

C언어를 이용한 웹서버 구현

컴퓨터 네트워크 과제

2020.06.05

소프트웨어학부

2016003654 안채령

목차

1. 개요

2. Part 1 웹서버

2-1. 서버 디자인에 대한 대략적인 설명 및 도식

2-2. 출력된 Client의 request message 스크린샷

2-3. Request message 와 각 field 값의 의미 해석(RFC 1945)

3. Part 2 웹서버

3-1. 서버 디자인에 대한 대략적인 설명 및 도식

3-2. 출력된 Client의 request message 스크린샷

3-3. Request message 와 각 field 값의 의미 해석(RFC 1945)

4. 구현 시 어려웠던 점

5. 클라이언트와 서버 동작

1. 개요

본 레포트는 '컴퓨터 네트워크' 수업의 과제로, C 언어를 이용한 웹서버를 구현에 대해 작성되었다. 과제의 목표는 다음과 같은 두가지 파트의 웹서버를 완성하는 것이다.

Part1) Client의 request message를 화면에 출력하는 Web server 제작

Part2) Server가 browser의 request에 response할 수 있도록 확장한 웹서버

세부적으로 각 Part1과 Part2의 웹서버에 대해서 설명할 것들은 다음과 같다.

- ➔ 서버 디자인에 대한 대략적인 설명 및 도식
- ➔ 출력된 Client의 request message 스크린샷
- ➔ Request message 와 각 field 값의 의미 해석(RFC 1945)

이후, 서버 구현 시 어려웠던 점과 해결 방법에 대한 설명과 함께, 본인의 컴퓨터에서 작동한 예시 스크린샷이 가장 마지막에 첨부 되어있다.

2. Part 1 웹서버

2-1. 서버 디자인에 대한 대략적인 설명 및 도식

TCP 소켓 통신을 위해, 웹서버는 `socket()`, `bind()`, `listen()`, `accept()` 등 과 같은 socket API 시스템 콜들을 이용해 만들어질 수 있고, 다음과 같은 과정을 통해 생성된다.

① 소켓 생성

```
sockfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)
```

이때, 각 인자는 인터넷 프로토콜 체계(IPv4), TCP 통신방식, TCP 프로토콜이며 이를 따르는 소켓을 생성하여 `sockfd(file descriptor)`로 반환한다.

이때, Internet address 를 담을 수 있는 `server_addr`, `cli_addr` 도 정의한다.

(Internet Address 를 담을 수 있는 구조체 – `cli_addr` 와 `serv_addr`)

```
struct sockaddr_in serv_addr, cli_addr;
```

`sockaddr_in` 구조체를 이용하여 주소체계, IP address, Port # 등의 정보를 할당해줌. 이때, IP address 와 port #는 network byte order 를 사용하고, 서버의 IP address 를 자동으로 찾아주기 위해 `INADDR_ANY` 를 사용함.

② 소켓을 Identify

```
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
    error("ERROR on binding");
```

Bind 시스템 콜을 이용하여 IP 와 port # 를 할당해줌

③ Listen(): 연결을 위해 기다림

```
listen(sockfd, 5);
```

연결 요청 대기 큐(5개까지 가능)를 이용해 연결을 기다리는 과정

④ Accept(): 큐의 연결을 가져와서 새로운 소켓 생성

```
if((newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &cli  
ilen))<0)  
{  
    perror("accept");  
    exit(EXIT_FAILURE);  
}
```

⑤ Read(): 클라이언트의 요청(newsockfd)을 읽어옴

```
n = read(newsockfd, buffer ,511);
```

이때, 요청 메시지는 buffer 에 저장하고, 읽은 요청을 프린트함으로써 어떤 메시지가 왔는지 terminal 에서 확인할 수 있게 된다.

⑥ Write(): Socket 에 전달하려는 메시지를 씀

```
write(newsockfd, "Hello" , 512);
```

⑦ Close() : 각 소켓을 닫음

```
close(sockfd);  
close(newsockfd);
```

이때 클라이언트의 입장에서선 연결을 요청하고, 먼저 연결 대기 큐에 들어가게 된다.(listen()) 이후 서버가 큐의 연결을 가져와서 새로운 소켓을 생성(accept())하고, 클라이언트가 보내는 요청을 서버가 읽을 수 있게 된다(read()). Accept()하고 read()하고, write() 하는 과정동안 서버는 계속 클라이언트를 기다리는 상태, 즉 blocking 된 상태이다.

2-2. 출력된 Client의 request message 스크린샷

```
tanya@tanya-UX410UQK:~/Desktop/ClientServer_Example$ ./server 8080
Here is the message: GET / HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Enc
```

2-3. Request message 와 각 field 값의 의미 해석(RFC 1945)

- 위의 스크린샷에서 "Here is the message : " 이하의 내용들이 Client 의 request message 이다. RFC 1945 문서를 참조해 해석한 Request Message 의 내용은 다음과 같다.

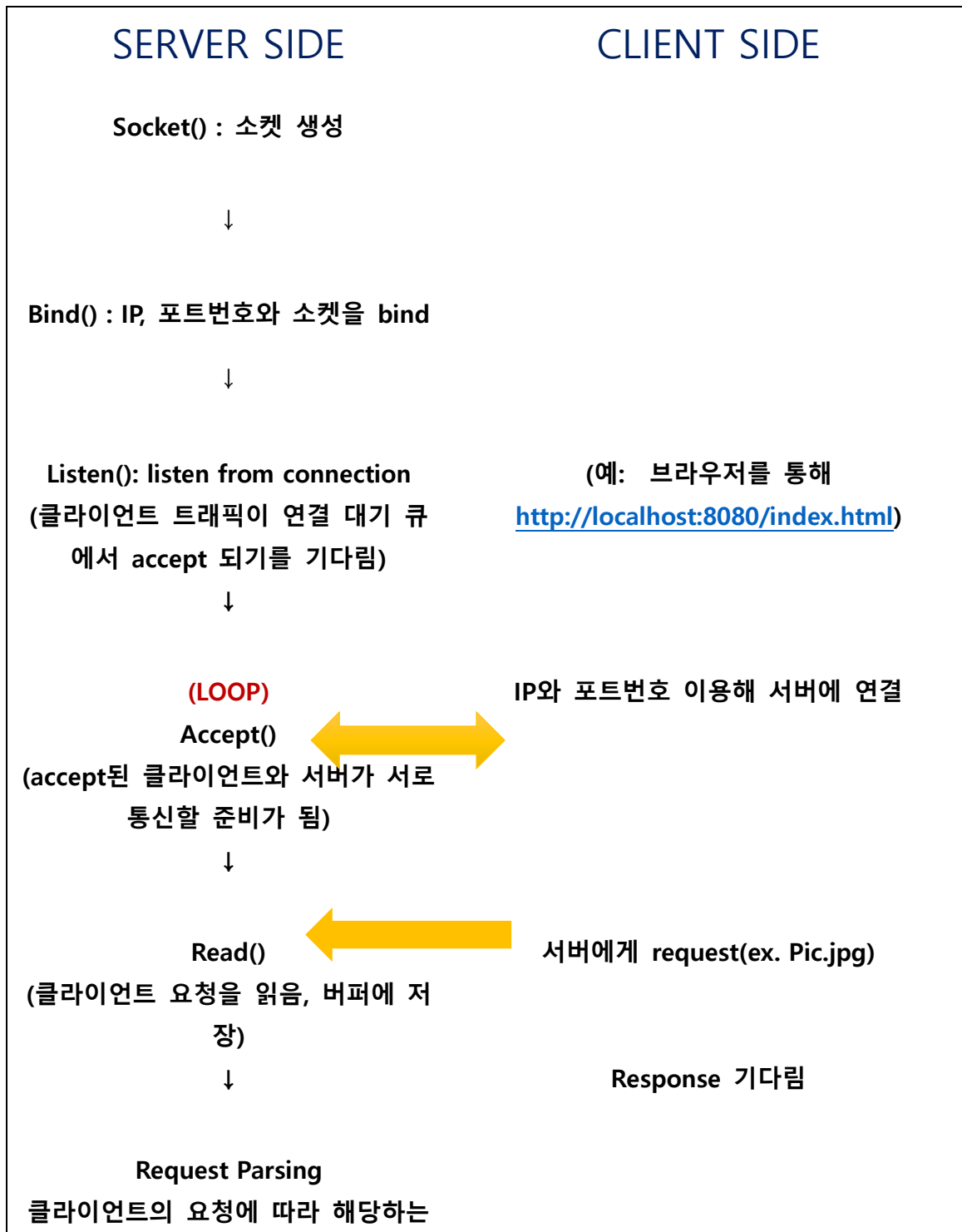
- ① GET /: GET 메소드는 Request URI(URI : 통합자원식별자) 로부터 식별된 정보를 되돌려주는 것이다. 다시 말해, 서버로부터 정보를 조회하기 위한 메소드이고, GET 다음에 오는 것이 클라이언트가 서버로부터 요청하는 정보이다. 이를테면, GET /index.html 이 온다면 index.html 페이지를 클라이언트가 요청하는 것이다.
- ② HTTP/1.1: HTTP 1.1 프로토콜을 따르겠다는 이야기이다.
- ③ Host: localhost:8080 으로, 인터넷 호스트 도메인 이름 혹은 IP 주소가 적혀있다. 이 경우, 로컬호스트이며 포트번호는 8080이다.
- ④ User-Agent : 유저 에이전트는 사용자 대신에 일을 수행하는 소프트웨어 에이전트로, 각 브라우저별로 정보가 다르다. 이 경우, 파이어폭스 브라우저를 이용하였다.
- ⑤ Accept: 접근가능한(acceptable) 미디어들을 알려준다. 이 경우, text/html 등의 미디어 종류가 접근가능하다.
- ⑥ Accept-langauge : Accept 와 비슷한 역할의 헤더로, 요청에 대한

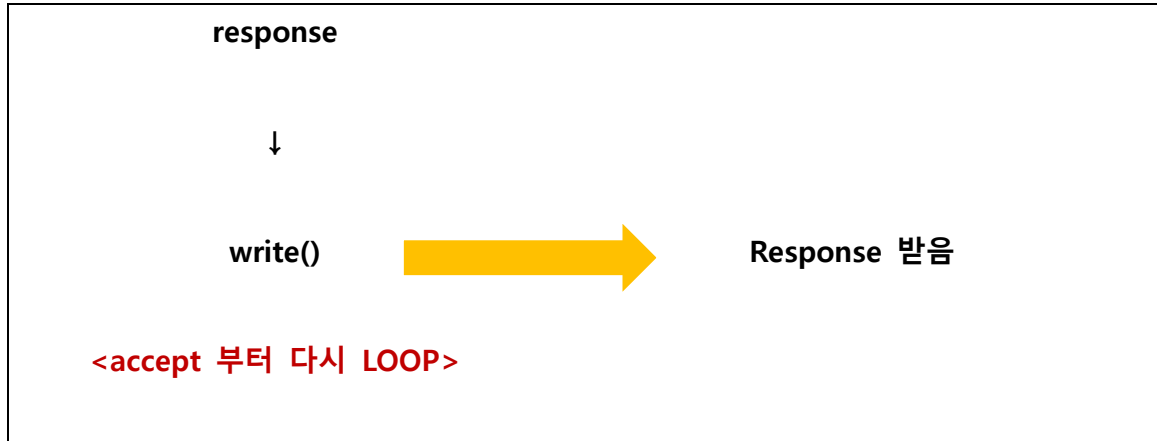
응답으로 선호되는 언어를 알려준다. 이 경우, English이다.

- ⑦ Accept-Enc: encoding 의 줄임말로, 어떤 content-codings 가 acceptable 한지 알려준다. No encoding 일 경우 identify token 이 같은 의미로 쓰이기도 한다.

3. Part 2 웹서버

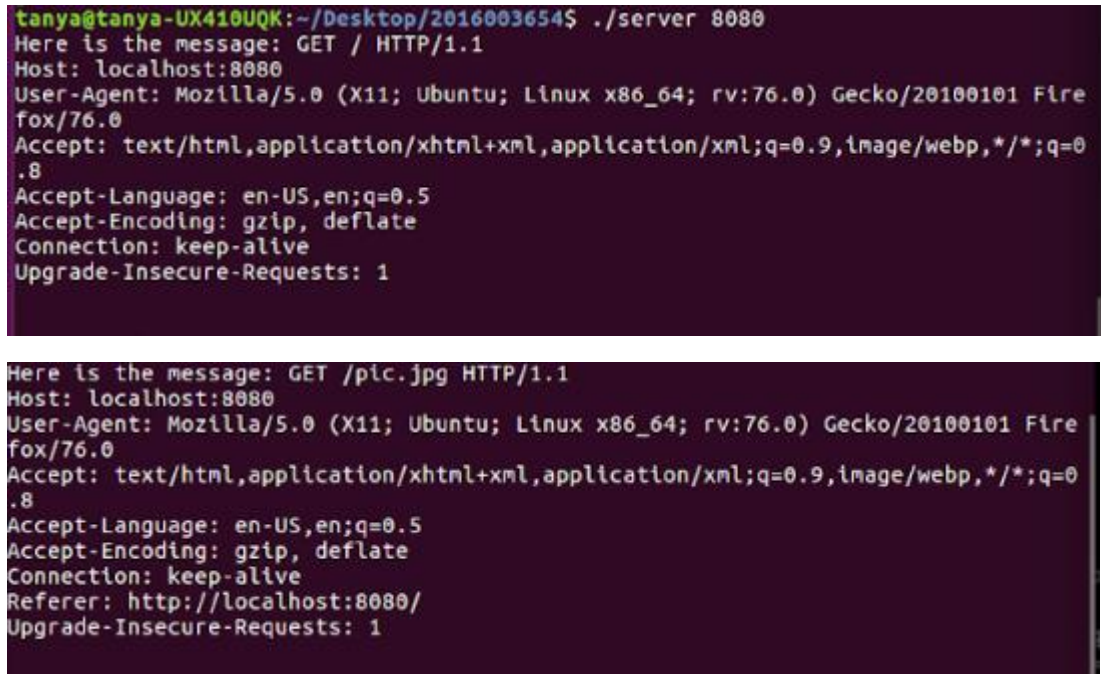
3-1. 서버 디자인에 대한 대략적인 설명 및 도식





- strcmp 를 사용하여 클라이언트의 요청을 파싱하였다. 클라이언트의 요청버퍼에서
장하고, "GET /index.html", "GET /music.mp3", "GET /pic.jpg" 등 버퍼에 저장된 글
자를 비교하여 해당하는 파일을 열고, 읽은 후 newsockfd 에 write 하여 웹
페이지에 프린트할 수 있도록 하였다. 이때, 디폴트로 index.html 페이지가 연결
되도록 하였다.
- 각각의 링크에 대해 매번 새로 HTTP connection을 하는게 아닌, 단일한 TCP
connection을 가지고 여러 번의 링크에 대해 요청과 응답이 가능하도록
Persistent HTTP 로 만들었다. 이를 위해 while 문을 만들고, while 문이 끝나고 나
서 모든 소켓을 닫았다.
- 각 요청에 해당되는 파일을 가져올 때는 fcntl.h 헤더파일의 fopen, fread 등의 메소
드를 사용하였다.

3-2. 출력된 Client의 request message 스크린샷



```
tanya@tanya-UX410UQK:~/Desktop/2016003654$ ./server 8080
Here is the message: GET / HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1

Here is the message: GET /pic.jpg HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://localhost:8080/
Upgrade-Insecure-Requests: 1
```

3-3. Request message 와 각 field 값의 의미 해석(RFC 1945)

- Here is the message: 이하의 부분이 request message로, part 1에서 봤던 "GET /" 부터 "Accept-Encoding" 의 부분은 동일하여 part 2 에서는 설명을 생략하고, 이후 Connection 부분부터 보면 다음과 같은 내용들이 있다. 이때, part1 에서 설명한 accept-encoding 부분까지는 HTTP header 중에서도 Request Headers에, Connection 과 Upgrade-Insecure-Request 는 General Headers에, 그리고 나머지 Content-type, Content-length 등은 Entity headers 에 속한다. Entity Headers 는 Body 의 내용을 담고 있다.

① Connection: Keep-Alive

위의 도식에서 볼 수 있듯이, Accept 부터 Read(), Request Parsing, Write() 후 다시 Accept 로 루프를 도는 것을 알 수 있다.

② Upgrade-Insecure-Request:

Upgrade-Insecure-Requests 가 1이라는 것은 사이트의 secure 버전에 리다이렉트할 수 있음을 뜻한다.

③ Accept-encoding

앞서 Part1에서 accept encoding의 내용에 아무것도 없었던 것에 비해, 여기서는 gzip 형태로 인코딩되어 리소스가 저장됨을 뜻한다.

4. 구현 시 어려웠던 점

- 연결이 성사되고 난 후부터 계속 링크에 대한 요청이 오면 받을 수 있게 만들어야 되기 때문에 루프를 돌 게 만들어야 했는데, while문을 만들어 루프를 만드니 크롬에서는 잘 동작하였지만 firefox에서는 아예 페이지가 뜨지 않다가 모든 연결을 종료시키고 나서야 뜨는 문제가 발생했다. 이 문제는 아래와 같이 while 문의 마지막에 빈 버퍼를 써줌으로서 (빈 페이지) 해결하였다.

```
bzero(buffer, 512);  
write(newsockfd, buffer, 512);
```

- 처음 jpg 파일 등 다른 형식의 파일들을 html을 통해 가지고 와야한다고 생각했다. 그래서 헤더의 Content-type을 text/html으로 만들려고 했는데, 빈 html 페이지를 샘플로 만들어서 코드를 쓴 것은 잘 돌아가는 반면, 소켓에 write하는 이 서버코드 위에서는 원하는 대로 이미지 등이 잘 뜨지 않았다. 결국 다시 과제 설명 영상을 들으면서 교수님의 시연영상을 보니, .html 로 파일을 가져오는 것이 아닌 .jpg, .pdf, .gif, .mp3 의 형식으로 가지고 온다는 것을 알았다. 그래서 그대로 파일을 오픈해주는 fcntl의 메소드들을 써서 newsockfd에 write해주니 잘 동작하였다. 위의 파일을 열고 쓰는 동작에서도 처음 바이너리로 열지 않아 제대로 써지지 않는 문제가 발생했었다. 이것 역시 바이너리로 바꿔주니 잘 동작하였다.

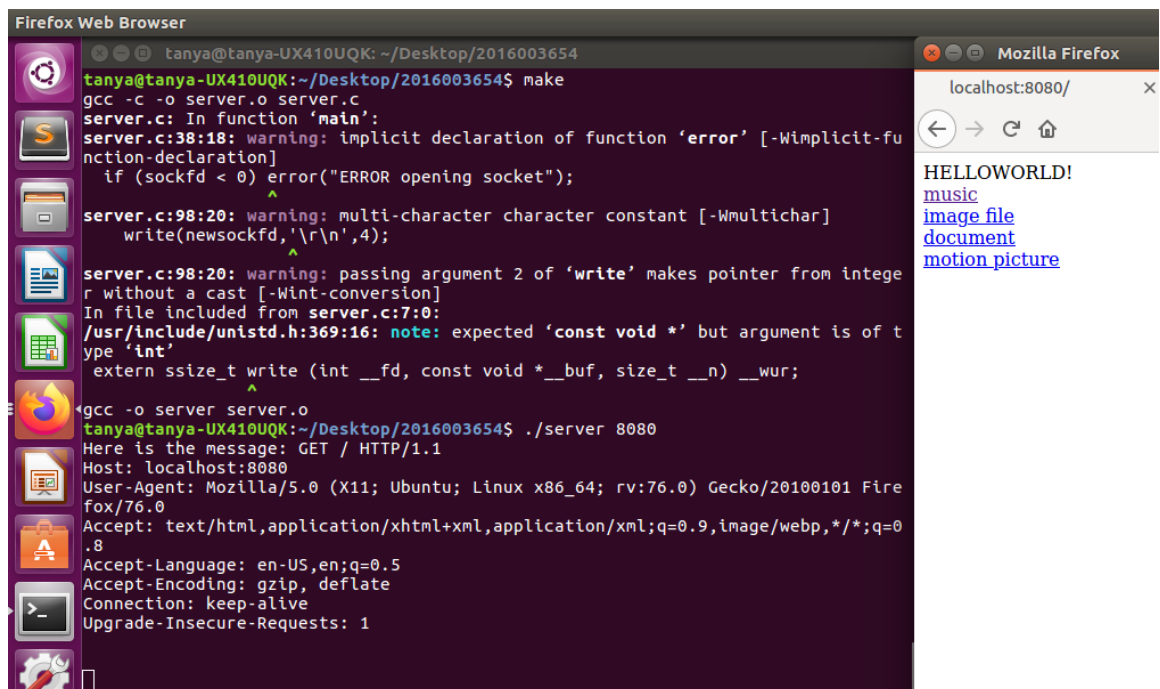
- 요청받은 파일(ex. .jpg)에 대해 write할 때, 한번만 소켓에 write 해야한다고 생각하여 모든 헤더와 바디를 한번에 쓴 상태로 보낼려고 하였다. 그러나 잘 되지 않았고, write을 여러 번 쓰는 것으로(Content-type 에 대한 헤더,

Content-length에 대한 헤더, 그리고 파일내용이 들어가있는 바디) 그 문제를 해결하였다.

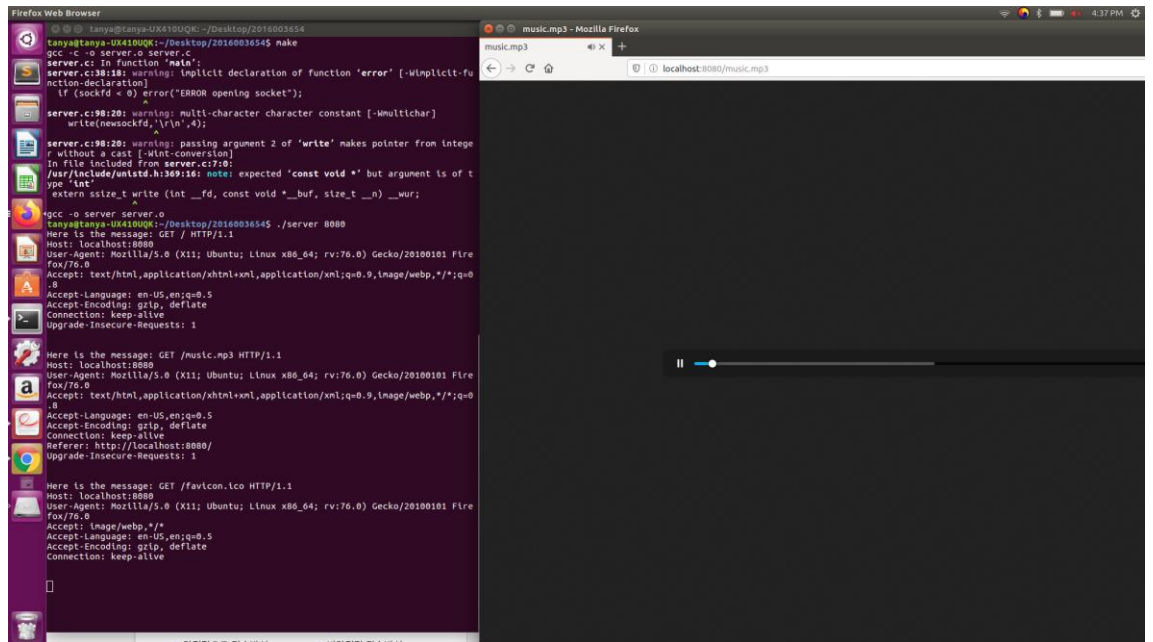
- 나머지 부분들에 대해선 교수님이 처음 알려주신 예제 TCP 코드가 있었기 때문에 크게 어려운 부분이 없었다.

6. 클라이언트와 서버 동작(스크린샷)

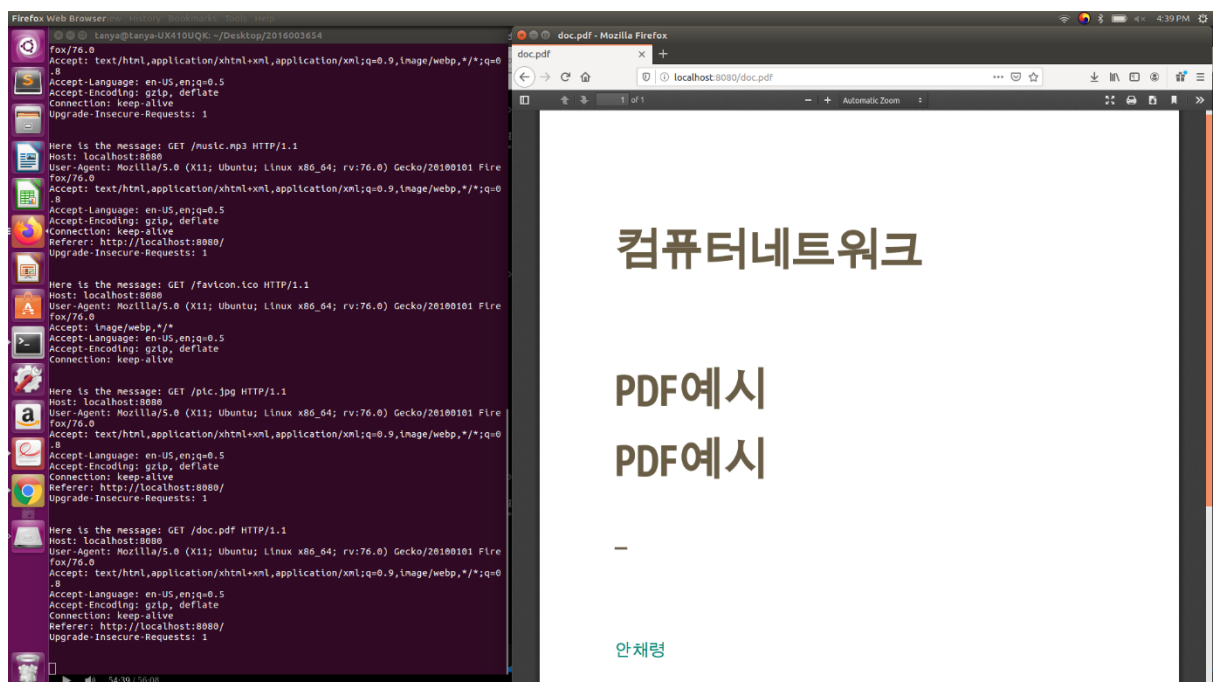
(1) Make 로 컴파일 및 8080포트로 서버 실행 → 브라우저에서 localhost:8080



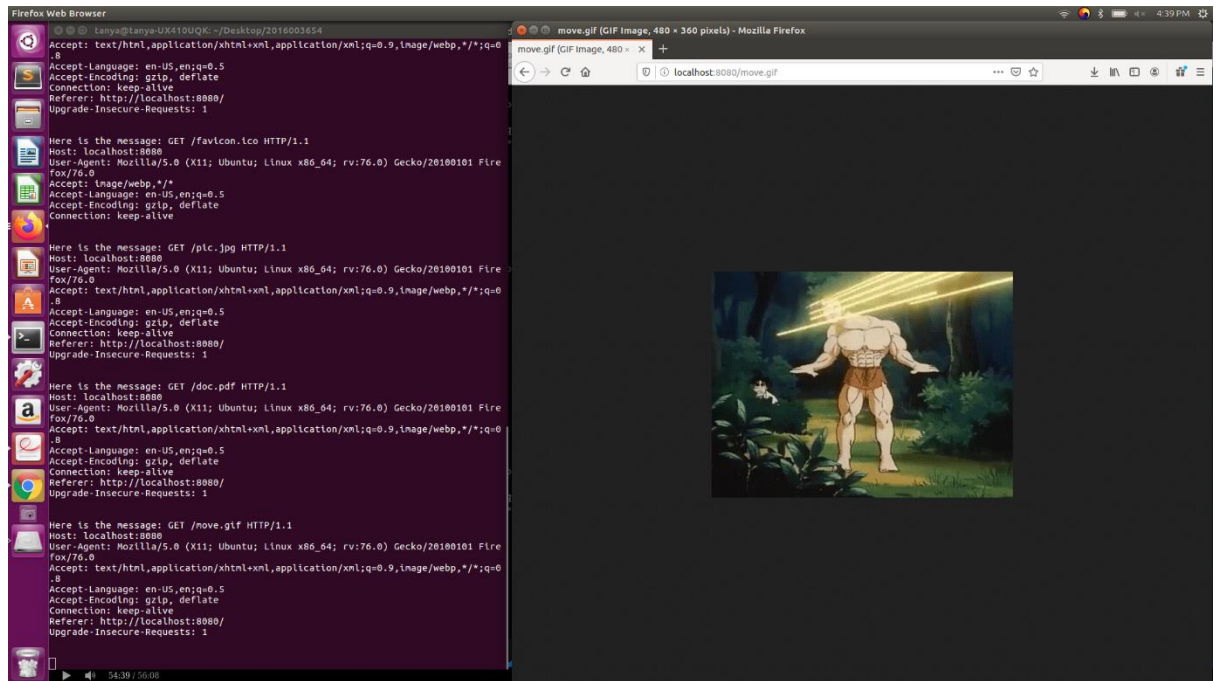
(2) Music.mp3 실행화면



(3) PDF 실행화면



(4) GIF 실행화면



(5) Jpg 실행 화면

