컴퓨터 네트워크
중간고사 대체 보고서

TCP 기반 소켓프로그래밍 / HTTP 통신

2019198 이세희

1. 동작환경

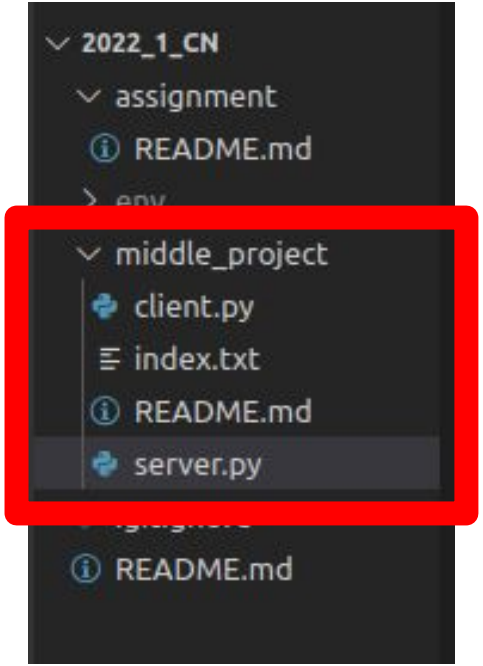


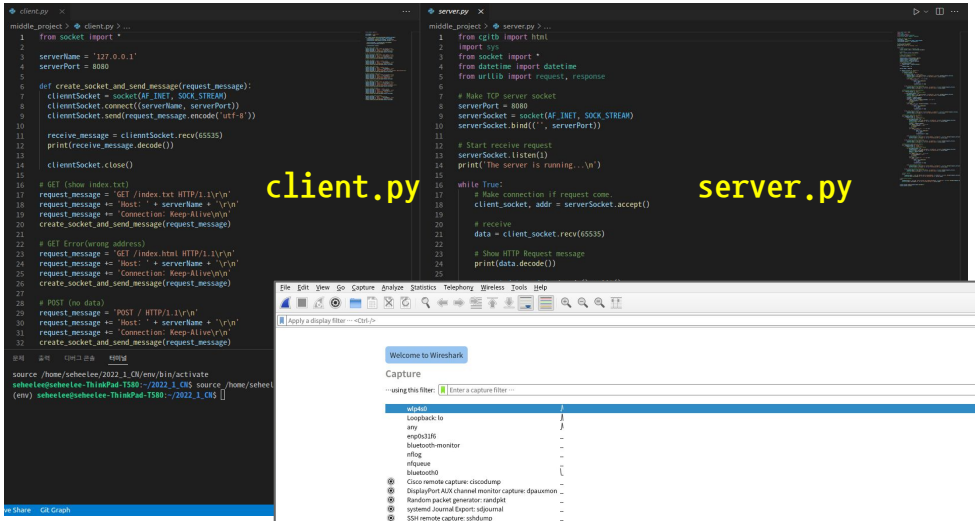
그림 1. 과제 파일 구성

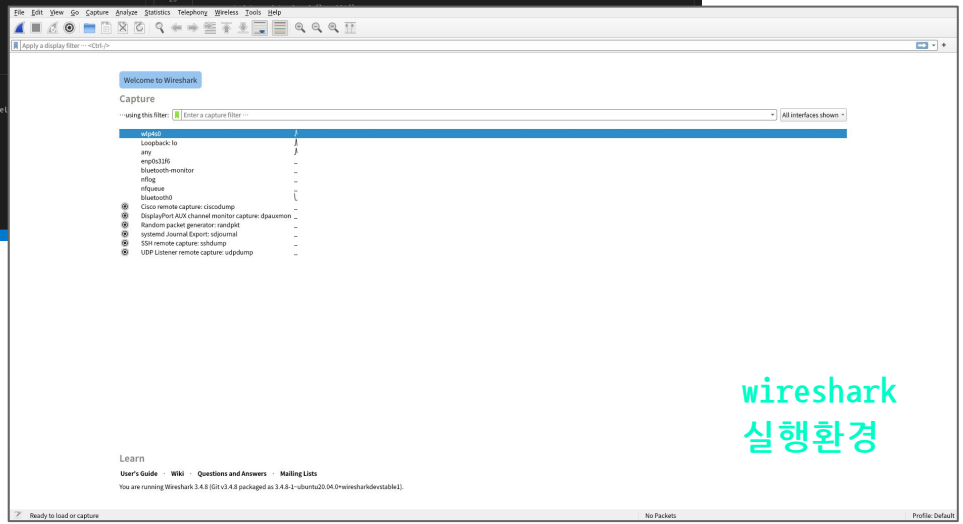
운영체제 Ubuntu LTS 20.04 환경에서 Local TCP socket 프로그래밍을 위한 server.py와 client.py를 작성하였습니다.

ip: 127.0.0.1(localhost)
port: 8080

위 정보를 TCP socket 프로그래밍 대상 주소로 설정한 후 과제를 진행하였습니다.

1. 동작환경

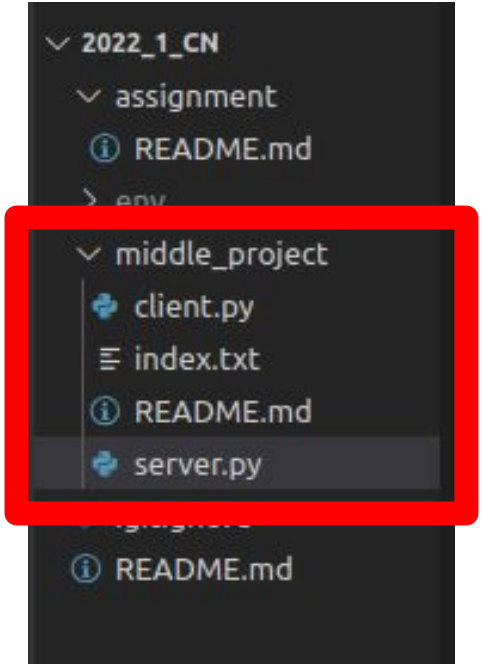




Visual Studio Code IDE를 사용하여 로컬 환경에서 진행되었고

패킷 분석은 Wireshark를 이용했습니다.

2. 소스파일 설명 - 디렉토리 구성



과제 주요 파일 구성

1. 서버 구현을 위한 **server.py**
2. 파일 생성 및 업데이트 실행을 위한 **index.txt**
3. 클라이언트 서버 구현을 위한 **client.py**

위처럼 크게 세 개의 파일로 구성되어있습니다.

2. 소스파일 설명 - client socket

```
from socket import *

serverName = '127.0.0.1'
serverPort = 8080

def create_socket_and_send_message(request_message):
 clienntSocket = socket(AF_INET, SOCK_STREAM)
 clienntSocket.connect((serverName, serverPort))
 clienntSocket.send(request_message.encode('utf-8'))

 receive_message = clienntSocket.recv(65535)
 print(receive_message.decode())

 clienntSocket.close()
```

Client Socket 생성

Socket 라이브러리를 이용하여 클라이언트 소켓을 구현했습니다.

GET, POST, PUT, PATCH 네 종류 method 요청이 정상 응답(state 200)을 하도록 구현하였습니다.

설정한 Client request는 총 8가지 입니다.

2. 소스파일 설명 - server

```
from cgitb import html
import sys
from socket import *
from datetime import datetime
from urllib import request, response

# Make TCP server socket
serverPort = 8080
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))

# Start receive request
serverSocket.listen(1)
print('The server is running...\n')
```

1

Server 생성 1

Socket, Urlib 라이브러리를 이용하여 socket 통신 서버를 구현하였습니다.

로컬환경에서 진행했기 때문에 ip주소는 127.0.0.1이고 port 번호는 8080으로 설정하였습니다.

2. 소스파일 설명 - server

```
while True:                                                    2
    # Make connection if request come.
    client_socket, addr = serverSocket.accept()

    # receive
    data = client_socket.recv(65535)

    # Show HTTP Request message
    print(data.decode())

    request_data = data.decode().split()
                    ...(중간 생략)...
    client_socket.send(response_data.encode())
    client_socket.close()
```

Server 생성 2

지속적으로 Client socket을 인식하기 위해 while loop를 사용하였습니다.

Client 요청을 받으면 데이터를 해석하여 그에 맞는 응답을 보내도록 구성하였습니다.

생략된 부분은 뒤에 method에 따라 정리하였습니다.

1) HTTP Method: GET

GET?

요청받은 URI의 정보를 검색하여 응답한다.

1) HTTP (GET) - client.py

```
# GET (show index.txt)
request_message = 'GET /index.txt HTTP/1.1 \r\n'
request_message += 'Host: ' + serverName + '\r\n'
request_message += 'Connection: Keep-Alive \n\n'
create_socket_and_send_message (request_message)

# GET Error(wrong address)
request_message = 'GET /index.html HTTP/1.1 \r\n'
request_message += 'Host: ' + serverName + '\r\n'
request_message += 'Connection: Keep-Alive \n\n'
create_socket_and_send_message (request_message)
```

GET Request format

Request 1) index.txt를 url 주소로 담아 파일 내용을 보여주도록 서버에 요청(state 200)

Request 2) 잘못된 url 주소로 담아 서버에 요청 (state 404)

1) HTTP (GET) - server.py

```
if request_method == "GET":
            # Valid address
            if request_item == "/index.txt":
                response_data = "{0} 200 OK\nDate: {1}\nServer: {2}\n\n".format(request_version,
                datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
                # read file
                with open ('index.txt', 'r') as txt:
                        while True:
                            body = txt.readline()
                            if not body:
                                break
                            response_data += body

            else: # Invalid address(Send Data)
                response_data = "{0} 404 Error\nDate: {1}\nServer: {2}\n\n".format(request_version,
                datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
```

State 200:
정상 응답.
기존의 파일을 읽어
Response Data에
포함시킨다.

State 404:
파일 이름이 올바르지
않은 경우

1) HTTP (GET) - 실행 결과

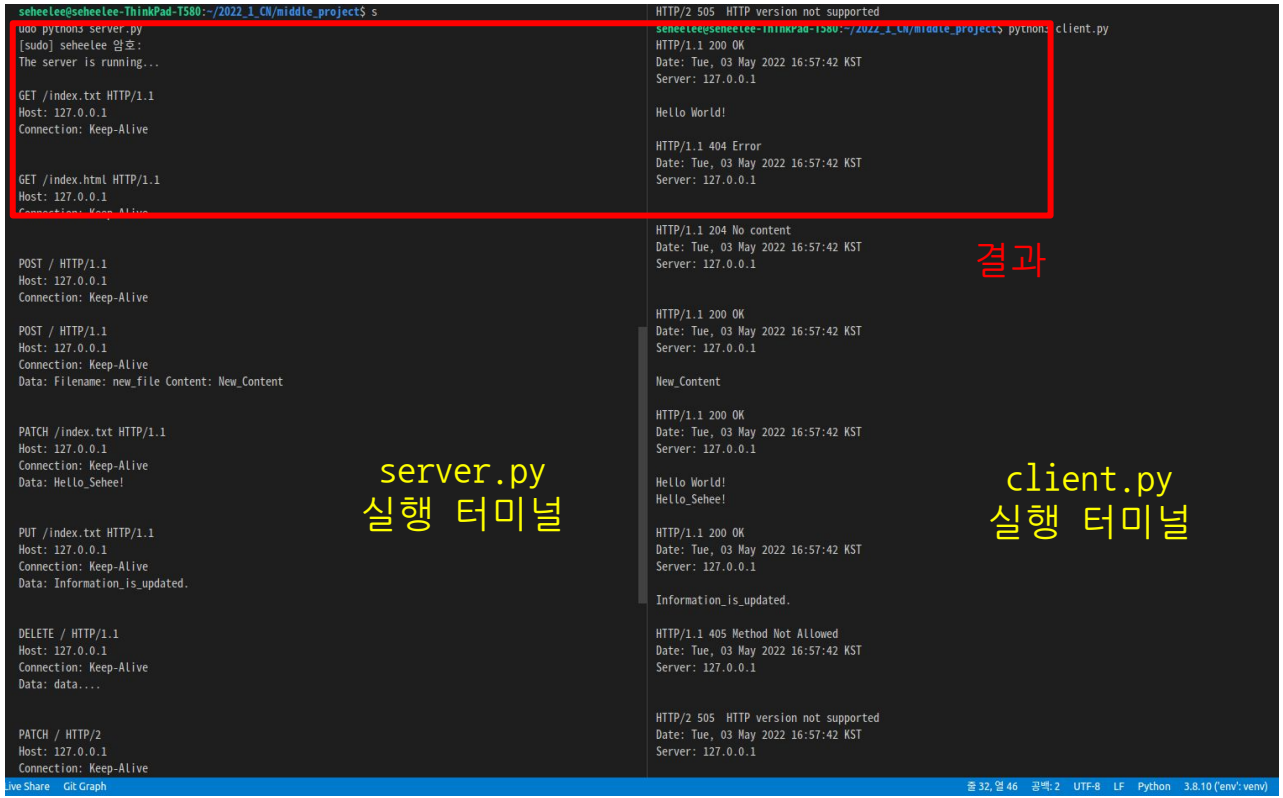
server.py
실행 터미널

```
The server is running...

GET /index.txt HTTP/1.1
Host: 127.0.0.1
Connection: Keep-Alive


GET /index.html HTTP/1.1
Host: 127.0.0.1
Connection: Keep-Alive
```

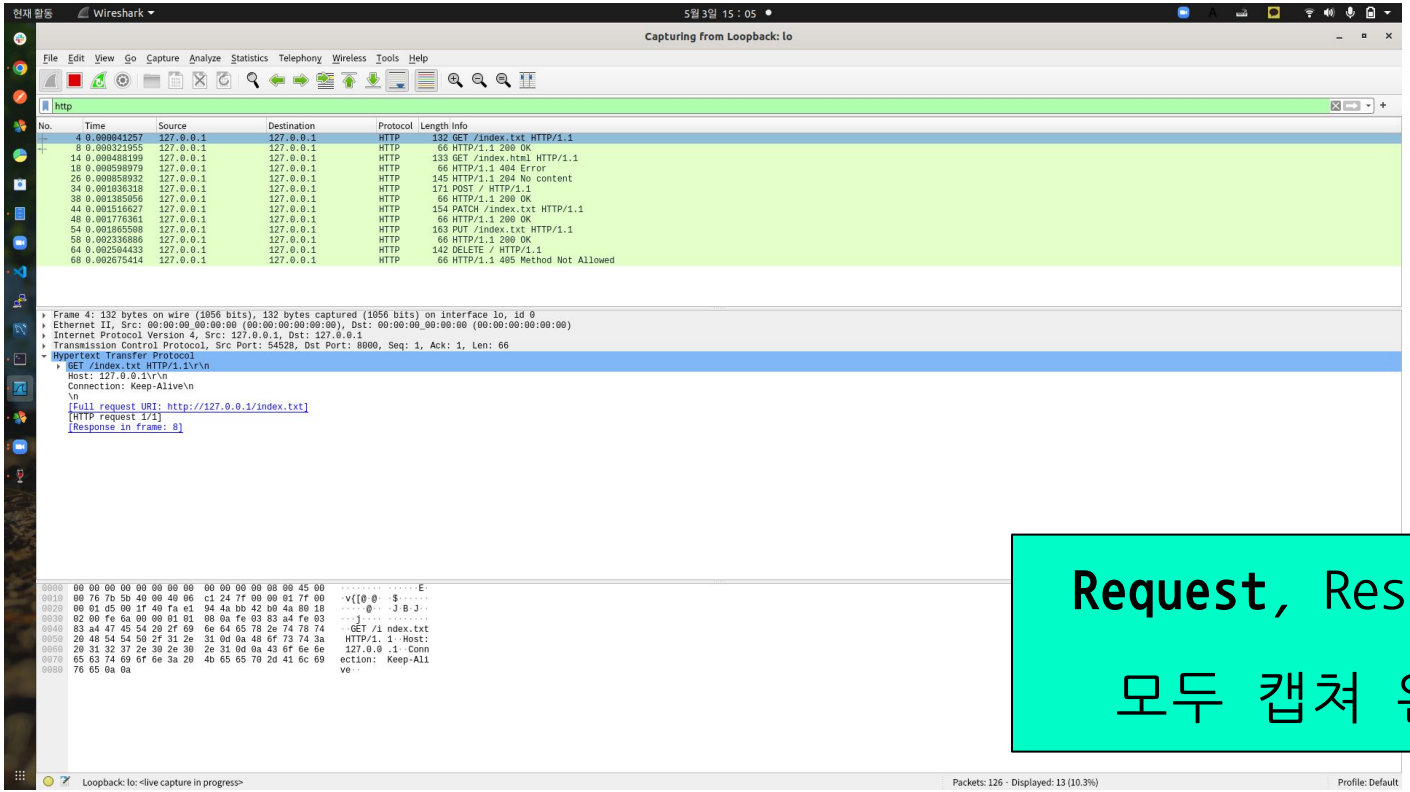
client.py
실행 터미널

```
HTTP/1.1 200 OK
Date: Tue, 03 May 2022 16:57:42 KST
Server: 127.0.0.1

Hello World!

HTTP/1.1 404 Error
Date: Tue, 03 May 2022 16:57:42 KST
Server: 127.0.0.1
```

1) HTTP (GET) - 캡쳐본



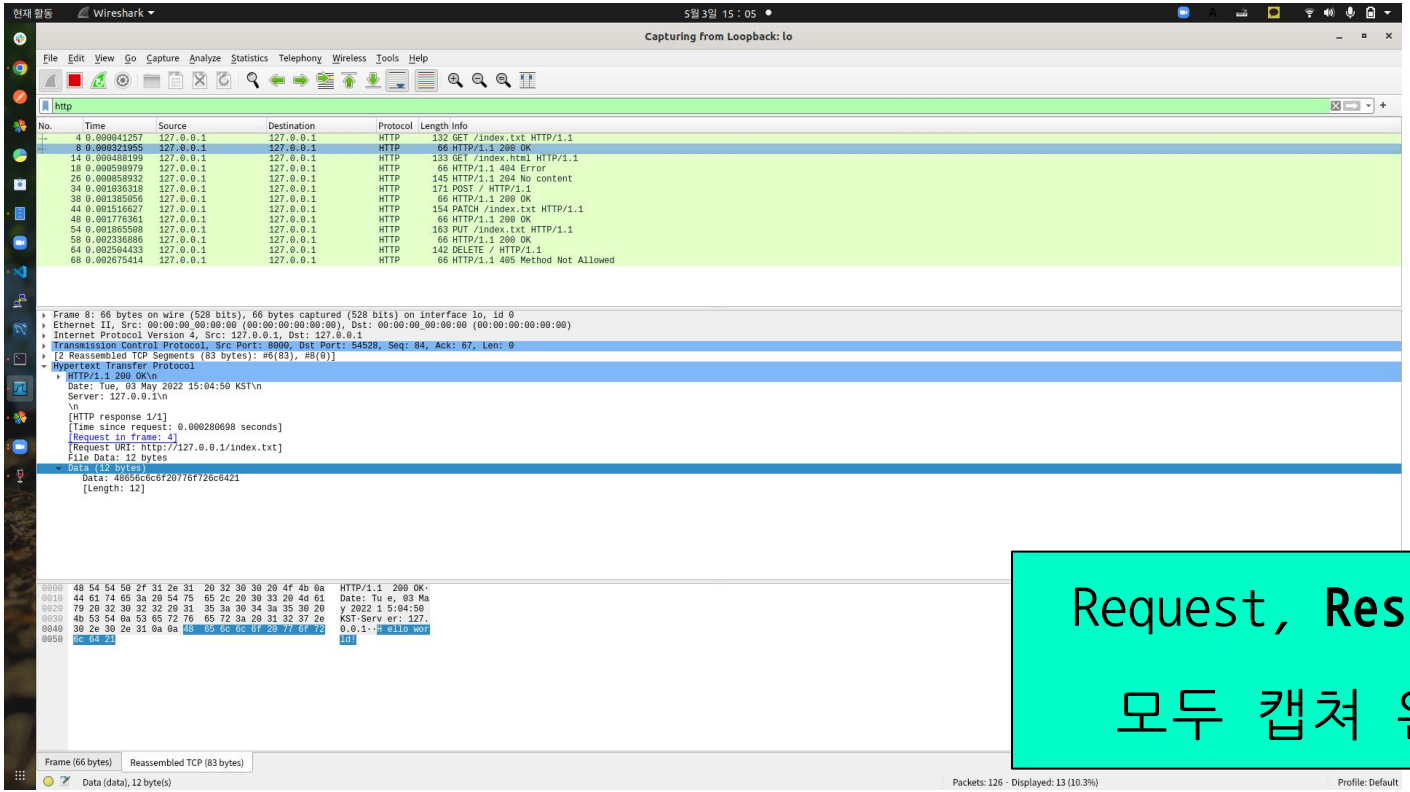
1) WireShark - GET Request 1



Request, Response

모두 캡쳐 완료

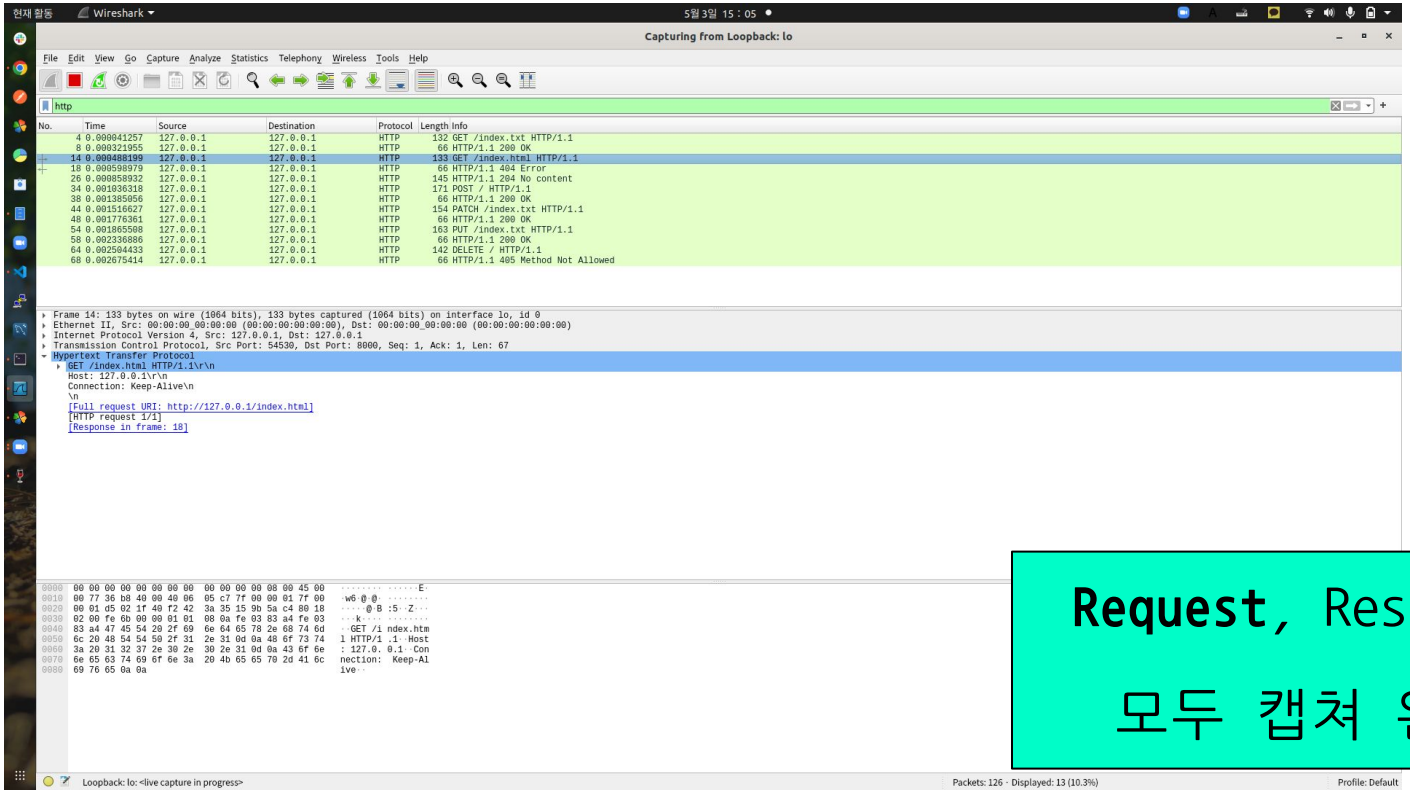
1) WireShark - GET Response 1 (200)



Request, **Response**

모두 캡쳐 완료

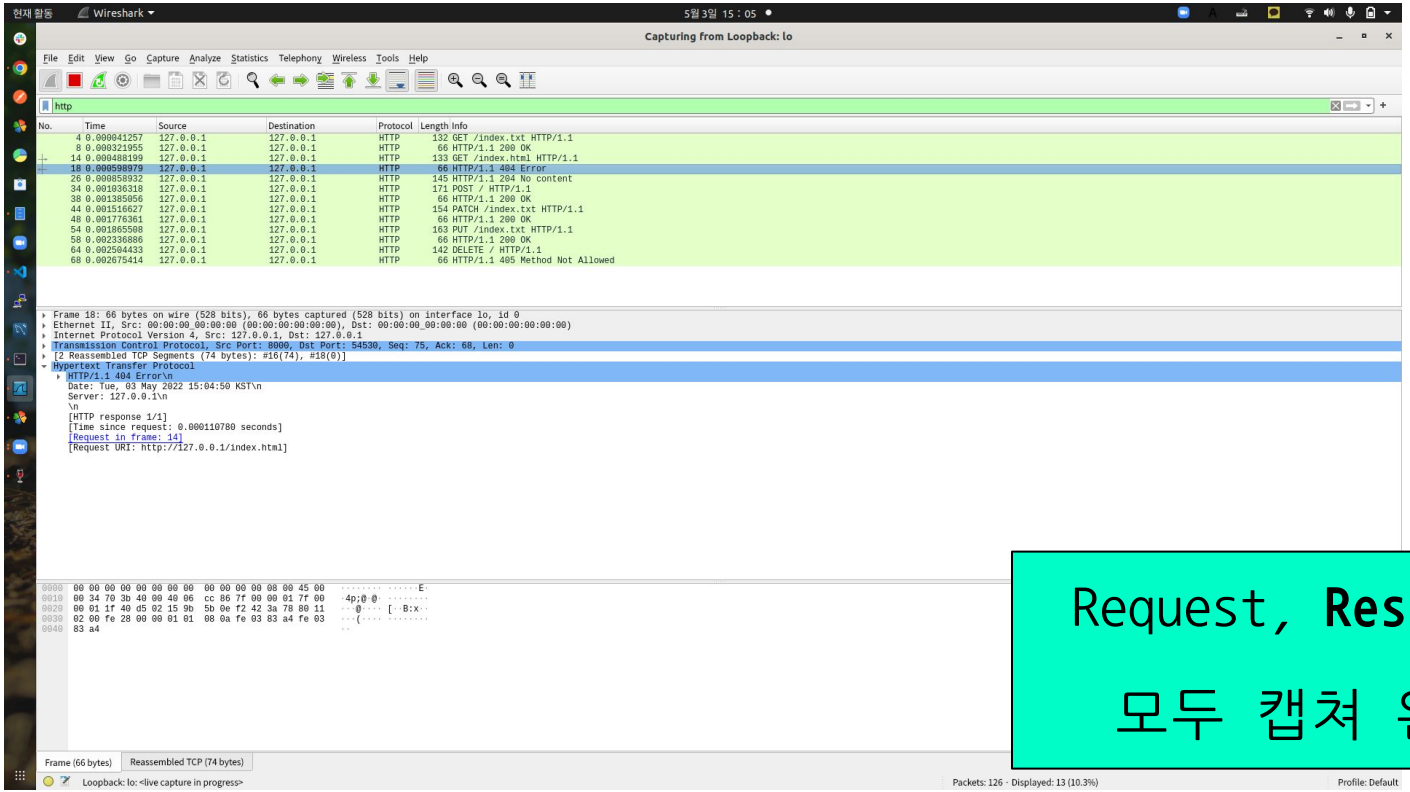
1) WireShark - GET Request 2



Request, Response

모두 캡쳐 완료

1) WireShark - GET Response 2(404)



Request, **Response**

모두 캡쳐 완료

2) HTTP Method: POST

POST?

요청된 자원을 생성(CREATE)한다.

2) HTTP (POST) - client.py

```
# POST (no data)
request_message = 'POST / HTTP/1.1\r\n'
request_message += 'Host: ' + serverName + '\r\n'
request_message += 'Connection: Keep-Alive\r\n'
create_socket_and_send_message(request_message)

# POST (make new file)
request_message = 'POST / HTTP/1.1\r\n'
request_message += 'Host: ' + serverName + '\r\n'
request_message += 'Connection: Keep-Alive\r\n'
request_message += 'Data: Filename: new_file Content:
New_Content\n\n'
create_socket_and_send_message(request_message)
```

POST Request format

Request 1) 빈 body(data)를 담아 서버에 요청 (state 204)

Request 2) 새로운 파일 이름과 파일 안에 쓸 내용을 body(data)를 담아 서버에 요청(state 200)

2) HTTP (POST) - server.py

```
elif request_method == "POST" :
            # If client has the data to post
        if len(request_body) > 0:
            response_data = "{0} 200 OK\nDate: {1}\nServer: {2}\n\n".format(request_version,
            datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
            # create new file
            fliename = request_body[1]
            content = request_body[3]
            with open('{0}.txt'.format(fliename), 'w') as f:
                f.write(content + "\n")
                f.close()
            # read file
            with open ('{0}.txt'.format(fliename), 'r') as txt:
                    while True:
                        body = txt.readline()
                        if not body:
                            break
                        response_data += body

        else: # no data to modify
            response_data = "{0} 204 No content\nDate: {1}\nServer: {2}\n\n".format(request_version,
            datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
```

State 200:
정상 응답.
새로운 파일을 생성하고
읽어 Response Data에
포함시킨다.

State 204:
생성 할 데이터가 없는
경우

2) HTTP (POST) - 실행 결과

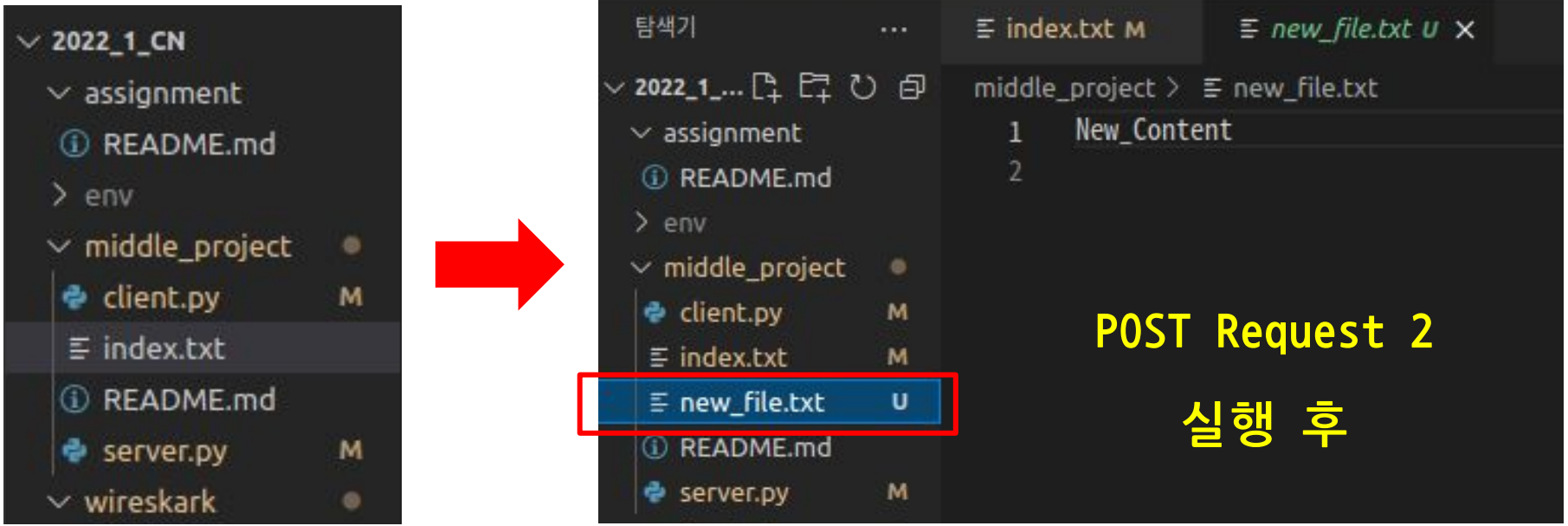
server.py
실행 터미널

```
POST / HTTP/1.1
Host: 127.0.0.1
Connection: Keep-Alive

POST / HTTP/1.1
Host: 127.0.0.1
Connection: Keep-Alive
Data: Filename: new_file Content:
New_Content
```

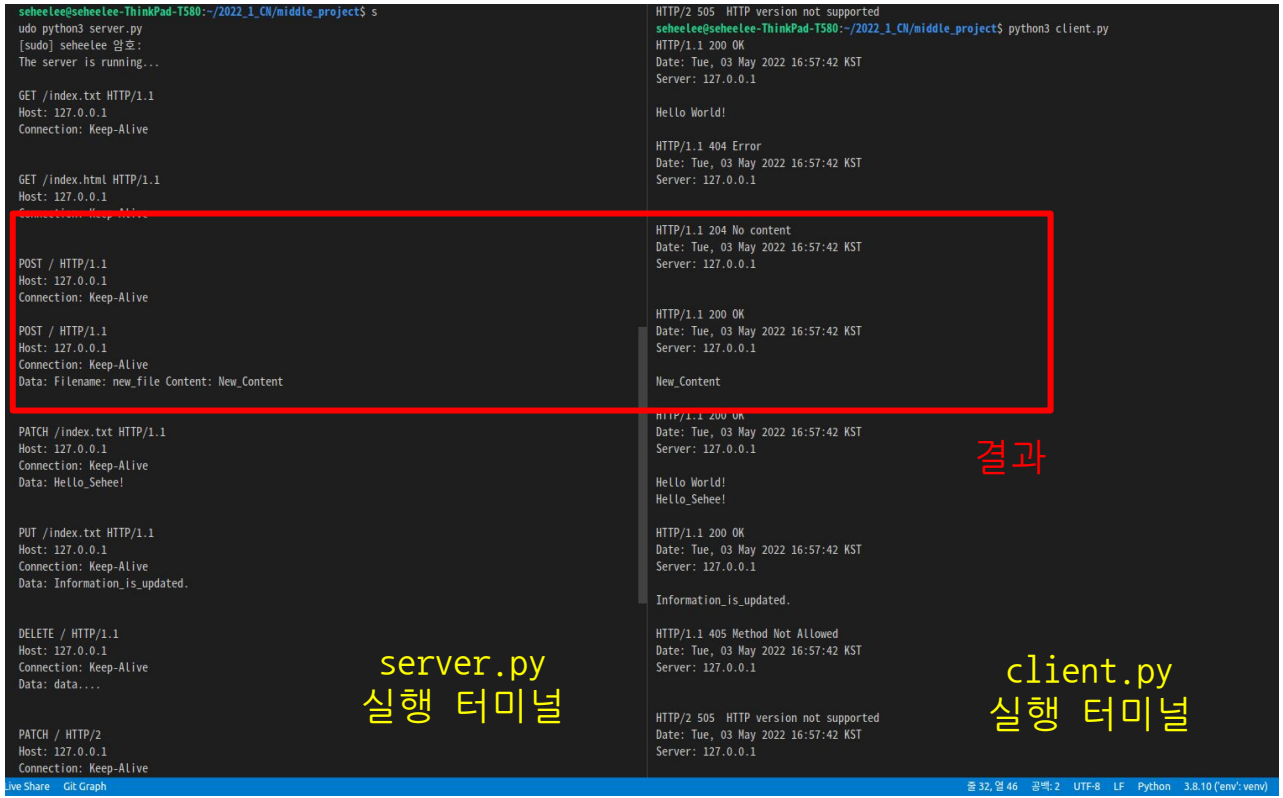
client.py
실행 터미널

```
HTTP/1.1 204 No content
Date: Tue, 03 May 2022 16:57:42 KST
Server: 127.0.0.1


HTTP/1.1 200 OK
Date: Tue, 03 May 2022 16:57:42 KST
Server: 127.0.0.1

New_Content
```

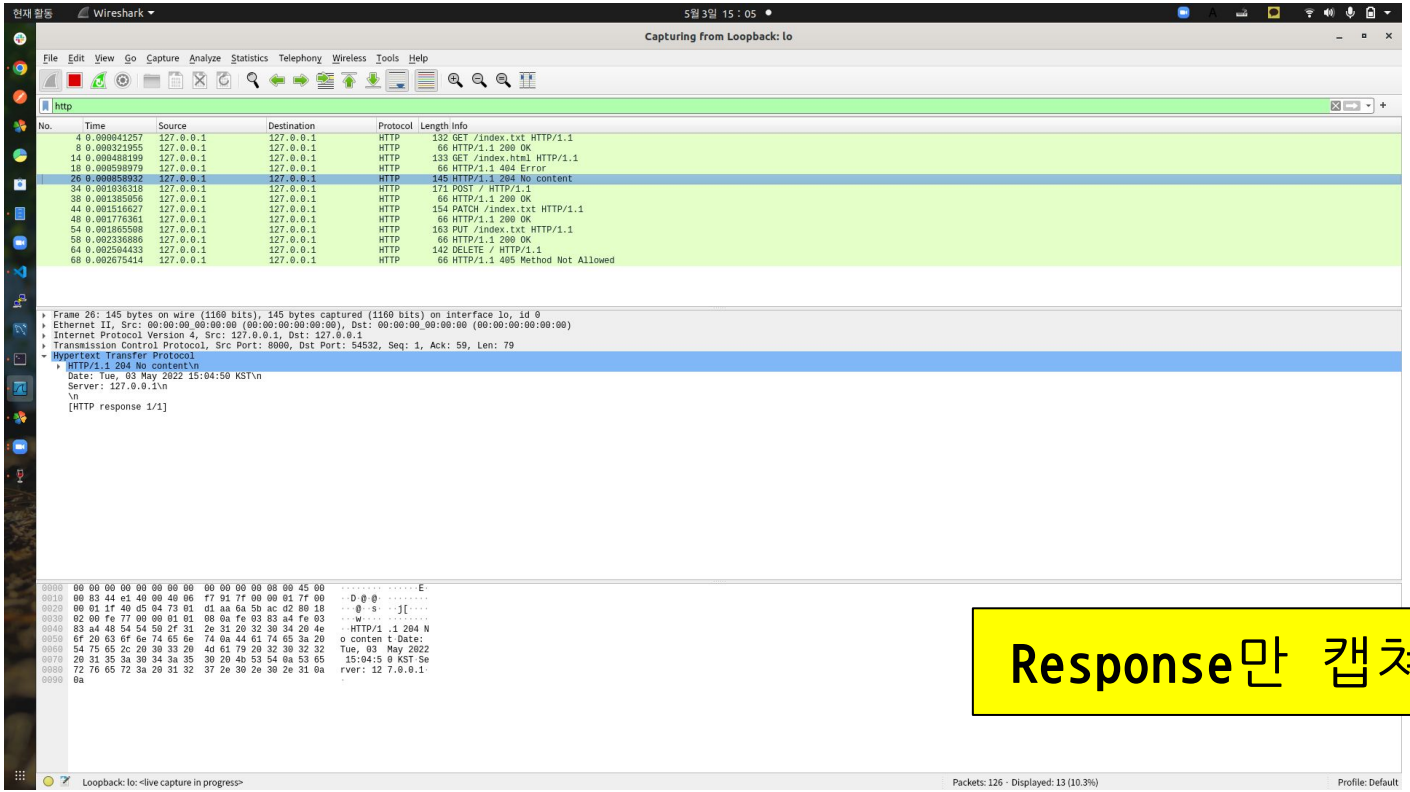
2) HTTP (POST) - 캡쳐본



3. HTTP (POST) - 캡쳐본

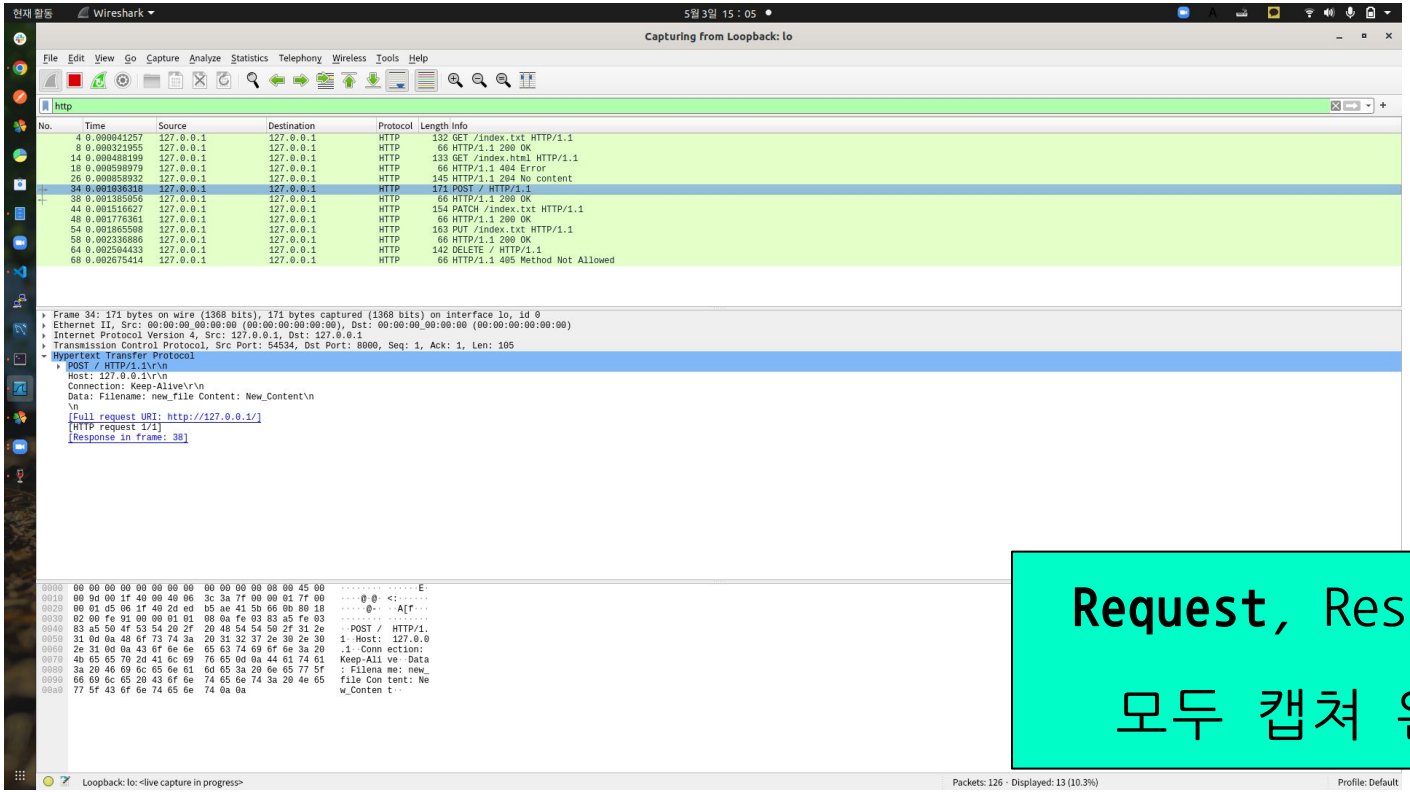


2) WireShark - POST Response 1 (204)



Response만 캡쳐 완료

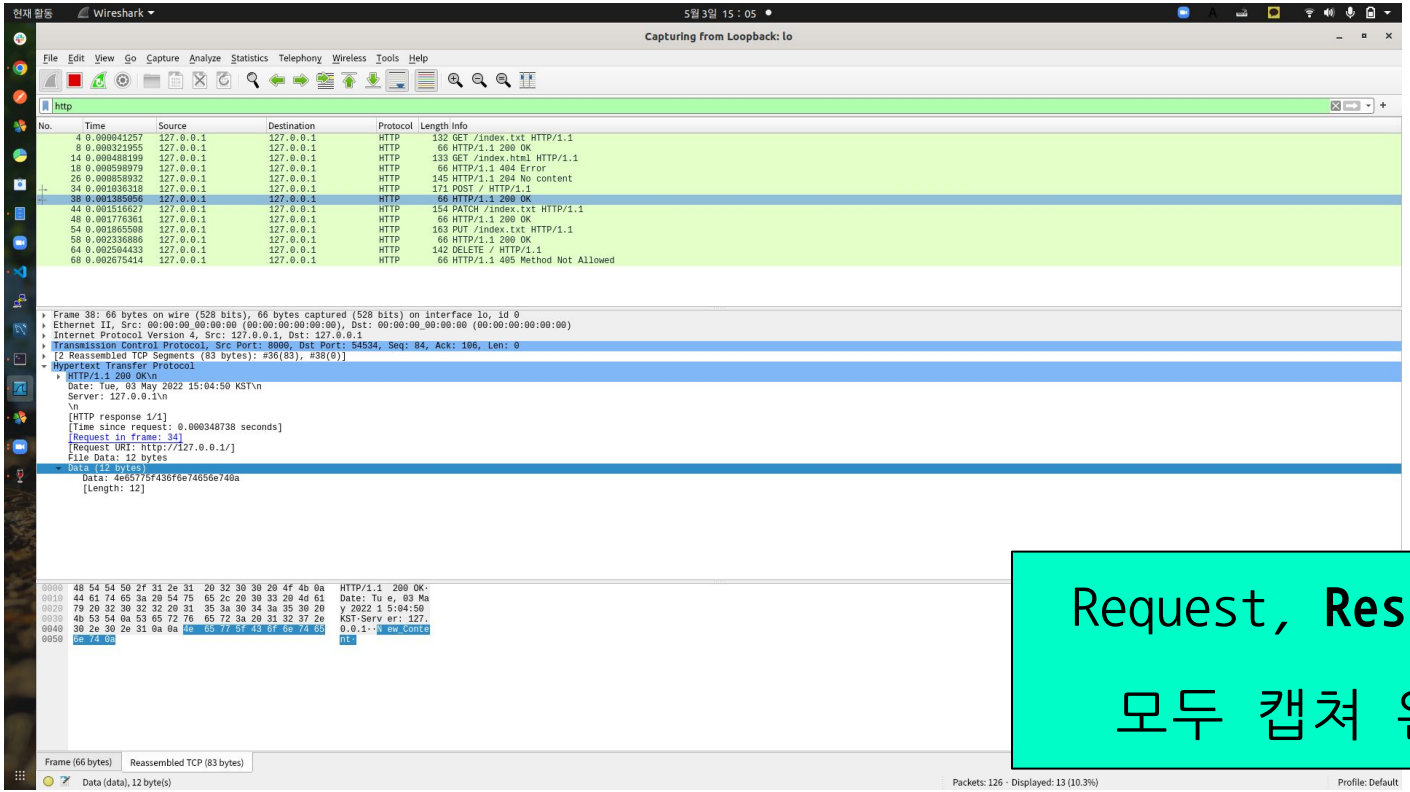
2) WireShark - POST Request 2



Request, Response

모두 캡쳐 완료

2) WireShark - POST Response 2(200)



Request, **Response**

모두 캡쳐 완료

3) HTTP Method: PUT

PUT?

요청된 자원을 수정(UPDATE)한다.

3) HTTP (PUT) - client.py

```
# PUT (update index.txt)
request_message = 'PUT /index.txt HTTP/1.1 \r\n'
request_message += 'Host: ' + serverName + '\r\n'
request_message += 'Connection: Keep-Alive \r\n'
request_message += 'Data:
Information_is_updated. \n\n'
create_socket_and_send_message (request_message )
```

PUT Request format

Request) 업데이트 파일인 index.txt를 url에 담고 이 파일의 업데이트 내용을 body(data)에 담아 서버에 요청(state 200)

3) HTTP (PUT) - server.py

```
elif request_method == "PUT" :
        if request_item == "/index.txt":
            # If client has the data to put
            if len(request_body) > 0:
                response_data = "{0} 200 OK\nDate: {1}\nServer: {2}\n\n".format(request_version,
                datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
                # update file
                content = request_body[0]
                with open('index.txt', 'w') as f:
                    f.write(content + "\n")
                    f.close()
                # read file
                with open ('index.txt', 'r') as txt:
                        while True:
                            body = txt.readline()
                            if not body:
                                break
                            response_data += body

            else: # no data to modify
                response_data = "{0} 204 No content\nDate: {1}\nServer: {2}\n\n".format(request_version,
                datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)

        else: # Invalid address(Send Data)
            response_data = "{0} 404 Error\nDate: {1}\nServer: {2}\n\n".format(request_version,
            datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
```

State 200:
정상 응답.
업데이트 된 기존의 파일을 읽어 Response Data에 포함시킨다.

State 204:
업데이트 할 데이터가 없는 경우

State 404:
올바른 파일 이름을 제시하지 않은 경우

3) HTTP (PUT) - 실행 결과

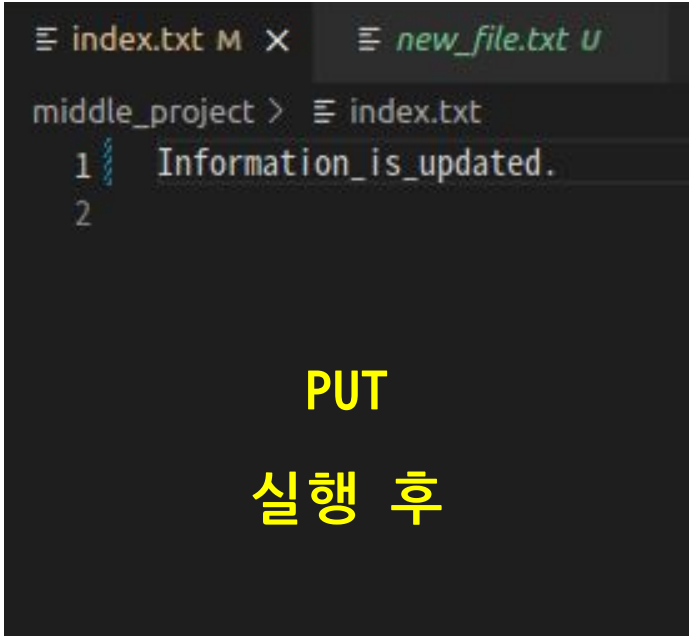
server.py
실행 터미널

```
PUT /index.txt HTTP/1.1
Host: 127.0.0.1
Connection: Keep-Alive
Data: Information_is_updated.
```

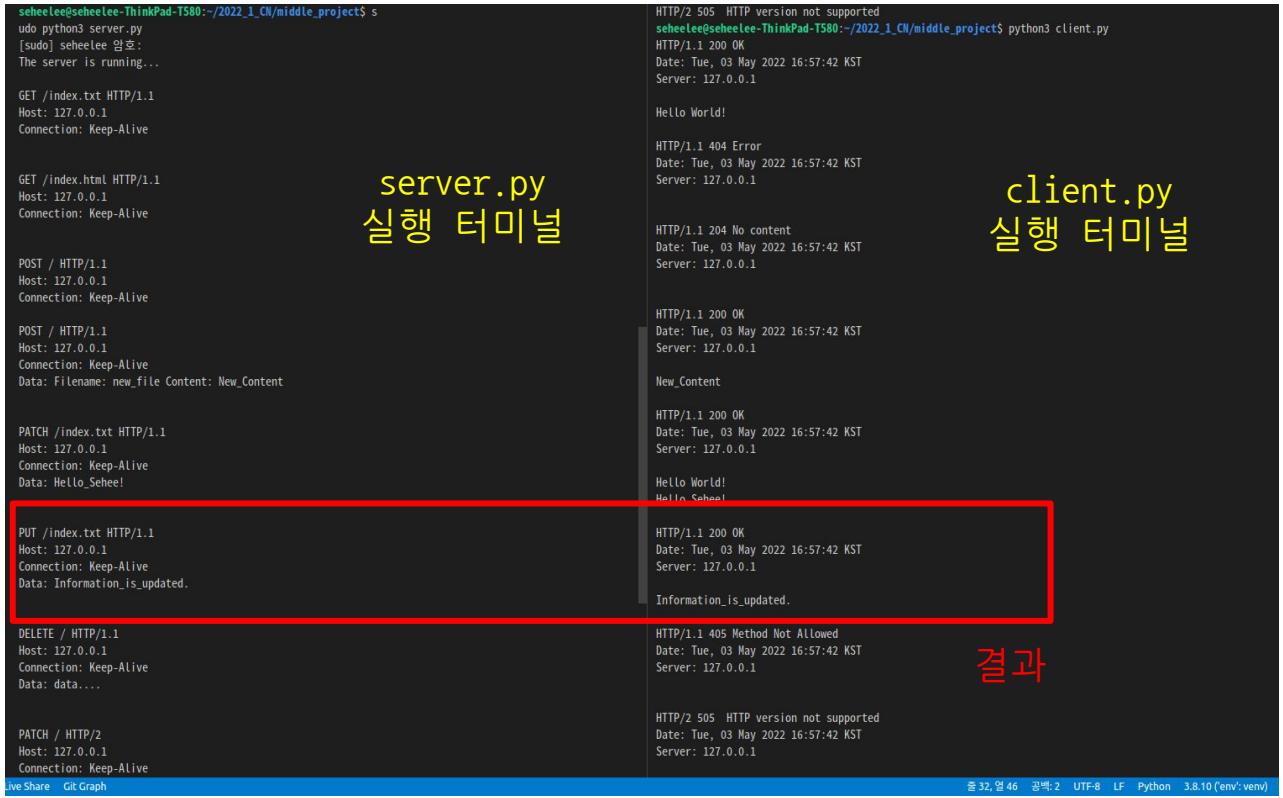
client.py
실행 터미널

```
HTTP/1.1 200 OK
Date: Tue, 03 May 2022 16:57:42 KST
Server: 127.0.0.1

Information_is_updated.
```

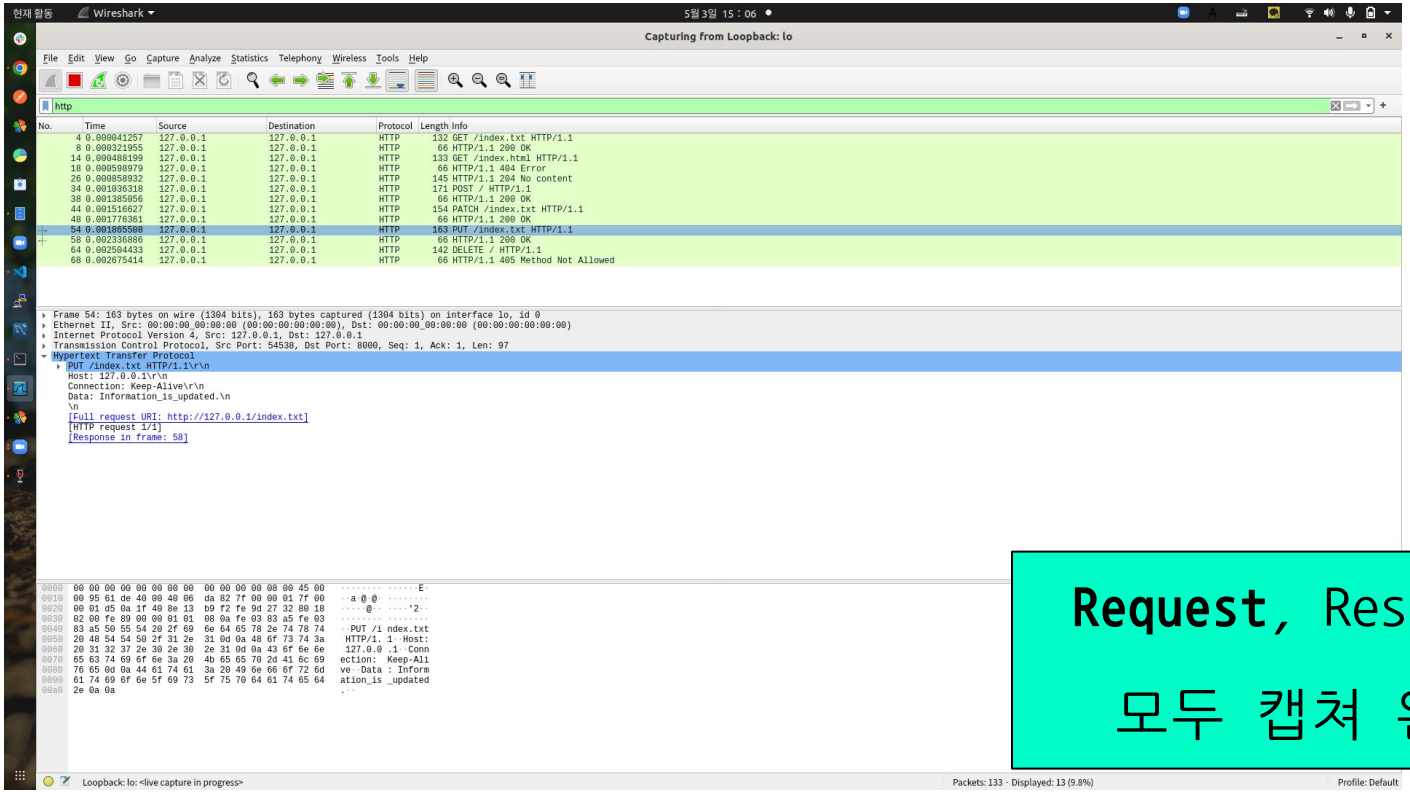
3) HTTP (PUT) - 캡쳐본



3. HTTP (PUT) - 캡쳐본



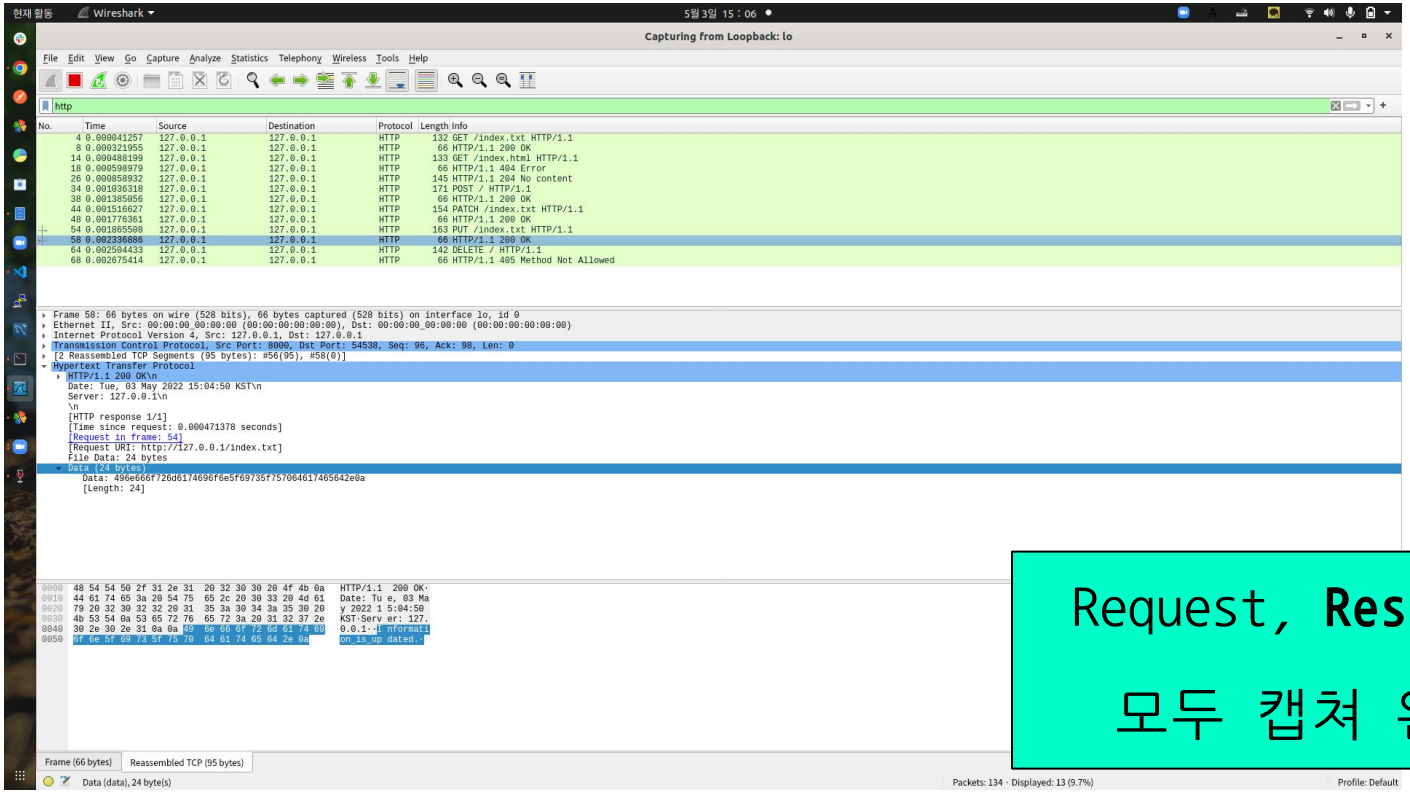
3) WireShark - PUT Request



Request, Response

모두 캡쳐 완료

3) WireShark - PUT Response(200)



Request, **Response**

모두 캡쳐 완료

4) HTTP Method: PATCH

PATCH?

PUT과 유사하게 요청된 자원을
수정(UPDATE)할 때 사용한다.
PUT의 경우 자원 전체를 갱신하는
의미지만, PATCH는 주로
해당자원의 일부를 교체하는
의미로 사용한다.

4) HTTP (PATCH) - client.py

```
# PATCH (modify index.txt)
request_message = 'PATCH /index.txt HTTP/1.1 \r\n'
request_message += 'Host: ' + serverName + '\r\n'
request_message += 'Connection: Keep-Alive \r\n'
request_message += 'Data: Hello_Sehee! \n\n'
create_socket_and_send_message (request_message)
```

PATCH Request format

Request) 수정할 파일인 index.txt를 url에 담고 이 파일의 수정 내용을 body(data)에 담아 서버에 요청(state 200)

4) HTTP (PATCH) - server.py

```
elif request_method == "PATCH" :
        if request_item == "/index.txt":
            # If client has the data to patch
            if len(request_body) > 0:
                response_data = "{0} 200 OK\nDate: {1}\nServer: {2}\n\n".format(request_version,
                datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
                append = request_body[0]
                # update file
                with open('index.txt', 'a') as f:
                    f.write(append + "\n")
                    f.close()
                # modify file
                with open ('index.txt', 'r') as txt:
                    while True:
                        body = txt.readline()
                        if not body:
                            break
                        response_data += body

            else: # no data to modify
                response_data = "{0} 204 No content\nDate: {1}\nServer: {2}\n\n".format(request_version,
                datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)

        else: # Invalid address(Send Data)
            response_data = "{0} 404 Error\nDate: {1}\nServer: {2}\n\n".format(request_version,
            datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
```

State 200:
정상 응답.
수정된 기존의 파일을 읽어 Response Data에 포함시킨다.

State 204:
수정해야할 데이터가 없는 경우

State 404:
올바른 파일 이름을 제시하지 않은 경우

4) HTTP (PATCH) - 실행 결과

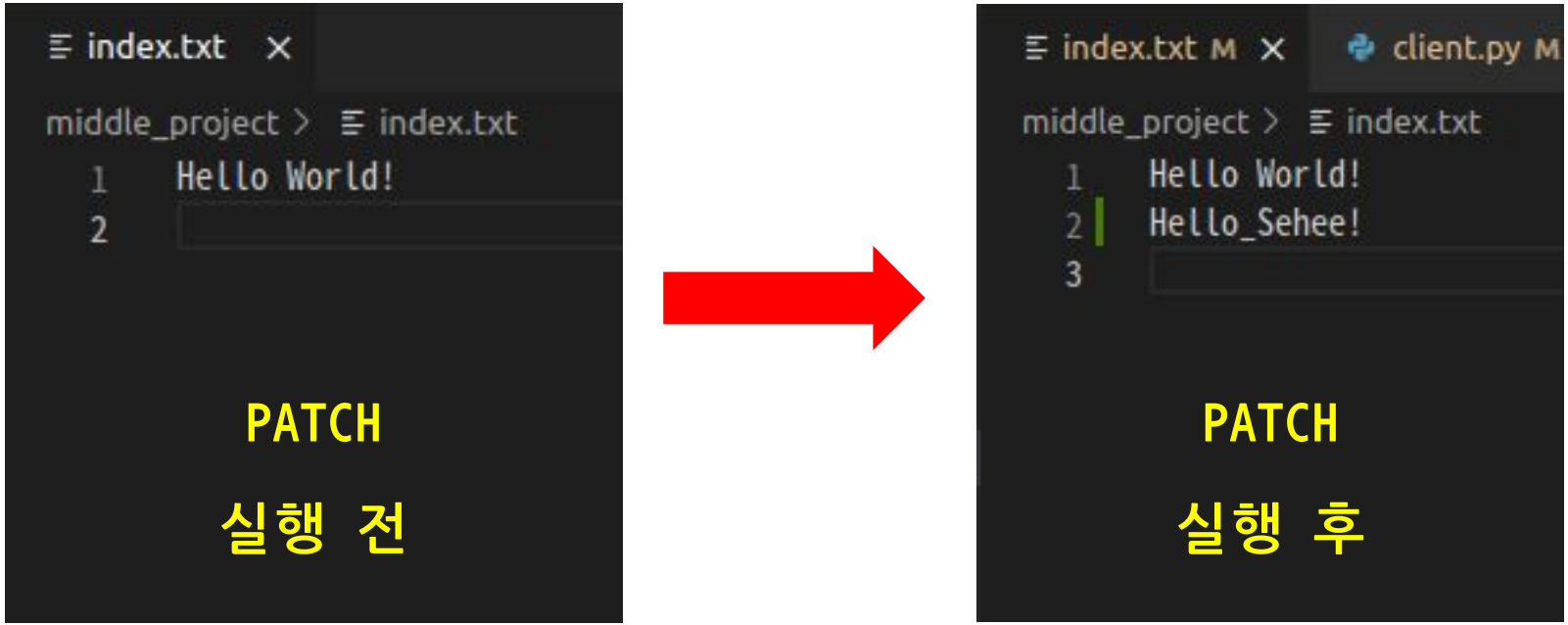
server.py
실행 터미널

```
PATCH /index.txt HTTP/1.1
Host: 127.0.0.1
Connection: Keep-Alive
Data: Hello_Sehee!
```

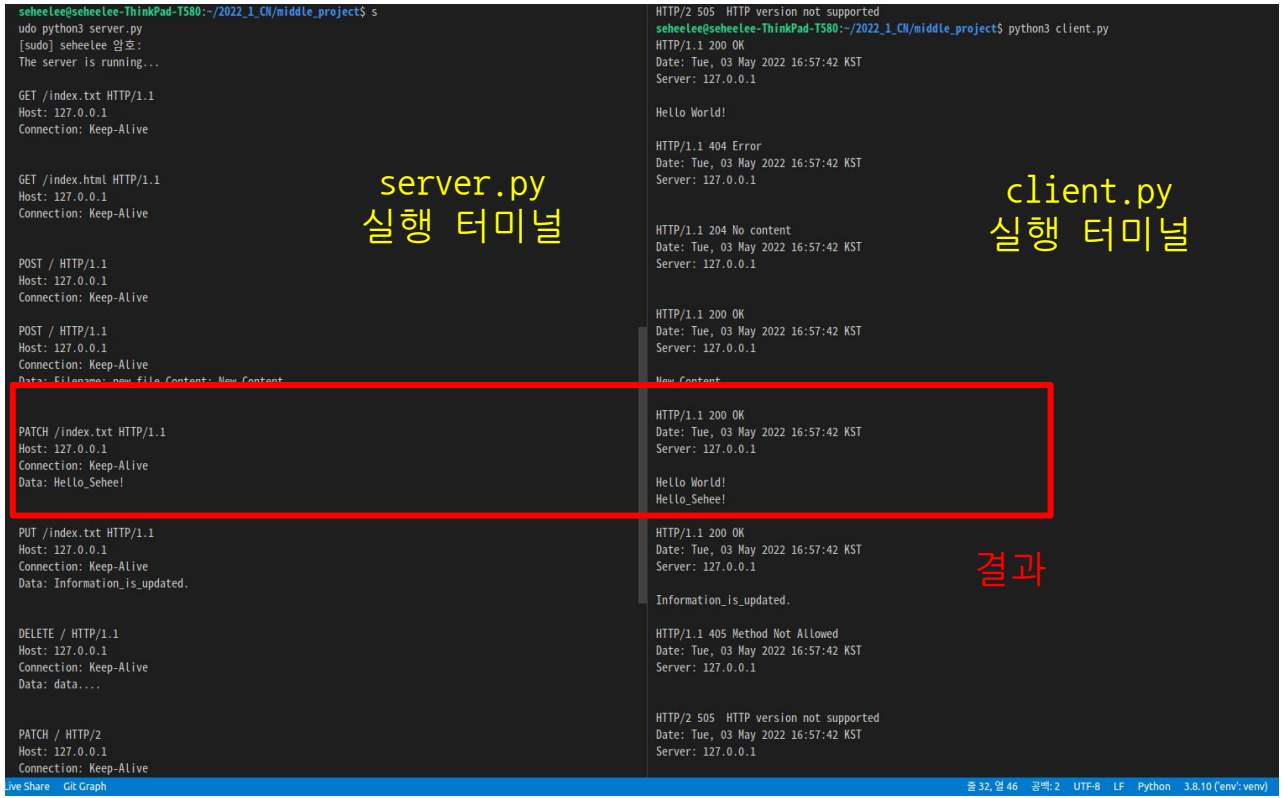
client.py
실행 터미널

```
HTTP/1.1 200 OK
Date: Tue, 03 May 2022 16:57:42 KST
Server: 127.0.0.1

Hello World!
Hello_Sehee!
```

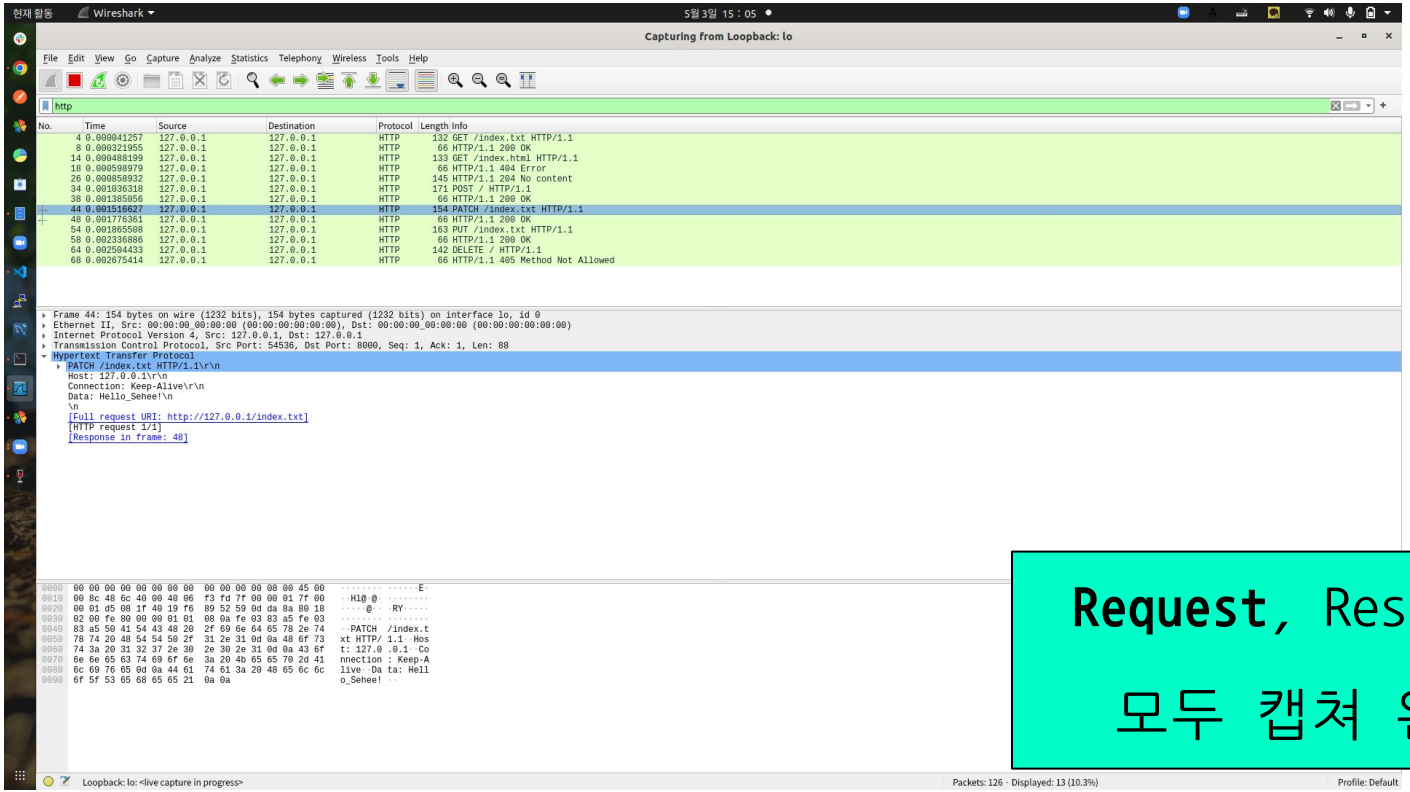
4) HTTP (PATCH) - 캡쳐본



4) HTTP (PATCH) - 캡쳐본



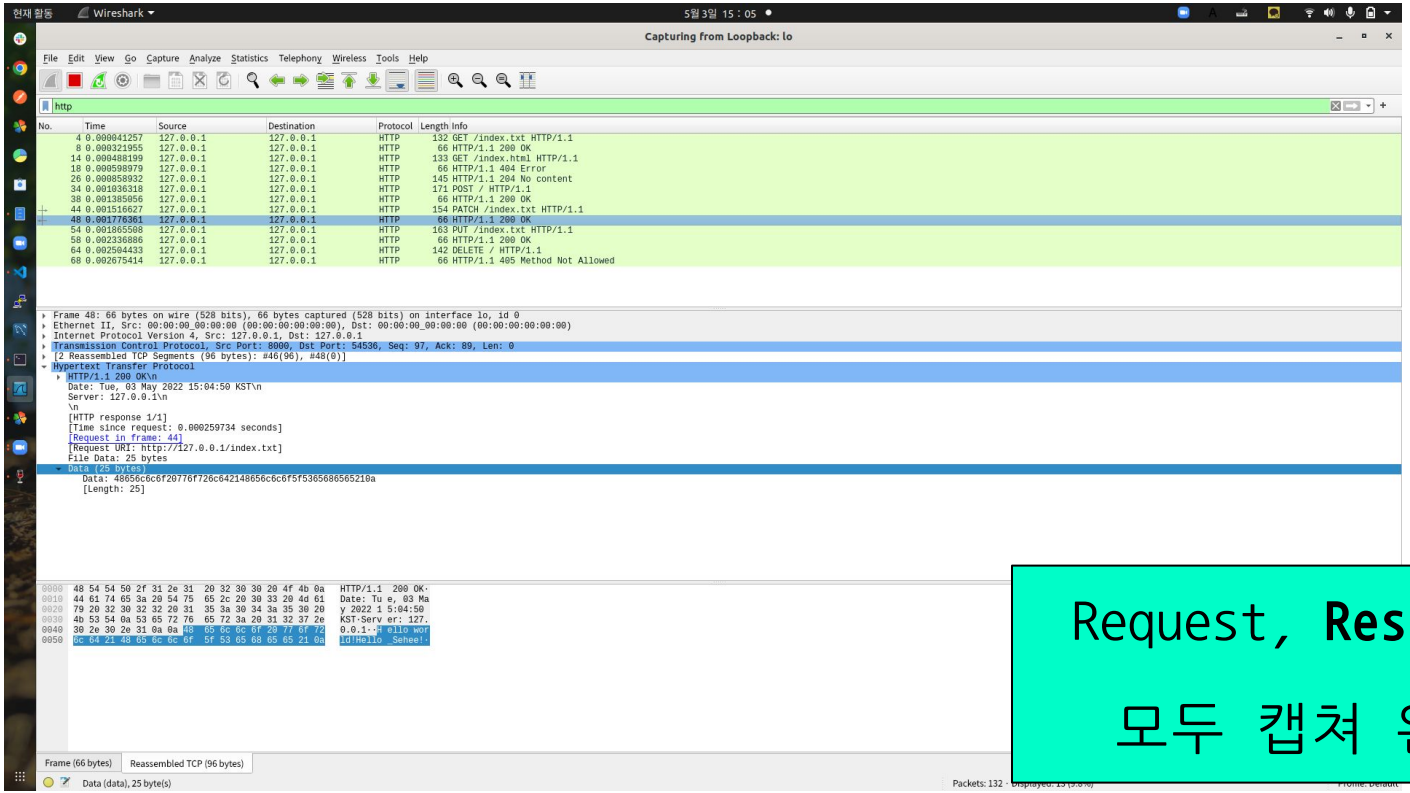
4) WireShark - PATCH Request



Request, Response

모두 캡쳐 완료

4) WireShark - PATCH Response(200)



Request, **Response**

모두 캡쳐 완료

5) HTTP State: Error

5) HTTP (State 405) - client.py

```
# Error (Other Method)
request_message = 'DELETE / HTTP/1.1\r\n'
request_message += 'Host: ' + serverName +
'\r\n'
request_message += 'Connection:
Keep-Alive\r\n'
request_message += 'Data: data....\n\n'
create_socket_and_send_message(request_messag
e)
```

DELETE Request format

Request) 서버에서 구현하지 않은 DELETE 메소드를 서버에 요청 (state 405)

5) HTTP (State 405) - server.py

```
# Other method
        else:
            response_data = "{0} 405 Method Not Allowed\nDate: {1}\nServer: {2}\n\n".format(request_version,
            datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
```

State 405:
정의 되지 않은 다른
Method Request가
오는 경우

5) HTTP (State 405) - 실행 결과

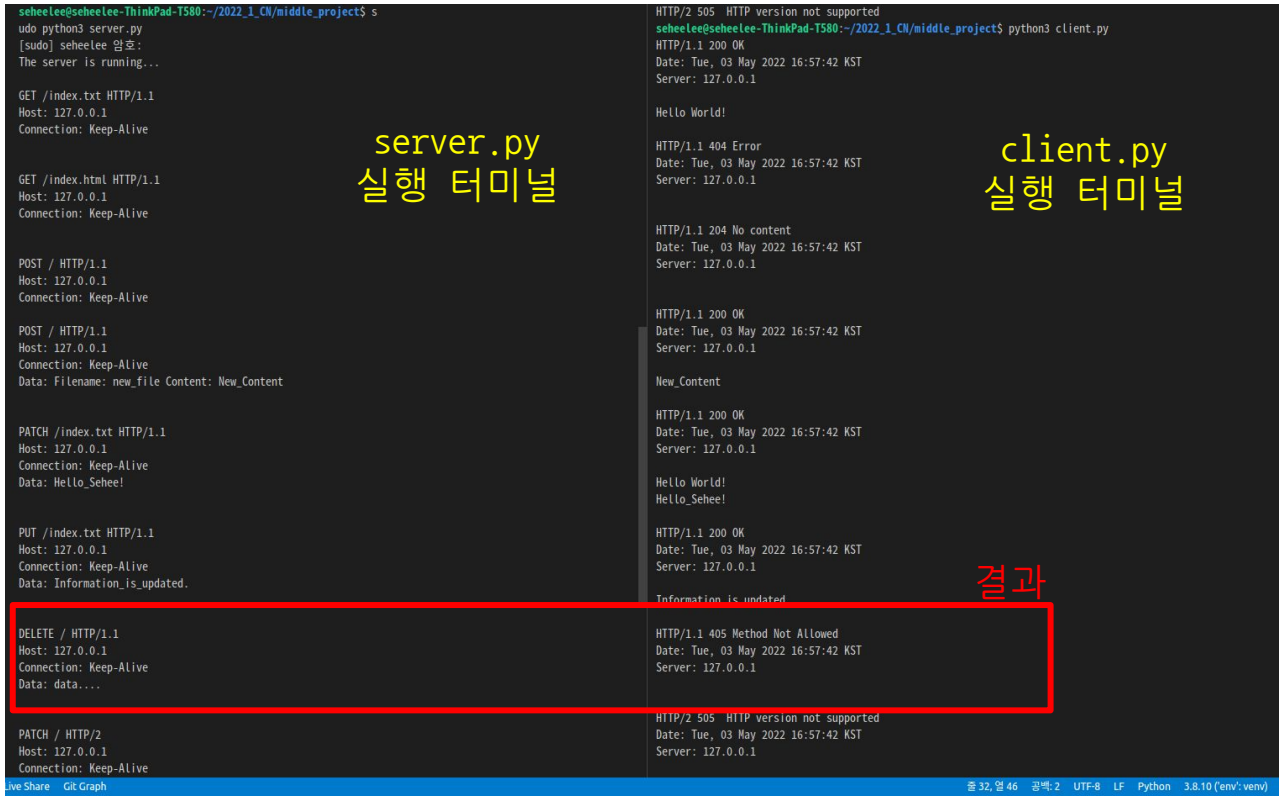
server.py
실행 터미널

```
DELETE / HTTP/1.1
Host: 127.0.0.1
Connection: Keep-Alive
Data: data....
```

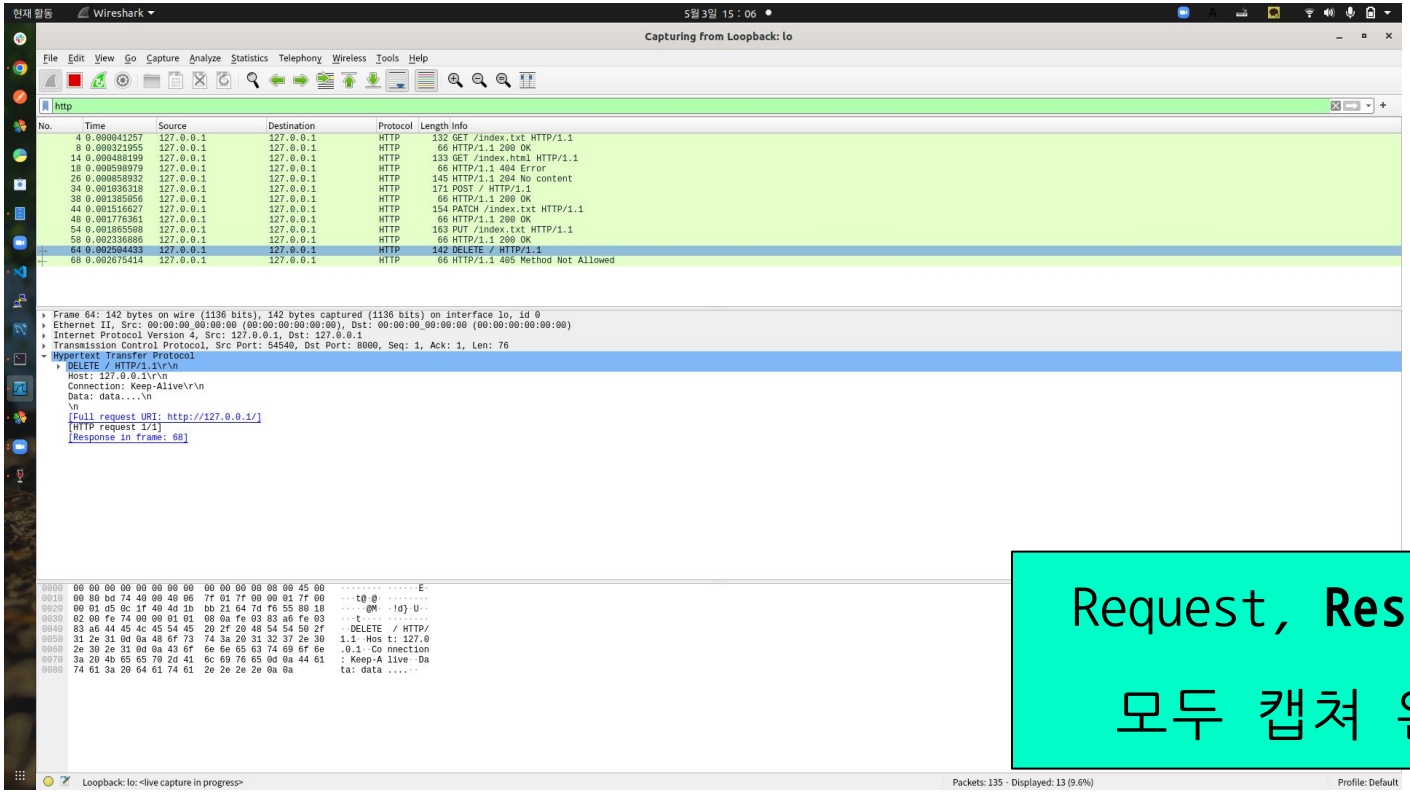
client.py
실행 터미널

```
HTTP/1.1 405 Method Not Allowed
Date: Tue, 03 May 2022 16:57:42 KST
Server: 127.0.0.1
```

5) HTTP (State 405) - 캡쳐본



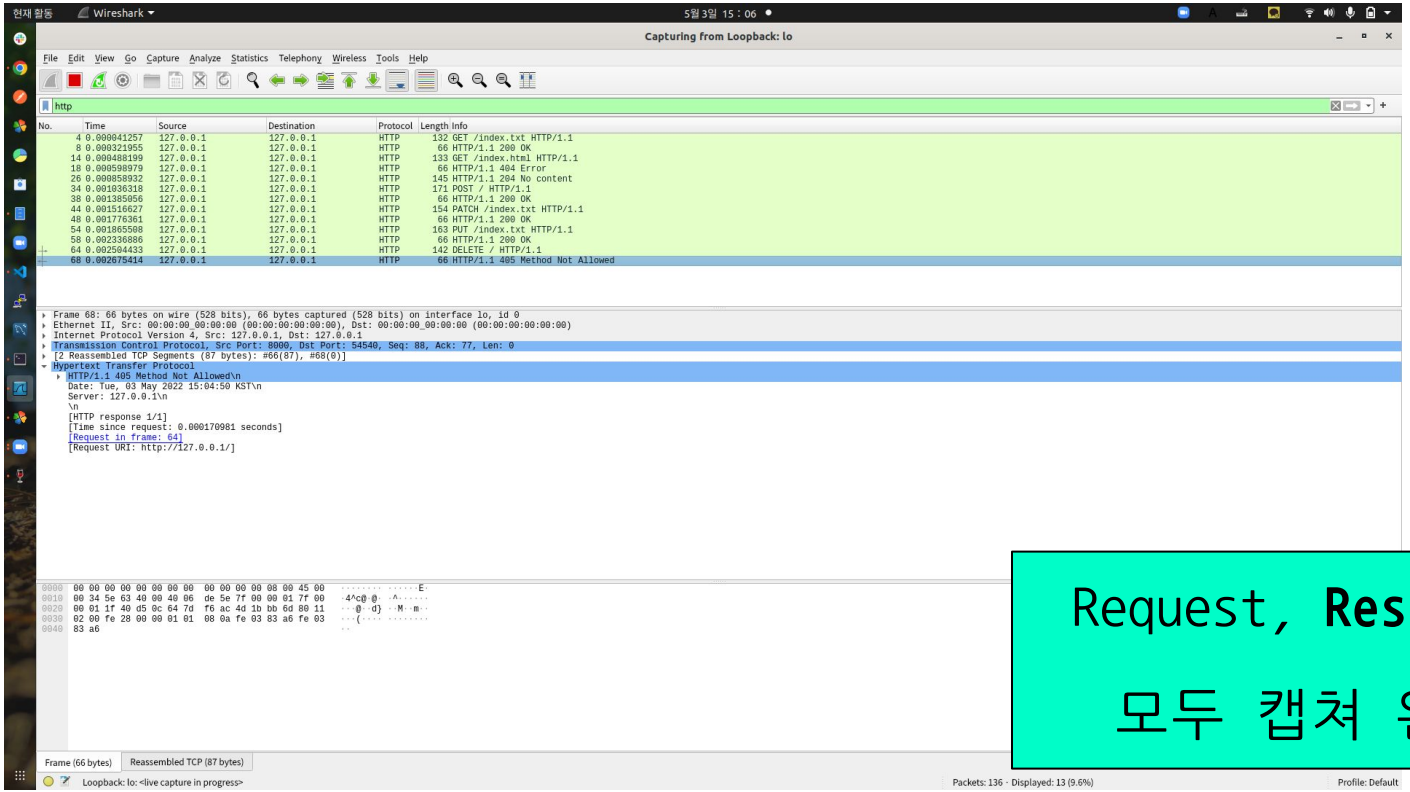
5) WireShark - DELETE Request



Request, **Response**

모두 캡쳐 완료

5) WireShark - DELETE Response(405)



Request, **Response**

모두 캡쳐 완료

5) HTTP (State 505) - client.py

```
# Error (Other HTTP version)
request_message = 'PATCH / HTTP/2\r\n'
request_message += 'Host: ' + serverName +
'\r\n'
request_message += 'Connection:
Keep-Alive\n\n'
create_socket_and_send_message(request_messag
e)
```

HTTP/2 Request format

Request) HTTP/1.1 요청 방식을 다루는 서버에 HTTP/2 방식으로 요청

(state 505)

5) HTTP (State 505) - server.py

```
    # Other HTTP version
    else:
        response_data = "{0} 505  HTTP version not supported\nDate: {1}\nServer: {2}\n\n".format(request_version,
        datetime.now().strftime('%a, %d %b %Y %H:%M:%S KST'), server_name)
```

State 505:
HTTP 버전이 다른 경우

5) HTTP (State 505) - 실행결과

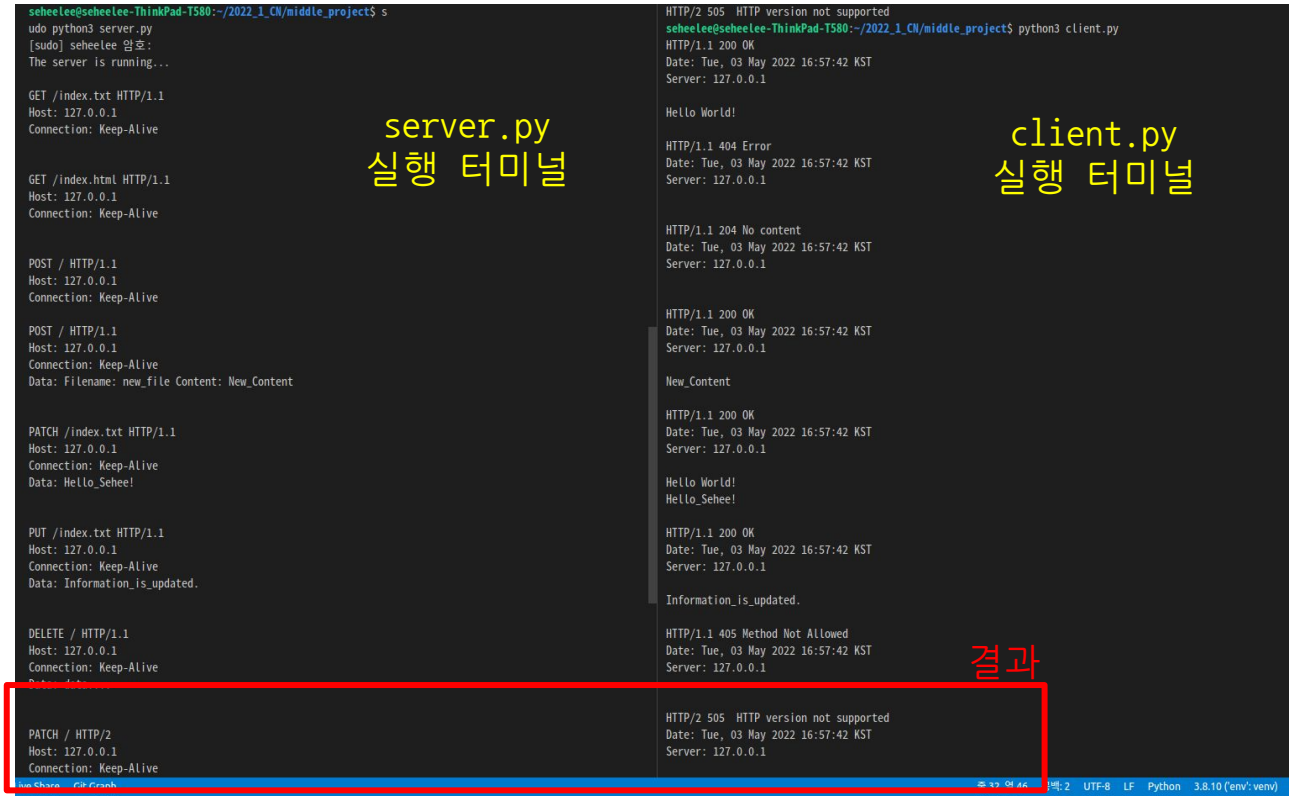
server.py
실행 터미널

```
PATCH / HTTP/2
Host: 127.0.0.1
Connection: Keep-Alive
```

client.py
실행 터미널

```
HTTP/2 505  HTTP version not
supported
Date: Tue, 03 May 2022 16:57:42 KST
Server: 127.0.0.1
```

5) HTTP (State 505) - 캡쳐본



마무리

1) **TCP 소켓 통신**

   서버와 클라이언트의 통신 방법을 더 직관적으로 알게되었다. HTTP라는 프로토콜 형식에 대한 이해가 더 깊어진 것 같다.

2) **Wireshark**

   와이어샤크를 통해 내가 직접 작성한 패킷이 HTTP 통신으로 캡쳐되는 것이 흥미로웠다. 다만 Post Request 1, Response State 505가 나오는 Request, Response가 와이어샤크에 잡히지 않는다는 것이 아쉬웠다.