Работа с threading и multiprocessing в Python
САМОСТОЯТЕЛЬНАЯ РАБОТА

http://www.kimrt.ru
Онлайн-курс
Суперкомпьютерные технологии в задачах моделирования

Threading

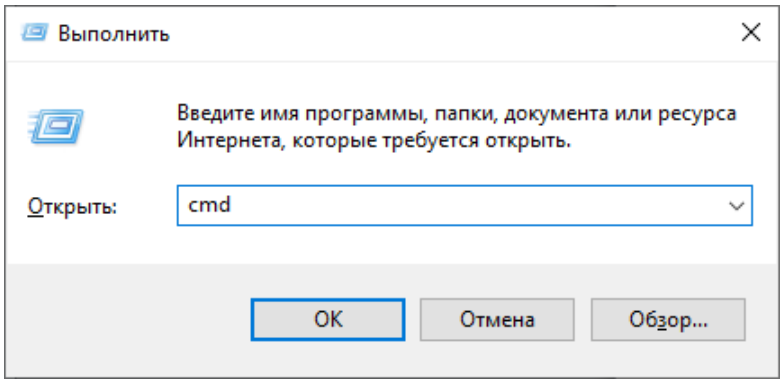
Этот модуль создает интерфейсы многопоточности более высокого уровня поверх модуля *_thread* [1] более низкого уровня [2–4]. Из-за глобальной блокировки интерпретатора (*GIL*) [5] только один поток может одновременно выполнять код Python, поэтому потоки лучше всего работают с операциями ввода/ вывода [6].

Multiprocessing

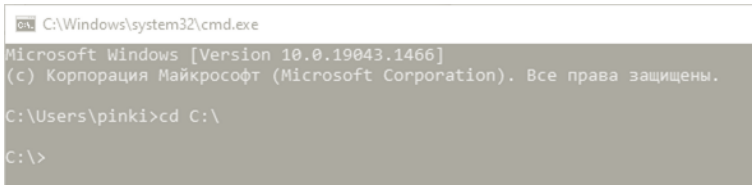
Multiprocessing — это пакет, который поддерживает создание процессов с использованием API, аналогичного модулю потоковой обработки. Многопроцессорный пакет предлагает как локальный, так и удаленный параллелизм, эффективно обходя глобальную блокировку интерпретатора за счет использования подпроцессов вместо потоков. Благодаря этому модуль многопроцессорности позволяет программисту полностью использовать несколько процессоров на данной машине [7, 8].

Multiprocessing и Windows

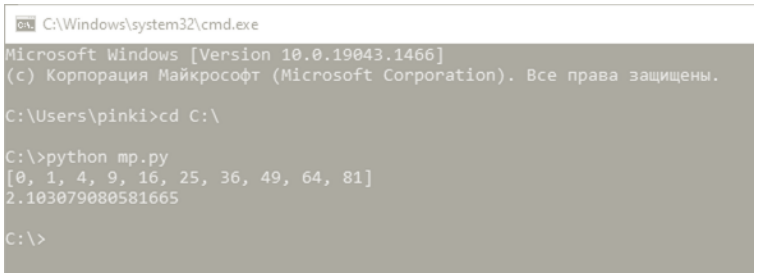
Multiprocessing нестабилен при использовании Windows [9, 10]. Программы использующие этот пакет могут работать некорректно. Если вы используете какую-либо ide при запуске таких программ, одним из вариантов решения проблемы является запуск скрипта через командную строку. Для этого запустите командную строку. Это можно сделать следующим образом: нажмите сочетание *Win+R*, пропишите в строку «открыть:» *cmd* и нажмите «ОК».



После чего откроется командная строка. Далее перейдите к расположению вашего скрипта с помощью команды *cd*, например, *cd C:*.



Запустите нужный скрипт с помощью *python [name]*, например, *python mp.py*.



Либо можно запускать такие скрипты, используя ОС Linux. Для этого откройте терминал с помощью сочетания клавиш *Ctrl + Alt + T*, перейдите в папку со скриптом с помощью *cd* и выполните следующую команду [11].

python mp.py

Multiprocessing

Рассмотрим следующий код.

Листинг 1.

```
# импорт библиотек
import time
# таймер
start_time = time.time()
# заполнение массива
arr = []
for i in range(10):
    arr.append(i)
```

```
# функция возведения в квадрат
def square(num):
    time.sleep(1)
    arr[num] **= 2


# поочередное возведение в квадрат
for num in range(len(arr)):
    square(num)
print(arr)
# таймер
stop_time = time.time()
print(stop_time - start_time)
```

В начале и в конце программы в переменные записывается системное время. Это необходимо для вычисления времени, которое затрачено программой на выполнение, *start_time = time.time() stop_time = time.time()*. Последняя строка выводит это время. Также в программе создается список и заполняется числами от 0 до 9. После чего все эти числа возводятся в квадрат и выводятся на экран. В функции с помощью которой производится возведение присутствует задержка перед выполнением. Эта задержка реализуется с помощью команды *sleep()* [12] и служит для симуляции вычислительной нагрузки. Запустим программу и посмотрим, как долго она будет выполняться.

```
C:\Users\pinki\AppData\Local\Programs\Python\Python310\python.exe C:/Users/pinki/Downloads/nomp.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
10.031839370727539

Process finished with exit code 0
```

Программа выполнилась примерно за 10 секунд. А теперь рассмотрим параллельную версию этой же программы.

Листинг 2

```
# импорт библиотек
import multiprocessing
import time


# функция возведения в квадрат
def square(num):
```

```
    time.sleep(1)
    return num ** 2


if __name__ == '__main__':
    # таймер
    start_time = time.time()
    # заполнение массива
    arr = []
    for i in range(10):
        arr.append(i)
    # создание пула процессов
    pool = multiprocessing.Pool(processes=6)
    print(pool.map(square, arr))
    # таймер
    stop_time = time.time()
    print(stop_time - start_time)
```

В этой программе аналогично предыдущей присутствует таймер, который выводит время затраченное на выполнение программы. Так же создается массив заполненный числами от 0 до 9, после чего эти числа возводятся в квадрат, но не с помощью поочередного перебора циклом, а с помощью *multiprocessing*. Сначала создается пул процессов с помощью команды *Pool()* [13]. С помощью аргумента processes задается количество процессов создаваемое в пуле. После чего *pool.map()* передается функция, которую нужно выполнить параллельно (*square*) и набор данных, который необходимо обработать с помощью этой функции (*arr*) [14]. Метод *pool.map()* параллельный эквивалент встроенной функции *map()*. В свою очередь *map()* возвращает итератор, который применяет функцию к каждому элементу итерации, получая результаты [15]. В модуле *multiprocessing* класс *Pool* в основном применяется для реализации некоего пула процессов, каждый из которых будет заботиться о задачах, поставляемых некому объекту *Pool*. Обычно класс *Pool* более удобен нежели класс *Process*, в особенности если необходимо упорядочивать данные, получаемые от ваших совместных приложений. Класс *Pool*: имеет методы *Pool.map()* и *Pool.apply()*, которые следуют соглашениям традиционных методов Python *map()* и *apply()*, гарантируя, что все возвращаемые значения упорядочиваются в точности в том порядке, как они поступают на вход. Тем не менее, эти методы блокируют свою основную программу до тех пор, пока

процесс не завершит обработку. Вследствие этого, класс *Pool* также имеет функции *map_async()* и *apply_async()* для лучшего обслуживания одновременной обработки и параллелизма [16].

Запустим программу. Время выполнения сократилось примерно в 5 раз.

```
C:\Users\pinki\AppData\Local\Programs\Python\Python310\python.exe C:/Users/pinki/Downloads/mp.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
2.094305992126465

Process finished with exit code 0
```

Можно создавать процессы вручную с помощью *Process()* [17].

Например:

Process1 = multiprocessing.Process(target=square, args=(num,))

В *target=* записывается функция, которую необходимо выполнить параллельно, а в *args=()* передаются ее аргументы.

Threading

Пакеты *threading* и *multiprocessing* схожи. Для создания потока используется *thread()* аналогичная *Process()*. Например:

thread1 = threading.Thread(target=web, args=(index,))

Рассмотрим два примера подробнее. Как говорилось ранее, *threading* хорошо работает с задачами не связанными с вычислениями на процессоре, поэтому рассмотрим задачу ввода/ вывода информации [18].

Листинг 3

```
import time
import urllib.request


start_time = time.time()


urls = ['https://docs.python.org/3/tutorial/index.html',
        'https://docs.python.org/3/library/index.html',
        'https://docs.python.org/3/library/threading.html',
        'https://docs.python.org/3/library/multiprocessing.html',
        'https://docs.python.org/3/installing/index.html',
```

```
        'https://docs.python.org/3/distributing/index.html',
        'https://docs.python.org/3/using/cmdline.html',
        'https://docs.python.org/3/using/unix.html#getting-and-
installing-the-latest-version-of-python',
        'https://docs.python.org/3/using/configure.html',
        'https://docs.python.org/3/using/windows.html',
        'https://docs.python.org/3/using/mac.html',
        'https://docs.python.org/3/using/editors.html',

'https://wiki.python.org/moin/IntegratedDevelopmentEnvironments',
        'https://wiki.python.org/moin/PythonEditors',
        'https://docs.python.org/3/glossary.html']


sites = []


for i in range(len(urls)):
    sites.append(urllib.request.urlopen(urls[i]))


stop_time = time.time()
print(stop_time - start_time)
```

Эта программа поочередно открывает ссылки из списка *urls* с помощью *urlopen* [19]. На открытие всех страниц ушло около 6 секунд.

```
E:\Python\Python39\pythonw.exe "E:/Zolotarev/самостоятельные работы/multiproc python/nothurls.py"
6.3443686962127686

Process finished with exit code 0
```

Листинг 4

```
import threading
import time
import urllib.request

def web(index):
    urllib.request.urlopen = urls[index]

if __name__ == '__main__':
```

```
    start_time = time.time()
    urls = ['https://docs.python.org/3/tutorial/index.html',
            'https://docs.python.org/3/library/index.html',
            'https://docs.python.org/3/library/threading.html',

'https://docs.python.org/3/library/multiprocessing.html',
            'https://docs.python.org/3/installing/index.html',
            'https://docs.python.org/3/distributing/index.html',
            'https://docs.python.org/3/using/cmdline.html',
            'https://docs.python.org/3/using/unix.html#getting-
and-installing-the-latest-version-of-python',
            'https://docs.python.org/3/using/configure.html',
            'https://docs.python.org/3/using/windows.html',
            'https://docs.python.org/3/using/mac.html',
            'https://docs.python.org/3/using/editors.html',

'https://wiki.python.org/moin/IntegratedDevelopmentEnvironments',
            'https://wiki.python.org/moin/PythonEditors',
            'https://docs.python.org/3/glossary.html']
    threads = []
    for index in range(len(urls)):
        thread = threading.Thread(target=web, args=(index,))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()
    stop_time = time.time()
    print(stop_time - start_time)
```

Эта программа тоже открывает ссылки, но для каждой ссылки создается отдельный поток. Время затраченное программой составило около 0,02 секунды.

```
E:\Python\Python39\pythonw.exe "E:/Zolotarev/самостоятельные работы/multiproc python/thurls.py"
0.015623331069946289

Process finished with exit code 0
```

http://www.kimrt.ru/index/course_stm/0-24

Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры 2020/2021 благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Источники

1. Модуль _thread.URL: https://docs.python.org/3/library/_thread.html#module-_thread
2. Модуль threading. URL: https://docs.python.org/3/library/threading.html#module-threading
3. Работа с потоками в Python. URL: http://onreader.mdl.ru/MasteringConcurrencyInPython/content/Ch03.html
4. Модуль threading на примерах. URL: http://python-3.ru/page/import-threading
5. GIL. URL: https://docs.python.org/3/glossary.html#term-global-interpreter-lock
6. Разбираемся с параллельными и конкурентными вычислениями в Python. URL: https://habr.com/ru/company/wunderfund/blog/581994/
7. Multiprocessing. URL: https://docs.python.org/3/library/multiprocessing.html
8. Модуль multiprocessing на примерах. URL: http://python-3.ru/page/multiprocessing
9. Почему Python multiprocessing нестабилен? URL: https://qna.habr.com/q/280506
10. Python multiprocessing is different under Linux and Windows. URL: https://rhodesmill.org/brandon/2010/python-multiprocessing-linux-windows/
11. Запуск python скрипта в Linux. URL: https://losst.ru/zapusk-python-skripta-v-linux
12. Time.sleep. URL: https://docs.python.org/3/library/time.html#time.sleep
13. Pool. URL: https://docs.python.org/3/library/multiprocessing.html#multiprocessing.pool.Pool
14. Pool.map. URL: https://docs.python.org/3/library/multiprocessing.html#multiprocessing.pool.Pool.map
15. Map. URL: https://docs.python.org/3/library/functions.html#map
16. Работа с процессами в Python. URL: http://onreader.mdl.ru/MasteringConcurrencyInPython/content/Ch06.html
17. Process. URL: https://docs.python.org/3/library/multiprocessing.html#multiprocessing.Process
18. Зачем, когда и как использовать multithreading и multiprocessing в Python. URL: https://habr.com/ru/company/otus/blog/501056/
19. urllib.request. URL: https://docs.python.org/3/library/urllib.request.html#module-urllib.request