

# 컴퓨터 네트워크

## 6주차

(다중 클라이언트 연결 웹 서버 구조와 성능 테스트)

학번: 201404377

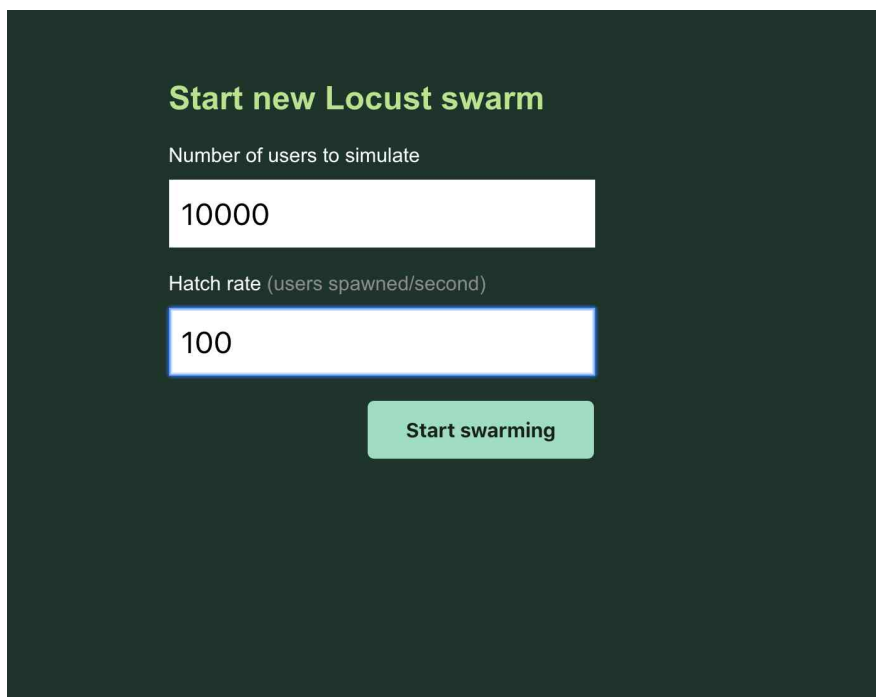
이름: 진승언

## -과제 목표-

이번 과제의 목표는 다중 클라이언트 연결 웹 서버 구조와 성능 테스트이다. 테스트를 위해 Locust라는 서버 테스트 도구를 사용하도록 한다. 그리고 도커 컨테이너 안에 서버를 두도록 한다. 서버는 단일 프로세스, 멀티 프로세스, 멀티 스레드 소켓 서버가 있다. python 코드로 구현하였다. 각 소켓 서버의 성능을 테스트하고 차이점을 알아보았다.

## -해결 방법-

가장 먼저 도커 안에 단일 프로세스, 멀티 프로세스, 멀티 스레드 소켓 서버를 만들고 클라이언트에서 locust를 사용해서 도커의 서버에 접속했다. 테스트 환경은 내 노트북 윈도우의 vmware 리눅스 안에 도커 서버를 두고 같은 와이 파이 환경인 친구의 맥북을 클라이언트로 사용하였다. 도커에서는 포트 8000번을 열고 외부에서는 1234 포트로 접속하면 된다. (포트 포워딩)



**Start new Locust swarm**

Number of users to simulate

10000

Hatch rate (users spawned/second)

100

Start swarming

모든 테스트의 기준은 위와 같이 하였다. 각각 가상 유저 수와 초당 실행 수를 의미한다.

```
u201404377@ubuntu:~/u201404377/network_week6/network_3$ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:cbff:fe70:2928 prefixlen 64 scopeid 0x20<link>
    ether 02:42:cb:70:29:28 txqueuelen 0 (Ethernet)
    RX packets 265998 bytes 21655081 (21.6 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 311209 bytes 28596588 (28.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.30.1.53 netmask 255.255.255.0 broadcast 172.30.1.255
    inet6 fe80::4620:4937:56cc:caf4 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:7d:55:88 txqueuelen 1000 (Ethernet)
    RX packets 542272 bytes 58548840 (58.5 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 447531 bytes 41520959 (41.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
```

ifconfig 결과는 다음과 같다. ip:172.30.1.53으로 접속하면 된다. (같은 와이파이를 사용해야한다.)

차례대로 코드와 locust 테스트 결과를 보면 다음과 같다.

## 1. 단일 프로세스 소켓 서버

```
network@49df03a8dc27: /home/network_3
File Edit View Search Terminal Help
import time
import os
import socket

def send_recv(client_socket):
    data = client_socket.recv(1024)
    print("[client {}] {}".format(os.getpid(), data.decode()))
    response = "HTTP/1.1 200 OK\r\n"
    client_socket.send(response.encode('utf-8'))
    client_socket.send(data)
    client_socket.close()

def main(FLAGS):
    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serversocket.bind('', FLAGS.port)
    serversocket.listen()

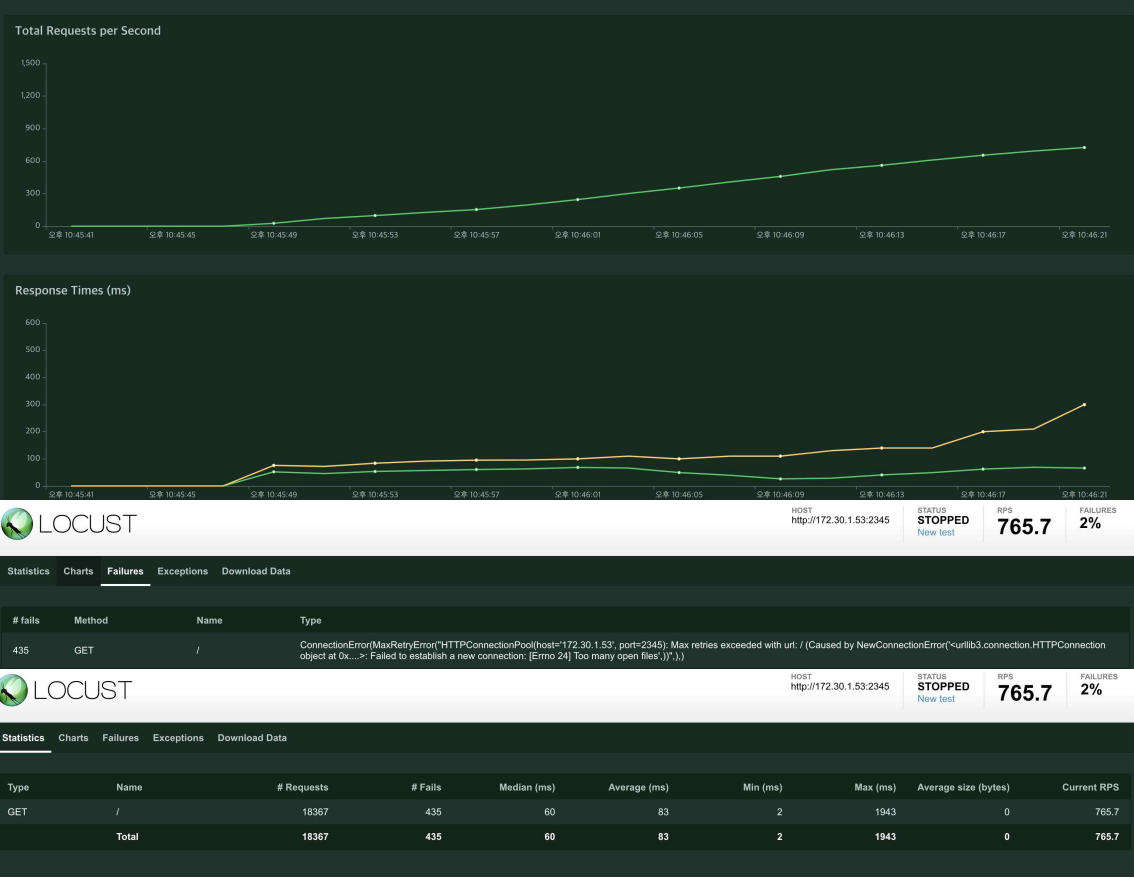
    while True:
        (clientsocket, address) = serversocket.accept()
        print("accept client from", address)
        send_recv(clientsocket)

if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--port',
                        type = int,
                        default = 8000,
                        help="input port number")
    FLAGS, _ = parser.parse_known_args()
    main(FLAGS)
```

단일 프로세스는 데이터가 send/recv가 완료될 때까지 block되고, 다른 프로세스에게 자원을 넘긴다. 즉 동시에 처리하지 못한다. 응답메세지도 200 OK로 보내준다.

단일 프로세스의 locust 결과는 다음과 같다.



클라이언트에서 get 요청을 단일 프로세스 소켓 서버에게 보낸다. RPS(requests per second) 가 765.7이다. 18367번의 요청을 했을 시 failure가 발생한다. failure는 단일 프로세스가 처리할 수 있는 양을 넘어서 발생하게 된다. 평균 응답시간은 83ms이다.

## 2. 멀티 프로세스 소켓 서버

```
network@49df03a8dc27: /home/network_3
File Edit View Search Terminal Help
from multiprocessing import Process
import socket
import os

def send_recv(client, address):
    msg = client.recv(1024)
    print("[client {}] {}".format(os.getpid(), msg.decode()))
    response = "HTTP/1.1 200 OK\r\n"
    client.send(response.encode('utf-8'))
    client.close()

def main(FLAGS):
    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serversocket.bind(('', FLAGS.port))
    serversocket.listen(5)
    clients = list()

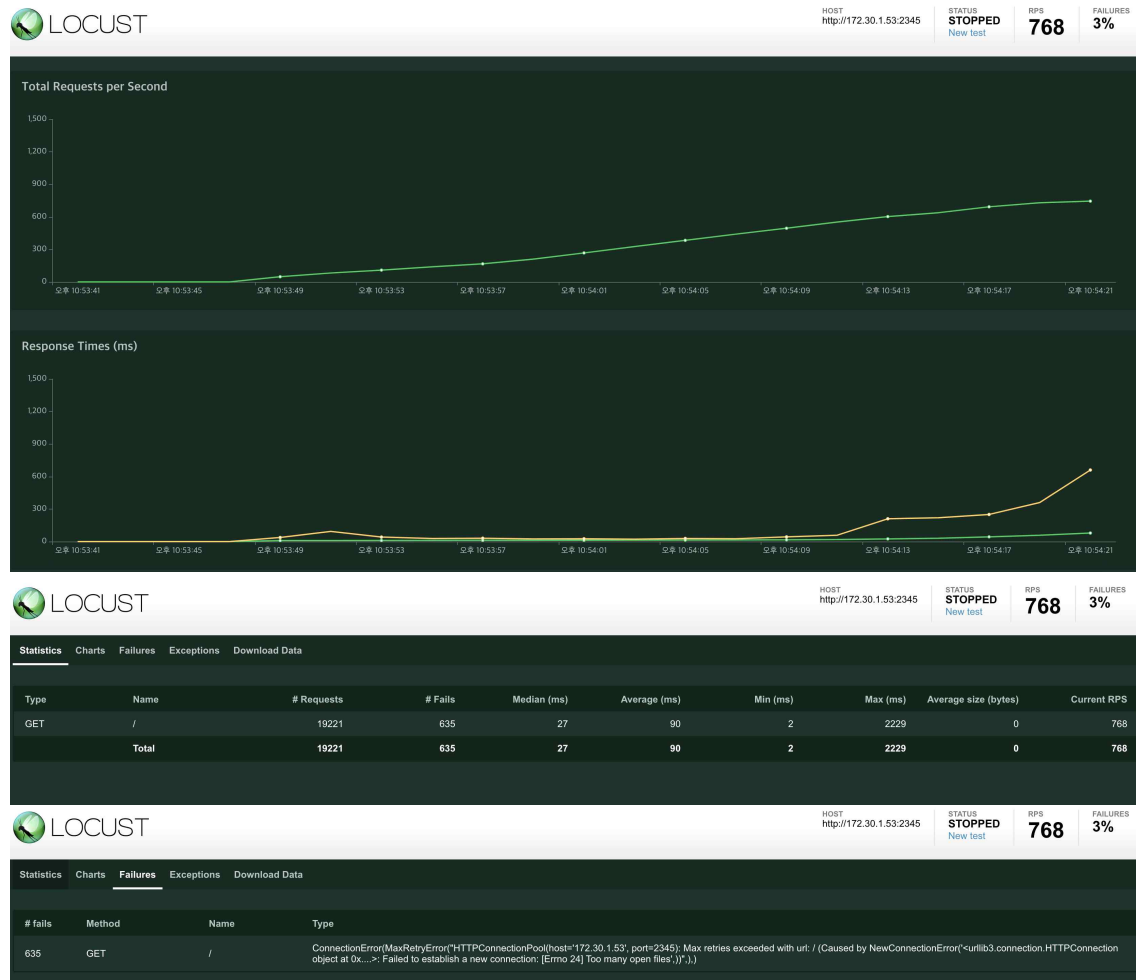
    while True:
        client, address = serversocket.accept()
        print("accept client from", address)
        clients.append(client)
        proc = Process(target=send_recv, args = (client, address))
        proc.start()
        proc.join()

if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--port', type = int, default=8000, help="input port number")
    FLAGS, _ = parser.parse_known_args()
    main(FLAGS)
```

멀티프로세스는 클라이언트를 동시에 처리할 수 있다. fork()를 통해 클라이언트가 accept되면 process를 복사한다. 다수의 프로세스로 클라이언트의 요청을 처리한다. 그리고 200 OK 응답 메시지를 보내준다.

멀티프로세스의 locust 테스트 결과는 다음과 같다.



클라이언트에서 get 요청을 단일 프로세스 소켓 서버에게 보낸다. RPS 가 768이다. 19221번의 요청을 했을 시 failure가 발생한다. failure는 멀티 프로세스가 처리할 수 있는 양을 넘어서 발생하게 된다. 평균 응답시간은 90ms이다.



### 3. 멀티 스레드 소켓 서버

```
network@49df03a8dc27: /home/network_3
File Edit View Search Terminal Help
from threading import Thread
import socket
import os

def send_recv(client_socket, address):
    data = client_socket.recv(1024)
    print("[client {} ] {}".format(os.getpid(), data.decode()))
    msg = "HTTP/1.1 200 OK\r\n"
    client_socket.send(msg.encode('utf-8'))
    client_socket.send(data)
    client_socket.close()

def main(FLAGS):
    serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serversocket.bind(('', FLAGS.port))
    print("listening...")
    serversocket.listen(5)
    client_socket = list()

    while True:
        client, address = serversocket.accept()
        print("accept client from", address)

        th = Thread(target=send_recv, args=(client, address))
        th.start()
        th.join()

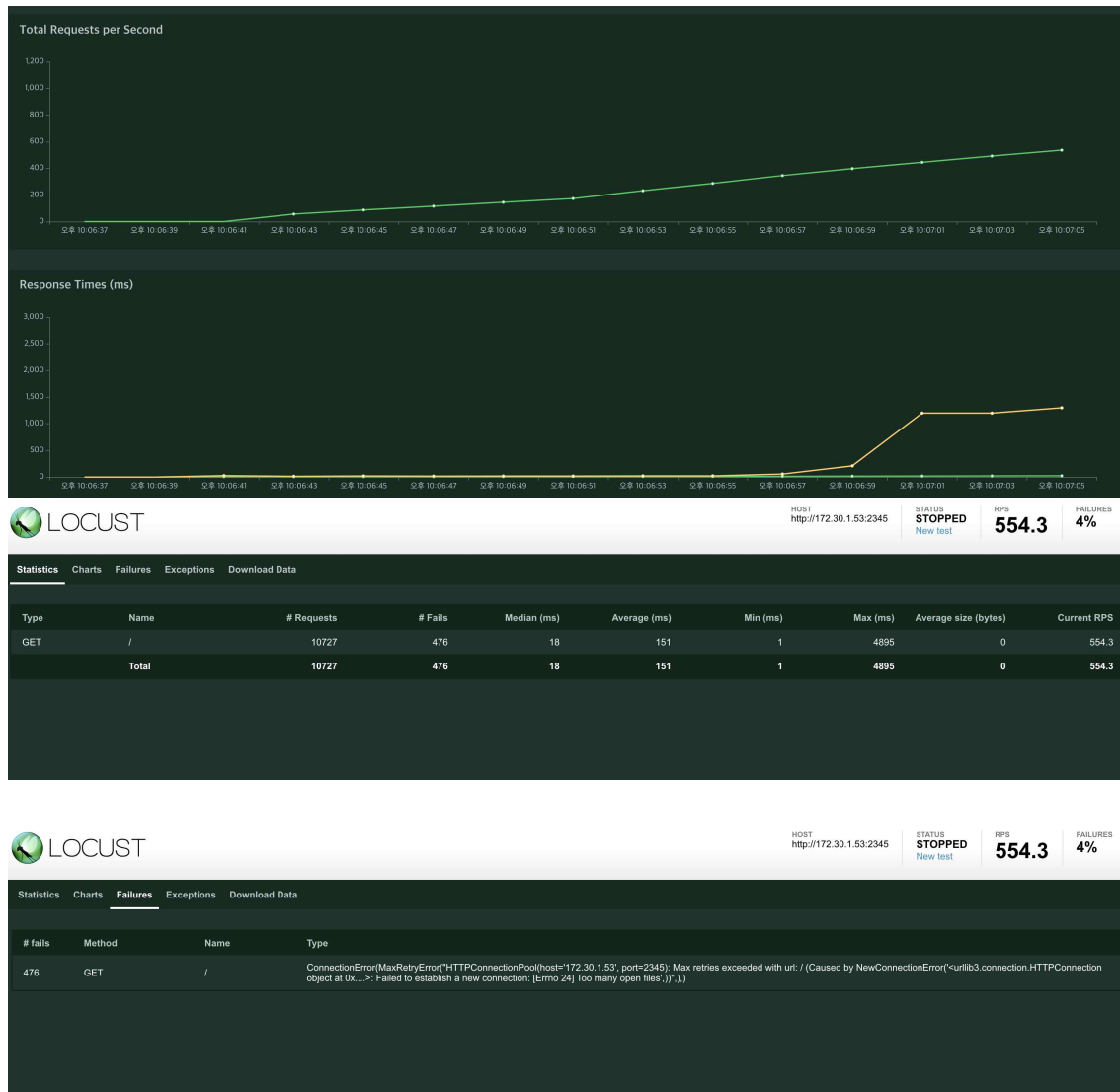
if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--port', type = int, default=8000, help="input port number")
    FLAGS, _ = parser.parse_known_args()
    main(FLAGS)
```

멀티 스레드도 클라이언트를 동시에 처리할 수 있다. 하나의 프로세스 안에서 스레드를 이용해서 클라이언트의 요청을 처리한다.  
그리고 클라이언트에게 200 OK 응답 메시지를 보내준다.



멀티 스레드의 locust 테스트 결과는 다음과 같다.



클라이언트에서 get 요청을 단일 프로세스 소켓 서버에게 보낸다. RPS 가 554.3이다. 10727번의 요청을 했을 시 failure가 발생한다. failure는 멀티 스레드가 처리할 수 있는 양을 넘어서 발생하게 된다. 평균 응답시간은 151ms이다.

## 비교 (단일 프로세스 vs 멀티 프로세스 vs 멀티 스레드 소켓 서버)

### 1. RPS

멀티 스레드(554.3) < 단일 프로세스(765.7) < 멀티 프로세스(768)

### 2. 평균 응답 시간(ms)

단일 프로세스(83) < 멀티 프로세스(90) < 멀티 스레드(151)

### 3. Failure 뜰 때까지의 요청 수

멀티 스레드(10727) < 단일 프로세스(18367) < 멀티 프로세스(19221)