Параллелизм в Python

> **Процесс** — это экземпляр программы, выполняемой на компьютере.
>
> **Поток** — это единица выполнения внутри процесса.
>
> **Потоки-демоны** — это потоки, которые выполняют задачи в фоновом режиме.

Когда стоит использовать потоки

Потоки оптимизированы для задач, связанных с вводом-выводом. Например: сетевые запросы, соединения с базами данных, ввод/вывод файлов.

- Используйте модуль `threading` для создания многопоточной программы.
- Используйте `Thread(function, args)` для создания нового потока.
- Вызовите метод `start()` класса `Thread`, чтобы запустить поток.
- Вызовите метод `join()` класса `Thread`, чтобы дождаться завершения потока в главном потоке.
- Используйте потоки только для программ, связанных с обработкой ввода-вывода.

Однопоточная программа

```
def replace(filename, substr, new_substr):
    print(f'Обрабатываем файл {filename}')
    # получаем содержимое файла
    with open(filename, 'r') as f:
        content = f.read()

    # заменяем substr на new_substr
    content = content.replace(substr, new_substr)

    # записываем данные в файл
    with open(filename, 'w') as f:
        f.write(content)


def main():
    filenames = [
        'c:/temp/test1.txt',
        'c:/temp/test2.txt',
        'c:/temp/test3.txt',
        'c:/temp/test4.txt',
        'c:/temp/test5.txt',
        'c:/temp/test6.txt',
        'c:/temp/test7.txt',
        'c:/temp/test8.txt',
        'c:/temp/test9.txt',
        'c:/temp/test10.txt',
    ]
```

```
    for filename in filenames:
        replace(filename, 'ids', 'id')


if __name__ == "__main__":
    start_time = perf_counter()

    main()

    end_time = perf_counter()
    print(f'Выполнение заняло {end_time- start_time :0.2f} секунд.')
```

Многопоточная программа

```
from threading import Thread
from time import perf_counter


def replace(filename, substr, new_substr):
    print(f'Обрабатываем файл {filename}')
    # получаем содержимое файла
    with open(filename, 'r') as f:
        content = f.read()

    # заменяем substr на new_substr
    content = content.replace(substr, new_substr)

    # записываем данные в файл
    with open(filename, 'w') as f:
        f.write(content)


def main():
    filenames = [
        'c:/temp/test1.txt',
        'c:/temp/test2.txt',
        'c:/temp/test3.txt',
        'c:/temp/test4.txt',
        'c:/temp/test5.txt',
        'c:/temp/test6.txt',
        'c:/temp/test7.txt',
        'c:/temp/test8.txt',
        'c:/temp/test9.txt',
        'c:/temp/test10.txt',
    ]

    # создаем потоки
    threads = [Thread(target=replace, args=(filename, 'id', 'ids'))
            for filename in filenames]

    # запускаем потоки
    for thread in threads:
        thread.start()

    # ждем завершения потоков
    for thread in threads:
        thread.join()


if __name__ == "__main__":
    start_time = perf_counter()

    main()

    end_time = perf_counter()
    print(f'Выполнение заняло {end_time- start_time :0.2f} секунд.')
```

Пример потокобезопасной очереди

```
import time
from queue import Empty, Queue
from threading import Thread


def producer(queue):
    for i in range(1, 6):
        print(f'Вставляем элемент {i} в очередь')
        time.sleep(1)
        queue.put(i)


def consumer(queue):
    while True:
        try:
            item = queue.get()
        except Empty:
            continue
        else:
            print(f'Обрабатываем элемент {item}')
            time.sleep(2)
            queue.task_done()


def main():
    queue = Queue()

    # создаем поток-производитель и запускаем его
    producer_thread = Thread(
        target=producer,
        args=(queue,)
    )
    producer_thread.start()

    # создаем поток-потребитель и запускаем его
    consumer_thread = Thread(
        target=consumer,
        args=(queue,),
        daemon=True
    )
    consumer_thread.start()

    # дожидаемся, пока все задачи добавятся в очередь
    producer_thread.join()

    # дожидаемся, пока все задачи в очереди будут завершены
    queue.join()


if __name__ == '__main__':
    main()
```