Multiple Processors

This lesson discusses the implementation of the web-service using multiple processors.

Multiple Processors

> We'll rewrite our service from the previous section to use a **`ProcessPoolExecutor`** to process requests instead of threads. The changes appear in the widget below.

```
from threading import Thread
from concurrent.futures import ProcessPoolExecutor

import socket, time, random, sys


def find_nth_prime(nth_prime):
    i = 2
    nth = 0

    while nth != nth_prime:
        if is_prime(i):
            nth += 1
            last_prime = i

        i += 1

    return last_prime


def is_prime(num):
    if num == 2 or num == 3:
        return True

```

```
25      div = 2
26
27      while div <= num / 2:
28          if num % div == 0:
```

Note that the GIL and switch intervals become irrelevant when we move the implementation based on multiple processors. Also, note that the total number of requests completed per second without the long-running request being injected is less than the total number of requests completed when using threads or a single thread. The reason is that there's additional overhead because of inter-process communication. However, from the output, you can verify that the requests completed per second aren't meaningfully impacted when the long-running request is received by the server. The long-running request is contained to one process on a single processor and the rest of the CPUs can be used to serve other requests through the process pool.

Back
Multiple Threads

Next
Asynchronous

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/multiple-processors__global-interpreter-lock__python-concurrency-for-senior-engineering-interviews)