

# Linux-like CLI CS 통신 프로그램

컴퓨터네트워크[01] 20203039 김선우

## 프로젝트 설명

### Overview

소켓 통신을 활용하여 **Client**에서 **Server**를 **txt**파일을 관리 생성하는 프로그램 작성을 하였습니다. 프로그래밍 언어는 **Python** 으로 구현하였습니다. **CLI**형식으로 구현하였으며, 좀 더 명령어에 친숙함을 느끼기 위해, 리눅스 몇개의 명령어(**ls**, **echo**, **rm**, **more**)을 모방하였습니다. 아래의 시나리오로 데모영상을 제작하였습니다.

Client command -Server Response 시나리오.

| Client command               | Response        | 발생하는 일, 부가 설명      |
|------------------------------|-----------------|--------------------|
| ls                           | 200 OK          | DB 파일 내역 보여줌       |
| dir                          | 400 Bad request | 구현되지 않은 명령어.       |
| more sunwoo                  | 200 OK          | sunwoo 내용을 보여줌.    |
| echo "new file" > test2      | 201 Created     | db에 파일 test2 파일 생성 |
| echo "hi I'm Sunwoo" > test1 | 200 OK          | tes1 파일 내용 수정      |
| rm test1                     | 200 OK          | test1 파일 삭제        |
| ls                           | 200 OK          | DB 파일 내역 보여줌       |
| rm test18                    | 404 Not Found   | DB에 없는 파일을 지우려 함.  |

## 시작 가이드

### Requirements

- Python 3.8
  - Python STL
    - os
    - socket
    - json
- Ubuntu 20.04 LTS

## 주요 기능

Client에서 수행할 수 있는 명령어는 총 4개로, **echo**, **ls**, **more**, **rm**이 있습니다. 아래에 각 명령어 사용법이 있습니다. 해당 명령어는 Linux 명령어 **format**을 사용했습니다.

### 1. echo

- 설명1 : Server의 DB에 새로운 txt 파일을 하나 생성합니다.
- 예시1 : 내용이 "hello world"인 **test1.txt** 파일 생성.
- CMD : **echo "hello world" > test1**
- 결과1 :



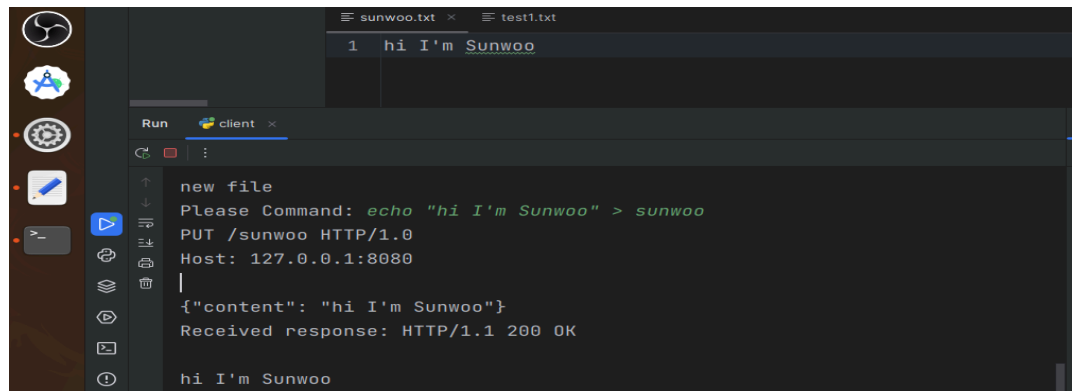
```

Run client x
Kim Sun Woo
Please Command: echo "new file" > test2
PUT /test2 HTTP/1.0
Host: 127.0.0.1:8080

{"content": "new file"}
Received response: HTTP/1.1 201 Created

new file
  
```

- 설명2 : 만약 DB에 이미 파일이 있다면, 파일 내용을 수정합니다.
- 예시2 : 내용이 "hello world"인 **test1.txt** 파일의 내용이 "hi"로 변경.
- CMD : **echo "hi" > test1**
- 결과2 :



```

sunwoo.txt x test1.txt
1 hi I'm Sunwoo

Run client x
new file
Please Command: echo "hi I'm Sunwoo" > sunwoo
PUT /sunwoo HTTP/1.0
Host: 127.0.0.1:8080

{"content": "hi I'm Sunwoo"}
Received response: HTTP/1.1 200 OK

hi I'm Sunwoo
  
```

### 2. ls

- 설명 : DB에 있는 **file list**를 출력합니다.
- 예시 : DB에 있는 날짜순(오름차순)으로 정렬된 **file list**가 response온다. Client는 출력.
- CMD : **ls**
- 결과 :



```

Run client x
/usr/bin/python3.8 /home/seonu/kmu/2nd/sem_win/20203039_김선우/workspace/cli
Please Command: ls
GET / HTTP/1.0
Host: 127.0.0.1:8080

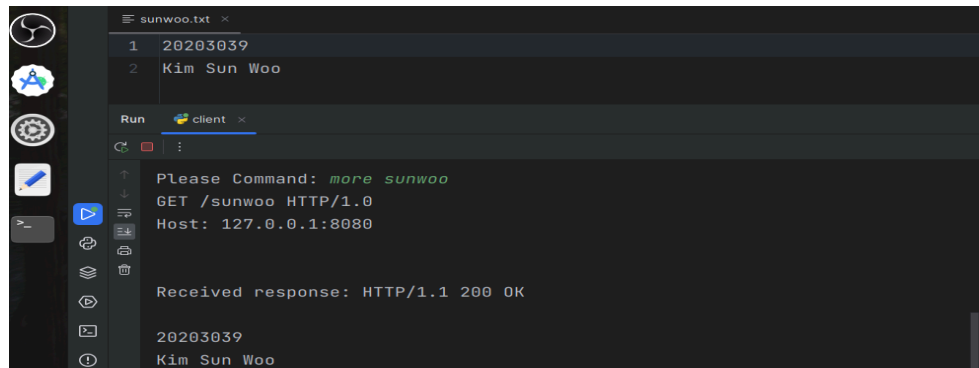
Received response: HTTP/1.1 200 OK

Mon Jan 8 20:58:41 2024 sunwoo.txt
Mon Jan 8 22:33:02 2024 test1.txt

```

### 3. more

- 설명 : 파일의 내용을 출력합니다.
- 예시 : sunwoo.txt의 내용인 “20203039 Kim Sun Woo”을 Client에서 출력.
- CMD : more test1
- 결과 :



```

sunwoo.txt x
1 20203039
2 Kim Sun Woo

Run client x
Please Command: more sunwoo
GET /sunwoo HTTP/1.0
Host: 127.0.0.1:8080

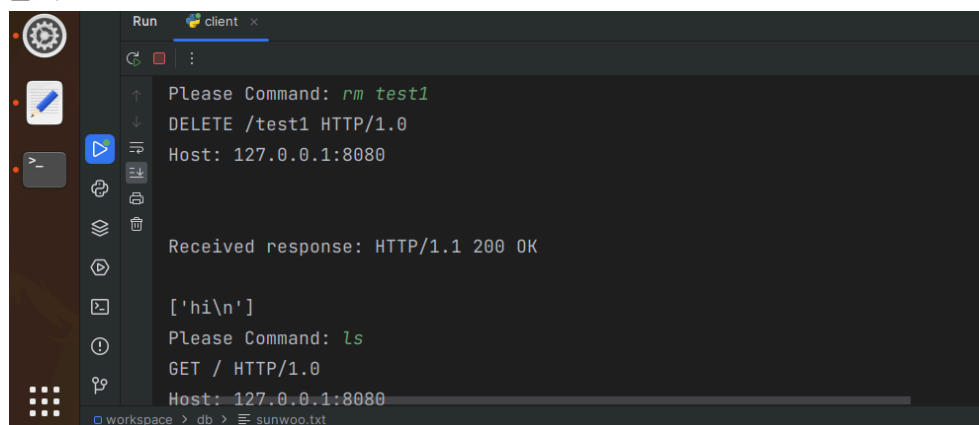
Received response: HTTP/1.1 200 OK

20203039
Kim Sun Woo

```

### 4. rm

- 설명 : 입력한 하나의 파일을 제거합니다.
- 예시 : DB에 있는 file “test1.txt”을 제거한다.
- CMD : rm test1
- 결과 :



```

Run client x
Please Command: rm test1
DELETE /test1 HTTP/1.0
Host: 127.0.0.1:8080

Received response: HTTP/1.1 200 OK

['hi\n']
Please Command: ls
GET / HTTP/1.0
Host: 127.0.0.1:8080

```

## 프로그램 구현 설명

## client.py

- **Help Function : extract\_content\_within\_quotes(command)**
- Description :
  - 명령에서 큰따옴표 안의 내용을 추출합니다. 명령어 `ehco`의 file content를 처리하기 위한 help function 입니다.
- Parameter :
  - `command(str)`, 큰따옴표가 포함된 입력 명령어 입니다.
- Returns :
  - 큰따옴표 안에 있는 내용을 반환합니다. 그렇지 않으면 빈 문자열 (`str`)
- Example :
  - input : `echo "hi" > test1`
  - return : `hi`
- **Key Function : msgToRequest(msg)**
- Description : 사용자 명령을 HTTP 요청으로 변환합니다.
- Parameter : `msg(str)`, user command을 받습니다.
- Returns : HTTP Request Message(`str`)로 변환합니다.
- Example :
  - input: `ls`
  - return : `GET / HTTP/1.0 /r/n HOST: 127.0.0.1:8080`
- In Detail : 아래는 각 cmd에 따른 HTTP 요청, 구현 방법의 근거를 기술합니다.
  - `echo` : PUT, URI = `"/file_name"`, request\_body = `{"content": "user input content"}`
    - `echo`는 해당 프로젝트에서 파일을 새로 생성하거나 수정하는 명령어입니다.
    - MDN에 따르면, HTTP PUT method는 요청 페이로드를 사용해 새로운 리소스를 생성하거나, 대상 리소스를 나타내는 데이터를 대체합니다.
    - PUT과 POST의 차이는 멍등성으로, PUT은 멍등성을 가집니다. PUT은 한 번을 보내도, 여러 번을 연속으로 보내도 같은 효과를 보입니다. 즉, 부수 효과가 없습니다
    - file system에선 path라는 identifier로 file을 생성해야합니다. 이는 멍등성이 필요한 부분이기때 생성을 POST 대신 PUT으로 구현했습니다.
  - `ls` : GET, URI = `"/"` request\_body = `""`
    - `ls`는 DB에 있는 file list를 Server로부터 받아와 출력하는 명령어입니다.

- MDN에 따르면, HTTP GET method는 특정한 리소스를 가져오도록 요청하고, 데이터를 가져올 때만 사용해야 합니다. file list만 받아오므로 GET으로 구현했습니다.
- Request시 URI field는 "/" root 입니다.
- more : GET, URI = "/file\_name", request\_body = ""
  - more는 DB에 있는 file의 내용을 Server로부터 받아와 출력하는 명령어입니다.
  - file 내용만 받아오므로 GET으로 구현했습니다.
  - Request시 URI field는 "/file\_name"입니다.
- rm : DELETE, URI = "/file\_name", request\_body = ""
  - rm은 지정한 파일을 DB에서 삭제하는 명령어 입니다.
  - MDN에 따르면, HTTP DELETE 는 지정 자원을 삭제하는 method 이기에, 이로 구현했습니다.
- 잘못된 명령어를 입력했을 경우 : FOO, URI = "/file\_name", request\_body = ""
  - FOO는 구현상에서 임의로 정한 method입니다.
  - 이 요청을 보낼 시 400 Bad Request을 받습니다.
- **Main Block**
- **Description :**
  - Server와 소켓 통신을 통합합니다. Linux-like cmd을 통해 Server의 파일을 조작할 수 있습니다. HTTP/1.0 요청을 보내며, 응답을 받습니다.
- **In Detail :**
  - HOST, PORT는 127.0.0.1, 8080 상수 값으로 고정되어 있습니다. 이는 연결하려는 Server의 End Point을 가르킵니다. Loop Back 하는 구조입니다.
  - 아래 반복문을 돌며 진행됩니다.
    1. socket 통신을 시도합니다.
    2. accept 되었으면, 사용자 명령어를 기다립니다.
    3. 이를 HTTP 1.0 Request로 변환 후 Server에 send 합니다.
    4. Response을 받으면 이를 출력합니다.
    5. 이후 HTTP1.0 이기 때문에 TCP 연결을 종료합니다.
    6. 빠른 연결 요청이 실패 할 수 있기 때문에 2초간 sleep 합니다.

**server.py**

- **Help Function : put\_data(query, request\_body)**

- Description :

- request 부분에서 file 내용을 추출하고 'db' 디렉터리에서 파일을 업데이트하거나 생성하여 PUT 요청을 처리합니다.

- Parameter :

- query(str) : 쿼리 경로입니다.
- request\_body (str): JSON 형식의 콘텐츠가 포함된 HTTP 요청의 본문입니다

- Returns :

- status, response\_body: HTTP 상태와 업데이트되거나 생성된 콘텐츠를 포함하는 튜플입니다.

- In Detail:

- python 파일 표준 입출력을 이용하여 txt파일을 생성, 수정합니다.
- query가 나타내는 파일이 없어 생성할 경우에 status는 “201 Created”입니다.
- 존재하여 수정할 경우엔 status는 “200 OK”입니다.
- 그 외 Exception 시 status는 “500 Internal Sever Error” 입니다.

- **Help Function : delete\_data(query)**

- Description :

- 'db' 디렉터리의 쿼리 경로에 지정된 파일을 제거하여 DELETE 요청을 처리합니다.

- Parameter : query(str) : 쿼리 경로입니다.

- Returns :

- status, response\_body : HTTP 상태와 삭제된 파일의 내용(존재하는 경우)을 포함하는 튜플입니다.

- In Detail:

- python의 os 모듈을 이용하여 파일을 삭제합니다.
- 존재하여 정상적으로 삭제할 경우엔 status는 “200 OK”입니다.
- query가 나타내는 파일이 존재하지 않을 경우, status는 “404 Not Found” 입니다.
- 그 외 Exception 시 status는 “500 Internal Sever Error” 입니다.

- **Help Function : read\_data(query)**

- Description :

- 쿼리 경로로 지정된 파일에서 콘텐츠를 읽거나 'db' 디렉터리에 있는 파일을 나열하여 GET 요청을 처리합니다.
- **Parameter : query(str)** : 쿼리 경로입니다.
- **Returns :**
  - **status, response\_body** : HTTP 상태와 요청에 따른 콘텐츠 또는 파일 목록을 포함하는 튜플입니다.
- **In Detail:**
  - query가 "/", root일 시에는 response\_body는 날짜순으로 정렬된 file\_list 정보입니다.
  - query가 "/test1"와 같이 특정 파일을 나타낸다면, response\_body는 파일내의 내용입니다.
  - 둘 다 정상작동했으면 "200 OK" 그렇지 않은 경우 "500 Internal Sever Error" 입니다.
  
- **Key Function : handle\_request(request)**
- **Description :**
  - HTTP 요청에서 메서드, 쿼리 경로 및 요청 본문을 추출하고 적절한 메서드 핸들러에 위임합니다.
- **Parameter : request(str)**: 클라이언트로부터 받은 HTTP 요청입니다.
- **Returns : str**: 형식화된 HTTP 응답입니다.
  
- **Main Block**
- **Description:**
  - 서버를 구성하고, 들어오는 연결을 수신하고, 요청을 수신하고, 처리하고, 응답을 다시 보냅니다.
- **Note:**
  - 서버는 무한정 실행되며 루프에서 한 번에 하나의 요청을 처리합니다.
  - Client에 request 따라 Handler function을 사용하고, 잘못된 request을 보낼 경우, "400 Bad Request" status을 전송합니다.

## 시나리오 패킷 간단 분석

## Client command -Server Response 시나리오.

| Client command               | Response        | 발생하는 일, 부가 설명      |
|------------------------------|-----------------|--------------------|
| ls                           | 200 OK          | DB 파일 내역 보여줌       |
| dir                          | 400 Bad request | 구현되지 않은 명령어.       |
| more sunwoo                  | 200 OK          | sunwoo 내용을 보여줌.    |
| echo "new file" > test2      | 201 Created     | db에 파일 test2 파일 생성 |
| echo "hi I'm Sunwoo" > test1 | 200 OK          | tes1 파일 내용 수정      |
| rm test1                     | 200 OK          | test1 파일 삭제        |
| ls                           | 200 OK          | DB 파일 내역 보여줌       |
| rm test18                    | 404 Not Found   | DB에 없는 파일을 지우려 함.  |

## Common Things

- HTTP 1.0 방식으로 구현하였기 때문에, 매번 HTTP request- response할 때 new TCP connection을 시도합니다. 그리고 Client의 Port도 Fixed 되어있지 않기에, OS의 할당에 따라 Dynamic합니다.
- Seq, Ack bit는 HTTP request에 따라 Random한 것 처럼 보입니다.
- 3-way handshaking, 4-way hand shaking에서 흥미로운 점은, Ack가 Client와 Server가 서로 주고받을 때마다. 1씩만 증가한다는 점입니다.
  - 4-way handshaking을 살펴보겠습니다. Server가 Client 요청에 대해 응답 할 때 (Seq = 74, Ack= 33)을 지닌 Segment로 전송합니다.
  - 한번의 HTTP 통신을 했으므로, Server에서는 Client에 Close요청을 보내는데, (Seq = 74, Ack = 34)을 지닌 Segment로 전송합니다. ACK가 1이 증가된 모습입니다.
  - Client에서는 (Seq = 34, Ack = 75)을 지닌 ACK 응답을 합니다.
  - 이는 1인 FIN bit를 Ack에 더하는 scheme 을 사용했음을 알 수 있습니다.

시나리오 마다 Wire-shark Packet 분석 캡처본



## 1. ls

Wireshark - Follow TCP Stream (tcp.stream eq 0) - paket\_analisis.pcapng

GET / HTTP/1.0  
Host: 127.0.0.1:8080

HTTP/1.1 200 OK

Wed Jan 10 21:54:08 2024 test1.txt  
Wed Jan 10 21:54:29 2024 sunwoo.txt

## 2. dir

Wireshark - Follow TCP Stream (tcp.stream eq 1) - paket\_analisis.pcapng

FOO / HTTP/1.0  
Host: 127.0.0.1:8080

HTTP/1.1 400 Bad Request

## 3. more sunwoo

Wireshark - Follow TCP Stream (tcp.stream eq 2) - paket\_analisis.pcapng

GET /sunwoo HTTP/1.0  
Host: 127.0.0.1:8080

HTTP/1.1 200 OK

28283039  
Kim Sun Woo

## 4. echo "new file"&gt; test2

Wireshark - Follow TCP Stream (tcp.stream eq 3) - paket\_analisis.pcapng

PUT /test2 HTTP/1.0  
Host: 127.0.0.1:8080

["content": "new file"] HTTP/1.1 201 Created

new file

## 5. echo "hi I'm Sunwoo" &gt; test1

| No. | Time          | Source    | Destination | Protocol | Length | Info   |
|-----|---------------|-----------|-------------|----------|--------|--|
| 46  | 184.777750472 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 49102 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3631843544 TSecr=0 WS=128                             |
| 47  | 184.777750468 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 8080 → 49102 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3631843544 TSecr=3631843544 WS=128         |
| 48  | 184.777750470 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 49102 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3631843544 TSecr=3631843544   |
| 57  | 139.994416700 | 127.0.0.1 | 127.0.0.1   | HTTP     | 139    | PUT /test1 HTTP/1.0  |
| 58  | 139.994439993 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 8080 → 49102 [ACK] Seq=1 Ack=74 Win=65536 Len=0 TSval=3631878760 TSecr=3631878760  |
| 59  | 139.995731586 | 127.0.0.1 | 127.0.0.1   | TCP      | 98     | 8080 → 49102 [PSH, ACK] Seq=1 Ack=74 Win=65536 Len=32 TSval=3631878762 TSecr=3631878760 [TCP segment of a reassembled PDU] |
| 60  | 139.995769950 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 49102 → 8080 [ACK] Seq=74 Ack=33 Win=65536 Len=0 TSval=3631878762 TSecr=3631878762   |
| 61  | 139.995847189 | 127.0.0.1 | 127.0.0.1   | HTTP     | 66     | HTTP/1.1 200 OK  |
| 62  | 139.995920465 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 49102 → 8080 [FIN, ACK] Seq=74 Ack=34 Win=65536 Len=0 TSval=3631878762 TSecr=3631878762                                    |
| 63  | 139.995938983 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 8080 → 49102 [ACK] Seq=34 Ack=75 Win=65536 Len=0 TSval=3631878762 TSecr=3631878762   |

## 6. rm test1

| No. | Time          | Source    | Destination | Protocol | Length | Info   |
|-----|---------------|-----------|-------------|----------|--------|--|
| 64  | 141.997823405 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 50896 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3631880764 TSecr=0 WS=128                             |
| 65  | 141.997836987 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 8080 → 50896 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3631880764 TSecr=3631880764 WS=128         |
| 66  | 141.997848617 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 50896 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3631880764 TSecr=3631880764   |
| 71  | 167.171146905 | 127.0.0.1 | 127.0.0.1   | HTTP     | 114    | DELETE /test1 HTTP/1.0   |
| 72  | 167.171156564 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 8080 → 50896 [ACK] Seq=1 Ack=49 Win=65536 Len=0 TSval=3631905937 TSecr=3631905937  |
| 73  | 167.171732258 | 127.0.0.1 | 127.0.0.1   | TCP      | 102    | 8080 → 50896 [PSH, ACK] Seq=1 Ack=49 Win=65536 Len=36 TSval=3631905938 TSecr=3631905937 [TCP segment of a reassembled PDU] |
| 74  | 167.171744971 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 50896 → 8080 [ACK] Seq=49 Ack=37 Win=65536 Len=0 TSval=3631905938 TSecr=3631905938   |
| 75  | 167.171932206 | 127.0.0.1 | 127.0.0.1   | HTTP     | 66     | HTTP/1.1 200 OK  |
| 76  | 167.171897750 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 50896 → 8080 [FIN, ACK] Seq=49 Ack=38 Win=65536 Len=0 TSval=3631905938 TSecr=3631905938                                    |
| 77  | 167.171909281 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 8080 → 50896 [ACK] Seq=38 Ack=50 Win=65536 Len=0 TSval=3631905938 TSecr=3631905938   |

## 7. ls

| No. | Time          | Source    | Destination | Protocol | Length | Info   |
|-----|---------------|-----------|-------------|----------|--------|--|
| 78  | 169.173782700 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 43504 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3631907940 TSecr=0 WS=128                             |
| 79  | 169.173795123 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 8080 → 43504 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3631907940 TSecr=3631907940 WS=128         |
| 80  | 169.173806049 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 43504 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3631907940 TSecr=3631907940   |
| 81  | 189.89757513  | 127.0.0.1 | 127.0.0.1   | HTTP     | 106    | GET / HTTP/1.0   |
| 82  | 189.897580353 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 8080 → 43504 [ACK] Seq=1 Ack=41 Win=65536 Len=0 TSval=3631928663 TSecr=3631928663  |
| 83  | 189.89811160  | 127.0.0.1 | 127.0.0.1   | TCP      | 155    | 8080 → 43504 [PSH, ACK] Seq=1 Ack=41 Win=65536 Len=89 TSval=3631928664 TSecr=3631928663 [TCP segment of a reassembled PDU] |
| 84  | 189.898123278 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 43504 → 8080 [ACK] Seq=41 Ack=90 Win=65536 Len=0 TSval=3631928664 TSecr=3631928664   |
| 85  | 189.898148640 | 127.0.0.1 | 127.0.0.1   | HTTP     | 66     | HTTP/1.1 200 OK  |
| 86  | 189.898160271 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 43504 → 8080 [FIN, ACK] Seq=41 Ack=91 Win=65536 Len=0 TSval=3631928664 TSecr=3631928664                                    |
| 87  | 189.898196195 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 8080 → 43504 [ACK] Seq=91 Ack=42 Win=65536 Len=0 TSval=3631928664 TSecr=3631928664   |

## 8. rm test18

| No. | Time          | Source    | Destination | Protocol | Length | Info   |
|-----|---------------|-----------|-------------|----------|--------|--|
| 88  | 191.908881821 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 34282 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3631930666 TSecr=0 WS=128                             |
| 89  | 191.908891293 | 127.0.0.1 | 127.0.0.1   | TCP      | 74     | 8080 → 34282 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3631930666 TSecr=3631930666 WS=128         |
| 90  | 191.908932542 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 34282 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3631930666 TSecr=3631930666   |
| 91  | 213.069810943 | 127.0.0.1 | 127.0.0.1   | HTTP     | 115    | DELETE /test18 HTTP/1.0  |
| 92  | 213.069826134 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 8080 → 34282 [ACK] Seq=1 Ack=50 Win=65536 Len=0 TSval=3631951836 TSecr=3631951836  |
| 93  | 213.069945800 | 127.0.0.1 | 127.0.0.1   | TCP      | 92     | 8080 → 34282 [PSH, ACK] Seq=1 Ack=50 Win=65536 Len=26 TSval=3631951836 TSecr=3631951836 [TCP segment of a reassembled PDU] |
| 94  | 213.069951664 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 34282 → 8080 [ACK] Seq=50 Ack=27 Win=65536 Len=0 TSval=3631951836 TSecr=3631951836   |
| 95  | 213.069969559 | 127.0.0.1 | 127.0.0.1   | HTTP     | 66     | HTTP/1.1 404 Not Found   |
| 96  | 213.069982195 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 34282 → 8080 [FIN, ACK] Seq=50 Ack=28 Win=65536 Len=0 TSval=3631951836 TSecr=3631951836                                    |
| 97  | 213.069987788 | 127.0.0.1 | 127.0.0.1   | TCP      | 66     | 8080 → 34282 [ACK] Seq=28 Ack=51 Win=65536 Len=0 TSval=3631951836 TSecr=3631951836   |

Furthermore - 개선 및 더 생각할 수 있는 것

## TCP connection

- 해당 프로젝트에서는 Local Host을 사용하여 Port구분 만으로, Client, Server를 구분했다. 이는 엄밀하게는 Process간의 Protocol을 사용해도 프로젝트는 작동했을 것이다.

- 보다 TCP을 사용하기 위해선, Client Server의 Device구분이 필요하다. 유선랜을 활용하는 경우에는 Raspberry-Pi, Laptop 2개의 Device을 구비하여 Connection을 진행한다. 무선랜을 활용하는 경우에는 내 외부의 IP주소가 다르기에, Forwarding 등의 작업이 추가적으로 필요해 보인다.

## HTTP method

- 해당 프로젝트에선, HTTP 1.0을 사용하였다. 그 이유는 TCP connection을 유지하며 HTTP 통신을 하는데 실패했기 때문이다. TCP connection을 유지한다면 HTTP 1.1을 사용할 수 있을 것이다.
- Single Client로 구현하였는데 Multi Clients라면, HTTP 2.0 프로토콜과 Server단에서 Round Robin, Thread를 고려해야한다.

## CLI to GUI or WEB

- CLI을 사용했는데, HTML문서 등을 이용해 프로젝트를 진행했으면 하는 아쉬움이 있다. 그 이유는 다음과 같다.

### 1. POST 사용:

- POST는 HTTP method 중 가장 대표적인 것 중 하나이다. FILE system에서의 비역등성, POST를 사용할 때 쓰는 특징을 잘 표현할 수 없었다. 따라서 Create할때도 PUT을 사용하였다. 만약 HTML문서를 사용했다면, TODO list 처럼 중복된 것들(할일들, 예를 들면 div tag) 생성을 보이기 더욱 용이했을 것이다.

### 2. PATCH 사용:

- HTTP method 중 Update 하는 기능이 PUT 말고도 PATCH라는 method가 있다. 이 둘은 다음의 차이점을 갖는다.
- PUT은 client에서 모든 정보를 알고있다. Server에 있는 자원을 client request 자원으로 전체 수정한다. 그에 반해 PATCH는 client request body에 있는 정보를 가지고, server 자원 부분 수정한다.
- 위에서 들었던 ToDoList를 예로 들어보자. Client 단에서 request\_body를 { 'name' : '장보기', 'content' : '수박 1, 메론1'} 처럼 보내 전체 Page을 수정하는게 아니라, 하나의 할 일만 수정 가능 할 것이다.

## Implementation

- Cookie, Session 등 적용으로 301 Redirected 등의 status 받기 :
  - 본래 프로젝트 구상시 이름을 변경하는 method, 'mv'도 계획했다. 예를 들어 Client가 'origin\_name'의 파일 이름을 'modified\_name'로 바꾼다. 이후 GET 등의 method, URI = 'origin\_name' 로 접근한다면, Server단에서, '301 Redirected' status를 반환하여 접근하게 해준다.
  - 이는 HTTP의 stateless와 관련되어 있는 문제이다. Client의 Cookie나 Server의 Session을 구현해야하는데 이에 성공하지 못하였다.

## Reference

### DOCS

- Python STL Docs, socket, <https://docs.python.org/ko/3/library/socket.html>
- MDN WEB Docs, HTTP method and status, <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET>

### Blog Post

- 꾸꾸웁스 storage, [Linux] 리눅스 echo 명령어, echo 옵션 및 사용법, <https://rhrhth23.tistory.com/21>
- happy\_tiger.log, CS-HTTP의-모든것, <https://velog.io/@tiger/CS-HTTP%EC%9D%98-%EB%AA%A8%EB%93%A0%EA%B2%83>
- Studio u by kingjakeu, RESTful API POST와 PUT 차이, <https://velog.io/@tiger/CS-HTTP%EC%9D%98-%EB%AA%A8%EB%93%A0%EA%B2%83>
- Techoble, 자원을 수정하는 HTTP 메서드 - PUT vs PATCH
- Codzaram, [Python] socket 통신으로 채팅하기, <https://codezaram.tistory.com/31>
- ok-lab, [python] os, os.path로 파이썬 경로 다루기, [https://ok-lab.tistory.com/163#os.path.isfile,\\_os.path.isdir](https://ok-lab.tistory.com/163#os.path.isfile,_os.path.isdir)

### Dev Community

- stack overflow, How do you get a directory listing sorted by creation date in python?,  
<https://stackoverflow.com/questions/168409/how-do-you-get-a-directory-listing-sorted-by-creation-date-in-python>
-