Multiprocessing in Python

The multiprocessing module was added to Python in version 2.6. It was originally defined in PEP 371 by Jesse Noller and Richard Oudkerk. The multiprocessing module allows you to spawn processes in much that same manner than you can spawn threads with the threading module. The idea here is that because you are now spawning processes, you can avoid the Global Interpreter Lock (GIL) and take full advantages of multiple processors on a machine.

The multiprocessing package also includes some APIs that are not in the threading module at all. For example, there is a neat Pool class that you can use to parallelize executing a function across multiple inputs. We will be looking at Pool in a later section. We will start with the multiprocessing module's **Process** class.

Getting Started With Multiprocessing

The **Process** class is very similar to the threading module's Thread class. Let's try creating a series of processes that call the same function and see how that works:

```
import os

from multiprocessing import Process


def doubler(number):
    """
    A doubling function that can be used by a process
    """
    result = number * 2
    proc = os.getpid()
    print('{0} doubled to {1} by process id: {2}'.format(
        number, result, proc))

if __name__ == '__main__':
    numbers = [5, 10, 15, 20, 25]
    procs = []
```

```
    for index, number in enumerate(numbers):
        proc = Process(target=doubler, args=(number,))

        procs.append(proc)
        proc.start()

    for proc in procs:
        proc.join()
```

For this example, we import Process and create a **doubler** function. Inside the function, we double the number that was passed in. We also use Python's **os** module to get the current process's ID (or pid). This will tell us which process is calling the function. Then in the block of code at the bottom, we create a series of Processes and start them. The very last loop just calls the **join()** method on each process, which tells Python to wait for the process to terminate. If you need to stop a process, you can call its **terminate()** method.

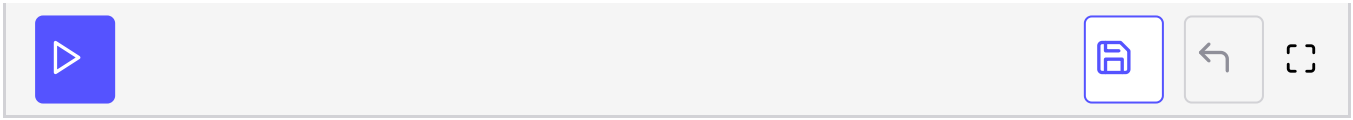
When you run this code, you should see output that is similar to the following:

```
5 doubled to 10 by process id: 10468
10 doubled to 20 by process id: 10469
15 doubled to 30 by process id: 10470
20 doubled to 40 by process id: 10471
25 doubled to 50 by process id: 10472
```

Sometimes it's nicer to have a more human readable name for your process though. Fortunately, the Process class does allow you to access the name of your process. Let's take a look:

```
import os

from multiprocessing import Process, current_process


def doubler(number):
    """
    A doubling function that can be used by a process
    """
    result = number * 2
    proc_name = current_process().name
    print('{0} doubled to {1} by: {2}'.format(
        number, result, proc_name))


if __name__ == '__main__':
    numbers = [5, 10, 15, 20, 25]
```

```
    procs = []
    proc = Process(target=doubler, args=(5,))


    for index, number in enumerate(numbers):
        proc = Process(target=doubler, args=(number,))
        procs.append(proc)
        proc.start()

    proc = Process(target=doubler, name='Test', args=(2,))
    proc.start()
    procs.append(proc)

    for proc in procs:
        proc.join()
```



This time around, we import something extra: **current_process**. The current_process is basically the same thing as the threading module's **current_thread**. We use it to grab the name of the thread that is calling our function. You will note that for the first five processes, we don't set a name. Then for the sixth, we set the process name to "Test". Let's see what we get for output:

```
5 doubled to 10 by: Process-2
10 doubled to 20 by: Process-3
15 doubled to 30 by: Process-4
20 doubled to 40 by: Process-5
25 doubled to 50 by: Process-6
2 doubled to 4 by: Test
```

The output demonstrates that the multiprocessing module assigns a number to each process as a part of its name by default. Of course, when we specify a name, a number isn't going to get added to it.