
System Programming

Assignment#2-3

07_Proxy 2-3



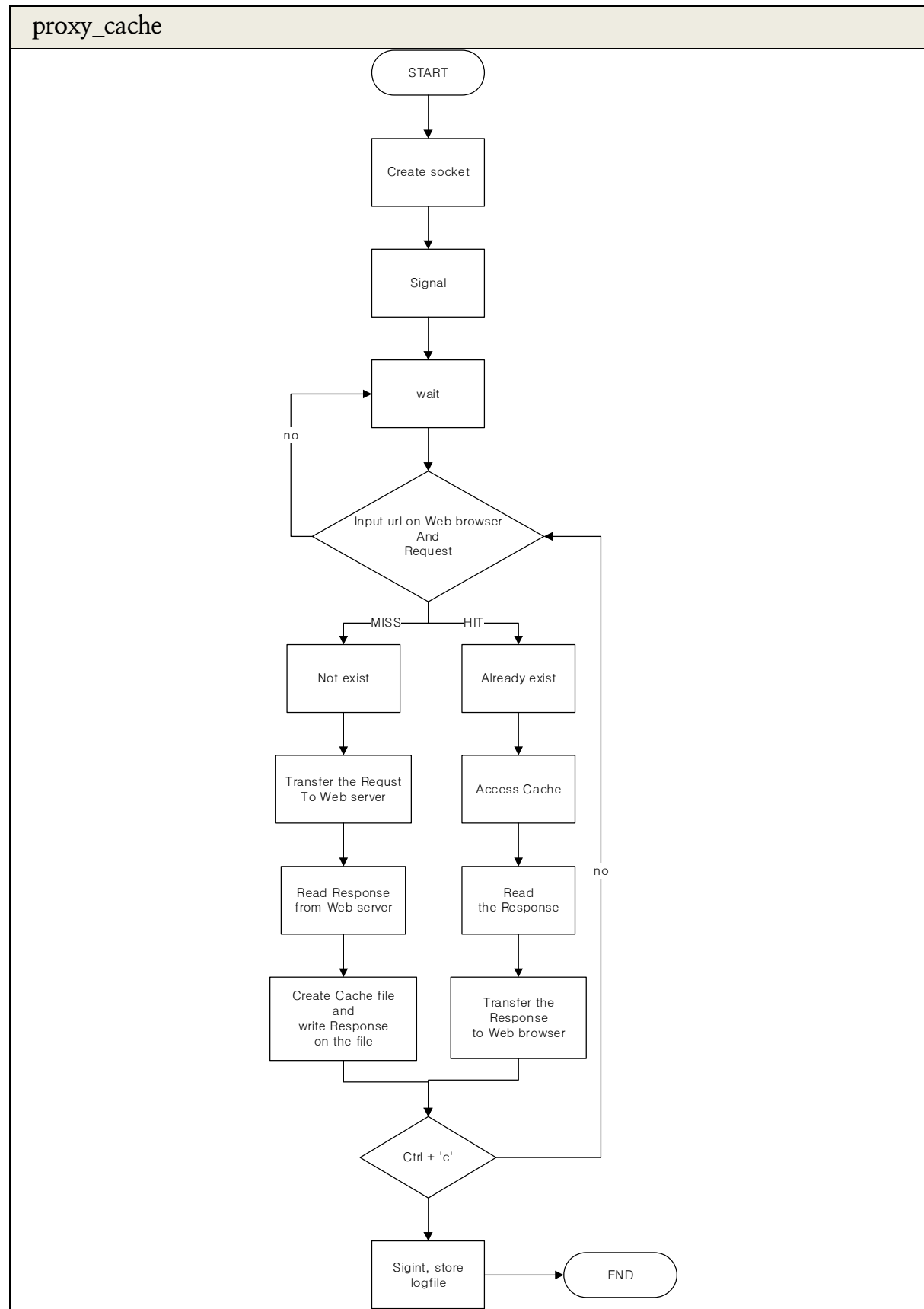
Professor	목 3 4 황호영 교수님
Department	컴퓨터공학과
Student ID	2012722028
Name	장 한 별
Date	2018. 05. 18

Introduction.

시스템 프로그래밍 강의 시간에 배운 proxy server 를 구현하는 것을 목표로 한다.

이번 과제는 web browser 에서 url 을 입력하면 해당 url 의 HTTP response 파일 이 있는지 proxy server 내 cache 에 있는지 확인하고, 없으면 web server 와 통신하여 request 를 보내고, response를 받아와 response 를 cache 파일의 내용으로 저장하고, web browser 에도 write 하도록 한다. 있으면 cache 파일을 read 한 후, web browser에 write 하도록 한다. Signal(SIGINT, SIGALRM)이 추가된다.

Flow chart.



위 그림은 이번 과제 proxy_cache 의 flow chart 이다. Server 측에서 socket 을 생성 후, web browser 에서 url 을 입력할 때까지 대기한다. Web browser 에서 url 을 입력시 proxy server 의 cache 를 확인한 후 파일이 없으면 MISS, 있으면 HIT 가 된다. MISS 상황에서는 다시 web server 에 request 한 후 web server 로부터 온 response 를 모두 read 해 web browser 에 write 하도록 한다. 또한 그 response 를 cache file 의 내용으로 저장한다. HIT 상황에서는 web server 에 접근할 필요 없이 이 cache file 을 read 한 후 그 정보를 그대로 web browser 에 write 하도록 한다.

이번 과제에선 몇가지의 signal 이 추가 되었는데 SIGALRM 과 SIGINT 이다. SIGALRM 은 web server 로부터 온 response 를 read 시 10초가 초과되면 알람 기능이 있는 signal 이고, SIGINT 는 ctrl + 'c' 로 proxy_cache 를 종료 시, server 내용이 log file 에 저장되도록 한다. 이때, Run time 과 process count 가 저장된다.

Pseudo code.

proxy_cache

```
Int main(void)
{
    Socket();
    Setsockopt();
    Bind(socket);
    Listen(socket, 5);
    Signal(SIGCHLD);
    Signal(SIGALRM);
    Signal(SIGINT);

    While(1)
    {
        Accept();
        Read(client);

        Printf("%s", Request );
        Get url;

        Assignment #1-2;
        If(MISS)
        {
            Socket(web);
            Connect(web);
            Write(web, request);
            Alarm(10);
            While(read(web, buf))
            {
                Write(client, buf);
            }
        }
    }
}
```

```

        Alarm(0);
    }
    Close(web);


    Fprintf(fp, response);
    Fclose(fp);
    Fprintf(logfile, "MISS");
    Fclose(logfile);
}
Else
{
    Read(cache, response);
    Write(client, response);

    Fprintf(logfile, "MISS");
    Fclose(logfile);
}
Close(client);
}
Close(socket);
Return 0;
}

```

위 그림은 proxy_cache 의 pseudo code 이다. Server 측에서 socket 을 생성 후, web browser 에서 url 을 입력할 때까지 대기한다. Web browser 에서 url 을 입력시 response data 가 있는지 proxy server 내 cache 를 찾는다. 없으면 MISS 있으면 HIT 가 된다. MISS 시, Web server 에 request 한 후 response message 를 read 해 cache 파일의 내용으로 write 한다. 이때 read 가 10초가 넘어가면 SIGALRM 을 통해 'no response' 메시지를 출력하도록 한다. 또한 web browser(client) 에도 response 를 write 한다. HIT 시, response 가 cache 파일의 내용으로 존재하므로, 그 response 내용을 read 한 후, 바로 web browser 에 write 하도록 한다.

Result.

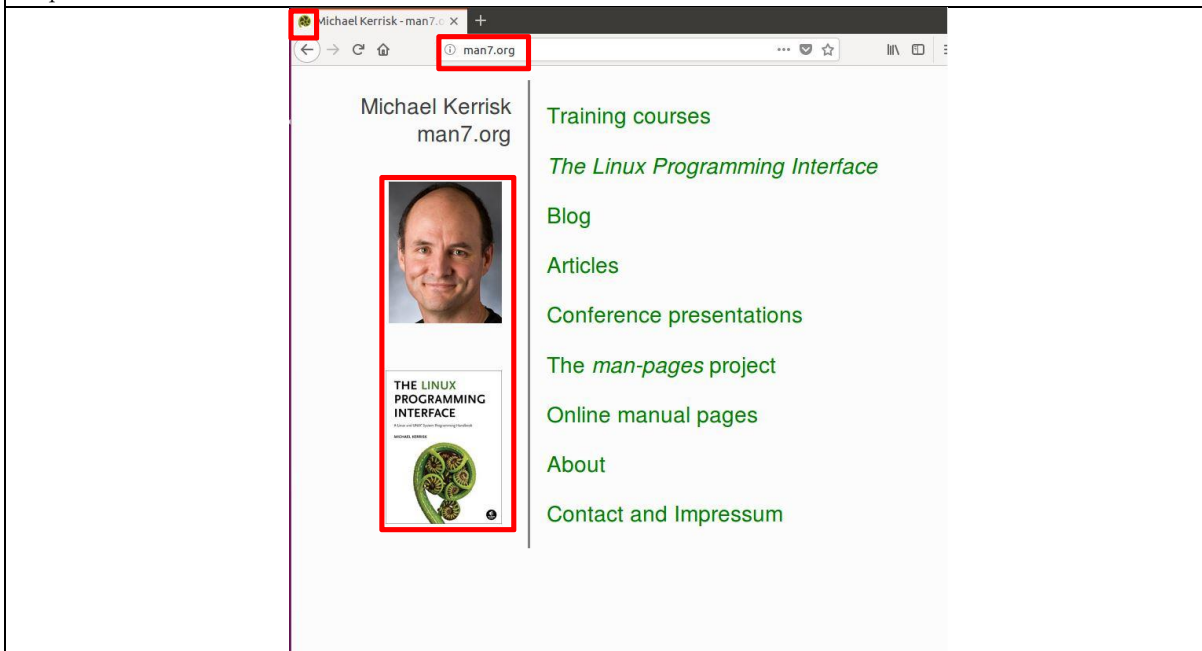
2-3.(1)	
<pre> 2012722028@sslab-desktop:~\$ make gcc -o proxy_cache proxy_cache.c -lcrypto 2012722028@sslab-desktop:~\$ ls examples.desktop Makefile proxy_cache proxy_cache.c 2012722028@sslab-desktop:~\$./proxy_cache </pre>	
<p>IP address : 223. 194. 46. 163.</p> <p>Port num : 38077</p> <p>위 IP address 와 Port number 를 이용해 이번 과제의 proxy server는 연구실 서버를 이용하도록 한다. 위 그림은 Xshell 을 이용해 정상적으로 연결이 완료되었다는 것을 확인할 수 있다. 그 후 proxy_cache 를 실행시킨다.</p>	
	

위 그림과 같이 Ubuntu, firefox 에 'man7.org' 를 입력시킨다.

```
=====
Request from [128.134.49.20 : 54216]
GET http://man7.org/ HTTP/1.1
Host: man7.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
=====
```

```
url:===man7.org===
host:===man7.org===
IPaddr:===213.131.240.174===
hash url:===man7.org===
hashed:===8933dcc5c486944f791c885
```

그러면 위 그림과 같이 web server 에 요청한 request 를 terminal 에 출력하게 된다. 추가로 문제 없이 통신이 이루어지는지, 혹은 후에 필요한 정보는 없는지를 고려해 오른쪽 그림과 같이 url, host, IP address, hash 예정인 url, hash 된 url 을 출력하게 했다. 위 그림과 같이 정상적으로 web server 에 request 했다는 것을 확인할 수 있다.



다시 Ubuntu, firefox 를 확인하게 되면 위 그림과 같이 출력되었음을 확인할 수 있다. Web server 로 부터 온 response 를 모두 read 하여 web browser(client) 에 write 했다. Jpg 파일이나 png 파일이 출력 되지 않는 문제는 response 를 client 에 write 시, 해당 buffer 의 size 만큼 write 했던 것을 read 함수가 반환하는 값으로 대체하는 것으로 해결했다. 또한, HTTP 정보가 web browser 에 출력되던 문제는 write 시 모든 response 를 한번에 write 하지않고 한 줄씩 read해 곧 바로 그 한 줄을 write 함으로써 해결했다. 행복한 미소를 띄고 있는 아저씨를 보면서 정상적으로 이루어졌음을 확인할 수 있다.

2-3.(2)



위 그림과 같이 Ubuntu, firefox 에 'man7.org/linux/man-pages' 를 입력한다.

```

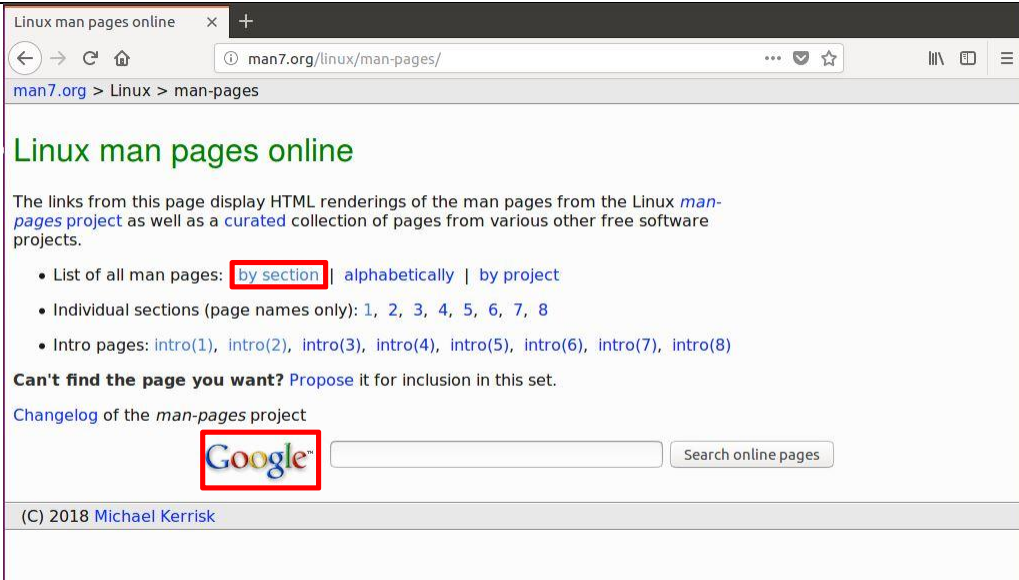
=====
Request from [128.134.49.20 : 16073]
GET http://man7.org/linux/man-pages/ HTTP/1.1
Host: man7.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: sc_is_visitor_unique=rx6129341.1526634963.6E74C08383F64F5F12BA07FAEE
8A3E2C.1.1.1.1.1.1.1.1; __utma=141474094.482797898.1526634963.1526634963.1
526634963.1; __utmb=141474094.1.10.1526634963; __utmc=141474094; __utmz=1414
74094.1526634963.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); __utmt=1
Connection: keep-alive
Upgrade-Insecure-Requests: 1

=====

url:==man7.org/linux/man-pages==
host:==man7.org==
IPAddr:==213.131.240.174==
hash url:==man7.org/linux/man-pages==
hashed:==f3afe9a64ddd001d5bff149d718dab6cb68a43f9==

```

그 후 위 그림과 같이 Request 가 올바르게 출력됨을 확인할 수 있고,



위 그림을 확인하여 response 역시 올바르게 client 에 write 되었음을 확인할 수 있다. Response 를 read 하여 한번에 write 하는 방식에서 한 줄 한 줄 write 하는 방식을 사용함으로써 구글 로고가 출력되 지 않던 문제 역시 해결 할 수 있었다.

그 후 'by section'을 클릭한다.


```
[128.134.49.20 : 18633] client was connected.
=====
Request from [128.134.49.20 : 18633]
GET http://man7.org/linux/man-pages/dir_all_by_section.html HTTP/1.1
Host: man7.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://man7.org/linux/man-pages/
Cookie: __utma=141474094.482797898.1526634963.1526634963.1526634963.1; __utmb=141474094.2.10.1526634963; __utmc=141474094; __utmz=141474094.1526634963.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); __utmt=1; sc_is_visitor_unique=rx7422636.1526635042.6E74C08383F64F5F12BA07FAEE8A3E2C.1.1.1.1.1.1.1-6129341.1526634963.1.1.1.1.1.1.1
Connection: Keep-alive
Upgrade-Insecure-Requests: 1

=====

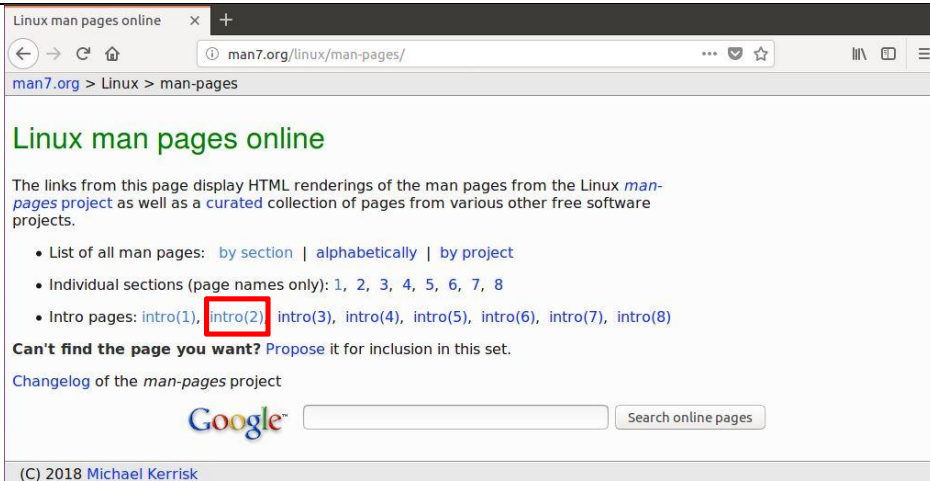
url:==man7.org/linux/man-pages/dir_all_by_section.html==
host:==man7.org==
IPaddr:==213.131.240.174==
hash url:==man7.org/linux/man-pages/dir_all_by_section.html==
hashed:==b00a81d4f740647c6b1b53e80d3d96058b60aab6==
```

위 그림과 같이 terminal 에서 request가 출력됨을 확인한 후,

The screenshot shows a web browser window with the address bar displaying `man7.org/linux/man-pages/dir_all_by_section.html`. The page title is "Linux man pages: list of all" and the URL bar shows "man7.org > Linux > man-pages". The main content area is titled "Linux man pages: all pages, by section" and includes a "Jump to section:" link with options 0, 1, 2, 3, 4, 5, 6, 7, 8. Below this, a "top" link is visible. The "Section 0" section lists various header files and their descriptions, such as `aio.h(0p)` for asynchronous input and output, `arpa_inet.h(0p)` for definitions for internet operations, and `assert.h(0p)` for verifying program assertions. The list continues with `complex.h(0p)`, `cpio.h(0p)`, `ctype.h(0p)`, `dirent.h(0p)`, `dlfcn.h(0p)`, `errno.h(0p)`, `fcntl.h(0p)`, `fenv.h(0p)`, `float.h(0p)`, `fmtmsg.h(0p)`, `fnmatch.h(0p)`, `ftw.h(0p)`, `glob.h(0p)`, `grp.h(0p)`, `iconv.h(0p)`, `inttypes.h(0p)`, `iso646.h(0p)`, `langinfo.h(0p)`, `libgen.h(0p)`, `limits.h(0p)`, `locale.h(0p)`, `math.h(0p)`, `monetary.h(0p)`, `mqueue.h(0p)`, `ndbm.h(0p)`, `netdb.h(0p)`, and `net_if.h(0p)`.

위 그림과 같이 올바르게 client 에 write 되었음을 확인할 수 있다. 이로써 MISS 상황에서 또 다른 MISS 상황이 일어나도 문제가 없음을 확인할 수 있다.

위 그림과 같이 firefox에 다시 'man.7.org/linux/man-pages' 를 입력한다.



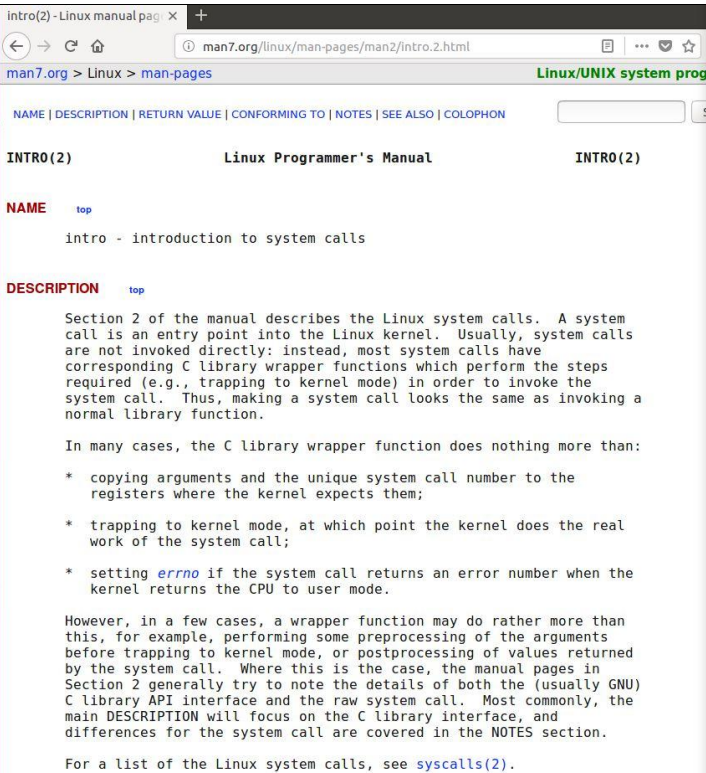
2-3.(2) 에서 입력했던 url 이지만 위 그림을 확인하여 알 수 있듯이 HIT 상황에서도 문제가 없음을 확인할 수 있다. 그 후 intro(2) 를 클릭한다.

```
XshellXshell[128.134.49.20 : 21449] client was connected.
=====
Request from [128.134.49.20 : 21449]
GET http://man7.org/linux/man-pages/man2/intro.2.html HTTP/1.1
Host: man7.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://man7.org/linux/man-pages/
Cookie: __utma=141474094.482797898.1526634963.1526634963.1526634963.1; __utm
b=141474094.4.10.1526634963; __utmc=141474094; __utmz=141474094.1526634963.1
.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); __utmt=1; sc_is_visitor_un
ique=rx7422636.1526635219.6E74C08383F64F5F12BA07FAEE8A3E2C.1.1.1.1.1.1.1.1
-6129341.1526634963.1.1.1.1.1.1.1.1
Connection: keep-alive
Upgrade-Insecure-Requests: 1

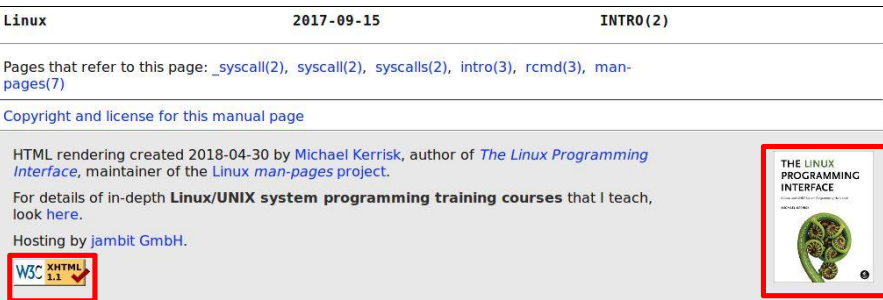
=====

url===man7.org/linux/man-pages/man2/intro.2.html===
host===man7.org===
IPAddr===213.131.240.174===
hash url===man7.org/linux/man-pages/man2/intro.2.html===
hashed===74f7f7f588545852a3916746e27e44a7dc72a215===
```

Terminal 에서 request 가 출력됨을 확인하고,

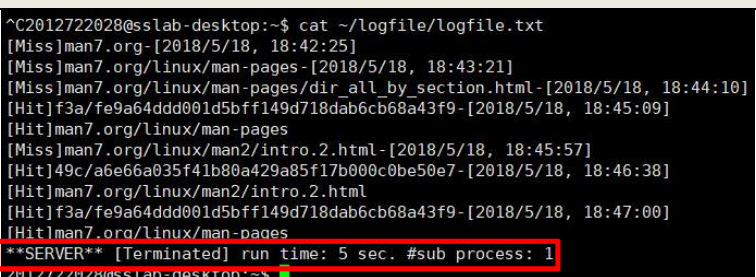


Firefox 역시 문제없이 작동 됨을 확인할 수 있다.



해당 url 맨 밑에 있는 사진, 로고도 문제없이 출력됨을 확인할 수 있다.

2-3.(4)



위 그림은 'ctrl + C' 키를 눌러 종료한 뒤 logfile.txt 내용을 출력한 결과 화면이다. 2-3.(1) ~ 2-3.(3) 과정이 올바르게 작성되었음을 확인할 수 있다. 또한 'ctrl + c' 키를 눌러 종료시킬 때 signal 을 보내는데 이 signal 을 받아 서버 정보를 작성하는 것까지 문제없이 되었음을 확인할 수 있다.

```
2012722028@sslab-desktop:~$ ls
cache  examples.desktop  logfile  Makefile  proxy_cache  proxy_cache.c
2012722028@sslab-desktop:~$ tree -/cache
/home/2012722028/cache
├── 086
│   └── a596584f5f75a319fc349faec63535be07ef1
├── 139
│   └── adf2d7603d7033d2baa1eb4062bf54734890f
├── 3fb
│   └── be067807f2755bc6a656df3e1bdc74629d5eb
├── 551
│   └── 493c4e8d35727625323e537e2e68679389790
├── 74f
│   └── 7f7f588545852a3916746e27e44a7dc72a215
├── 893
│   └── 3dcc5c486944f791c8851fd450ffe562ef992
├── 8dc
│   └── e3036abc0228de0101561b03ed6491f7b5b3
├── 958
│   └── 291e1c24981f7d0e744b6041673502f67889e
├── ade
│   └── 53d15e83846d2f631e4390beb2860bcdbe09
├── b00
│   └── a81d4f740647c6b1b53e80d3d96058b60aab6
├── b59
│   └── d5533ae6d56375d47191af8dfdbb6fa83af87
├── g91
│   └── 507dc37537a2dd3d2a68d0c2f7ef38d9c377e
├── dee
│   └── 7366a252e389181a7137b106783110b45b7c3
├── f2d
│   └── 58c847f65515046eeba53800699c34924217d
├── f3a
│   └── fe9a64ddd001d5bfff149d718dab6cb68a43f9
├── f51
│   └── 081771bf3d52ea7e8de8e271c75295754c736
```

위 그림은 위 과정들을 통해 생성된 cache file 들을 tree 로 출력한 화면이다.

Conclusion.

Input 으로는 HTTP request, Output 으로는 web server 로부터 proxy server 를 통해 온 HTTP response. 이번 과제는 이것들을 적절히 잘 활용하여 proxy server 구조를 구현했다. 지난 과제에선 Socket 관련 함수인 socket(), bind(), listen(), accept(), connect() 가 중요 했었다면, 이번 과제는 read() 와 write 를 더욱 자세하게 활용할 수 있었어야했다. Web browser(clinet) 상에 url 을 입력하면 해당 HTTP request 를 create 한다. 그 후 web server 로부터 온 HTTP response 를 web browser 에 write 하게 되면 web page 가 출력되게 된다. 이때, 처음으로 입력한 url 이면 response message 를 cache file 의 내용으로 저장하게 된다. 다음 시도에 그 url 을 입력하면 web server 와 통신할 필요없이 저장되어 있는 cache 파일을 read 한 후 그 response message 를 곧 바로 web browser 에 write 하므로 훨씬 빠른 결과를 확인할 수 있다.

Web server 의 response 를 10초 안에 read 하지 못한다면, 그것을 나타내주는 signal 인 SIGALRM, ctrl+ 'c' 로 종료 시, 그것을 나타내주는 signal 인 SIGINT 를 활용하여 구현했는데, SIGALRM 은 크게 문제가 없었지만 SIGINT 는 어려웠다. 종료가 되면서 signal 을 보내주고 그 signal 이 발생시 log file 에 server 정보(실행시간, 생성된 프로세스 수)를 출력해야 하는데, 생각보다 쉽게 해결되지 않았다. 이 부분은 최용락 군의 도움을 받아 함수 호출과, 전역 변수를 이용해야한다는 아이디어를 얻었고, 여러 번의 시행착오 끝에 구현할 수 있었다. 이때 process가 parent 일 때와 child때 총 2가지 모두 출력 될 수 있으므로 child process 라는 하나의 flag 를 추가해 그것만 log file에 저장될 수 있도록 해 더욱 완성도를 높일 수 있었다.

Web server 로부터 의 response 를 read 한 후 그 response 를 write 하는 과정에서 처음에는 처음부터 끝까지 모두 read 한 후 그 내용을 한번에 write 하는 방식으로 구현 했었는데 로고나 이미지가 출력이 안 되는 문제가 있었고, 가끔은 HTTP 내용이 그대로 web browser 에 출력되는 문제가 있었다. 그래서 한 줄 read 할 때마다 그 한 줄을 바로 write 하는 방식으로 바꿔보니 올바르게 출력되는 것을 확인할 수 있었다.

이번 과제는 그 동안의 과제의 난이도와는 차원이 달랐다. 개념 자체도 쉽게 이해할 수 있는 부분이 아니었고, 수많은 ERROR가 있었고, 네트워크 상황에 따라 그때 그때 결과가 달라 결과를 확인하기도 힘들었다. 그렇지만 포기하지않고 2주라는 기간 동안 끝까지 노력해서 구현할 수 있었고, 혼자만의 생각으로는 한계가 있어 다른 사람들과 아이디어를 공유하고 장점을 극대화하고, 단점을 피드백 해서 개개인의 실력을 향상시키고, 동료들과의 협동심을 기를 수 있었던 과제였던 것 같다. 3차 과제까지 포기하지 않고 끝까지 완수할 수 있도록 노력할 것이다.