Single Thread

This lesson discusses the implementation of the web-service using a single thread.

Single thread

In this lesson, we'll present a simulation that bombards the server with a trivial request asking for the 1st prime number in an infinite loop. The server has a single thread that serves the request as well as finds the requested prime number. Additionally, we spin up another thread, **monitor thread** that counts the number of requests the server is able to complete in a second. We keep a counter that is reset to zero after every second and the number of requests being completed is output every second. The implementation from the previous section changes as below:

```
    def moniter_thread(self):
        while True:
            time.sleep(1)
            print("{0} requests/min".format(self.requests))
            self.requests = 0

    def run_service(self):

        s = socket.socket()
        s.bind(('', self.server_port))

        # put the socket into listening mode
        s.listen(5)

        while True:
            client_socket, addr = s.accept()

            data = client_socket.recv(4096).decode()
            nth_prime = int(data)
            prime = self.find_nth_prime(nth_prime)
            client_socket.send(str(prime).encode())
            self.requests += 1
            client_socket.close()
```

We'll also need to modify the client code to run an infinite loop of requests as follows:

```
def run_client(server_host, server_port):
    while True:
        server_socket = socket.socket(socket.AF_INET, socket.S
OCK_STREAM)
        server_socket.connect((server_host, server_port))
        server_socket.send("1".encode())
        server_socket.recv(4096).decode()
        server_socket.close()
```

We'll also initiate a thread with a delay that asks for the 10,000th prime number. Evidently, this would completely consume the single thread that is serving the requests and the number of requests per second would drop to zero.

The simulation in the code widget below runs the service, the monitor thread and the client which makes trivial requests in an infinite loop for roughly ten seconds. Three seconds into the program, we spawn the thread that asks for the 10,000-th prime number and the number of requests completed per seconds drops to zero.

```
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
        self.requests = 0

    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 moniter_thread(self):
        while True:
            time.sleep(1)
            print("{0} requests/min".format(self.requests), flush=True)
            self.requests = 0

    def run_service(self):

        s = socket.socket()
        s.bind(('', self.server_port))

        # put the socket into listening mode
        s.listen(5)
        while True:

            client_socket, addr = s.accept()

            data = client_socket.recv(4096).decode()
            nth_prime = int(data)
            prime = self.find_nth_prime(nth_prime)
            client_socket.send(str(prime).encode())
            self.requests += 1
            client_socket.close()


def run_client(server_host, server_port):
    while True:
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.connect((server_host, server_port))
        server_socket.send("1".encode())
        server_socket.recv(4096).decode()
        server_socket.close()


def run_long_request(server_host, server_port):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.connect((server_host, server_port))

    while 1:
        server_socket.send("100000".encode())
        server_socket.recv(4096)


if __name__ == "__main__":
    random.seed(time.time())
    server_port = random.randint(10000, 65000)
    server_host = "127.0.0.1"
    server = PrimeService(server_port)

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

    monitor_thread = Thread(target=server.moniter_thread, daemon=True)
    monitor_thread.start()

    small_reqs_thread = Thread(target=run_client, args=(server_host, server_port),
    small_reqs_thread.start()

    time.sleep(3)

    long_reqs_thread = Thread(target=run_long_request, args=(server_host, server_po
    long_reqs_thread.start()

    time.sleep(100)
```

The output from the above program demonstrates that the single thread fails to field any new requests once it starts processing a larger request.

The usual way out of this predicament is to spawn a separate thread to deal with each new client connection, that way the main thread that listens for incoming requests doesn't block. In the next section, we'll implement that version of the program.

Back
Introduction

Next
Multiple Threads

✓ Mark as Completed

Report an Issue

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