

2021 SYSTEM PROGRAMMING

Lab6 Report – Proxy Lab

자유전공학부 2012-13311 안 효 지

0. 이 랩의 목적

- 1) 프록시를 직접 구현함으로써 client와 server의 역할을 모두 처리할 수 있다.
- 2) concurrent server로도 구현해봄으로써 thread와 thread safe 에 대해 알 수 있다.
- 3) 이미 알고 있는 cache를 프록시에 구현해봄으로써 프록시의 유용함을 알 수 있다.

1. How to implement

<Part A-sequential proxy>

1. main() : client로부터 request를 받는다.

- 1) listenfd와 connfd를 선언한다.
- 2) 아래와 같이 listenfd를 열고 accept하기 위해 계속 기다린다.

```
56 listenfd = Open_listenfd(argv[1]);
57 while(1){
58     clientlen = sizeof(clientaddr);
59     connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
60     sequential_proxy(connfd, &req, &req_hdr);
61     Close(connfd);
62 }
63 return 0;
64 }
```

- 3) accept가 되면 해당 connfd를 sequential_proxy()라는 함수로 넘겨주어 처리되도록 한다.

2. void* sequential_proxy(): request를 parsing하여 server에게 보낼 request를 만든다.

1) Request, Request_header structure

- Request structure에는 method(GET), uri, port, path, version이 들어가도록 하였다.
- Request_header에는 hostname, connection, proxy_connection이 들어가도록 하였다.

Hostname은 request로 들어오는 정보이나 서버로 request를 보낼 때에는 header에 들어가는 정보여서 request가 아닌 request_header에 들어가도록 하였다.

2) init structure elements

```
/* init request elements */
memset(req.method, 0, 4);
memset(req.uri, 0, MAXLINE);
memset(req.port, 0, 100);
memset(req.path, 0, MAXLINE);
memset(req.version, 0, 10);

/* init header elements */
strcpy(req_hdr.host, "Host: ");
memset(req_hdr.hostname, 0, sizeof(req_hdr.hostname));
strcpy(req_hdr.connection, "Connection: close");
strcpy(req_hdr.proxy_connection, "Proxy-Connection: close");
```

- 현재 설명하고 있는 함수는 thread가 시작되는 함수이다. 따라서 thread가 시작될 때 앞서 설명한 structure의 모든 element들을 초기화 시켜준다. 그래야 기존 메모리에 남아있던 trash value들이 리셋되어 다음 작업을 할 때 영향을 미치지 않는다.
- request의 원소는 0으로, request_header의 원소는 hostname을 제외하고는 다 정해진 값을 갖고있기 때문에 각각이 해당하는 값으로 hard coding하여 설정해주었다.

3) parsing request line

```
// parse HTTP request from browser
sscanf(buf, "%s %s %s", req.method, req.uri, req.version);
```

- 아주 간단하게 sscanf를 사용하여 method, uri, version으로 나누어주었다. 그 후 method가 GET인 경우만 다를 수 있도록 설정해주었다.

4) parsing uri

```
void parse_uri(Req* req, Req_hdr* req_hdr){
    // Parse uri
    char* path_bgn, *hostname_bgn, *port_bgn;
    char* http= strstr(req->uri, "http://");

    if (!http){
        printf("Request not begin with \"http://\"\n");
        exit(1);
    }

    /* 1) extract path */
    hostname_bgn = http + 7;
    // http:// 이후에 /로 시작하는 거 있느냐.
    if ((path_bgn = strstr(hostname_bgn, "/"))){ strcpy(req->path, path_bgn); }
    // no path => make default path as /
    else{ strcpy(req->path, "/"); }
```

```

5  /* 2) extract port for open_clientfd() */
6  /* 3) extract hostname for req_hdr */
7  if ((port_bgn = strstr(hostname_bgn, ":"))){
8      strncpy(req->port, port_bgn+1, path_bgn-port_bgn-1); // remove ':'
9      strncpy(req_hdr->hostname, hostname_bgn, port_bgn-hostname_bgn);
10 }else {
11     strncpy(req->port, "80"); // default port: 80
12     strncpy(req_hdr->hostname, hostname_bgn, path_bgn-hostname_bgn);
13 }
14 /* 4) set version */
15 strcpy(req->version, "HTTP/1.0\r\n");
16 }

```

- uri를 파싱하여 server에게 보낼 request를 만든다.
- `http://`로 시작하지 않으면 exit을 한다.
- `http://` 이후에 `/`로 시작하는 것이 있으면 그 이후의 값들을 받아와 path에 넣는다. 만약 없으면 기본 path는 `/`로 설정해준다.
- uri에 `:`가 있으면 port의 값을 `:`뒤의 값으로 설정해준다. 없으면 기본 80.
- hostname은 `http://` 이후부터 `/`, 혹은 `:`가 시작되기 전까지로 한다.
`curl --proxy localhost:4500 http://csapp.cs.cmu.edu` 이와 같이 명시적으로 input에 `/`가 없더라도 client로부터 proxy에 들어올 때에는 `uri: http://csapp.cs.cmu.edu/` 이 형태로 들어오므로 고려하지 않아도 된다.

5) make request

```

int requestfd = Open_clientfd(req_hdr.hostname, req.port);

// build request buffer
strcat(request_buf, req.method);
strcat(request_buf, " ");
strcat(request_buf, req.path);
strcat(request_buf, " ");
strcat(request_buf, req.version);
strcat(request_buf, req_hdr.host); // "Host: "
strcat(request_buf, req_hdr.hostname);
strcat(request_buf, "\r\n");
strcat(request_buf, user_agent_hdr);
strcat(request_buf, "\r\n");
strcat(request_buf, req_hdr.connection);
strcat(request_buf, "\r\n");
strcat(request_buf, req_hdr.proxy_connection);
strcat(request_buf, "\r\n");
strcat(request_buf, "\r\n");

// send request
Rio_writen(requestfd, request_buf, strlen(request_buf));

```

버퍼에 request 형식에 따라 request line과 request header를 넣고 서버에게 request를 보낸다.

3. 서버에게 response가 오면 그걸 받아서 client에게 다시 보내준다.

```
// receive response
ssize_t n = 0;
Rio_readinitb(&rio, requestfd);
while((n = Rio_readnb(&rio, response_buf, MAXLINE)) > 0){
    Rio_writen(fd, response_buf, (size_t)n);
}
```

<Part B – Concurrent proxy>

Concurrent proxy를 구현하기 위해 Posix thread를 이용하였다. Part A에서 구현한 코드를 아주 조금만 수정하면 되어서 큰 어려움은 없었다.

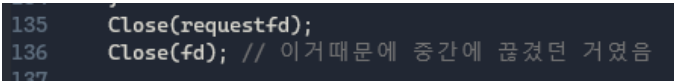
1. main함수의 while문에 있던 sequential_proxy() 호출 대신, Pthread_create를 호출하여 sequential_proxy를 thread에서 실행되도록 만든다.

```
listenfd = Open_listenfd(argv[1]);
while(1){
    clientlen = sizeof(clientaddr);
    connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
    //sequential_proxy(connfd, &req, &req_hdr);
    Pthread_create(&tid, NULL, sequential_proxy, &connfd);
}
return 0;
```

2. 하지만 이렇게 하면, 기존 sequential_proxy()에는 connfd, request_line, request_header, 이렇게 총 3개의 param을 설정 해놓았던 것을 한 개의 param으로 줄여야 했다. 따라서 원래는 그냥 main function에서 request_line과 request_header를 선언하고 init해준 후, thread에 arg로 전달을 해주었는데, 굳이 그럴 필요가 없는 것 같아 sequential_proxy()로 옮겨주었다.

```
52
53 void* sequential_proxy(void* vargp){
54     //printf("\n**[d]sequential에 들어옴\n");
55
56     int fd = *((int*)vargp);
57     Pthread_detach(pthread_self());
58
```

3. 그 후, thread에 들어가면 Pthread_detach(pthread_self())를 통해 자기 자신을 분리한다.

4. 마지막에  fd를 닫는 것을 깜빡하여 concurrent가 제대로 동작되지 않았던 것을 수정하였다.

<Part C – Proxy Caching>

1. Cache structure

```
typedef struct _Cache{
    char data[MAX_OBJECT_SIZE];
    char* key_uri;        // key for finding cache
    struct _Cache* next;
    size_t LRUcnt;
} Cache;
```

캐쉬의 구조는 위와 같다. 일단, 실제 data가 담겨있는 버퍼를 가리키는 `data`가 있다. Key_uri는 uri를 key로 하여 cache hit/miss를 판단하기 위함이다. Cache node는 linked list로 구성하였기 때문에 다음 node를 가리키는 pointer `next`가 있다. 그리고 eviction policy가 LRU이기 때문에 이를 위한 element도 있다.

2. Cache initialize

```
// Init cache dummy block
root = (Cache*)malloc(sizeof(Cache));
memset(root->data, 0, MAX_OBJECT_SIZE);
root->key_uri = NULL;
root->next = NULL;
root->LRUcnt = 0;
```

일단, main function에서 dummy node를 만들고 그것을 init한다.

3. Figure out cache hit/miss

1) HIT

Client의 request가 GET인지 확인한 후에, cache를 사용하지 않을 때에는 바로 uri parsing을 하여 server에게 보낼 request를 만들고, send request를 하고, receive response를 하고 그것을 다시 client에게 보내주었다. 하지만 cache를 사용하면 cache hit일 때에는 굳이 앞의 복잡한 과정을 거칠 필요가 없이, cache에서 꺼내서 바로 client에게 보내주면 된다. `Rio_writen(fd, node->data, strlen(node->data));`를 통해서 cache node에 들어있는 data를 바로 client buffer로 보내준다.

2) MISS

MISS인 경우에는 원래의 과정에 더하여 cache node를 build하여 insert cache하는 과정이 필요하다. Cache에 들어갈 수 있는 data는 MAX_OBJECT_SIZE보다 작아야 하므로 이를 판단해주는 과정이 필요하다. 만약 이보다 작으면 그대로 cache에 넣어주고, 크면 캐쉬에 넣는 대상에서 제외한다.

이렇게 cache node를 완성했으면 cache list에 넣어야 한다. 하지만 이 단계에서도 max

cache size를 넘지 않았는지를 체크해야 한다. 만약 새로 만든 node가 들어갈 자리가 없으면 cache list에서 eviction을 하고, 현재 allocated 된 cache size에서 evicted 된 data의 사이즈를 빼준다.

Eviction policy는 LRU로, 우리의 lab4. Cache lab과 동일한 방법을 사용하면 된다.

또한, cache list를 write하는 것은 multi thread 중에서 하나만 할 수 있고, read는 여러 thread가 동시에 해도 된다. 이를 보장하기 위해서는 insert_cache()의 과정에 writing mutex를 설정해주어, 하나의 thread가 writing하고 있을 때에는 다른 thread가 write하지 못하게 만든다.

2. What was difficult

① 처음에 디버깅하는 방법을 잘 몰라서 어려웠다. 평소 하나하나 printf문 써가면서 디버깅하면서 코딩을 하는 습관이 있는데, lab ppt에 나와있는 curl, telnet을 어떻게 사용할 지 잘 모르기도 했고, 그것은 다 완성한 후에 디버깅을 하는 용도로 사용하는 것이라고 생각해서 처음부터 사용할 생각 자체를 하지 못했다. 그래서 ./proxy portnum 만 친 상태에서 어떻게 client가 서버에게 request를 보낼 지 삽질을 하고 있었다. 하다하다 안 되어 다른 분들께 디버깅하는 방법만 질문했더니 터미널 하나를 더 키고나서 curl을 쓰면 된다고 했다. 그 답변을 듣고 curl에 대해 ppt에서 본 것 같아 찾아보니 프록시를 사용하여 curl을 이용하는 방법이 ppt자료에 그대로 나와있는 것을 발견하였다. 몇 시간 버리긴 했지만 이 방법을 알고 난 후부터는 진도가 빨라져서 괜찮았다.

② 그리고 web 내용 자체를 잘 이해를 못하고 있던 상태여서 시작이 더 어려웠다. 하지만 과제를 하고, 디버깅을 하면서 하나하나 실험을 하다 보니 이제 구조가 어떻게 되는지는 이해를 하였다. 내가 client로서 터미널에 curl -proxy localhost:4500 <http://www.sk.co.kr> 을 치면 이것이 내가 작성한 proxy server로 가고, 거기서 www.sk.co.kr로 다시 request를 보내는 것이라는 것을 알았다.

③ 또한 처음에 아무 생각 없이 concurrent proxy를 구현하면서 global var을 사용하였다. ./driver.sh로는 점수가 제대로 나왔다. 하지만 global static으로 하였던 점수가 0점이 나왔다. global이든 global static이든 thread unsafe한 것은 마찬가지일텐데 왜 결과가 다르게 나오는 지 잘 모르겠지만, 어쨌든 thread safe하게 만들기 위해 global/static global 대신 local var로 변경해주었다.

3. What was surprising

① thread를 끝낼 때에는 열려있는 fd를 다 닫아주어야 한다는 것은 알고 있었지만, 깜빡하고 닫아주지 않았더니 concurrent 점수가 갑자기 0점이 나왔다. 뭐든지 할당한 후에는 닫아주어야 한다는 교훈을 다시금 얻었다.

② request를 보낼 때 어떻게 보내는 지 굉장히 추상적으로 느껴졌었는데 직접 구현해보니 그냥 버퍼에 담고 rio를 이용해 써주기만 하면 된다는 것을 알았다. 유닉스는 모든 것을 파일로 여

긴다는 사실도 잘 와 닿지 않았었는데 조금 더 구체화된 개념으로 머리에 들어온 것 같다. 간단하면서도 신기했다.

- ③ 과제를 하기 전에는 정말로 networking programming이 복잡하고 붕 뜬 개념으로 느껴졌었는데, 과제를 하면서 훨씬 구체적으로 이해할 수 있게 되었다.

4. Result screen shot

```
get g: nate proxy to csapp to proxy opened
> ./driver.sh
*** Basic ***
Starting tiny on 1841
Starting proxy on 26014
1: csapp.c
  Fetching ./tiny/csapp.c into ./proxy using the proxy
  Fetching ./tiny/csapp.c into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
2: home.html
  Fetching ./tiny/home.html into ./proxy using the proxy
  Fetching ./tiny/home.html into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
3: tiny.c
  Fetching ./tiny/tiny.c into ./proxy using the proxy
  Fetching ./tiny/tiny.c into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
4: godzilla.jpg
  Fetching ./tiny/godzilla.jpg into ./proxy using the proxy
  Fetching ./tiny/godzilla.jpg into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
5: tiny
  Fetching ./tiny/tiny into ./proxy using the proxy
  Fetching ./tiny/tiny into ./noproxy directly from Tiny
  Comparing the two files
  Success: Files are identical.
Killing tiny and proxy
basicScore: 40/40
```

```

*** Concurrency ***
Starting tiny on port 16827
Starting proxy on port 32485
Starting the blocking NOP server on port 19387
Trying to fetch a file from the blocking nop-server
Fetching ./tiny/home.html into ./noproxy directly from Tiny
Fetching ./tiny/home.html into ./proxy using the proxy
Checking whether the proxy fetch succeeded
Success: Was able to fetch tiny/home.html from the proxy.
Killing tiny, proxy, and nop-server
concurrencyScore: 15/15

*** Cache ***
Starting tiny on port 30654
Starting proxy on port 4310
Fetching ./tiny/tiny.c into ./proxy using the proxy
Fetching ./tiny/home.html into ./proxy using the proxy
Fetching ./tiny/csapp.c into ./proxy using the proxy
Killing tiny
Fetching a cached copy of ./tiny/home.html into ./noproxy
Success: Was able to fetch tiny/home.html from the cache.
Killing proxy
cacheScore: 15/15

totalScore: 70/70

```

1) <http://www.sk.co.kr>

```

</li>^M
<!-- mib200423 네이버 추가 -->^M
<li class="mLinkSocial_li mLinkSocial_naverpost">^M
    <a href="https://post.naver.com/mediask_post" class="mLinkSocial_a" target="_blank" title
="SK그룹 공식 네이버포스트로 이동: 새 창으로 열기"><span class="blind">SK그룹 공식 네이버포스트</span></a>^M
</li>^M
<li class="mLinkSocial_li mLinkSocial_navertv">^M
    <a href="https://tv.naver.com/sktv" class="mLinkSocial_a" target="_blank" title="SK그룹 >
공식 네이버TV로 이동: 새 창으로 열기"><span class="blind">SK그룹 공식 네이버TV</span></a>^M
</li>^M
<!-- //mib200423 네이버 추가 -->^M
<!-- <li class="mLinkSocial_li mLinkSocial_twitter">^M
    <a href="https://twitter.com/skstory_blog" class="mLinkSocial_a" title="트위터로 이동" ta
rget="_blank"><span class="blind">트위터</span></a>^M
</li> -->^M
<!-- //mib190618 소셜미디어 텍스트 수정 -->^M
</ul>^M
</div>^M
<button class="sideMenu_exitPrevent blind" data-name=".sideMenu_close_btn">팝업이 아직 열린 상태입니다</button>^M
</nav>^M
<!-- //공통 모바일 메뉴 끝 -->^M
^M
<!-- 공통 스크립트 -->^M
<script src="/lib/script/common.js"></script>^M
</body>^M
</html>^M
^M

```



```

stu3@ubuntu:~$ curl --proxy localhost:30074 http://www.sk.co.kr > skproxy.txt
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 48678    0 48678    0     0  23.2M      0 --:--:-- --:--:-- --:--:-- 23.2M
stu3@ubuntu:~$ curl http://www.sk.co.kr > sknoproxy.txt
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 48678    0 48678    0     0   848k      0 --:--:-- --:--:-- --:--:-- 848k
stu3@ubuntu:~$ diff skproxy.txt sknoproxy.txt
stu3@ubuntu:~$

```

2) <http://csapp.cs.cmu.edu>

```

<!-- ----->
<!-- END CONTENT -->
<!-- ----->
<p class="footer">Copyright &copy; 2015,
Randal E. Bryant and David R. O'Hallaron</p>
</div> <!-- content_box -->

<!-- Feed sidebar -->
<div id=sidebar>
<h2 class="">Recent articles from the <a href="http://csappbook.blogspot.com">CS:APP blog</a></h2>
<script
src="http://feeds.feedburner.com/csapp?format=sigpro&
nItems=15&
displayExcerpts=true&
excerptFormat=plain&
excerptLength=25&
displayDate=true&
dateLocation=below"
type="text/javascript" ></script>
</div> <!-- sidebar -->

<!-- Google Analytics tracking code -->
<script type="text/javascript">
var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." : "http://www.");
document.write(unescape("%3Cscript src='" + gaJsHost + "google-analytics.com/ga.js' type='text/javascript'%3E%3C/script%3E"));
</script>
<script type="text/javascript">
try {
var pageTracker = _gat._getTracker("UA-5405607-4");
pageTracker._trackPageview();
} catch(err) {}</script>

</body>
</html>

```

```

stu3@ubuntu:~$ curl --proxy localhost:30074 http://csapp.cs.cmu.edu > csappproxy.txt
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 6274 100 6274    0     0 3063k      0 --:--:-- --:--:-- --:--:-- 3063k
stu3@ubuntu:~$ curl http://csapp.cs.cmu.edu > csappnoproxy.txt
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 6274 100 6274    0     0 16381      0 --:--:-- --:--:-- --:--:-- 16381
stu3@ubuntu:~$ diff csappnoproxy.txt csappproxy.txt
stu3@ubuntu:~$

```