# CPU bound операции

> run_in_executor/cpu.py

```
import asyncio
import concurrent.futures
import datetime


def blocking_task():
    counter = 50000000
    while counter > 0:
        counter -= 1


async def blocking_worker():
    while True:
        blocking_task()


async def async_thread_worker():
    loop = asyncio.get_event_loop()
    with concurrent.futures.ThreadPoolExecutor(2) as pool:
        while True:
            await loop.run_in_executor(pool, blocking_task)


async def async_process_worker():
    loop = asyncio.get_event_loop()
    with concurrent.futures.ProcessPoolExecutor(2) as pool:
        while True:
            await loop.run_in_executor(pool, blocking_task)
```

```
async def ticker():
    while True:
        print(datetime.datetime.now())
        await asyncio.sleep(1)


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.create_task(ticker())
    loop.create_task(blocking_worker())
    # loop.create_task(async_thread_worker())
    # loop.create_task(async_process_worker())
    loop.run_forever()
```

Код похож на предыдущий пример, но в **blocking_task** теперь CPU bound операция: уменьшение счетчика.

Если запустить **blocking_worker**, то, как и в предыдущем примере, event loop заблокируется. В результате ticker выполнит только одну итерацию цикла, и мы увидим только одну строчку с датой.

Давайте оставим только **async_process_worker** и запустим cpu.py:

```
coder@code-server-admin-38-94c54d494-m248j:~/mercury$ python3 run_in_executor/cpu.py
2021-09-30 10:57:29.264471
2021-09-30 10:57:30.271911
2021-09-30 10:57:31.273314
2021-09-30 10:57:32.296778
2021-09-30 10:57:33.298125
2021-09-30 10:57:34.298504
```

Как и ожидалось, cpu bound опреация выполняется в отдельном процессе и не блокирует event loop. Так как запускается отдельный процесс, а межпроцессное взаимодействие работает через socket, при таком подходе мы как будто обращаемся к другому сервису по сети (io bound опрерация) и ждем выполнения запроса.

Теперь давайте оставим только **async_thread_worker** и запустим cpu.py:

```
coder@code-server-admin-38-94c54d494-m248j:~/mercury$ python3 run_in_executor/cpu.py
2021-09-30 11:03:07.334160
2021-09-30 11:03:08.351872
2021-09-30 11:03:09.358425
2021-09-30 11:03:10.380154
2021-09-30 11:03:11.392263
2021-09-30 11:03:12.399658
```

На первый взгляд, ничего не поменялось. Все работает так же, как с **async_process_worker**.

Но нужно понимать, что так или иначе, но из-за GIL в один момент времени будет исполняться только один поток. Поэтому CPU bound операция будет отнимать процессорное время у event loop. При этом операционная система принимает решение — в какой момент включить один или другой поток. Также добавляются накладные расходы на переключение между потоками.

Поэтому в случае с cpu bound операциями ThreadPoolExecutor лучше не применять. Есть механизм, который позволяет контролируемо передавать управление другой корутине из cpu bound операции средствами asyncio.