Python Multiprocessing

Summary: in this tutorial, you'll learn how to run code in parallel using the Python multiprocessing module.

Introduction to the Python multiprocessing

Generally, programs deal with two types of tasks:

1. I/O bound tasks: if a task does a lot of input/output operations, it's called I/O-bound tasks. Typical examples of I/O-bound tasks are reading from files, writing to files, connecting to the databases, and making a network request. For I/O bound tasks, you can use multithreading to speed them up.

2. CPU bound tasks: when a task does a lot of operations using CPU, it's called a CPU-bound task. For example, number calculation, image resizing, and video streaming are CPU-bound tasks. To speed up the program with lots of CPU-bound tasks, you use multiprocessing.

Multiprocessing allows two or more processors to simultineously process two or more different part of a program. In Python, you use the `multiprocessing` module to implement multiprocessing.

Python multiprocessing example

See the following program:

```
import time

def task(n=100_000_000):
    while n:
        n -= 1

if __name__ == '__main__':
    start = time.perf_counter()
    task()
    task()
    finish = time.perf_counter()

    print(f'It took {finish-start: .2f} second(s) to finish')
```

Output:

```
It took  12.94 second(s) to finish
```

How it works.

First, define the `task()` funciton that has a big while loop from 10 mil to 0. The `task()` function is CPU-bound because it deals with calcualtion.

```
def task(n=100_000_000):
    while n:
        n -= 1
```

Second, call the `task()` functions twice and record the processing time:

```
if __name__ == '__main__':
    start = time.perf_counter()
```

```
    task()
    task()
    finish = time.perf_counter()

    print(f'It took {finish-start: .2f} second(s) to finish')
```

In our computer, it took 0.78 second to complete.

Using multiprocessing module

The following program uses multiprocessing module but takes less time:

```
import time
import multiprocessing

def task(n=100_000_000):
    while n:
        n -= 1

if __name__  ==  '__main__':
    start = time.perf_counter()

    p1 = multiprocessing.Process(target=task)
    p2 = multiprocessing.Process(target=task)

    p1.start()
    p2.start()

    p1.join()
    p2.join()

    finish = time.perf_counter()

    print(f'It took {finish-start: .2f} second(s) to finish')
```

Output:

```
It took  6.45 second(s) to finish
```

How it works.

First, import the multiprocessing module:

```
import multiprocessing
```

Second, create two processes and pass the task function to each:

```
p1 = multiprocessing.Process(target=task)
p2 = multiprocessing.Process(target=task)
```

Note that the `Process()` constructor returns a new `Process` object.

Third, call the `start()` method of the `Process` objects to start the process:

```
p1.start()
p2.start()
```

Finaly, wait for the processes to complete by calling the `join()` method:

```
p1.join()
p2.join()
```

Python multiprocessing practical example

We'll use the multiprocessing module to resize the high resolution images.

First, install the `Pillow` library for image processing:

```
pip install Pillow
```

Second, develop a program that creates the thumbnails of the pictures in the `images` folder and save them to the `thumbs` folder:

```
import time
import os
from PIL import Image, ImageFilter

filenames = [
    'images/1.jpg',
    'images/2.jpg',
    'images/3.jpg',
    'images/4.jpg',
    'images/5.jpg',
]

def create_thumbnail(filename, size=(50,50), thumb_dir ='thumbs'):
    img = Image.open(filename)
    img = img.filter(ImageFilter.GaussianBlur())
    img.thumbnail(size)
    img.save(f'{thumb_dir}/{os.path.basename(filename)}')
    print(f'{filename} was processed...')


if __name__ == '__main__':
    start = time.perf_counter()

    for filename in filenames:
        create_thumbnail(filename)

    finish = time.perf_counter()
```

```
    print(f'It took {finish-start: .2f} second(s) to finish')
```

In our computer, it took about 1.28s to complete:

```
images/1.jpg was processed...
images/2.jpg was processed...
images/3.jpg was processed...
images/4.jpg was processed...
images/5.jpg was processed...
It took  1.28 second(s) to finish
```

Third, modify the program to use multiprocessing. Each process will create a thumbnail for a picture:

```
import time
import os
import multiprocessing
from PIL import Image, ImageFilter


filenames = [
    'images/1.jpg',
    'images/2.jpg',
    'images/3.jpg',
    'images/4.jpg',
    'images/5.jpg',
]

def create_thumbnail(filename, size=(50,50),thumb_dir ='thumbs'):
    img = Image.open(filename)
    img = img.filter(ImageFilter.GaussianBlur())
    img.thumbnail(size)
    img.save(f'{thumb_dir}/{os.path.basename(filename)}')
```

```
    print(f'{filename} was processed...')


if __name__ == '__main__':
    start = time.perf_counter()

    # create processes
    processes = [multiprocessing.Process(target=create_thumbnail, args=[filename]
                for filename in filenames]

    # start the processes
    for process in processes:
        process.start()

    # wait for completion
    for process in processes:
        process.join()

    finish = time.perf_counter()

    print(f'It took {finish-start: .2f} second(s) to finish')
```

Output:

```
images/5.jpg was processed...
images/4.jpg was processed...
images/1.jpg was processed...
images/3.jpg was processed...
images/2.jpg was processed...
It took  0.82 second(s) to finish
```

In this case, the output shows that the program processed the pictures much faster.

Summary

- Use Python multiprocessing to run code in parallel to deal with CPU-bound tasks.