

컴퓨터 네트워크

-4주차-

학번: 201404377

이름 : 진승언

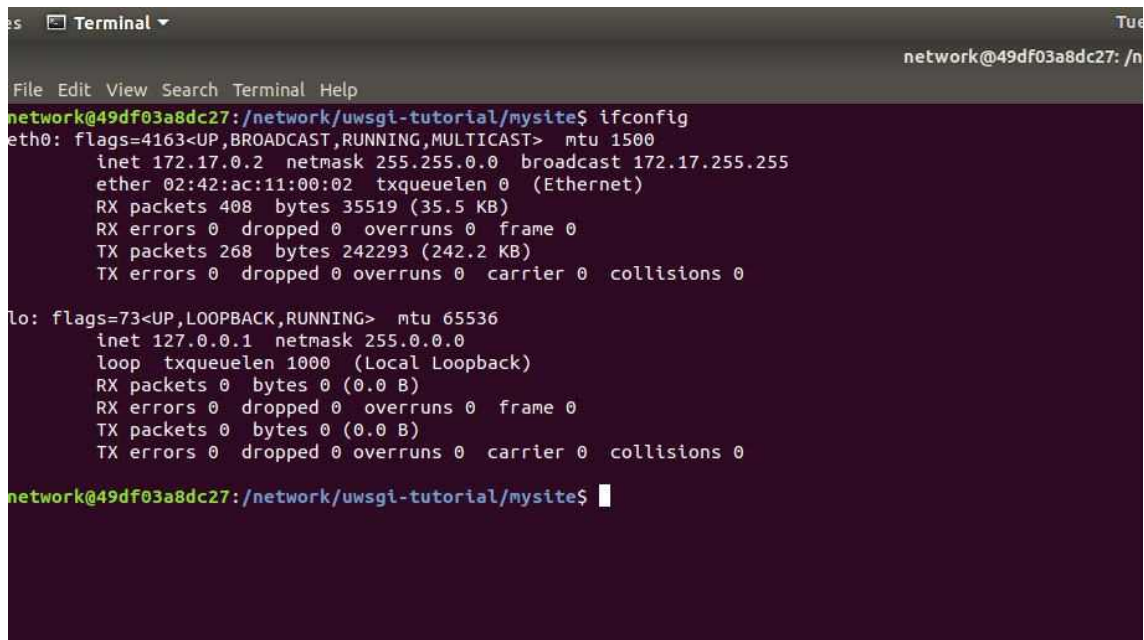
<과제목표>

이번 과제의 목표는 도커를 서버로 두고 다른 컴퓨터를 클라이언트로 두어 소켓 통신을 하는 거다.

TCP 소켓을 사용해서 HTTP 서버를 만들었다. 클라이언트와 서버 소켓 파이썬 코드를 각각 만들어 통신을 해보고, cURL, Telnet, Web browser를 통해서도 소켓 서버와 통신을 해보았다. 그리고 이 결과들을 출력도 하고 내가 만든 웹페이지를 띄워주어야 했다. 마지막으로 wireshark를 통해서 packet capture를 하면 된다. 주의할 점은 http.server 파이썬 라이브러리를 사용하는건 금지였다.

<과제 해결방법>

먼저 서버가될 docker 컨테이너 서버 소켓의 ip를 설정해주기 위해 docker 컨테이너내에서 ifconfig로 확인한 결과이다.

A terminal window titled 'Terminal' with a dark background. The prompt is 'network@49df03a8dc27: /network/uwsgi-tutorial/mysite\$'. The command 'ifconfig' has been executed, showing details for 'eth0' and 'lo'.

```
network@49df03a8dc27:/network/uwsgi-tutorial/mysite$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.2 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:ac:11:00:02 txqueuelen 0 (Ethernet)
    RX packets 408 bytes 35519 (35.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 268 bytes 242293 (242.2 KB)
    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
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

network@49df03a8dc27:/network/uwsgi-tutorial/mysite$
```

172.17.0.2임을 알 수 있다.

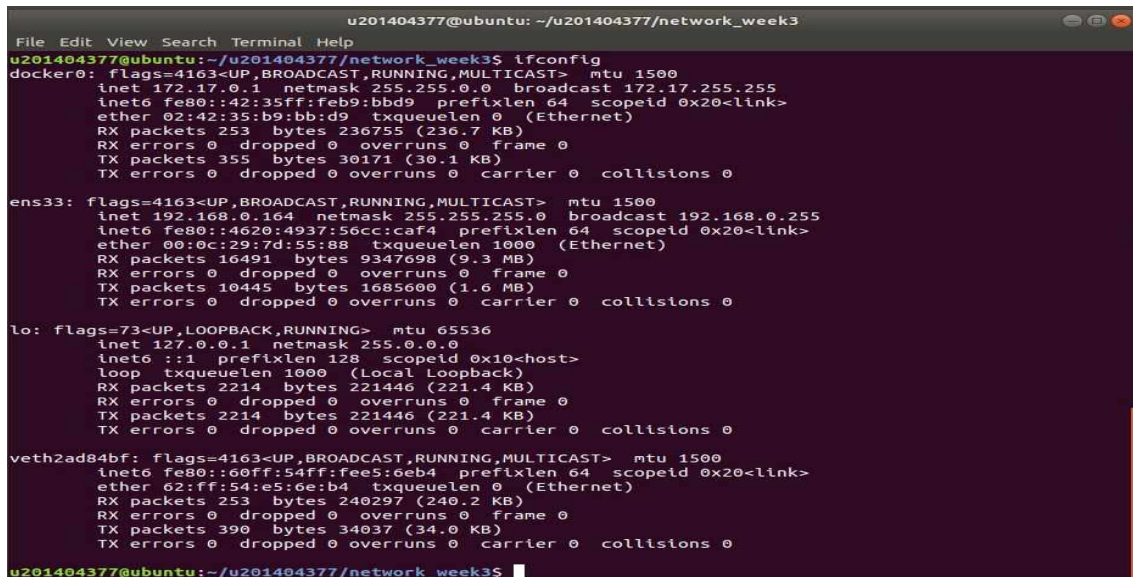
그래서 http_tcp.py(서버 소켓) IP를 다음과 같이 설정하였다.
(부분 코드)

```
if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-i', '--ipaddress', type=str, default='172.17.0.2')
    parser.add_argument('-p', '--port', type=int, default=80)

    FLAGS, _ = parser.parse_known_args()
    main(FLAGS)
```

그리고 와이파이를 통해서 윈도우 컴퓨터나 외부에서 도커 컨테이너의 소켓 서버에 접속할 것이므로 클라이언트에 ifconfig 명령어를 통해 ens33 IP를 확인했다.(Bridge) 192.168.0.164임을 알 수 있다. (Telnet과 client.py server.py 간의 소켓 통신은 다른 장소에서 했으므로 IP가 다릅니다.)



```
u201404377@ubuntu: ~/u201404377/network_week3
File Edit View Search Terminal Help
u201404377@ubuntu:~/u201404377/network_week3$ 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::42:35ff:feb9:bbd9 prefixlen 64 scopeid 0x20<link>
    ether 02:42:35:b9:bb:d9 txqueuelen 0 (Ethernet)
    RX packets 253 bytes 236755 (236.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 355 bytes 30171 (30.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.164 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::4620:4937:56cc:caf4 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:7d:55:88 txqueuelen 1000 (Ethernet)
    RX packets 16491 bytes 9347698 (9.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10445 bytes 1685600 (1.6 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
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 2214 bytes 221446 (221.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2214 bytes 221446 (221.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

veth2ad84bf: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::60ff:54ff:fee5:6eb4 prefixlen 64 scopeid 0x20<link>
    ether 62:ff:54:e5:6e:b4 txqueuelen 0 (Ethernet)
    RX packets 253 bytes 240297 (240.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 390 bytes 34037 (34.0 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

u201404377@ubuntu:~/u201404377/network_week3$
```

소켓 서버 코드인 http_tcp.py는 다음과 같이 만들었다.

```
network@49df03a8dc27: /home/network_3
File Edit View Search Terminal Help
import socket
from http.server import BaseHTTPRequestHandler

def open_server_socket(IP):
    s_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s_socket.bind((IP.ipaddress, IP.port))
    s_socket.listen()
    return s_socket

def accept_client(s_socket):
    while True:
        print("wait ... ")
        conn, addr = s_socket.accept()
        data = conn.recv(1024).decode('utf-8')
        try:
            html_dir = data.split(' ')[1]
            print('html_dir ==> ' + html_dir)
            print(html_dir[1:])
            f = open(html_dir[1:])
            outputdata = f.read()
            print("connected by ", end='')
            print(conn.getpeername())
            print("connect client... ", end='')
            print(s_socket.getsockname()[1])
            print("IP:{0}, data:{1} ".format(addr[0], data))
            print(outputdata)
            conn.send('HTTP/1.1 200 OK\r\n'.encode('utf-8'))
            conn.send('Content-Type: text/html\r\n'.encode('utf-8'))
            conn.send(outputdata.encode('utf-8'))
            conn.close()
        except Exception as ex:
            print('exception state')
            f = open('noIndex.html')
            outputdata = f.read()
            print("connected by ", end='')
            print(conn.getpeername())
            print("connect client... ", end='')
            print(s_socket.getsockname()[1])
            print("IP:{0}, data:{1} ".format(addr[0], data))
            print(outputdata)
            conn.send('HTTP/1.1 404 Not Found\r\n'.encode('utf-8'))
            conn.send('Content-Type: text/html\r\n'.encode('utf-8'))
            conn.send(outputdata.encode('utf-8'))
            conn.close()

def main(FLAGS):

    server_socket = open_server_socket(FLAGS)
    accept_client(server_socket)

if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-i', '--ipaddress', type=str, default='172.17.0.2')
    parser.add_argument('-p', '--port', type=int, default=80)

    FLAGS, _ = parser.parse_known_args()
    main(FLAGS)
```

기본 ip와 port는 앞서 확인한 도커 컨테이너 ip로 설정하였다. 그리고 포트는 도커 컨테이너를 만들 때 1234:80 으로 매칭하였으므로 80으로 설정하였다.

처음 파이썬 코드를 실행시키면 __main__이 실행이 되어 옵션도 입력해주었다면 파싱하고 아니면 기본값으로 알맞게 명령어가 실행된 후 main(Flags)으로 간다.

여기서는 먼저 서버 소켓을 열어줍니다. open_server_socket 서버를 연 후 클라이언트의 접속에 대기한다.

그리고 클라이언트의 접속 요청이 오면 http를 읽어들이어 파일 경로를 파싱해준 후 해당 경로의 파일을 open()읽어 들인다. 추가로 String형태로 outputdata에 저장한다. 저는 /index.html만 준비해놨고 다른 경로가 오면 예외처리로 404 코드와 함께 noIndex.html을 불러오게 했다.

조교님의 실행영상에서 출력문이 나왔던 것도 똑같이 출력되게 해주었다. conn.getpeername()으로 연결된 상대(클라이언트)의 address(주소)를 얻었다. getsockname() 으로는 포트 번호를 출력시켜주었다.

클라이언트에게 응답을 보내주는 것은 send() 메소드를 통해서 보내주었다. 내용은 HTTP 형식에 맞게 보내주고 요청한 데이터를 보내주었다. 응답을 보내줄 때는 utf-8로 인코딩해주어 보내줘야한다. 응답 데이터를 클라이언트에게 모두 보내준 후 연결은 close로 끝낸다.

- [HTTP Method] [파일경로] [프로토콜]
- ex) GET / HTTP/1.1

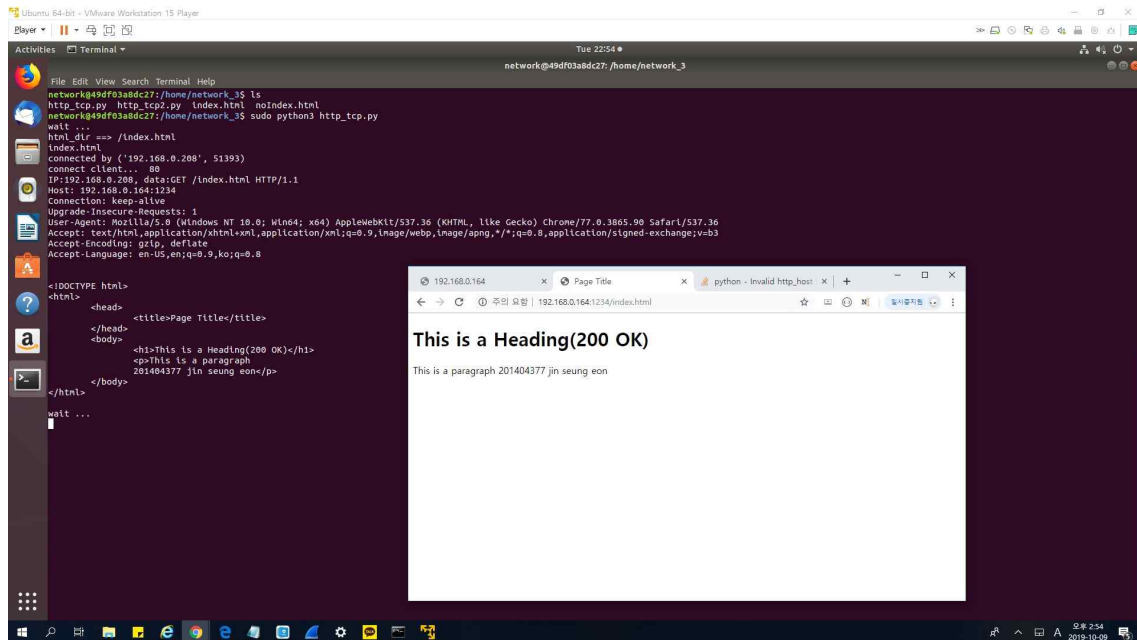
■ Response (Nginx, Django, uWSGI, Python 3 TCP Server)

- [프로토콜] [Status code] [Status Message]
- ex) HTTP/1.1 200 OK

요청과 응답은 위와 같이 보내줘야한다.

가장 먼저 Web browser와 통신한 결과이다.

(클라이언트는 노트북 윈도우, 서버는 노트북 vmware docker container이다.)



웹페이지가 잘 서빙됨을 알 수 있다. 또한 서버쪽에 클라이언트 메시지를 print문으로 출력해주었는데 192.168.0.208 IP이고 51393 포트를 통해 옴을 알 수 있다.

- [HTTP Method] [파일경로] [프로토콜]
- ex) GET / HTTP/1.1

또한 GET /index.html HTTP/1.1 즉 HTTP GET 요청을 하고 /index.html 파일 경로를 요청했음을 알 수 있다.


```
3browser.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

[+] Ip.addr == 192.168.0.164 && tcp.port == 1234

No. Time Source Destination Protocol Length Info
11 0.000497 192.168.0.208 192.168.0.164 TCP 54 51393 → 1234 [ACK] Seq=1 Ack=1 Win=1051136 Len=0
12 0.000500 192.168.0.208 192.168.0.164 TCP 54 [TCP Out-Of-Order] 1234 → 51393 [ACK] Seq=0 Ack=1 Min=1051136 Len=0
13 0.000593 192.168.0.164 192.168.0.208 TCP 66 1234 → 51394 [SYN, ACK] Seq=0 Ack=1 Min=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
14 0.000595 192.168.0.164 192.168.0.208 TCP 66 [TCP Out-Of-Order] 1234 → 51394 [SYN, ACK] Seq=0 Ack=1 Min=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
15 0.000635 192.168.0.208 192.168.0.164 TCP 54 51394 → 1234 [ACK] Seq=1 Ack=1 Win=1051136 Len=0
16 0.000617 192.168.0.208 192.168.0.164 TCP 54 [TCP Out-Of-Order] 51393 → 1234 [ACK] Seq=1 Ack=1 Min=1051136 Len=0
17 0.001003 192.168.0.208 192.168.0.164 HTTP 492 GET /index.html HTTP/1.1
18 0.001007 192.168.0.208 192.168.0.164 TCP 492 [TCP Retransmission] 51393 → 1234 [PSH, ACK] Seq=1 Ack=1 Min=1051136 Len=438
19 0.001122 192.168.0.164 192.168.0.208 TCP 66 1234 → 51393 [ACK] Seq=1 Ack=439 Win=30336 Len=0
20 0.001125 192.168.0.164 192.168.0.208 TCP 66 [TCP Out-Of-Order] 1234 → 51393 [ACK] Seq=1 Ack=439 Win=30336 Len=0
21 0.002000 192.168.0.164 192.168.0.208 TCP 71 1234 → 51393 [PSH, ACK] Seq=1 Ack=439 Win=30336 Len=17 [TCP segment of a reassembled PDU]
22 0.002003 192.168.0.164 192.168.0.208 TCP 71 [TCP Retransmission] 1234 → 51393 [PSH, ACK] Seq=1 Ack=439 Win=30336 Len=17
23 0.002150 192.168.0.164 192.168.0.208 HTTP 334 HTTP/1.1 200 OK (text/html)
24 0.002153 192.168.0.164 192.168.0.208 TCP 334 [TCP Out-Of-Order] 1234 → 51393 [FIN, PSH, ACK] Seq=18 Ack=439 Win=1050880 Len=280/Reassembly error, protocol TCP: New fragment overlaps old data (retransmit)
25 0.002207 192.168.0.208 192.168.0.164 TCP 54 51393 → 1234 [ACK] Seq=439 Ack=299 Win=1050880 Len=0
26 0.002210 192.168.0.208 192.168.0.164 TCP 54 [TCP Dup ACK 25#1] 51393 → 1234 [ACK] Seq=439 Ack=299 Win=1050880 Len=0
27 0.002859 192.168.0.208 192.168.0.164 TCP 54 51393 → 1234 [FIN, ACK] Seq=439 Ack=299 Win=1050880 Len=0
28 0.002863 192.168.0.208 192.168.0.164 TCP 54 [TCP Out-Of-Order] 51393 → 1234 [FIN, ACK] Seq=439 Ack=299 Win=1050880 Len=0
30 0.003004 192.168.0.164 192.168.0.208 TCP 60 1234 → 51393 [ACK] Seq=300 Ack=440 Win=30336 Len=0

> Frame 17: 492 bytes on wire (3936 bits), 492 bytes captured (3936 bits) on interface 0
> Ethernet II, Src: IntelCor_6f:59:91 (50:76:af:6f:59:91), Dst: Vmware_7d:55:88 (00:0c:29:7d:55:88)
> Internet Protocol Version 4, Src: 192.168.0.208, Dst: 192.168.0.164
> Transmission Control Protocol, Src Port: 51393, Dst Port: 1234, Seq: 1, Ack: 1, Len: 438
> Hypertext Transfer Protocol
  > GET /index.html HTTP/1.1\r\n
    > [Expert Info (Chat/Sequence): GET /index.html HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /index.html
      Request Version: HTTP/1.1
      Host: 192.168.0.164:1234\r\n
      Connection: keep-alive\r\n
      Upgrade-Insecure-Requests: 1\r\n
      User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36\r\n
      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3\r\n
      Accept-Encoding: gzip, deflate\r\n
      Accept-Language: en-US,en;q=0.9,kor;q=0.8\r\n
      \r\n
      [Full request URI: http://192.168.0.164:1234/index.html]
      [HTTP request 1/1]
      [Response in frame: 23]

Packets: 588 Displayed: 30 (5.0%) Profile: Default

23 0.002150 192.168.0.164 192.168.0.208 HTTP 334 HTTP/1.1 200 OK (text/html)
24 0.002153 192.168.0.164 192.168.0.208 TCP 334 [TCP Out-Of-Order] 1234 → 51393 [FIN, PSH, ACK] Seq=18
25 0.002207 192.168.0.208 192.168.0.164 TCP 54 51393 → 1234 [ACK] Seq=439 Ack=299 Win=1050880 Len=0
26 0.002210 192.168.0.208 192.168.0.164 TCP 54 [TCP Dup ACK 25#1] 51393 → 1234 [ACK] Seq=439 Ack=299
27 0.002859 192.168.0.208 192.168.0.164 TCP 54 51393 → 1234 [FIN, ACK] Seq=439 Ack=299 Win=1050880 Le
28 0.002863 192.168.0.208 192.168.0.164 TCP 54 [TCP Out-Of-Order] 51393 → 1234 [FIN, ACK] Seq=439 Ack
30 0.003004 192.168.0.164 192.168.0.208 TCP 60 1234 → 51393 [ACK] Seq=300 Ack=440 Win=30336 Len=0

> Frame 23: 334 bytes on wire (2672 bits), 334 bytes captured (2672 bits) on interface 0
> Ethernet II, Src: IntelCor_6f:59:91 (50:76:af:6f:59:91), Dst: IntelCor_6f:59:91 (50:76:af:6f:59:91)
> Internet Protocol Version 4, Src: 192.168.0.164, Dst: 192.168.0.208
> Transmission Control Protocol, Src Port: 1234, Dst Port: 51393, Seq: 18, Ack: 439, Len: 280
> [2 Reassembled TCP Segments (297 bytes): #21(17), #23(280)]
> Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    > [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
      Content-Type: text/html\r\n
      \r\n
      [HTTP response 1/1]
      [Time since request: 0.001147000 seconds]
      [Request in frame: 17]
      [Request URI: http://192.168.0.164:1234/index.html]
      File Data: 255 bytes
  > Line-based text data: text/html (11 lines)
    <!DOCTYPE html>\n
    <html>\n
    <head>\n
    <title>Page Title</title>\n
```

Wireshark의 결과이다. 요청과 응답이 잘 왔음을 볼 수 있다. HTTP 부분을 보면 클라이언트에서 서버로 요청시 GET /index.html HTTP/1.1로 통신했음을 볼 수 있다. 또한 서버에서 클라이언트로 요청할 때는 요청에 에러사항이 없으므로 200 OK 응답코드와 index.html의 데이터를 보내줌을 볼 수 있다.

다음은 404 에러페이지이다. /index.html로 요청을 보내지 않은 경우 모두 noIndex.html과 404 Not Found 응답을 보내게 해놓았다.

The screenshot displays a Kali Linux virtual machine environment. In the background, a terminal window shows the execution of a Python script that sets up a web server. The script defines a directory for HTML files and a function to handle HTTP requests. It then binds to IP 192.168.0.208 on port 80 and starts the server.

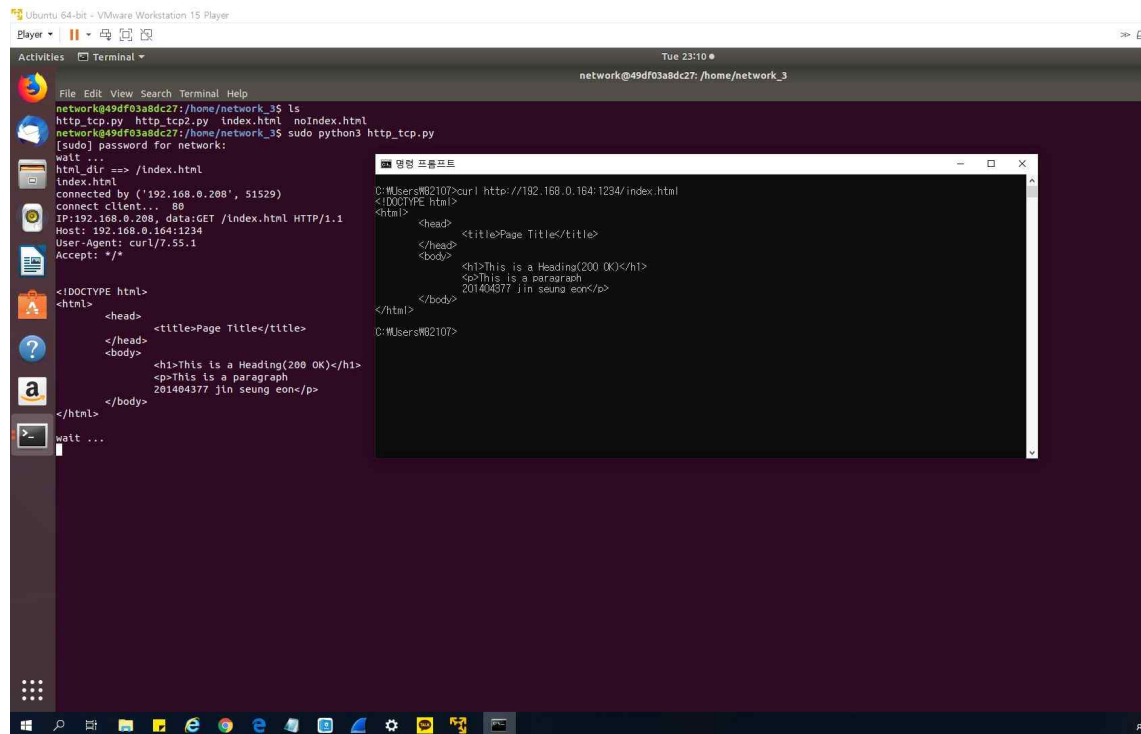
In the foreground, a web browser window shows a "404- The Page cannot be found" error page. The page content includes the title "404- The Page cannot be found" and a message "Please go index.html".

Below the browser, Wireshark is open, showing a packet capture of the HTTP request and response. The selected packet is a "404 Not Found" response from 192.168.0.208 to 192.168.0.164. The packet details show the HTTP response structure, including the status line "HTTP/1.1 404 Not Found", the content type "text/html", and the response body containing the error message.

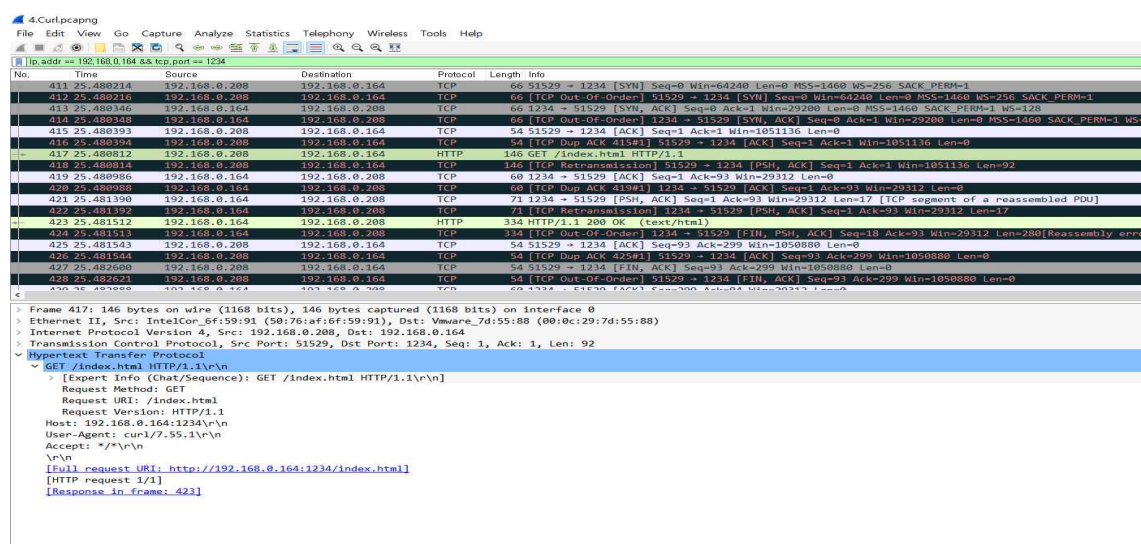
패킷 캡처 결과를 보면 내가 정하지 않는 경로 / 로 요청을 보내어 404 Not Found를 응답으로 보냈음을 볼 수 있다. 원

래의 웹페이지라면 /로 요청했을 시 기본페이지(ex index.html)을 서빙해주는게 맞지만 본 과제는 소켓 통신이 목적이므로 이렇게 구현을 했다. 추가로 소켓 서버를 80으로 열어 뒀으므로 그에 매칭된 1234 TCP 포트로 접속됨을 볼 수 있었다. 그리고 cURL, telnet도 결과는 거의 똑같이 나올거라고 예측할 수 있다.

cURL 결과입니다.



마찬가지로 노트북 윈도우 환경을 클라이언트로 하고 vmware 도커 컨테이너를 서버로 하였습니다. 요청과 응답이 잘 옳을 볼 수 있습니다.



wireshark 결과입니다. 앞서 웹브라우저로 접속 한거와 마찬가지로 요청과 응답이 잘 옳을 볼 수 있다.

다음은 telnet 요청의 결과이다.

윈도우 클라이언트에서 telnet 명령어 후 input을 치는게 안먹어서 친구의 노트북으로 클라이언트로 사용 했다.

(서버)

```
network@49df03a8dc27: /home/network_3$ sudo python3 http_tcp.py
wait ...
html_dir ==> /index.html
index.html
connected by ('192.168.43.158', 50333)
connect client... 80
IP:192.168.43.158, data:GET /index.html HTTP/1.1

<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>This is a Heading(200 OK)</h1>
    <p>This is a paragraph
    201404377 jin seung eon</p>
  </body>
</html>
wait ...
```

(클라이언트)

```
((base) jeongseong-ug-ui-MacBook-Pro:~ jeongseong-ug$ telnet 192.168.43.129 1234
Trying 192.168.43.129...
Connected to 192.168.43.129.
Escape character is '^]'.
GET /index.html HTTP/1.1
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>This is a Heading(200 OK)</h1>
    <p>This is a paragraph
    201404377 jin seung eon</p>
  </body>
</html>
Connection closed by foreign host.
(base) jeongseong-ug-ui-MacBook-Pro:~ jeongseong-ug$
```

(wireshark)

S.Telnet.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter: <<Ctrl-Z>>

No.	Time	Source	Destination	Protocol	Length	Info
130	30.621097	192.168.43.158	192.168.43.129	TCP	66	50330 → 1234 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSva
131	32.158967	52.11.109.54	192.168.43.129	TLSv1.2	97	Application Data
132	32.159128	192.168.43.129	52.11.109.54	TCP	66	55574 → 443 [ACK] Seq=1 Ack=32 Win=326 Len=0 TSval=1
133	32.159132	192.168.43.129	52.11.109.54	TCP	66	[TCP Dup ACK #12#1] 55574 → 443 [ACK] Seq=1 Ack=32 W
134	32.159316	192.168.43.129	52.11.109.54	TLSv1.2	101	Application Data
135	32.159318	192.168.43.129	52.11.109.54	TCP	101	[TCP Retransmission] 55574 → 443 [PSH, ACK] Seq=1 Ac
136	32.230635	192.168.43.158	192.168.43.129	TCP	92	50330 → 1234 [PSH, ACK] Seq=1 Ack=1 Win=131712 Len=2
137	32.230908	192.168.43.129	192.168.43.158	TCP	66	1234 → 50330 [ACK] Seq=1 Ack=27 Win=29056 Len=0 TSva
138	32.230912	192.168.43.129	192.168.43.158	TCP	66	[TCP Dup ACK #17#1] 1234 → 50330 [ACK] Seq=1 Ack=27
139	32.231792	192.168.43.129	192.168.43.158	TCP	83	1234 → 50330 [PSH, ACK] Seq=1 Ack=27 Win=29056 Len=1
140	32.231797	192.168.43.129	192.168.43.158	TCP	83	[TCP Retransmission] 1234 → 50330 [PSH, ACK] Seq=1 A
141	32.231917	192.168.43.129	192.168.43.158	HTTP	346	HTTP/1.1 200 OK (text/html)
142	32.231920	192.168.43.129	192.168.43.158	TCP	346	[TCP Out-Of-Order] 1234 → 50330 [FIN, PSH, ACK] Seq=
143	32.234959	192.168.43.158	192.168.43.129	TCP	66	50330 → 1234 [ACK] Seq=27 Ack=18 Win=131712 Len=0 TS
144	32.237651	192.168.43.158	192.168.43.129	TCP	66	50330 → 1234 [ACK] Seq=27 Ack=299 Win=131456 Len=0 T
145	32.237651	192.168.43.158	192.168.43.129	TCP	66	50330 → 1234 [FIN, ACK] Seq=27 Ack=299 Win=131456 Le
146	32.237889	192.168.43.129	192.168.43.158	TCP	66	1234 → 50330 [ACK] Seq=299 Ack=28 Win=29056 Len=0 TS
147	32.237893	192.168.43.129	192.168.43.158	TCP	66	[TCP Dup ACK #146#1] 1234 → 50330 [ACK] Seq=299 Ack=2
148	32.452220	52.11.109.54	192.168.43.158	TCP	66	443 → 55574 [ACK] Seq=33 Ack=326 Win=1024 Len=0 TSval=

< Frame 141: 346 bytes on wire (2768 bits), 346 bytes captured (2768 bits) on interface 0

> Ethernet II, Src: IntelCor_6f:59:91 (50:76:af:6f:59:91), Dst: Apple_79:14:d6 (f0:18:98:79:14:d6)

> Internet Protocol Version 4, Src: 192.168.43.129, Dst: 192.168.43.158

> Transmission Control Protocol, Src Port: 1234, Dst Port: 50330, Seq: 18, Ack: 27, Len: 280

> [2 Reassembled TCP Segments (297 bytes): #139(17), #141(280)]

> Hypertext Transfer Protocol

> HTTP/1.1 200 OK\r\n

Content-Type: text/html\r\n

\r\n

[HTTP response 1/1]

File Data: 255 bytes

> Line-based text data: text/html (11 lines)

<!DOCTYPE html>\r\n

<html>\r\n

<head>\r\n

<title>Page Title</title>\r\n

</head>\r\n

<body>\r\n

<h1>This is a Heading(200 OK)</h1>\r\n

<p>This is a paragraph\r\n

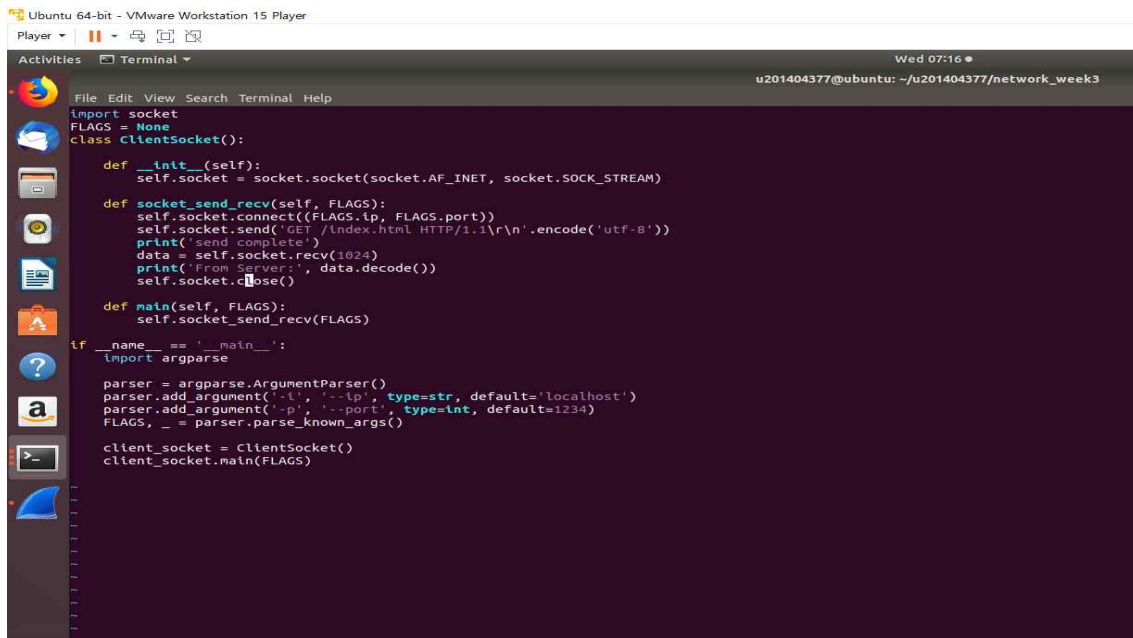
\t\t201404377 jin seung eon</p>\r\n

</body>\r\n

</html>\r\n

위에서 했던거와 동일하다.

클라이언트 소켓과 서버 소켓의 결과이다. 클라이언트 소켓코드는 다음과 같다.



```
import socket
FLAGS = None
class ClientSocket():
    def __init__(self):
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def socket_send_recv(self, FLAGS):
        self.socket.connect((FLAGS.ip, FLAGS.port))
        self.socket.send('GET /index.html HTTP/1.1\r\n'.encode('utf-8'))
        print('send complete')
        data = self.socket.recv(1024)
        print('From Server:', data.decode())
        self.socket.close()

    def main(self, FLAGS):
        self.socket_send_recv(FLAGS)

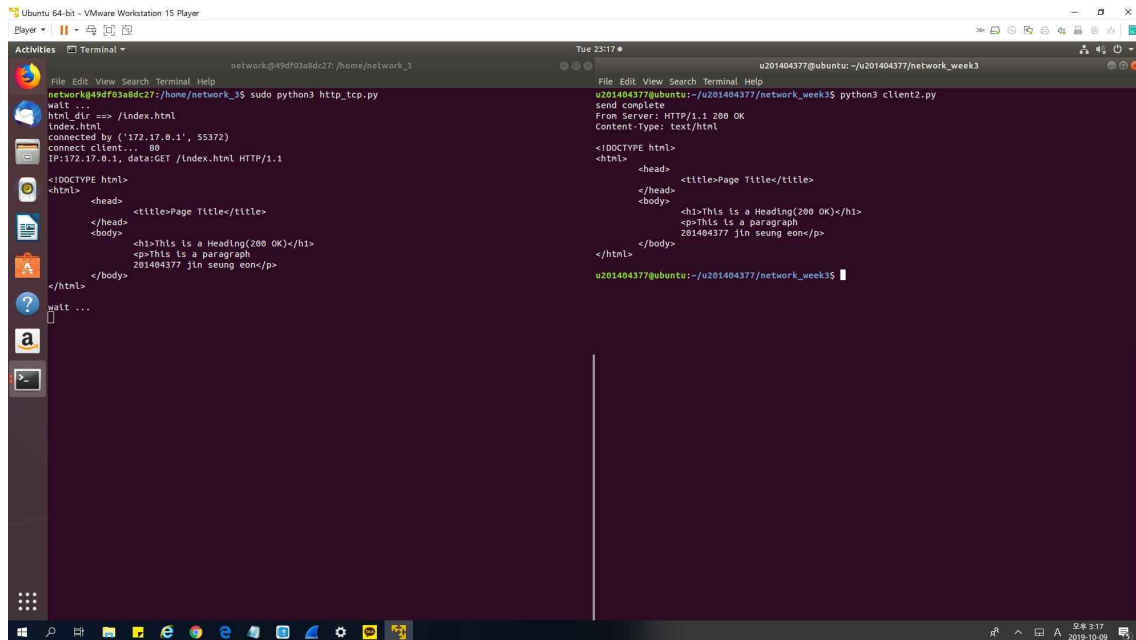
if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('-t', '--ip', type=str, default='localhost')
    parser.add_argument('-p', '--port', type=int, default=1234)
    FLAGS, _ = parser.parse_known_args()
    client_socket = ClientSocket()
    client_socket.main(FLAGS)
```

클라이언트 소켓은 지정된 ip와 TCP 포트로 연결을 시도한다.

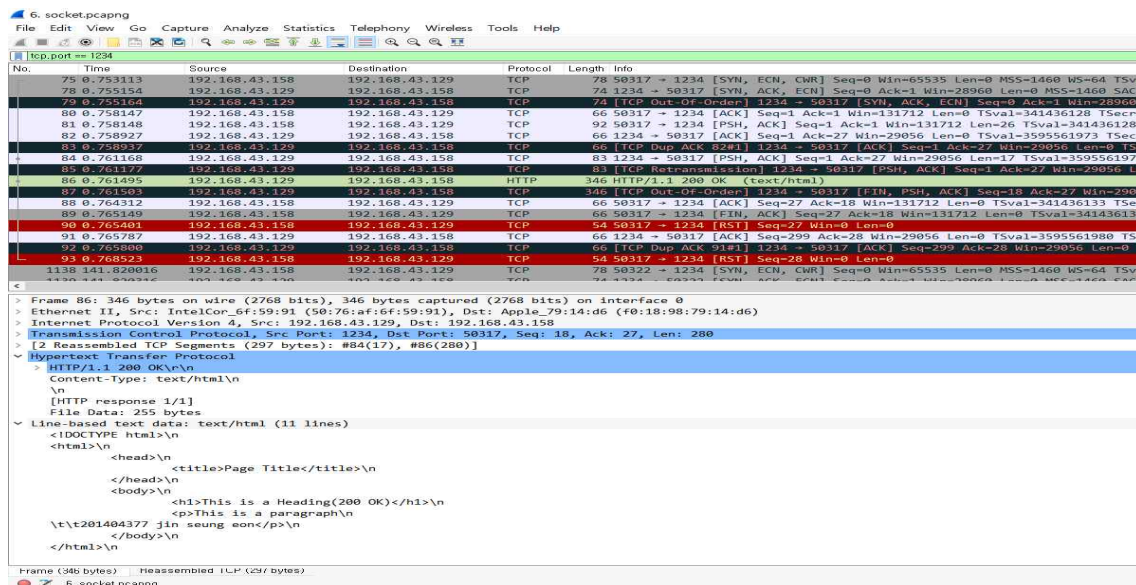
- [HTTP Method] [파일경로] [프로토콜]

그 후 위와 같이 요청을 보내준다. 난 GET 요청과 /index.html을 파일경로로 하여 보냈다. 그 후 응답이 오면 응답받은 데이터를 print문으로 출력해주고 연결을 끊게 했다.

(서버, 클라이언트)



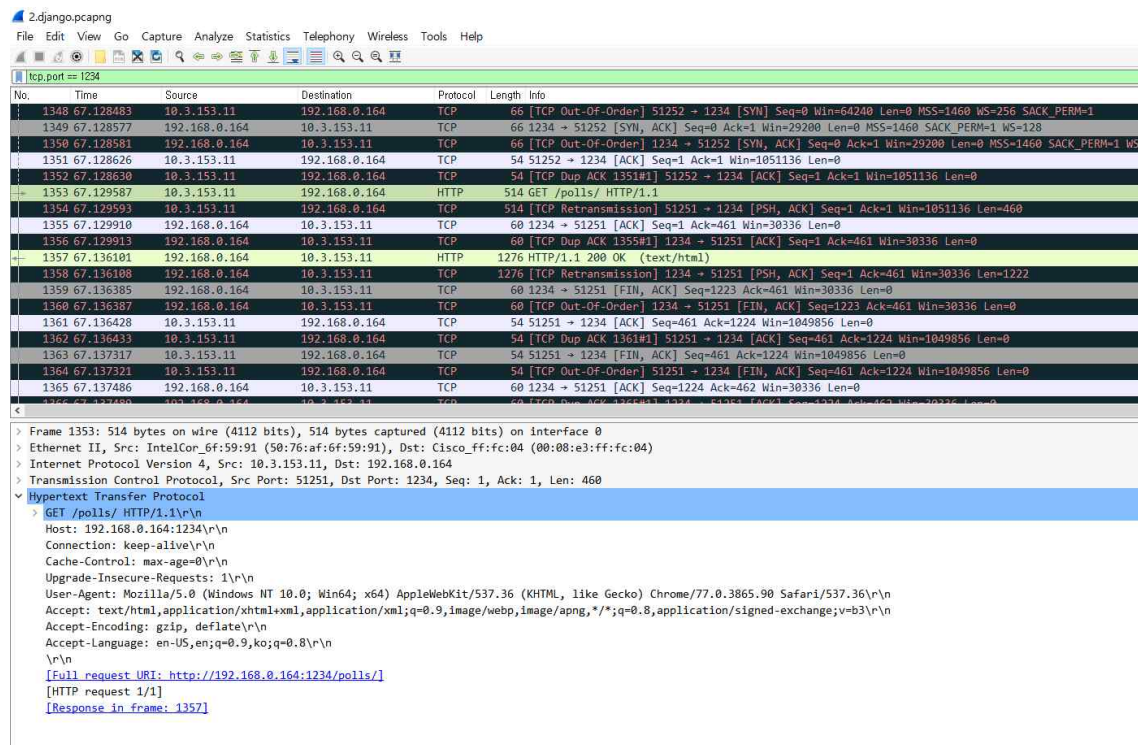
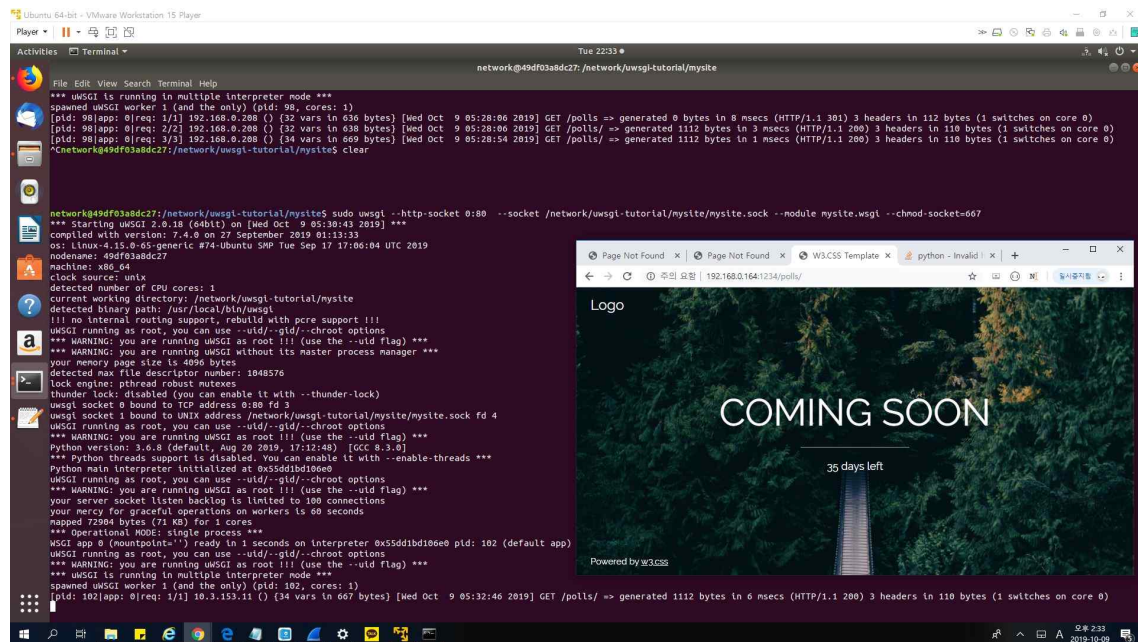
(wireshark)



결과는 마찬가지로 위에서 한 것들과 동일하다.

마지막으로 도커 내의 django 웹 어플리케이션과의 패킷캡처 결과이다.

(서버는 도커, 클라이언트는 윈도우 노트북)



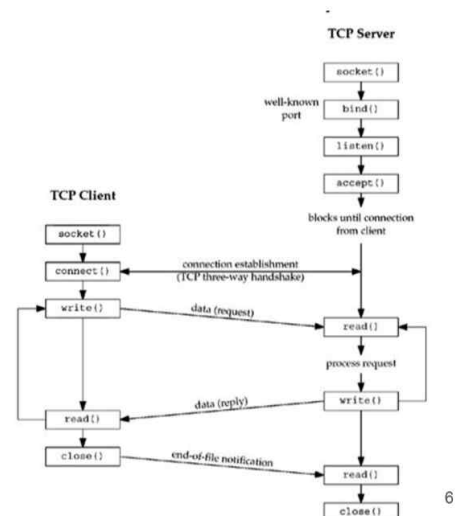
마찬가지로 요청과 응답이 잘 오고 HTTP도 /polls(내 장고 프

로젝트 경로) 로 GET요청이 가고 브라우저에 장고 어플리케이션이 잘 실행되고 HTTP 200 OK 응답을 받았음도 확인 할 수 있었다.

총정리하면, 모두 하는 방식만 다를 뿐

Berkeley Socket

- `bind()`: Process - Address, Port number 연결
- `listen()`: Listening State 진입
- `accept()`: TCP Connection 대기
- `send()/recv()`: 데이터 송수신
- `connect()`: TCP Connection 생성



이와 같은 구조로 http 소켓 통신이 이루어짐을 알 수 있었고 구현할 수 있었다. TCP server가 소켓을 열어놓고 클라이언트의 요청을 받을 준비를 하고 클라이언트가 소켓 서버에 연결을 하고 요청을 하면 소켓 서버는 요청을 받고 요청에 맞게 응답을 해주고 연결이 서로 끊어지게 된다.

<과제후기>

소켓 통신의 구조와 구현을 공부할 수 있어 좋았습니다.