Python: конкурентность, параллельность, асинхронность | Я.Шпора

Потоки и процессы в Python

Многопоточность и многопроцессность в Python может быть реализована с помощью модулей стандартной библиотеки:

- *threading* — работа с потоками;
- *multiprocessing* — работа с процессами.

Работа с потоками

Пример программы, которая замеряет время выполнения задачи в многопоточном выполнении:

```
import time
import threading


def task(n=100_000_000):
        """Функция, которая имитирует задачу."""
    while n:
        n -= 1


if __name__ == '__main__':
        # Засечь время с максимально возможной точностью.
    start = time.time()
        # Создать два потока.
    p1 = threading.Thread(target=task)
    p2 = threading.Thread(target=task)
        # Старт двух потоков.
    p1.start()
    p2.start()
```

```
        # Подождать, пока оба потока завершат свою работу.
    p1.join()
    p2.join()
    finish = time.time()
    print(f'Выполнение заняло {finish - start} секунд.')

# Вывод в терминал:
# Выполнение заняло 5.329088926315308 секунд.
```

Чтобы запустить новый дочерний поток, нужно создать экземпляр класса `Thread` и вызвать его метод `start()`.

Метод `join()` не даст продолжить выполнять инструкции из основного потока до тех пор, пока дочерние потоки, для которых вызван этот метод, не закончат работу.

Работа с процессами

Пример программы, которая замеряет время выполнения задачи в многопроцессном выполнении:

```
import time
import multiprocessing


def task(n=100_000_000):
    """Функция, которая имитирует задачу."""
    while n:
        n -= 1


if __name__ == '__main__':
    start = time.time()
        # Создать два процесса.
    p1 = multiprocessing.Process(target=task)
    p2 = multiprocessing.Process(target=task)
    # Старт двух процессов.
    p1.start()
    p2.start()
        # Подождать, пока оба процесса завершат свою работу.
```

```
    p1.join()
    p2.join()
    finish = time.time()
    print(f'Выполнение заняло {finish-start} секунд.')

# Вывод в терминал:
# Выполнение заняло 2.9015047000000322 секунд.
```

Синтаксис и формат работы с процессами точно такой же, как и с потоками, только вместо потоков запускаются процессы: создаётся экземпляр класса `Process` и у него вызывается метод `start`.

Потоки или процессы?

Чаще всего встречаются задачи двух типов:

- **I/O-bound** — операции, связанные с «медленными» действиями ввода/вывода, например, работа с файлами, сетевые запросы, запросы к базе данных. Для них предпочтительнее использовать модуль `threading` или асинхронное программирование.
- **CPU-bound** — операции, связанные только с использованием вычислительных возможностей процессора, например, обработка изображений. Выигрышнее будет использовать модуль `multiprocessing`.

Асинхронный Python

Асинхронный код — способ организации выполнения программы, когда операции, занимающие некоторое время (например, запросы к серверу, чтение файлов), выполняются одновременно с основным потоком выполнения, не блокируя его. Это позволяет программе продолжать работать в то время, как длительные задачи выполняются в фоне, и возвращать результаты после их завершения.

Асинхронный подход требует использования корутин и цикла событий для управления задачами.

Корутина — специальная функция с ключевым словом `async`. Корутины бывают двух видов:

- обычные, например, сами задачи;
- корутины верхнего уровня — в них создаётся список из задач, которые нужно выполнить асинхронно.

Цикл событий (*Event loop*) управляет выполнением обычных корутин из корутины верхнего уровня: регистрирует их поступление, запускает, ставит на паузу и запускает снова, когда они «сообщают», что готовы выполняться. Цикл событий позволяет переключаться между задачами в нужный момент.

Асинхронный подход предполагает, что в коде программы указаны специальные места, где такое переключение возможно. Точка переключения обозначается внутри корутины ключевым словом `await`. Любая функция, которая использует `await` обязана объявляться как `async`.

```
async def task(...):
        result = await async_task()
        ...
```

Полезные ресурсы

Документация модуля threading

Документация модуля multiprocessing

Документация модуля asyncio

Практикум