Forkserver

This lesson discusses the less-used forkserver option to spin off new processes.

Forkserver

The third and final option that can be chosen for the start method is the "forkserver". The official documentation states that with forkserver as the start method, a brand new single-threaded process, called server is started. Whenever, a new process needs to be created, the parent process connects to the server and requests that it forks a new process. Since the server process is single threaded, it can safely invoke `os.fork()` to create a new process.

The example below appears from the previous section and now uses the forkserver as the start method on **line 13**.

```
 1  from multiprocessing import Process
 2  import multiprocessing
 3
 4  class Test:
 5      value = 777
 6
 7
 8  def process_task():
 9      print(Test.value)
10
11
12  if __name__ == '__main__':
13      multiprocessing.set_start_method('forkserver')
14
15      # change the value of Test.value before creating
16      # a new process
17      Test.value = 999
18      process = Process(target=process_task, name="process-1")
19      process.start()
20      process.join()
21
```

Run Save Reset

```
Output                                                    0.92s
```

```
777
```

You can verify on your system that a new server process gets created, when we select the start method as forkserver. Note forkserver doesn't work on a Windows system but for others, you may want to save the following script in a file and run it. Make sure your version of python is atleast 3.6+

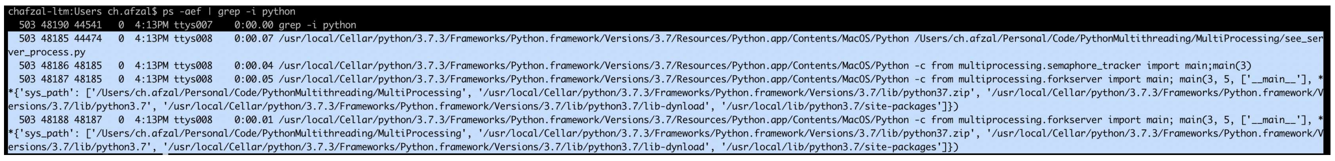
```
from multiprocessing import Process
import multiprocessing
import time


def process_task():
    # block the new process
    time.sleep(1000 * 1000)


if __name__ == '__main__':
    multiprocessing.set_start_method('forkserver')

    process = Process(target=process_task)
    process.start()
    print("New process created")
    # block the main process too
    process.join()
```

Run the above script in your terminal and the program would hang. Next execute **"ps -aef | grep python"** in a new terminal window. The output should list the python processes. Below, I share the screen-shot on my machine with python processes highlighted.



```
chafzal-ltm:Users ch.afzal$ ps -aef | grep -i python
  503 48190 44541   0  4:13PM ttys007    0:00.00 grep -i python
  503 48185 44474   0  4:13PM ttys008    0:00.07 /usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/Versions/3.7/Resources/Python.app/Contents/MacOS/Python /Users/ch.afzal/Personal/Code/PythonMultithreading/MultiProcessing/see_ser
ver_process.py
  503 48186 48185   0  4:13PM ttys008    0:00.04 /usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/Versions/3.7/Resources/Python.app/Contents/MacOS/Python -c from multiprocessing.semaphore_tracker import main;main(3)
  503 48187 48185   0  4:13PM ttys008    0:00.05 /usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/Versions/3.7/Resources/Python.app/Contents/MacOS/Python -c from multiprocessing.forkserver import main; main(3, 5, ['__main__'], *
*{'sys_path': ['/Users/ch.afzal/Personal/Code/PythonMultithreading/MultiProcessing', '/usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/Versions/3.7/lib/python37.zip', '/usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/V
ersions/3.7/lib/python3.7', '/usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/Versions/3.7/lib/python3.7/lib-dynload', '/usr/local/lib/python3.7/site-packages']})
  503 48188 48187   0  4:13PM ttys008    0:00.01 /usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/Versions/3.7/Resources/Python.app/Contents/MacOS/Python -c from multiprocessing.forkserver import main; main(3, 5, ['__main__'], *
*{'sys_path': ['/Users/ch.afzal/Personal/Code/PythonMultithreading/MultiProcessing', '/usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/Versions/3.7/lib/python37.zip', '/usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/V
ersions/3.7/lib/python3.7', '/usr/local/Cellar/python/3.7.3/Frameworks/Python.framework/Versions/3.7/lib/python3.7/lib-dynload', '/usr/local/lib/python3.7/site-packages']})
```

If you run the suggested command, you should see four python processes (not counting the grep). One is the parent process, second the child process, third the forkserver process and fourth a *semaphore tracker* process.

Semaphore Tracker

The semaphore tracker process gets created with both spawn and forkserver as start methods. There is a limit on the number of semaphores that can exist on a Unix based system. Semaphores when created by a process are linked to that process and may remain linked even if the process exits in an abnormal way e.g.

it is killed. The semaphore tracker process is tasked with making sure there are no leaking semaphores i.e. it unlinks semaphores which were linked to processes that have already exited.

Back
Spawn

Mark As Completed

Next
Queues & Pipes