

АСИНХРОННЫЙ ПОДХОД В ПРОГРАММИРОВАНИИ**Н. С. Железко, С. А. Лукашевич***Гомельский государственный университет имени Франциска Скорины,
Республика Беларусь*

Рассмотрены преимущества и недостатки асинхронного подхода с практическим примером реализации на Python.

Ключевые слова: python, асинхронность, AsyncIO, await, корутина, поллинг.

ASYNCHRONOUS APPROACH TO PROGRAMMING**M. S. Zhelezko, S. A. Lukashevich***Francisk Skorina Gomel State University, Republic of Belarus*

Discusses the advantages and disadvantages of the asynchronous approach with a practical example of implementation in Python.

Keywords: python, asynchrony, AsyncIO, await, coroutine, polling.

Современные программные системы часто требуют обработки большого количества одновременного количества запросов: взаимодействие с пользователем, обмен сообщениями, получение сетевых запросов, интеграция со сторонними сервисами. В таких условиях традиционный (последовательный) подход становится малоэффективным, поскольку каждая операция блокирует выполнение программы, что приводит к снижению производительности.

Одним из ключевых решений этой проблемы стало асинхронное программирование, которое позволяет выполнять множество задач параллельно, не дожидаясь завершения каждой из них. В python реализация данного подхода основана на механизме корутин, а стандартная библиотека AsyncIO предоставляет `async` и `await` для написания неблокирующего кода. В современном мире подобный подход особенно актуален в разработке Discord/Telegram-ботов, где программа должна обрабатывать сообщения от множества пользователей одновременно, взаимодействовать с внешними API и выполнять фоновые задачи, например, рассылку уведомлений.

В данной работе будет рассмотрен асинхронный код, его преимущества и недостатки, а также приведен практический пример реализации Telegram-бота с использованием асинхронной модели.

Цель работы – продемонстрировать, как асинхронный подход способствует повышению эффективности и масштабируемости приложений.

Асинхронное программирование – это подход к разработке программ, при котором выполнение операций ввода-вывода (например, запросов к серверу, чтения файлов или ожидания пользовательских действий) не блокирует основной поток выполнения. Вместо ожидания завершения одной задачи программа может переключаться на другие операции, тем самым повышая общую производительность и отзывчивость.

В отличие от многопоточности, где задачи выполняются параллельно с использованием нескольких потоков, асинхронность опирается на событийный цикл (event loop) и корутины – специальные функции, которые могут приостанавливать и возобновлять свое выполнение. Это позволяет экономно использовать системные ресурсы, особенно при работе с большим числом сетевых подключений.

В Python асинхронное программирование реализовано с помощью модуля AsyncIO, который предоставляет базовый механизм для запуска и управления корутинами. Пример простейшего асинхронного кода представлен на рис. 1.

```
import asyncio

async def greet(name): 3 usages
    print(f"Привет, {name}!")
    await asyncio.sleep(1)
    print(f"До свидания, {name}!")

async def main(): 1 usage
    await asyncio.gather(
        greet("Аня"),
        greet("Борис"),
        greet("Виктор")
    )

asyncio.run(main())
```

Рис. 1. Пример простейшего асинхронного кода

В этом примере три корутины выполняются «одновременно» – на самом деле программа просто переключается между ними, когда одна из них ожидает завершения операции (`await asyncio.sleep(1)`). Такой подход позволяет эффективно использовать время ожидания и не блокировать выполнение других задач.

К преимуществам асинхронного подхода относятся: высокая производительность при большом числе операций, более низкое потребление ресурсов вычислительной машины, простота масштабирования сетевых приложений и микросервисов, удобство интеграции с современными API и веб-протоколами.

К недостаткам следует отнести: повышенную сложность отладки и тестирования кода, необходимость строгого соблюдения контекста асинхронного кода, некоторые библиотеки не поддерживают `async/await`.

Асинхронное программирование играет важную роль в развитии современных приложений, особенно тех, которые связаны с сетевым взаимодействием и высокой нагрузкой на систему ввода-вывода. Использование асинхронного подхода в Python, основанного на библиотеке AsyncIO, позволяет значительно повысить эффективность и отзывчивость программ, избегая избыточного расхода ресурсов, характерного для традиционной многопоточности.

На практике асинхронность открывает широкие возможности при создании Telegram-ботов, которые должны одновременно обслуживать множество пользователей, обрабатывать команды, выполнять сетевые запросы и взаимодействовать с внешними API.

Литература

1. Фаулер, М. Asyncio и конкурентное программирование на Python / М. Фаулер. – СПб. : Питер, 2023. – 352 с.
2. Браунли, Д. Python's asyncio: A Hands-On Walkthrough // Real Python. – 2025 (онлайн).
3. Aiogram developers. aiogram – асинхронный фреймворк для Telegram Bot API // aiogram.dev. 2024 (онлайн).