-Description of my server's design

코드를 작성하기에 앞서서, 우분투 가상환경을 이용할 것이기 때문에, 터미널에 hostname -I 를 이용해 내 머신의 IP를 파악했다. 10.0.2.15 로, 후에 주소에 이 값이 들어가게 된다. 작성된 서버 코드는, 기본으로 주어진 뼈대 코드에서, 간단한 함수 세 가지와, main함수로 이루어져 있다. 기본으로 주어진 error함수, 파일의 길이를 알려줄 gettingLength 함수, 파일의 형식을 알려주는 gettingytype 함수가 있다.

Main 함수의 뼈대코드에 의하면, 포트를 만들고, socket을 만들었으며, accept함수를 이용해 소켓 연결까지 마친 상태이다. 뼈대코드 이후에, request message를 console로 dump 해주기 위해, request message를 받아와야 했다. 만들어져 있던 buffer에 read 함수를 이용해 request message를 담았다.

While문과 가상의 token변수, strtok() 함수를 이용해 path 변수안에 /를 포함한 파일명을 담았다. 파싱을 한 후에, client로 직접 HTTP response를 만들어 send해야 했다. 따라서, send에 사용할 int sender; 을 선언해 주었다. 또한, header부분을 만들어야 하므로, char* head 또한 선언하고 malloc함수를 이용해 메모리를 할당해 주었다. 크기는 buffer와 같은 256이다. 우선, head에 필수적으로 필요한 요소들이 뭐가 있을까 조사해 보았고, 결과적으로 요청이 성공했음을 나타내는 HTTP/1.1 200, 우리가 전달할 path 위치의 파일의 길이, 파일의 형식 3가지를 헤더에 추가하기로 하였다. head에는 strcat을 통해 문자열을 더하는 방식으로 만들어 주었다.

Gettinglength는 파일의 상태를 나타내는 stat 구조체를 이용해 길이를 쉽게 구할 수 있었다. Char *head에 값을 문자열 형식으로 더해야 했으므로, sprint()를 이용해 문자열형식으로 바꾸어 head에 더해주었다.

Gettingtype 함수는 단순히 path 파일명을 strstr 를 이용해 스캔하며, 해당 형식을 나타내는 문자가 있으면, 그 형식의 MIME 타입을 return 해주도록 하였다. HTML, jpeg, gif, pdf, mp3의 지원을 가능하게 만들었으나, pdf 형식에서만 오류가 나는 것을 확인하였다.

아까 사용했던 배열 buffer은 더 이상 필요가 없으므로, read함수를 이용해 buffer내의 남아있는 message 들을 모두 출력하고, bzero()를 이용해 buffer의 바이트 스트림들을 0으로 바꿔주었다. 그리고, 아까 만들어 두었던 head의 내용들을 buffer에 복사해주었다. FILE 타입의 *fp 를 선언해 read로 열고, sender을 이용해 초반에 만들어 두었던 newsockfd 소켓으로 buffer, 길이를 전달했다. 모든 작업이 끝난 뒤, while문을 이용해 buffer에 남은 정보들을 send로 소켓에 보내주었다. 모든 작업이 마무리된 뒤, 아까 열어

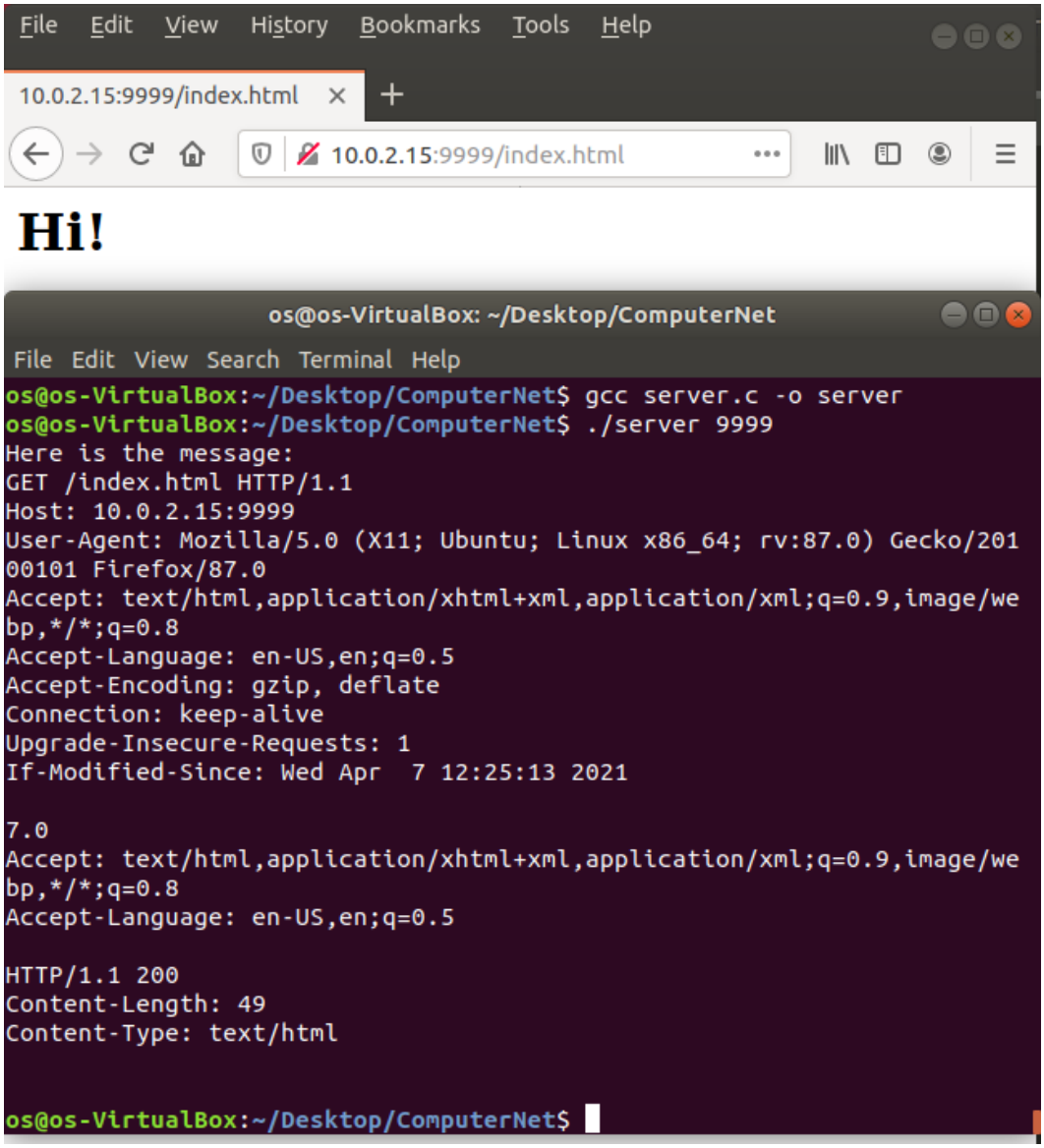
놓았던 fp, sockfd, newsockfd를 모두 닫아주었다.

-Difficulties I faced and how I solve them

어려움을 겪었던 부분이 많았다. 첫번째로는 gettinglength 함수 부분이다. 단순하게 길이를 구하는 방법으로 접근하려고 했으나, 파일이 그림이 담겨있는 경우에는 그 방식을 사용할 수 없었다. 따라서 파일의 상태를 알아볼 수 있는 함수를 검색해본 결과, stat 구조체와 fseek를 이용한 방법이었다. 직접 두개 다 구동 결과 stat 구조체를 이용한 방식이 낫다고 판단해서 사용하였다.

Header을 제작하는 부분에서 Header의 기본적인 형태를 이해하는데 어려움이 있었다. Mozilla 홈페이지와, 올바른 Header들의 예시를 많이 보면서 구조를 이해했다. Mozilla 홈페이지에서 HTTP의 기본적 형식, 위에서 언급되었던 MIME 타입의 예시 같은 것을 많이 참고하였다.

Buffer관련해서 어려움을 많이 겪었다. 이유를 확실히 알 수는 없지만, read함수로 buffer을 읽고 난 뒤에도, buffer에 내용이 계속 쌓이는 것 같았다. 실제로 코드가 완성되지 않았을 때에는, dump request를 확인하면 출력되다 중간에 끊긴 것과 같은 형태를 확인할 수 있었다. Read 함수로 중간중간 출력하는 것으로 개선했다.

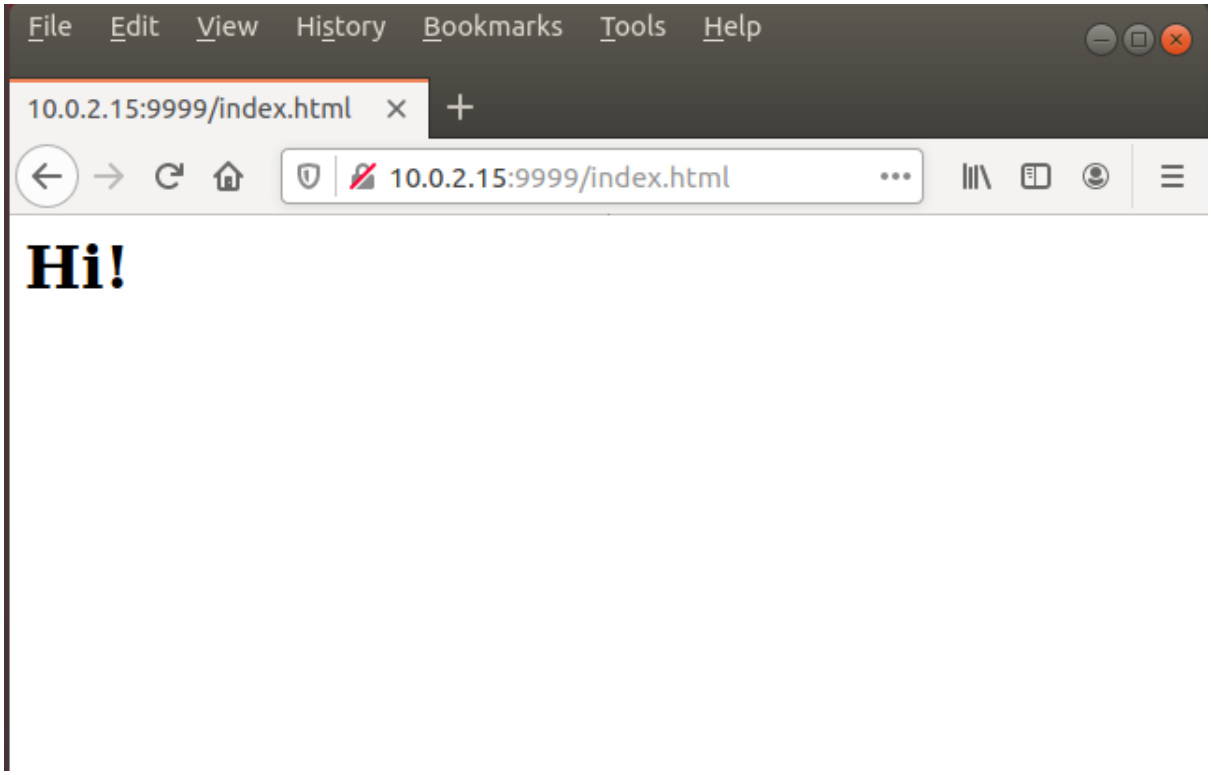


웹 브라우저를 이용해 index.html 을 요청한 모습이다. Here is the message 하단에 request message들이 dump 되었음을 확인 할 수 있다.

메시지들의 의미를 확인해보자.

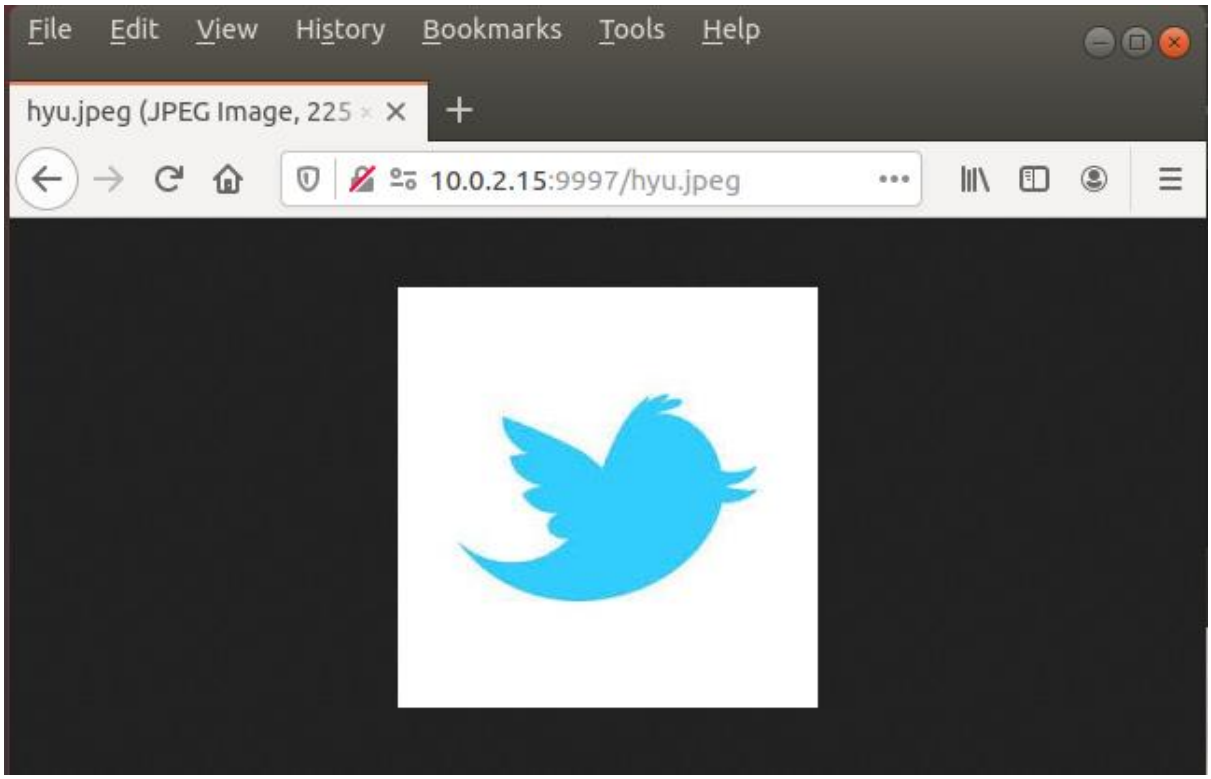
GET은 http 메소드 방식으로, 뒤쪽에 우리가 원하는 파일의 이름, 버전 등이 나와있다. Host는 말그대로 host의 주소, 포트가 나와있다. User-agent 에는 클라이언트 프로그램 정보가 담겨있다. Accept에는 클라이언트가 지원 가능한 타입의 종류들이 나와있다. */*는 모든 타입의 처리가 가능하단 뜻이라고 한다. 뒤에 따라오는 Accept-Language, Accept_Encoding 각각 지원 가능한 언어, 압축 포맷 등을 의미한다. Connection은 클라이언트와 서버의 연결 방식으로, keep-alive 상태가 디폴트이다. Upgrade- 는 프로토콜을 변경할 때 사용한다고 한다. If-Modified-Since 는 나와있는 시간 이후의 변경된 부분들을 얻어오는 헤더이다.

Html, jpeg, gif에 관한 실행 결과이다. Html 은 단순히 HI! 가 담겨있고, jpeg 에는 트위터의 사진이, gif는 캐릭터가 춤추고 있는 사진이다.

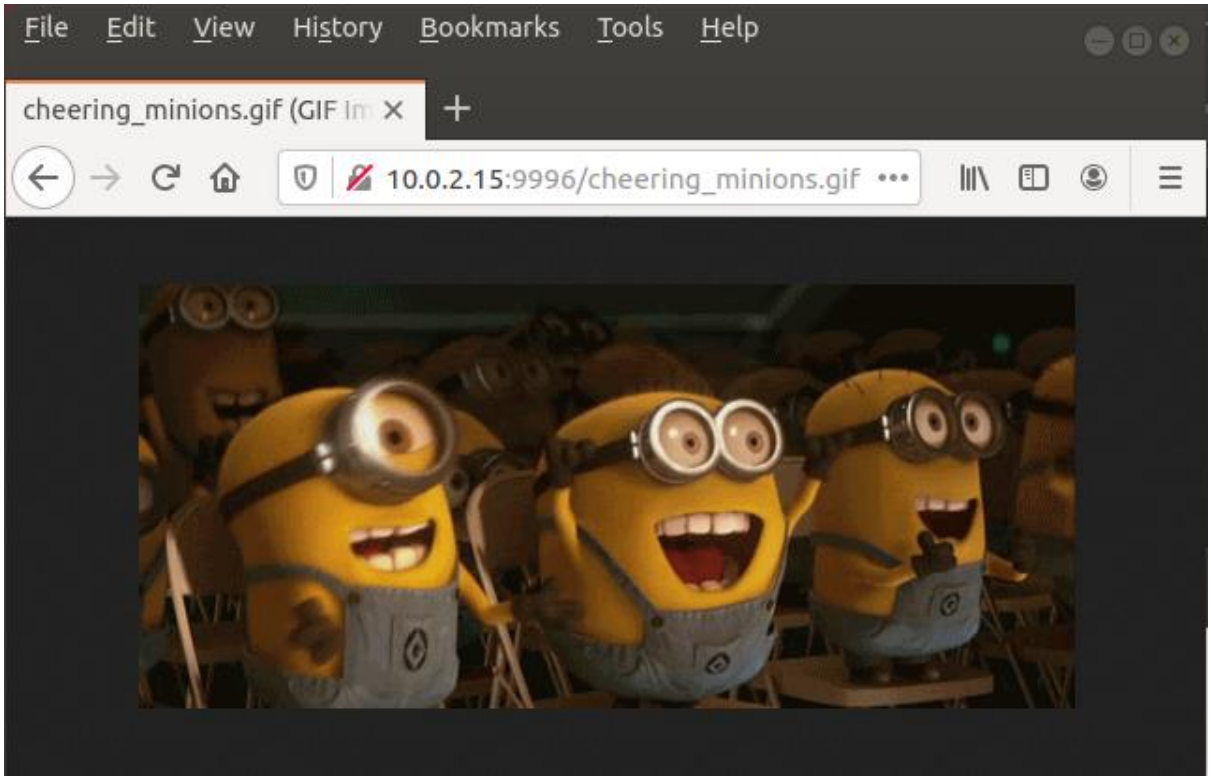


Port 9999에서 index.html 을 요청하였다.

Html 은 정상적으로 출력되었고, 직접 만든 헤더의 내용 또한 정상적으로 dump되었다.



Port 9997에서 hyu.jpeg 를 요청하였다. Hyu.jpeg 또한 정상적으로 출력되었다.



Port 9996에서 gif를 요청하였다.사진에서는 보이지 않지만, 이미지가 정상적으로 움직이는 것을 확인할 수 있다. 위에서 언급했듯, pdf 실행과정에서는 뷰어는 나타나지만, 내용이 보이지 않는 오류가 발생한다.