Lab 05 - Proxy

정성태 2018-15515

struct types

```c
typedef struct reqline {
    char method[10];
    char uri[300];
    char version[100];
    char hostname[200];
    char path[1000];
} reqline;

typedef struct reqheader {
    char name[MAXLINE];
    char data[MAXLINE];
    struct reqheader *next;        // next link to implement a linked list
} reqheader;

typedef struct cacheblock {
    char uri[300];
    size_t size;
    char* data;
    struct cacheblock *next;        // next link to implement a linked list
    clock_t access;                 // access time for LRU implementation
} cacheblock;
```

request line과 request header를 다루기 위한 struct type을 각각 정의했다. reqline은 method와 URI, HTTP version 정보를 포함하며 parsing의 결과인 hostname과 path를 따로 저장할 수 있도록 했다. reqheader는 name과 data를 포함하고, linked list 형태로 구현할 수 있도록 next link를 포함시켰다.

cacheblock은 cache에 저장된 block 하나에 해당하는 struct type으로, 식별자로 사용되는 URI와 함께 저장된 데이터의 내용과 크기, 접근 시각을 저장하도록 했다. cacheblock 역시 linked list 구현을 위한 next link를 포함한다. cacheblock들의 linked list는 모든 thread가 공유하게 되므로, root node는 global variable로 구현했다.

main

main()에서는 먼저 cmdline argument에서 port 번호를 추출하고, 이 정보를 사용해 Open_listenfd()를 호출해서 listenfd를 생성한다. 그 다음에는 while loop으로 진입하게 된다. loop 내부에서는 Accept()를 호출해 connfd를 생성하고, 이를 thread routine으로 전달해 각 thread가 요청을 맡아 처리하도록 한다.

proxy_thread

main()의 loop 내부에서 Pthread_create()에 의해 호출되는 thread routine이다. 먼저 Pthread_detache()를 호출해 thread를 detach mode로 전환한다. 이는 main thread에서 따로 reap 작업을 하지 않아도 종료 후 자동으로 reap 처리가 되도록 하기 위함이다. 그 다음에는 setjmp()를 호출해 jump 지점을 설정한다. 이는 이후에 호출되는 함수 내부에서 오류가 발생할 경우 적절히 처리한 뒤 proxy_thread()로 곧장 돌아올 수 있도록 하기 위함이다. longjmp()에 의해 이 지점으로 돌아올 경우 오류가 발생한 것이므로 return과 동시에 thread를 제거하게 된다.

이후에는 client의 request를 읽고 이를 target server에 전송한 뒤 response를 받아 다시 client에게 전달해주는 과정이 포함되어 있다. 이 과정은 아래에 정의되어 있는 하위 함수들이 수행한다.

read_request

client의 request를 읽은 뒤 request line과 request header를 따로 처리하는 역할을 하는 함수이다. request를 읽을 때는 RIO를 사용한다. 각 라인의 처리는 parse_req_line()과 parse_req_header() 함수가 담당한다. parsing이 종료되면 prepare_headers()를 호출해 부족한 header를 채운 뒤 return 한다.

parse_req_line

읽어 들인 request line의 처리를 담당한다. 먼저 method, URI, HTTP version을 각각 추출한다. 만약 GET 외의 다른 method가 입력되면 client에게 response로 501 Not Implemented를 전달하고 종료한다. method가 GET인 경우에는 URI에서 hostname과 path를 분리한 뒤 reqline 변수에 저장하고 종료한다.

parse_req_header

읽어 들인 request header를 처리한다. 각 라인은 “name: data”의 형식으로 되어 있는데, 만약 구분자 ‘:’가 포함되어 있지 않으면 유효한 header가 아닌 것으로 간주하고 400 Bad Request를 전달하며 종료한다. 정상적인 형식의 header인 경우 name과 data를 추출하고, insert_header() 함수를 사용해 reqheader로 구현된 linked list에 새로운 node를 추가한다.

prepare_headers

thread routine에서 호출되는 함수로, header의 parsing이 완료된 후 필수로 포함되어야 하는 header들을 추가한다. 여기에는 Host, User-Agent, Connection, Proxy-Connection 등이 포함된다. header list를 읽으며 name이 일치하는 것이 있는지를 검사한 뒤 없는 경우 미리 설정된 기본값을 data로 갖는 새로운 node를 추가하는 방식으로 작동한다.

send_request

parsing이 완료된 뒤 target server에 request를 전달하는 역할을 한다. 이때에는 proxy server 자체가 client의 역할을 하게 된다.

먼저 cache 내부에 URI가 일치하는 block이 존재하는지를 검사하고, 만약 존재하면 -1을 return 하면서 바로 종료한다. 이 경우의 처리는 receive_response()에서 담당한다.

만약 cache에 저장되어 있지 않은 request인 경우 request line에 포함된 정보를 이용해 target server에 request를 전송한다. 이때 Open_clientfd()를 호출해 얻은 socket에 request를 write 하는 방식을 사용한다.

receive_response

target server로부터 response를 받고 그것을 client에게 전달해주는 역할을 한다. 만약 send_request()의 return value가 -1이라면, 즉 요청된 content가 이미 cache에 저장되어 있다면 URI를 이용해 해당 content를 찾은 뒤 그대로 connfd가 가리키는 socket에 write 하고 종료한다.

content가 cache에 저장되어 있지 않다면, 먼저 send_request()에서 얻은 request socket을 read 해서 target server의 response를 한 줄씩 읽어 들이는 동시에 connfd를 통해 client에게 전송한다. 이때 response의 전체 크기를 추적하면서 cache의 max object size 조건을 만족하는지를 검사한다. 만약 content의 크기가 max object size를 만족하고 cache에 빈 공간이 충분하다면 content를 cache에 저장한다. 만약 공간이 부족하다면 evict_cache()를 호출해 LRU policy에 따라 공간을 확보한 후 cache에 content를 저장한다.

error_response

400 Bad Request, 501 Not Implemented 등 에러가 발생할 경우 해당 에러의 내용을 담은 response를 client에게 전송하는 함수이다. response body의 형식은 교재에 나와 있는 것을 따랐다. 전송이 끝나면 longjmp()를 호출해 proxy_thread() 돌아가 thread를 종료하게 된다.

insert_header / find_header

request header들의 linked list에 새로운 node를 추가하거나 name이 입력된 것과 일치하는 node를 탐색하는 함수이다.

free_header

request header들의 linked list에 포함된 모든 node를 free 하는 함수이다. insert_header()에서 새로운 node를 생성할 때 malloc()을 사용하므로, 한 thread가 종료되기 직전에 해당 thread에서 사용하던 request header들을 모두 free 하는 과정이 필요하다.

init_cache

proxy 가 시작할 때 호출되어 cache block 들의 linked list 를 사용할 수 있는 상태로 초기화해주는 함수이다. cacheblock list 에는 insert 과정을 간소화하고 root node 와 tail node 를 추적하기 위해 dummy root node 를 사용했다. 이 함수를 호출하고 나면 global variable 인 cache_root 는 비어 있는 cachenode 변수를 가리키게 된다.

insert_cache

receive_response() 내부에서 호출되어 새로운 response 를 cache 에 저장해주는 함수이다. tail node 의 뒤, 즉 list 의 가장 뒤에 새로운 node 를 추가하는 방식으로 작동한다. 이때도 reqheader list 와 유사하게 malloc()으로 새로운 node 를 저장할 공간을 할당한다.

evict_cache

LRU policy 에 따라 가장 오래전에 사용된 block 을 list 에서 제거한다. 새로운 block 을 생성하거나 특정 block 에 접근할 때마다 접근 시각이 수정되는데, 이 접근 시각이 가장 작은 (오래된) block 을 선택함으로써 LRU 를 구현할 수 있게 된다. 이렇게 victim 을 선택하고 나면 list 에서 제거한 뒤 종료한다. 이때 제거할 block 과 그 block 의 내부에 저장되어 있는 content 를 free 해주는 과정이 필요하다.

find_cache

입력된 URI 와 일치하는 cacheblock 을 찾아주는 함수이다.

Rwriten / Rreadlineb

rio_writen()과 rio_readlineb()의 wrapper 이다. csapp.h 에 선언되어 있는 기존의 wrapper 을 사용하면 오류가 발생할 경우 바로 프로그램을 종료해버린다. 하지만 proxy server 자체는 종료되면 안 되고 thread 만을 종료해야 하기 때문에 오류가 발생했을 때 longjmp()를 통해 proxy_thread()로 돌아갈 수 있도록 구현했다.

여기서 처리한 오류는 write 할 때 발생하는 EPIPE 와 read 할 때 발생하는 ECONNRESET 이다. 둘 모두 client 측에서 예기치 못한 이유로 socket 을 close 한 경우에 발생한다. 이 경우 특별한 처리 없이 해당 thread 를 종료하면 된다고 판단해 단순히 longjmp()를 호출하도록 구현했다.

실행 결과

```
 ~/system_programming/lab05_proxy ..........................
> curl --proxy localhost:10000 http://localhost:8888/
<html>
<head><title>test</title></head>
<body>
<img align="middle" src="godzilla.gif">
Dave O'Hallaron
</body>
</html>
```

```
 ~/system_programming/lab05_proxy ....................................
> curl --proxy localhost:10000 http://csapp.cs.cmu.edu
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3
.org/TR/html4/loose.dtd">

<html>
<head>
<title>CS:APP3e, Bryant and O'Hallaron</title>
<link href="http://csapp.cs.cmu.edu/3e/css/csapp.css" rel="stylesheet" type="
text/css">
</head>

<!-- Page-independent header  -->
<div id="cover_box">
<a href="http://csapp.cs.cmu.edu/3e/home.html">
<img src="http://csapp.cs.cmu.edu/3e/images/csapp3e-cover.jpg" width=70>
</a>
</div>
<div id="title_box">
<p class="title">Computer Systems: A Programmer's Perspective, 3/E (CS:APP3e)
</p>
<p class="author"> <a href="http://www.cs.cmu.edu/~bryant">Randal E. Bryant</
a> and <a href="http://www.cs.cmu.edu/~droh">David R. O'Hallaron</a>, Carnegi
e Mellon University</p>
<p>
<a href="http://csapp.cs.cmu.edu/2e/home.html">Legacy site for the second edi
```

정상 작동함을 확인했다.

어려웠던 점

- 웹 프로그래밍을 처음 접해보는 것이어서 request나 response 등 기본적 개념을 다루는 데 어려움을 겪었다. CSAPP 교재와 다른 자료들을 적극 참고했다.
- request header와 cache block을 어떻게 저장해야 할지를 많이 고민했다. 결과적으로 linked list 구조를 채택해 구현했다.

새로 알게 된 점

- proxy를 직접 구현해보며 proxy가 실제로 어떻게 동작하는지를 이해할 수 있었다.
- telnet이나 curl 등 웹 프로그램을 테스트할 수 있는 도구를 사용할 수 있게 되었다.
- strchr, strstr 등 문자열을 처리하는 C의 함수들을 새로 알게 되었다.