M1522.000800 System Programming
Fall 2019

System Programming Proxy Lab Report

Jinje Han
2018-18786

1. Introduction

이번 랩의 목적은 소켓 통신을 기반으로 동작하는 서버를 localhost에서 구현하고, 브라우저와 서버 사이에서 작동하는 프록시 서버를 구현하는 것이다. 동일한 컴퓨터의 브라우저의 search bar에 localhost를 기반으로 하는 URL을 작성하면, 브라우저가 이를 request로 바꾸어 서버에 보낸다. 이에 반응하여 서버는 response를 돌려주고 브라우저는 이를 창에 GUI의 형태로 표시한다.

2. Important Concepts

네트워크 응용 프로그램은 클라이언트-서버 모델에 기초하고 있다. 클라이언트와 서버는 각각이 프로세스이며, 서로 간의 connection을 통해 byte들을 주고받는 형식으로 통신을 진행한다. 소켓은 이러한 connection의 양 끝점이다. 소켓은 각 호스트의 주소와 포트 번호를 포함한다. 통신은 고도로 추상화된 과정이기 때문에 소켓 통신을 위한 함수는 많이 구현되어 있다. 소켓을 여는 함수, connection과 관련된 함수, 소켓과 호스트 정보를 상호 변환하는 함수 등이 있다. 이러한 함수들을 적극적으로 사용해야 잘 모듈화된 서버를 구축할 수 있다.

3. Part A

이 단계에서는 localhost에서 http request를 받고 response를 반환하는 서버를 구현해야 한다. http.c를 컴파일해 만든 http 프로그램은 정수 인자 하나를 받고, 이를 localhost의 port number로 사용하여 server가 된다. 이 프로그램이 하는 일은 아래와 같이 구현되었다.

먼저 Open_listenfd 함수를 통해 인자 port로 접근 가능한 listening socket을 하나 만들어 client로부터 connection을 요청 받을 준비를 해야 한다. Socket이 만들어지면 이제 프로그램이 종료할 때까지 connection와 request를 반복적으로 받을 것이다. 이를 위해 무한 루프 속에서 Accept를 call 한다. Browser가 이 socket에 Connect를 보내면 Accept는 그에 반응하여 connecting socket descriptor(integer 값)을 반환하는데, 이 descriptor에 write를 함으로써 browser에 입력 반응을 돌려줄 수 있다. 이제 connection이 성사된 것이다.

Connect된 후 request를 처리하는 것은 doit 함수에서 하는 일이다. doit은 넘겨받은 descriptor(이하 fd)를 robust I/O 구조체와 함께 Rio_readinitb 함수에 넣는 방식으로 fd에서 request를 읽을 수 있다. 첫 줄은 method, URI, http version이 차례로 들어오는데, http version은 고정이므로 신경 쓸 필요 없다. 이 서버는 GET에만 반응하는 server이므로 GET 이외의 method가 들어왔다면 501 not implemented error를 반환하면 된다. GET request인 것이 정해졌다면 URI를 parse하여 따라 반환할 값을 정하면 된다. 이 과정은 parse_uri 함수로 이미 구현되어 있다. 이 함수에서는 반환할 file의 directory와 name을 알 수 있는데, 만약 이러한 파일이 존재하지 않는다면 404 not found error를, 이게 파일이 아니라 directory라면 403 forbidden error를 반환하면 된다. 이 서버는 static content만을 처리하므로 실행 파일을 작동시킬 필요는 없다. error의 반환은 clienterror 함수를 통하여 이루어지는데, fd와 error number, error message 등을 argument로 넣어 주면 이 함수가 알아서 적절한 error response를 반환해 준다.

이제 어떤 file을 반환해 줄 지 알게 되었으므로 response의 형식에 맞춰 반환해 주기만 하면 된다. Response line은 header와 content 부분으로 나누어져 있다. Content 부분은 말 그대로 파일의 내용을 반환해 주면 된다. Open 함수를 통해 file을 read only로 연 다음 파일의 값과 길이를 알아 내고, 그대로 Rio_writen을 통해 fd에 적어 주면 된다. Header는 어느 정도 형식이 필요한데, 우선 첫 줄로 요청이 error 없이 정상적으로 처리되었다는 메시지를 내보내야 한다. 그 다음으로 서버의 이름, content의 길이, content의 type을 한 줄씩 적어 주어야 한다. 한 줄의 끝은 \n이 아닌 \r\n이며, header와 content는 빈 줄(\r\n)으로 구별한다. 이 규칙들을 지키면서 fd에 Rio_writen으로 적어 주면 response가 완료된다. 이제 다시 main의 루프로 되돌아가서 다른 response들을 처리할 차례이다.

4. Part B

B단계는 브라우저와 http 서버 사이에 proxy 서버를 추가하는 것이다. proxy 서버를 사용하면 브라우저는 http 서버에 직접 request를 보내는 대신 proxy 서버에 request를 보내게 되고, proxy서버는 받은 request를 잘 해석해서 http 서버에 request를 전송, response를 받아서 그걸 다시 브라우저로 전송하게 된다. 즉, proxy 서버는 request와 response를 모두 주고받는 중간 관계자인 것이다. 이렇게만 있으면 쓸데없이 중간 과정이 추가된 것처럼 보이지만, 이후 C단계에서 할 것처럼 proxy 서버에서 여러 유용한 작업들을 할 수도 있다.
가운데에 proxy server가 끼어 있다고 request 명령어가 달라지는 것은 아니기 때문에 기본적으로 브라우저로부터 전달받은 request를 그대로 http 서버에 전달해 주면 된다. 브라우저 쪽 file descriptor에서 Rio_readline으로 받은 줄을 http 서버 쪽 file descriptor로 Rio_writeen하면 된다. 다만 여기에서 URI를 parse하여 host와 port, filename을 직접 알아내는 과정이 필요한데, parse_uri_proxy에서 해 주어야 한다. http://는 버려도 되고, host-port-filename 순으로 붙어 있는데 port는 생략되어 default인 80번을 나타낼 수도 있다. 이렇게 알아낸 host와 port는 proxt server -> http server 방향 request의 두 번째 줄의 구성 요소가 된다.
이제 http 서버에서 다시 브라우저로 전달할 차례인데, 이 역시 받은 것들을 그대로 전달하면 된다. 그런데 content를 전달할 때 content의 길이를 알고 있어야 하고, 이것은 response header에 적혀 있기 때문에 header를 잘 parse하여 알아 내야 한다. 이 정보가 적혀 있는 줄은 “Content-Length: ”로 시작하기 때문에 strncmp 함수를 적절히 사용하여 검사하였다. length를 알아내는 과정에서 header는 한 줄씩 전달하고, 이후에 나오는 content는 한 번에 전달하면 된다.

5. Part C

C단계는 프록시 서버에 캐시를 추가하는 작업이다. B단계에서 proxy서버는 브라우저로부터 request를 받으면 그 reqeust를 그대로 http 서버로 전달하는 역할만 했는데, 만약 동일한 request가 여러 번 들어오면 그때마다 http 서버에 접속하는 것보다는 어딘가에 response를 저장 놓고 반환하는 것이 나을 것이다. 이 기능을 해 주는 캐시를 proxy 서버에 추가할 것이다.
cache와 관련되어 cache.h와 cache.c가 있다. cache.h에는 이번 단계에서 사용할 함수들과 cache block struct를 정의하는 것이다. cache block에 들어갈 field로는 request의 URI, request에 대한 response header, response의 content, content의 길이이다.

여기에 나는 linked list 방식으로 캐시를 만들었기 때문에 다음 block을 가리키는 cache block pointer도 하나 넣어 주었다. URI와 response header는 크기가 한정되어 있기 때문에 char의 배열로 직접 넣어 주었지만, content는 지나치게 길어질 수 있기 때문에 struct 내부에는 char pointer만 저장해 놓고 malloc하는 방식으로 사용했다. Cache block은 이렇게 만들어 놓고, cache.c에는 시작 block을 가리키는 start 포인터와 cache_size 변수를 적어 놓았다.

다음으로 cache와 관련된 함수들을 구현했다. 첫째는 find_cache_block으로, URI 값을 통해 원하는 cache block을 찾는 함수이다. Block들을 처음부터 하나씩 순회하다가 URI가 같은 게 나오면 바로 반환하고, 끝까지 갔는데 없으면 NULL을 반환한다. 둘째는 add_cache_block으로, response의 구성 요소들을 받아 cache block으로 저장한다. 이 과정에서 cache block을 malloc하고, 그 안에 있는 content 또한 malloc으로 할당한다. 또한 cache_size에 block + content의 크기를 더해 주어야 한다. 셋째는 cache_replacement_policy로, add 함수 끝마다 불리면서 전체 cache의 크기가 너무 커졌을 때 cache block들을 free하는 역할을 한다. FIFO 규칙을 활용하여 linked list의 앞에서부터 차례대로 free한다.

함수들을 다 만들었으니 이제 코드에서 적절히 사용할 차례였다. Proxy server의 코드에 적당히 끼워 넣으면 되었다. find 함수를 불러서 NULL이 반환되면 그냥 기존의 코드를 사용하고, 반환값이 있으면 그 포인터를 통해 cache block을 가져온다. Block 안에는 response header와 content가 들어 있으니 그대로 반환하면 되는 것이었다. 이때 response는 null-terminateed string으로 저장해 두어서 strlen으로 길이를 측정하면 되었지만 content는 그렇지 않았기 때문에 contentLength를 받아 와서 size_t 인자로 넣어 주어야 했다.

마지막으로 한 작업이 남았는데, 캐시 로그를 작성하는 것이었다. 브라우저로부터 request가 올 때마다 log를 작성하는 것이었다. Log의 한 줄에는 어떤 request가 cache hit이 되었는지 여부, request가 들어온 시각, request의 URI, content의 length가 들어가 있다. 날짜는 <time.h>의 라이브러리를 사용하여 알아내고, 나머지는 인자로 전달받아 string을 적절히 만들었다. 이제 open을 통해 proxy.log라는 파일을 쓰기 전용, append하는 방식으로 열고 그 안에 string을 적어 넣었다. 그리고 open에서 반환한 fd를 close했다.

6. Conclusion

네트워크 관련 코드를 작성하는 것은 처음이었기 때문에 이를 위해 많은 공부를 해야 했다. 서로 다른 두 프로세스 간에 정보를 전달하는 작업은 어떻게 적어야 하나 막막했는데, 다행히 놀라울 정도로 잘 추상화되어 있어서 놀랐다. 프록시 서버가 할 수 있는 간단한 일을 알아보았는데 또 할 수 있는 일에는 무엇이 있는지 궁금해졌다.