Introduction

This lesson lays the context for a hands-on exercise we'll undertake to study the effects of GIL on a web-service when designed and implemented using different concurrency paradigms.

Introduction

We have already discussed the Global Interpreter Lock and the challenges it presents when writing concurrent applications in Python. Depending on the use case we can either apply a multithreaded approach or an asynchronous one but in either case, we'll need to be mindful of the ways in which the GIL can impact our solution. In this section, we'll work with a web service that is implemented first using threads and then asyncio and examine the issues we'll face with compute-bound tasks. The examples in this section are inspired from Dave Beazley's (https://www.dabeaz.com/) PyCon talk (https://www.youtube.com/watch?v=MCs5OvhV9S4).

Setup

We'll create a very simple webservice that returns the **nth** prime number requested by a client. For instance, if the input is 1 the service returns the first prime number which is 2, if the input is 10 the service returns 29, which is the 10th prime number. We pick a very inefficient compute-intensive algorithm to find the nth prime number. You shouldn't focus on the code that finds the prime number as it can be substituted for any computationally intensive task.

Initially, we'll start with a multithreaded implementation of the server. The code which starts up the service is shown below:

```
    def run_service(self):
        s = socket.socket()
        s.bind(('', self.server_port))

        # put the socket into listening mode
        s.listen(5)
        client_socket, addr = s.accept()
        data = client_socket.recv(4096).decode()
        nth_prime = int(data)
        print("Asked to find the {0}-nth prime".format(nth_pri
me))

        prime = self.find_nth_prime(nth_prime)
        client_socket.send(str(prime).encode())
```

The complete code appears in the code-widget below.

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

import socket, time, random


class PrimeService():

    def __init__(self, server_port):
        self.server_port = server_port

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

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

            i += 1

        return last_prime

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

        div = 2

        while div <= num / 2:
            if num % div == 0:
                return False
            div += 1
        return True

    def run_service(self):
        s = socket.socket()
        s.bind(('', self.server_port))

        # put the socket into listening mode
        s.listen(5)
        client_socket, addr = s.accept()
        data = client_socket.recv(4096).decode()
        nth_prime = int(data)
        print("Asked to find the {0}-th prime".format(nth_prime))

        prime = self.find_nth_prime(nth_prime)
        client_socket.send(str(prime).encode())
```

```
def run_client(server_host, server_port):

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.connect((server_host, server_port))
    server_socket.send("7".encode())
    result = server_socket.recv(4096).decode()
    print("prime found to be {0}".format(result))


if __name__ == "__main__":
    server_port = random.randint(10000, 65000)
    server_host = "127.0.01"
    server = PrimeService(server_port)

    server_thread = Thread(target=server.run_service, daemon=True)
    server_thread.start()

    run_client(server_host, server_port)

    time.sleep(10)
```

← Back
Quiz

Next →
Single Thread