네트워크 프로젝트 #2 2013043337 컴퓨터공학과 이재현

* Give a high-level description of your server's design

<실행 과정>
1. 프로그램 동작에 필요한 버퍼 선언 및 정의
2. 웹페이지를 캐싱해온 데이터를 저장할 큐를 조작하는 함수가 들어있는 queue.h 파일 include
3. 캐싱할 데이터를 저장할 큐 선언 및 초기화
4. 프록시 -> 클라이언트, 클라이언트 -> 프록시를 각각 연결하는 2개의 socket descriptor (cli_sockfd, p_sockfd) 선언
5. 클라이언트가 서버에 전송한 request message를 저장할 버퍼 (c_request_buffer) 선언
6. p_sockfd의 IP주소, port 번호 설정, 바인딩, listen 함수 적용
7. 큐에 삽입할 노드 선언 및 초기화 (newNode)
8. cli_sockfd를 통해 클라이언트와 프록시를 소켓으로 연결
9. 클라이언트가 요청한 request message를 c_request_buffer에 저장
10. request message에서 end server host의 url정보와 클라이언트가 요청하는 object 이름 파싱
11. 파싱한 url 정보로 gethostbyname()함수를 사용하여 end server의 IP 주소를 받아옴
12. 클라이언트가 요청한 object의 이름으로 큐 검색
13. 큐에 요청한 object가 있으면 HIT로, 큐에 저장된 end server와 연결하지 않고 프록시에서 클라이언트로 object 바로 전송
14. 큐에 요청한 object가 없으면 MISS로, 프록시와 end server 연결
15. 프록시와 end server 연결할 socket descriptor (s_sockfd) 선언
16. 11번을 통해 알아낸 end server의 IP 주소, HTTP connection을 위한 port번호 80으로 s_sockfd의 IP주소, port 번호 설정, connect.
17. 프록시에서 end server로 request 메시지 전송
18. end server에서 프록시로 response message 전송
19. 정상적으로 object를 받아왔을 경우 로그 작성
20. 전송받은 response message를 소켓을 통해 클라이언트에게 전송
21. response message 중 object를 캐싱하기 위해 response message에서 response header 파싱
22. request message, object 파일, object 이름을 앞서 선언한 newNode에 저장
23. 큐의 남은 공간 및 저장되어 있는 object 갯수 고려하며 newNode를 큐에 삽입
24. 사용하지 않는 소켓은 닫아줌

<함수 설명>
void error(char* msg): 에러 발생시 에러 내용을 출력하고 프로세스 종료.

char* get_time(char* time_buffer): 로그에 저장할 현재 시각을 가져오기 위한 함수. "Tue 28 11 2017 23:25:04"의 형태로 시간 정보를 받아와 인자로 받은 time_buffer에 저장.

void write_log(char* log_buffer): 로그 파일을 쓰기 위한 함수. proxy.log파일이 없으면 생성하도록 하고, proxy.log파일을 열어 log_buffer에 담긴 내용을 쓴다.

int main(int argc, char* argv[]): 프로그램 메인 실행 내용

* What difficulties did you face and how did you solve them?

Part I: 저번 프로젝트에 소켓을 추가함으로써 별다른 어려움 없이 구현하였음.

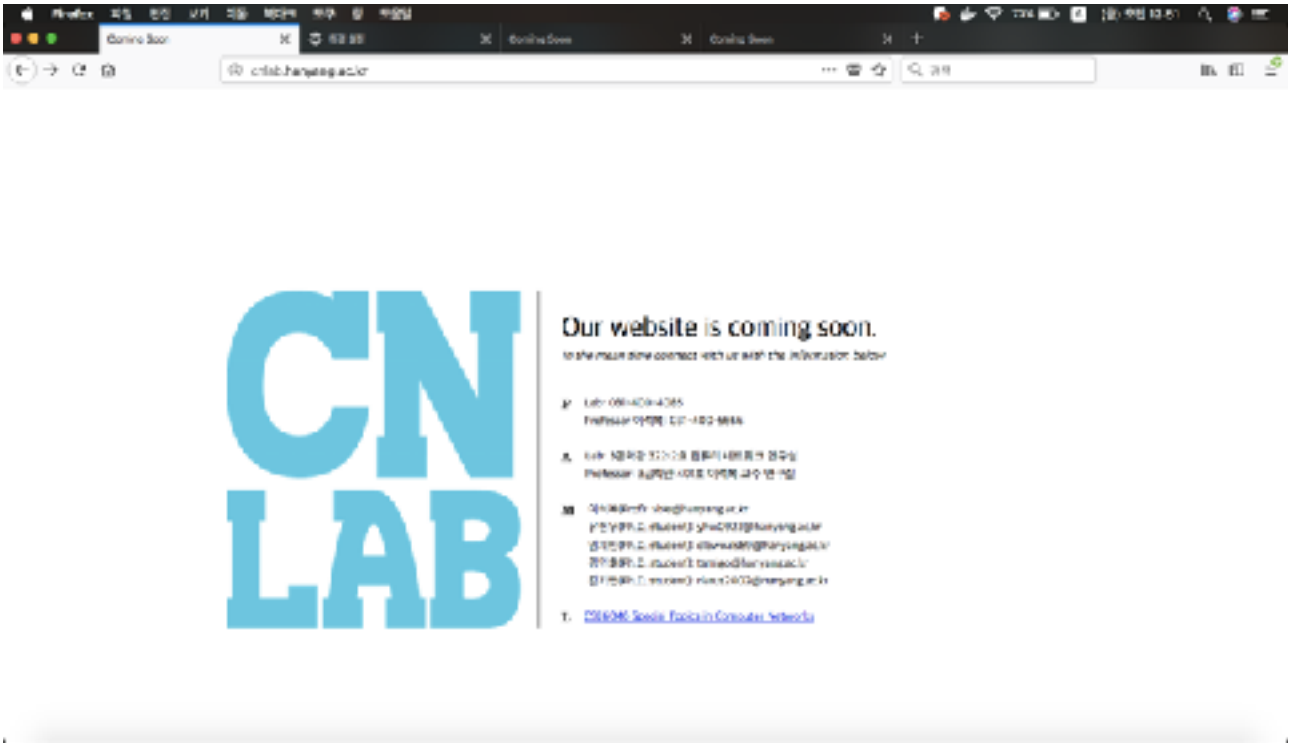
Part II: 처음에는 캐싱한 object를 별도의 binary 파일로 작성하여 이를 디스크에 저장하는 방식을 생각하였으나, 문제를 다시 이해한 결과 잘못된 생각이라 판단하여 프로그램 실행 중 메모리 상에서 object를 저장하는 방식으로 변경하여 구현하였다. 클라이언트에서 프록시로 간 request message 파싱은 이미 프로젝트 1에서 구현해 보았던 내용이기에 쉽게 구현할 수 있었으나, 이를 아무런 정제 과정 없이 프록시에서 end server로 보내주었더니 에러가 발생하였다. 구글링해본 결과 프록시에서 end server로 request message를 보내줄 때에는

```
GET http://cnlab.hanyang.ac.kr/ HTTP/1.1
Host: cnlab.hanyang.ac.kr
```

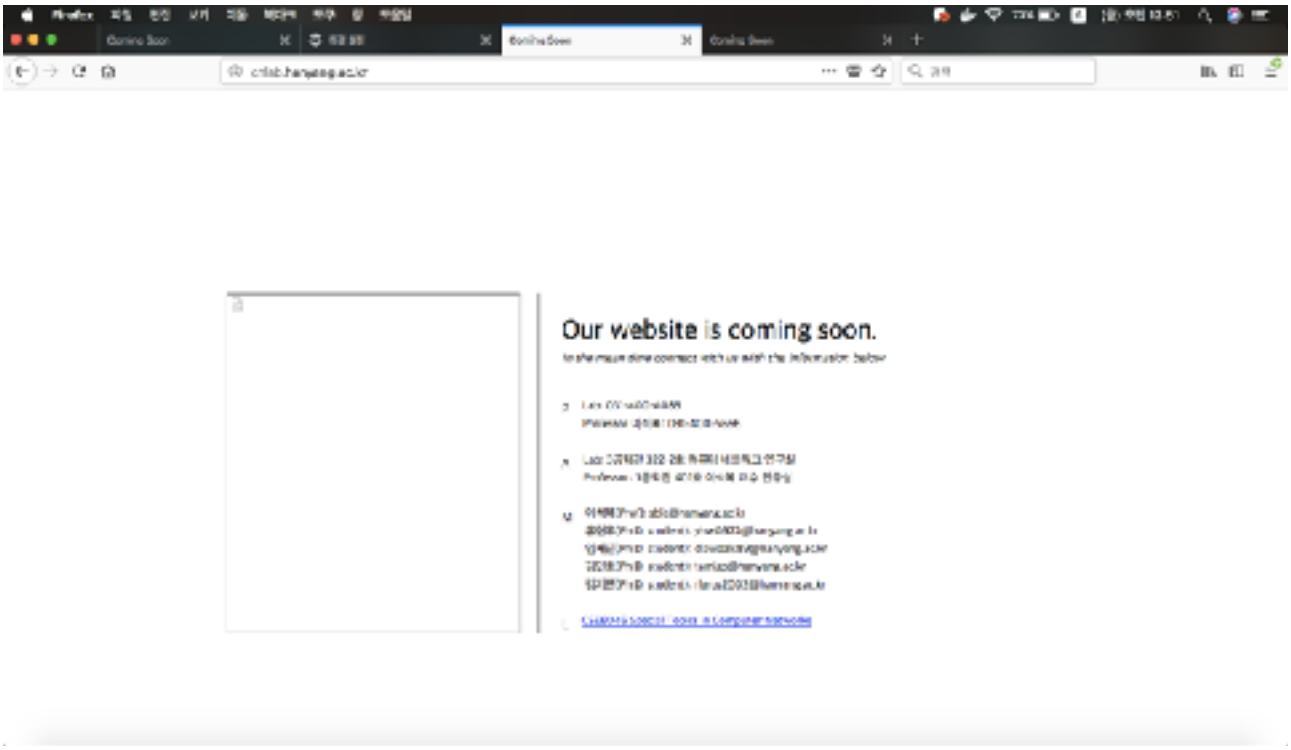
`Connection: close`
위와 같이 request message 중 저 세 줄만 보내줘야 한다는 것을 알게 되었고 문제를 해결하였다.
또 다른 문제는 end server에서 프록시가 받아온 response message를 클라이언트에 전달하는 과정에서 발생하였는데, read 함수가 전달해 준 인자의 바이트 수만큼 읽어오지 않고, 여러 차례에 걸쳐 나눠서 읽어온다는 것에서 발생한 문제였다. read 함수가 임의의 바이트 수만큼 끊어서 response message를 소켓에서 읽어오기 때문에 끊어서 전송받은 object를 클라이언트에게 전달해주기 위해 object 파일만을 위한 버퍼를 새로 만들었고, read 함수가 읽어온 바이트 수만큼 반복하여 버퍼에 저장하고 끊어진 내용은 이어 붙이도록 구현하였다.
이렇게 저장한 파일을 그대로 전송했을 때 또다른 문제가 발생하였는데, response message에 저장된 HTTP response header가 함께 전달되어 웹브라우저에서 제대로 화면에 띄우지 못하는 문제였다. 이를 해결하기 위해 response message 중 response header는 파싱하고 object 파일 내용만 저장하여 전달하도록 구현하였다.
마지막으로 발생한 문제는 이미지 파일을 캐싱하는 데에서 발생하였다. 여타 파일들과는 달리, 이미지 파일은 바이너리 파일로 읽었을 때 내용 중 공백이나 특수문자들이 많았고, 이러한 공백이나 특수 문자들로 인해 프록시에 캐싱할 때 저장이 제대로 되지 않았다. 이 문제를 해결하기 위해 여러 방법들을 찾아보았으나, 이 문제만큼은 해결하지 못하였다.

* 실행 결과 캡처

<페이지 첫 로드시>



<캐싱된 정보로 로드시(HIT 발생시)>



(위 사진과 같이, 이미지 파일은 제대로 저장되지 않아 불러오지 못한다는 점을 알 수 있음.)