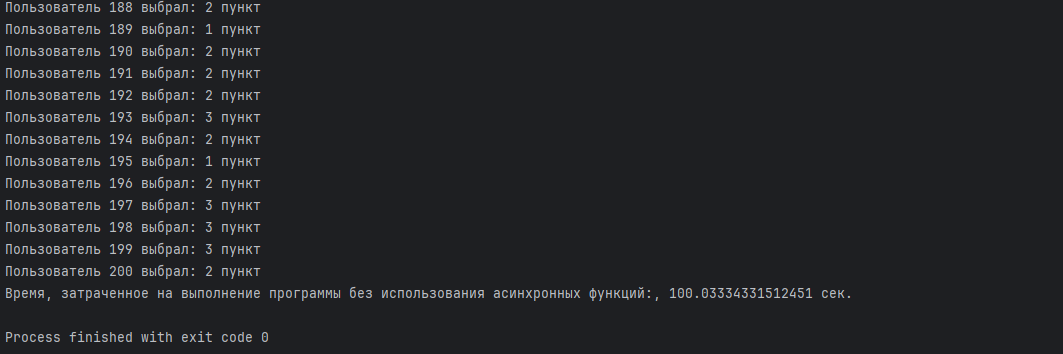


рис. 5.9. Голосование без использования асинхронных функций

* + 1. **Asyncio**

Если представим функцию take\_poll в виде корутины и просто их запустим, как в методе 1, то получим блокирующую функцию, которая не позволяет использовать корутины асинхронно, получим результат, аналогичный рис. 5.9 (рисунок 5.10).

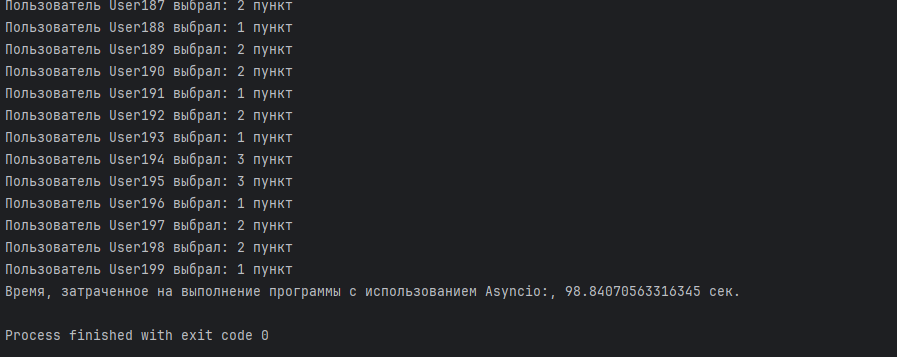


рис. 5.10. Голосование с использованием Asyncio, без цикла событий

Каждый запрос выполнялся последовательно и не был начат, пока не закончит работу предыдущий. Для решения проблемы нам поможет так называемый цикл событий (Event Loop), вернет функцию take\_poll в прежний вид (уберем async перед названием функции) и запустим функцию Asyncio.get\_event\_loop, и запустим её через executor – run\_in\_executor (рисунок 5.11).

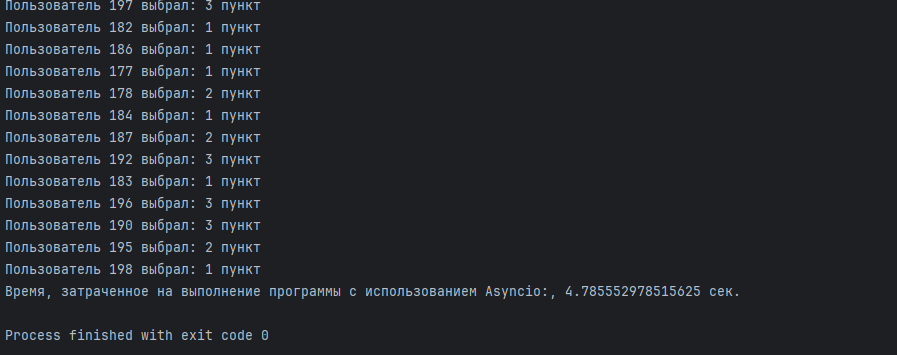


рис. 5.11. Голосование с использованием Asyncio, с циклом событий

Как видно из результатов, асинхронная функция сработала, почему так произошло? Дело в том, что функция запуска цикла событий возвращает текущий цикл событий, или создает новый, если текущего нет. Таким образом функция продолжает работу, не дожидаясь ответа от предыдущей.

* + 1. **Threading**

Запустим голосование через потоки (рисунок 5.12).

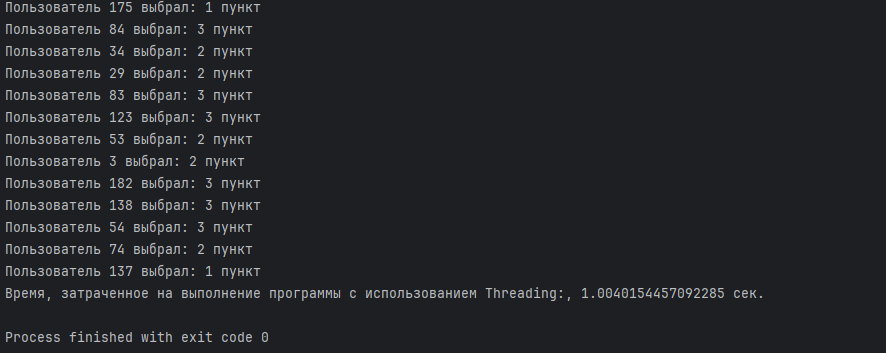


рис. 5.12. Голосование с использованием Threading

Результат получился быстрее чем с использованием Asyncio, даже запустив цикл событий. Это произошло потому, что потоки запустились одновременно, блокирующая функция не мешает запуску нового потока, а время выполнения программы ограничено лишь откликом последнего проголосовавшего.

* + 1. **Multiprocessing**

Теперь запустим голосование в режиме мультипроцессорности, позволим библиотеке multiprocessing управлять процессами самостоятельно (рисунок 5.13).

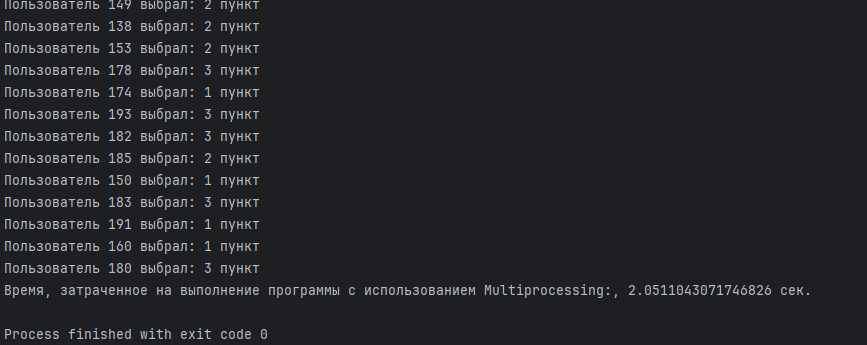


рис. 5.13. Голосование с использованием Multiprocessing

Результат также получился лучше, чем с использованием Asyncio, однако медленнее, чем с использованием Threading, это связано с тем, что функция take\_poll предельно простая, и для запуска процесса используется больше ресурсов и времени, чем на отработку функции. Для наглядности увеличим число пользователей в 10 раз (рисунок 5.14).

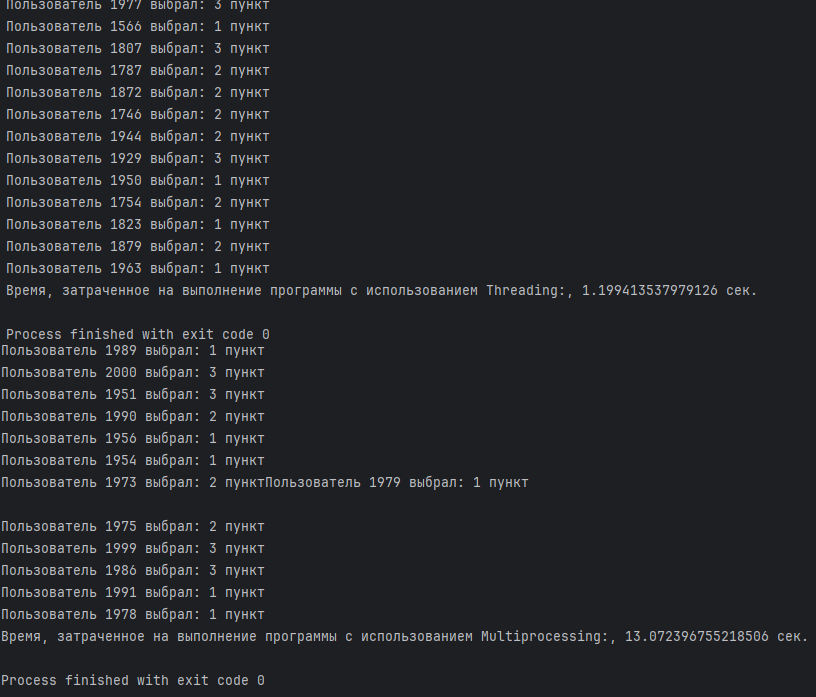


рис. 5.14. Голосование для 2000 пользователей

Сравнив полученные результаты, можно сделать вывод, что с увеличением числа пользователей разница между временем выполнения разных подходов возрастает экспоненциально.

1. **Анализ и интерпретация результатов**
   1. **Анализ полученных результатов**

Соберём полученные результаты:

1 метод – определение лиц:

- без использования асинхронных функций – 10.13 сек.;

- Asyncio – 10.19 сек.;

- Threading – 10.15 сек.;

- Multiprocessing – 6.46 сек.

2 метод – состояние погоды:

- без использования асинхронных функций – 28.49 сек.;

- Asyncio – 0.76 сек.;

- Threading – 1.44 сек.;

- Multiprocessing – 6.36 сек.

3 метод – голосование:

- без использования асинхронных функций – 100.03 сек.;

- Asyncio – 4.79 сек.;

- Threading – 1.00 сек.;

- Multiprocessing – 2.05 сек.

Для удобства восприятия составим диаграммы:

рис. 6.1. Анализ метода №1

рис. 6.2. Анализ метода №2

рис. 6.3. Анализ метода №3

* 1. **Интерпретация результатов**

Проанализировав полученные результаты, мы приходим к выводу, что асинхронное программирование помогает существенно сократить время выполнения программы, однако для разных задач разные подходы к асинхронному программированию будут показывать разную эффективность.

Asyncio – крайне полезен при выполнении задач, связанных с http запросами и ввод-вывод операциями, также благодаря современным библиотекам, появилась возможность легко и с низкими затратами ресурсов организовать работу автоматических систем обработки информации, например Telegram бот, с использованием библиотеки Aiogram. Благодаря async/await вызовам, мы можем сами управлять асинхронными процессами и более эффективно их использовать.

Threading – крайне прост в использовании, и по скорости во многих задачах не уступает asyncio, а в некоторых, в частности, где встречаются блокирующие функции – превосходит в несколько раз, однако организация управления потоков требует больших усилий. При выполнении параллельных простых функций, Threading – отличный инструмент.

Multiprocessing – при выполнении простых функций, является не эффективность, так как создание и управление процессами требует большего времени и ресурсов чем выполнение самой функции. Однако, при задачах, требующих высоких вычислительных мощностей и производительностей, мультипроцессорность даёт возможность распределить нагрузку между разными процессами и существенно разгрузить один процесс и сократить время выполнения.

* 1. **Выводы**

Проведя работу по сравнению выбранных инструментов асинхронного программирования, можно прийти к выводу, что наиболее эффективного инструмента нет, каждый в своей области будет лучше, чем другие и наоборот, в других областях будет существенно уступать. Понимание основных принципов работы этих инструментов поможет разработчику определить наиболее эффективный подход к решению определённой задачи.

1. **Заключение**
   1. **Обзор выполненной работы**

В ходе выполнения дипломной работы мною было определено понятие асинхронного программирования, обоснована тема выполняемой работы, рассмотрены основные инструменты асинхронного программирования. Предложены методы испытания данных инструментов. После прохождения испытаний, мною были проанализированы полученные результаты и определены эффективные стороны каждого из инструментов. Работу считаю выполненной в полном объеме.

* 1. **Дальнейшие планы**

В дальнейших планах предполагается дальнейшее глубокое изучение конкурентности и параллелизма, чтобы исследовать и улучшать методики, улучшающие эффективность уже известных программ, либо написание новых с учётом открытых возможностей. В частности, ничего не запрещает использовать все инструменты одновременно, используя лучшие стороны каждого из инструментов и закрывая тем самым слабые стороны.

Компьютерное зрение – одна из стремительно-развивающихся сфер машинного обучения, в рассмотренной работе была лишь упрощённая схема работы компьютерного зрения, которая использует библиотеку уже готовых изображений, обрабатывает их и возвращает результат. В дальнейших планах улучшение работы программы, чтобы она могла в реальном времени получать изображение с камеры видеонаблюдения, параллельно анализировать, обрабатывать и отправлять результат пользователю, если будут найдены лица, в этом и поможет нам асинхронность.

1. **Список используемой литературы**
2. Татьяна Валерьевна Любимова. Асинхронность В Программировании // Университетская Наука. — 2019. — Вып. 2 (8). — ISSN 2500-2724.
3. Савостин Петр Алексеевич, Ефремова Наталья Эрнестовна. Практическое применение асинхронного программирования на языке Python при помощи пакета asyncio // Программные системы и вычислительные методы. — 2018. — Вып. 2. — С. 11–16. Архивировано 18 января 2022 года.
4. ↑ Елена Васильевна Фешина, Дмитрий Александрович Омельченко, Руслан Геннадьевич Гонатаев. Многопоточность и асинхронность в языке программирования Python // Инновации. Наука. Образование. — 2021. — Вып. 28.
5. https://docs.python.org/3/library/asyncio-future.html
6. <https://habr.com/ru/articles/337420/>
7. https://habr.com/ru/companies/wunderfund/articles/710190/
8. <https://tproger.ru/translations/opencv-python-guide>
9. <https://codechick.io/tutorials/python/python-multiprocessing>
10. <https://habr.com/ru/articles/667630/>
11. https://pypi.org/project/opencv-python/