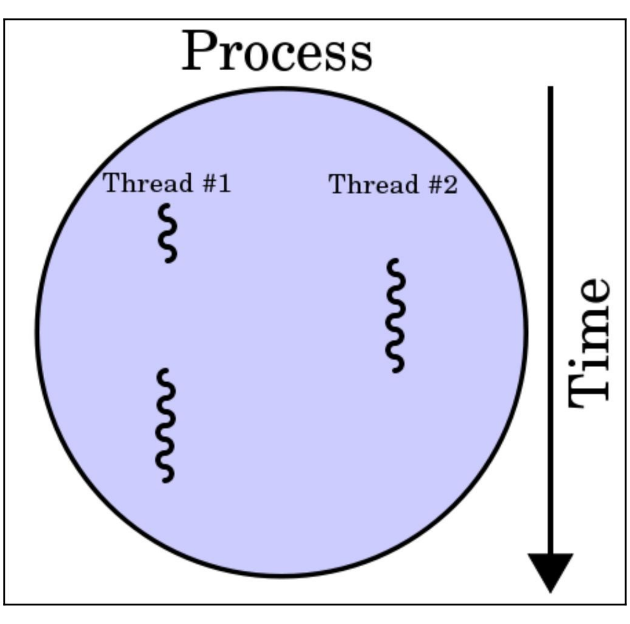
Треды (thread)

Тред — наименьшая единица обработки, исполнение которой может быть назначено ядром системы. Потоки в рамках одного и того же процесса могут совместно использовать ресурсы — такие, как память. А процессы явно не разделяют этих ресурсов.

Может показаться, что несколько работающих тредов внутри процесса будут выполняться параллельно, как и несколько запущенных процессов из предыдущего модуля. Но в Python это не работает. Из-за глобальной блокировки интерпретатора GIL только один поток может выполнять код Python одновременно. Это проиллюстрировано ниже:



Если необходимо выполнить CPU bound операцию, реализация с помощью тредов будет плохим решением. В обоих случаях только 1 поток будет обрабатываться одновременно. В случае с несколькими тредами система будет тратить ресурсы на переключение тредов. Давайте убедимся в этом на реальном примере.

Стоит отметить, что треды, созданные с помощью python threading — настоящие треды операционной системы. Их можно увидеть с помощью утилиты **ps**.

Команда:

> ps -T -p <pid>

```
import time
from threading import Thread


def t1():
    time.sleep(1000)


thread_1 = Thread(target=t1)
thread_2 = Thread(target=t1)

thread_1.start()
thread_2.start()
thread_1.join()
thread_2.join()
```

Если запустить скрипт и посмотреть список потоков, можно увидеть основной и 2 запущенных потока:

```
root@b4ef55eb2bd5:/# ps aux | grep python
root       424  1.1  0.3 157256  7036 pts/0    Sl+  11:09   0:00 python3 ttt.py
root       428  0.0  0.0   4800   648 pts/1    S+   11:09   0:00 grep python
root@b4ef55eb2bd5:/# ps -T -p 424
  PID  SPID TTY          TIME CMD
  424   424 pts/0    00:00:00 python3
  424   425 pts/0    00:00:00 python3
  424   426 pts/0    00:00:00 python3
```

Общая память

Давайте посмотрим на аналогичный пример из юнита про процессы, но реализованный на тредах. Threading — библиотека для работы с тредами.

```
import time
from threading import Thread

data = dict()

def t1():
    data["t1"] = True
    print(data)

def t2():
    data["t2"] = True
    print(data)

print(data)
thread_1 = Thread(target=t1)
thread_2 = Thread(target=t2)

thread_1.start()
thread_2.start()
thread_1.join()
thread_2.join()

print(data)
```

```
{}
{'t1': True}
{'t1': True, 't2': True}
{'t1': True, 't2': True}
```

Мы видим, что у тредов общая память. Треды запущенны внутри 1 процесса и разделяют его ресурсы.

Для передачи данных между тредами лучше использовать библиотеку queue. Она обеспечит безопасный обмен данными между тредами. О методах синхронизации тредов вы можете дополнительно узнать в библиотеке threading.

CPU-операции с тредами

Убедимся, что обработка CPU-операций в Python — плохое решение.

Сначала посмотрим на время выполнения синхронного кода без тредов:

```
import time


def countdown(id: int):
    i = 0
    begin = time.time()
    while i < 50_000_000:
        i += 1

    print(f"ID: {id} DONE duration: {time.time() - begin}")


COUNT = 2
begin = time.time()
print("START calculation")
for i in range(COUNT):
    countdown(i + 1)

print(f"FINISH calculation duration: {time.time() - begin}")
```

```
START calculation
ID: 1 DONE duration: 1.1998991966247559
ID: 2 DONE duration: 1.2359821796417236
FINISH calculation duration: 2.435983180999756
```

Теперь реализация «параллельного» вычисления в разных тредах:

```
import time
from threading import Thread


def countdown(id: int):
    i = 0
    begin = time.time()
    while i < 50_000_000:
        i += 1

    print(f"ID: {id} DONE duration: {time.time() - begin}")


print("START calculation")
begin = time.time()

t1 = Thread(target=countdown, args=(1,))
t2 = Thread(target=countdown, args=(2,))

t1.start()
t2.start()
t1.join()
t2.join()

print(f"FINISH calculation duration: {time.time() - begin}")
```

```
START calculation
ID: 1 DONE duration: 2.3408639430999756
ID: 2 DONE duration: 2.341341018676758
FINISH calculation duration: 2.347946882247925
```

Как мы уже сказали, нет смысла считать математику в тредах: появляются накладные расходы на их переключение. Это может иметь смысл только при работе с IO-операциями, хотя это тоже не лучшее решение. Позже расскажем, почему, и что же применять на практике.

А пока ответьте на вопросы в тесте на следующей странице и переходите к новому модулю. В нем расскажем, как с помощью тредов сделать старую синхронную библиотеку асинхронной.