컴퓨터네트워크

- 06. Web Performance Test -

제 출 일	2018.10.18
학 과	컴퓨터공학과
학 번	201402391
이 름	이 병 만

※ 과제 목표 및 해결 방법

· 과제 목표

- Python으로 성능 테스트 툴 개발
  - ▶ 라즈베리파이에 각자 구현한 HTTP Server 코드를 로드
  - ▶ 각자 PC에서 성능 측정 스크립트를 N개씩 동시 실행.
- N에 따른 페이지 로딩 시간 비교 및 시각화
  - ▶ python matplotlib 라이브러리 활용
  - ▶ 분석 결과 보고서에 정리

· 해결 방법

1. Server

```
from socket import *
import os
import sys
from threading import Thread
import traceback
```

[그림1. use_module]

➔ 구현에 필요한 모듈을 import한다.

```
def server_excute(clnt_sock):
    data = clnt_sock.recv(1500)
    parsing_data = (data.decode()).split(' ')
    file_name = parsing_data[1]
    print(file_name)

    # if method = GET
    if parsing_data[0] == 'GET':
        if os.path.isfile('.'+file_name):
            f = open('.' + file_name, 'rb')
            read_data = f.read(1024)
            file_size = str(os.path.getsize('.' + file_name)).encode() + b'\n\n'
            clnt_sock.send(is_file + file_size + read_data)

            while True:
                read_data = f.read(1024)
                if not read_data:
                    break
                clnt_sock.send(read_data)
            f.close()
            clnt_sock.close()
```

[그림2. server_excute function]

➔ 서버가 실행하는 함수이다. 소켓을 생성할 때마다 소켓을 인자로 받아서 처리해준다. 이전에는 인자를 받지 않고 실행하였더니 [OSError] Bad file descriptor라는 에러메세지가 생겼었다.
서버는 처음에 클라이언트가 요청한 정보를 받는다. 받은 요청을 파싱하여 요청이 들어온 method와 파일 이름으로 나눈다. 요청이 GET이고 파일이 존재한다면 파일을 열어 클라이언트에게 데이터를 1024씩 전송한다. 이때 처음에는 상태정보와 Content-length를 담고 있는 헤더와 데이터를 더해서 전송해준다. 두 번째부터는 1024씩 읽어서 전송을 해준다. 파일을 다 전송했으면 파일을 닫고 해당 소켓도 닫아준다. 앞으로 서버는 다음과 같은 처리를 한다.

```python
# variable
ip_addr = '127.0.0.1'
port_num = int(sys.argv[1])

is_file = b'HTTP/1.1 200 OK\n'
none_file = b'HTTP/1.1 404 Not Found\n'

server_sock = socket(AF_INET, SOCK_STREAM)
server_sock.bind((ip_addr, port_num))
print('Server socket open...')
print('Listening...')
server_sock.listen(1)

while True:
    clnt_sock, addr = server_sock.accept()
    try:
        Thread(target=server_excute, args=(clnt_sock,)).start()
    except:
        print("Thread did not start.")
        traceback.print_exc()

print("Send Message back to client")
```

[그림3. Code Progress]

➔ 처음에 코드가 진행되는 과정이다. IP주소와 port번호, 상태 정보를 선언해준다. 소켓을 생성하고 주어진 IP, Port번호로 연결한다. 반복문을 사용하여 서버는 클라이언트랑 접속을 하면 Thread로 서버가 처리하는 server_excute함수를 실행한다. 이 때의 인자값은 해당 clnt_sock을 받는다.

2. Client

```
from socket import *
import sys
from bs4 import BeautifulSoup
import concurrent.futures
import time as t
import matplotlib.pyplot as plt
```

[그림1. use_module]

➔ **구현에 필요한 모듈을 import한다.**

```
def vs(list1, list2, list3):
    box_plot_data = [list1, list2, list3]

    plt.boxplot(box_plot_data)
    plt.xlabel("Number of Clients")
    plt.ylabel("Download Time(5)")
    plt.xticks([1,2,3], [10,20,30])
    plt.show()
```

[그림2. vs function]

➔ **plot을 찍어주는 기능을 하는 함수이다. x축 y축 라벨을 지정해준다.**

```
def multiprocess_func(num, list):
    with concurrent.futures.ProcessPoolExecutor(max_workers=num) as executor:
        for i in range(0, num):
            executor.submit(single_client(list))
```

[그림3. multiprocess function]

➔ **프로세스를 N개를 생성하여 실행하도록 하는 함수이다. 인자값으로 생성 할 프로세스의 개수와 시간을 저장 할 리스트를 받는다.**

```
list1 = []
list2 = []
list3 = []

multiprocess_func(10, list1)
multiprocess_func(20, list2)
multiprocess_func(30, list3)
vs(list1, list2, list3)
```

[그림3. client progress]

➔ **Client가 실행되는 과정이다. 먼저 시간을 저장 할 리스트를 선언한다. multiprocess 함수를 호출한다. 테스트할 값이 10, 20, 30이므로 각 함수를 마치고 시간을 저장 할 리스트와 함께 인자값으로 넣어준다. 3번의 함수가 종료하면 vs함수를 통해서 plot을 띄워준다.**

```
def single_client():
    serverIP = '127.0.0.1'
    serverPort = int(sys.argv[3])

    clnt_sock = socket(AF_INET, SOCK_STREAM)
    clnt_sock.connect((serverIP, serverPort))
    print('Connect to Server....')
    clnt_msg = sys.argv[1] + ' ' + sys.argv[2]
    clnt_sock.send(clnt_msg.encode())
    print("Send Message to Server....")

    src_list = []
    recv_data = ''
```

[그림4. single_client(1)]

➔ 하나의 Client가 진행되는 과정이다. IP와 Port번호를 선언해준다. 여기서 라즈베리파이에서 진행하였때는 IP주소를 WIFI 주소로 변경해서 실행해주었다. 소켓을 생성하고 연결한다. 이미지 src를 넣은 리스트를 선언해준다.

```
    # recv request
    recv_data = clnt_sock.recv(1500).decode().split('\n\n')

    header = recv_data[0].split('\n')
    msg = header[0]
    file_size = int(header[1])

    html_data = recv_data[1].encode()
    download_status = 1024

    while download_status < file_size:
        html_data = html_data + clnt_sock.recv(1024)
        download_status = download_status + 1024

    img_list = parseHTML(html_data)
```

[그림5. single_client(2)]

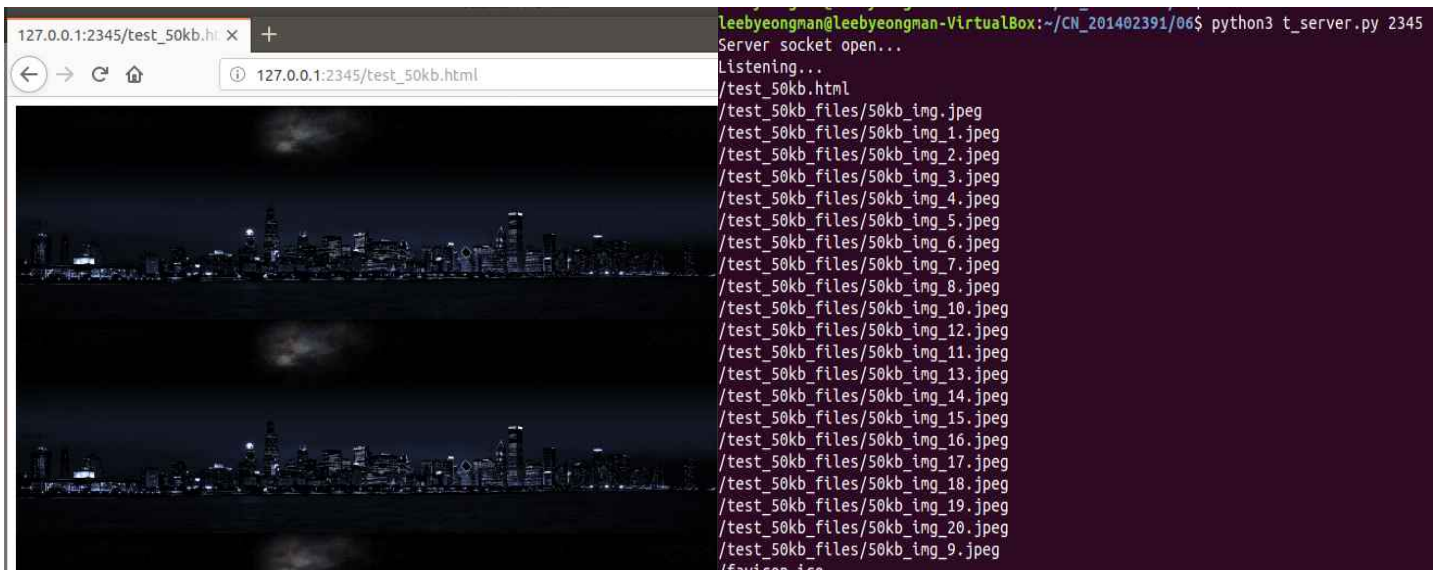
➔ 서버가 전송한 데이터를 받아서 파싱한다. 이 때 저장되는 recv_data에는 해더와 데이터가 함께 있다. 이제 헤더와 데이터를 파싱하면 헤더에서 파싱하여 상태정보와 Contetn-length를 저장한다. msg에는 상태정보가 저장되고 file_size에는 Content-length가 저장된다.
Client는 서버로부터 1024씩 데이터를 받는데 file_size만큼 받을 때까지 반복문을 실행한다. 서버가 전송한 데이터를 모두 받으면 parseHTML함수로 받은 HTML에서 이미지 태그만 반환하여 img_list에 저장한다.

```
for img in img_list:
    src = img['src'][1:]
    src_list.append(src)

for src in src_list:
    clnt_sock = socket(AF_INET, SOCK_STREAM)
    clnt_sock.connect((serverIP, serverPort))

    send_data = 'GET '+ src
    clnt_sock.send(send_data.encode())

    recv_data = clnt_sock.recv(1500).split(b'\n\n')

    header = recv_data[0].decode().split('\n')
    msg = header[0]
    file_size = int(header[1])

    download_status = 1024

    while download_status < file_size:
        clnt_sock.recv(1024)
        download_status = download_status + 1024
```

[그림6. single_client(3)]

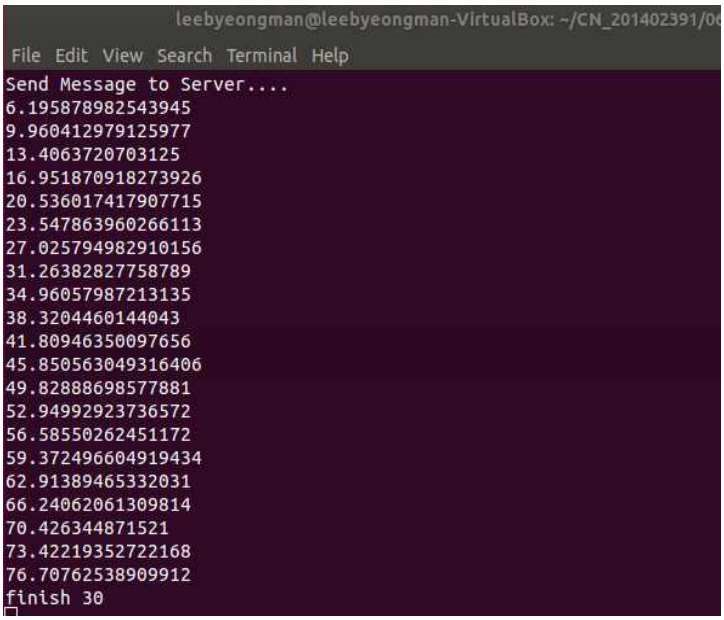
➔ 반환받은 이미지 태그 리스트에서 src만 추출하여서 src에 저장한 후 src_list에 삽입해준다.
모든 이미지 src를 담은 리스트를 이제 다시 서버한테 요청할 것이다. 반복문을 실행하여 소켓을 생성하여 연결하고 GET 방식으로 src를 서버한테 요청한다. 요청하고 서버한테서 온 정보를 바로 받아서 처리한다. 저리하는 과정은 이전에서 설명했던 방식과 동일하다.

※ 실행 결과 - (HTTP - 접속)

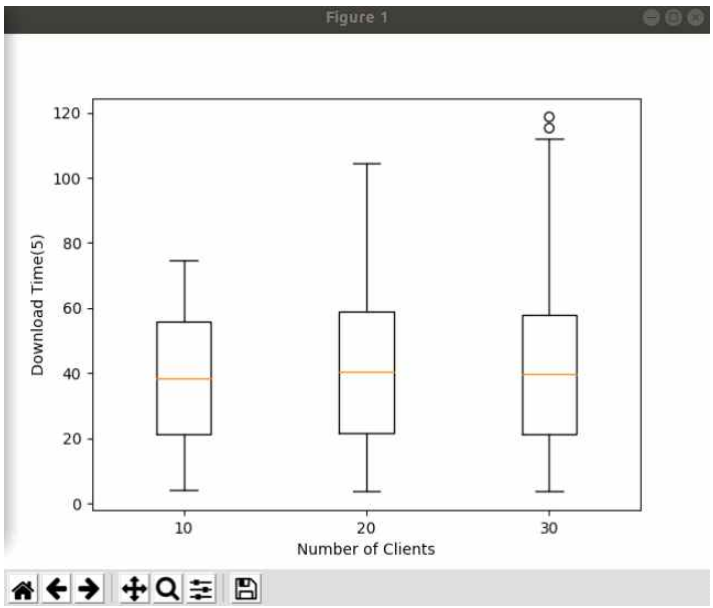


[그림1. HTTP]

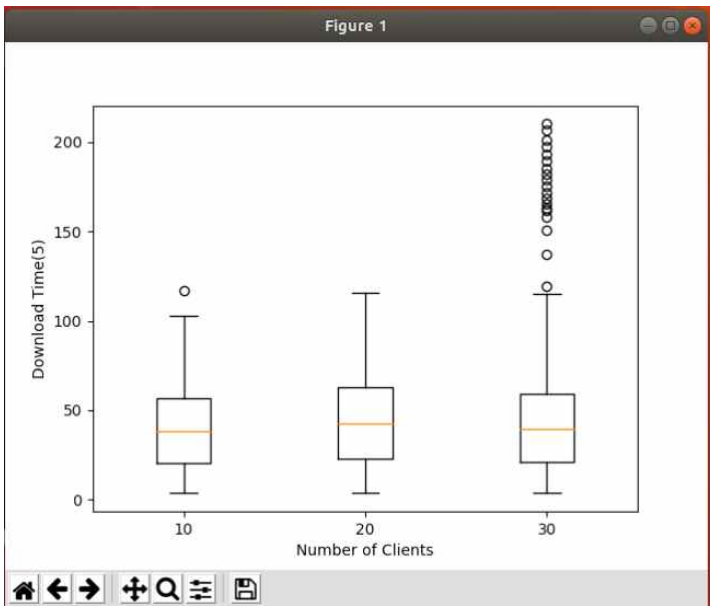
※ 실행 결과 - (Client - console)



[그림2. console]



[그림3. plot(1)]



[그림4. plot(2)]