Wednesday, April 11, 2018

EE 323, Assignment #2

Developing a Simple Web Proxy

1. Overview

A Web proxy is a program that acts as a middleman between a Web browser and an end server (See Figure 1). Instead of contacting the end server directly to get a Web page, the browser contacts the proxy, which forwards the request to the end server. When the end server replies to the proxy, the proxy sends the reply to the browser.

Proxies are used for many purposes. Sometimes proxies are used in firewalls, such that the proxy is the only way for a browser inside the firewall to contact an end server outside. The proxy may do translation on the page, for instance, to make it viewable on a Web-enabled cell phone. Proxies are also used as anonymizers. By stripping a request of all identifying information, a proxy can make the browser anonymous to the end server. Proxies can even be used to cache Web objects, by storing a copy of, say, an image when a request for it is first made, and then serving that image in response to future requests rather than going to the end server.

In this assignment, you will develop a simple web proxy program. Here, you need to implement the following functions for the proxy: First, your proxy needs to receive a HTTP request from a browser. Second, when a browser sends a HTTP request, the proxy should relay the request to a web server to get a content for the request (technically, the proxy should check whether a content for the request in a local repository. However, we will skip this part). Third, if the proxy receives the content from a web server, then the proxy should return this content to a client. Finally, the proxy should keep a log for each request of a client. Your ultimate goal is to successfully receive the requested web page through your implemented proxy from real web browser.

```
(browser)<---------------->HTTP<---------------->(proxy)<---------------->HTTP<---------------->(web server)
```

Figure 1 Overall Operation

2. Design Guide (and Requirements)

2.1. Proxy

The proxy acts as both client/server between a browser and a web server. The following requirements must be implemented in your proxy.

The proxy MUST

- be run as "`./proxy [port_number]`", and the argument should be listening port
- receive HTTP `GET` request from a browser (Firefox)
  * don't need to care about PUT, HEAD or other requests
  * don't need to care about a cookie or advanced options
- modify request/response header fields to notify the browser/web server that the packet is redirected from the proxy (See 2.2 for more information about the header fields)
- send the modified request to the web server
- receive content from the web server, and return to the web browser that asks the content
  * here, you should care about *bulk response handling* from the web server (i.e., the target resource file will be delivered as separate packets due to its size)
- store log information of each client request (log file name – ***proxy.log***)

* the log format should be "`[timestamp] [ip_address:port] [request-line]`"

* the timestamp should follow the format "`dd mmm yy hh:mm:ss zzz`"

* the following Figure 2 is an example of the log file:

```
02 FEB 08 07:59:01 PST 127.0.0.1 GET http://ee323.kaist.ac.kr/rfc2616.html  HTTP/1.1

02 FEB 08 08:01:01 PST 127.0.0.1 GET http://ee323.kaist.ac.kr/index.html HTTP/1.1
```

Figure 2 An example of the log file "proxy.log"

2.2 Processing HTTP Request/Response

When the proxy receives a HTTP GET request or a HTTP response from the web browser, it needs to modify some header fields according to the HTTP specification. The purpose of modification is to explicitly notify the web browser and the web server that the packets are being delivered though a proxy.

2.2.1 HTTP Request

Run Firefox and enter `http://ee323.kaist.ac.kr/` in the URL bar. If you look at packets using Wireshark, you can see the kind of the one in Figure 3. It is an HTTP GET request, which contains a URI (Uniform Resource Identifier) of the requested resource and other request-header fields. The fields have additional information about the request, and about the client itself (See Section 5 in [1]).

When the proxy receives a packet, it needs to change/add some fields: 1) request-URI of request-line, 2) `Via`, and 3) `X-Forwarded-For` field. For example, see red-colored texts of the packet in Figure 4. The request-URI "`http://ee323.kaist.ac.kr/rfc2616.html`" to "`/rfc2616.html`", which is a relative address version of the original one. And, the proxy adds the `Via` field, which indicates the intermediate *host* information, and `X-Forwarded-For`, which indicates an *IP address* information.

* NOTE:

  - the strings of `Via` and `X-Forwarded-For` in the example should be exactly inserted
  - each field is ending with CRLF (`\r\n`), so `\r\n\r\n` indicates that a HTTP header is end

```
//HTTP Request browser -> proxy
GET http://ee323.kaist.ac.kr/rfc2616.html HTTP/1.1\r\n /* request-line */
Host: ee323.kaist.ac.kr\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Cookie: _ga=GA1.3.1086174080.1523035245\r\n
Connection: keep-alive\r\n
Upgrade-Insecure-Requests: 1\r\n
\r\n
```

Figure 3 An example of a HTTP GET request from web browser

```
//HTTP Request: proxy -> web server
GET /rfc2616.html HTTP/1.1\r\n /* request-line */
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Cookie: _ga=GA1.3.1086174080.1523035245\r\n
Upgrade-Insecure-Requests: 1\r\n
Host: ee323.kaist.ac.kr\r\n
Via: 1.1 ubuntu (ee323_proxy/1.0.0)\r\n
X-Forwarded-For: 127.0.0.1\r\n
Cache-Control: max-age=259200\r\n
Connection: keep-alive\r\n
\r\n
```

Figure 4 An example of a HTTP GET request from proxy

2.2.2 HTTP Response

When the web server receives a HTTP GET request, it retrieves the requested resource as HTTP response. As you can see in Figure 5, it has a response-line "`HTTP/1.1 200 OK\r\n`" and some response fields. Here, you also need to insert the `Via` field to the response packet, as shown in the Figure 6. When you add the additional field to the header, you should be care about message-body, which is located at the next of "`\r\n\r\n`" in the header. If you incorrectly reassemble the message-body, a web browser cannot encode your request.

```
//HTTP Response web server -> proxy
HTTP/1.1 200 OK\r\n /* response-line */
Date: Tue, 10 Apr 2018 06:44:25 GMT\r\n
Server: Apache/2.4.7 (Ubuntu)\r\n
Last-Modified: Tue, 10 Apr 2018 06:24:32 GMT\r\n
ETag: "83393-56978935661aa-gzip"\r\n
Accept-Ranges: bytes\r\n
Vary: Accept-Encoding\r\n
Content-Encoding: gzip\r\n
Keep-Alive: timeout=5, max=100\r\n
Connection: Keep-Alive\r\n
Transfer-Encoding: chunked\r\n
Content-Type: text/html\r\n
\r\n
[message-body] /* contains requested resources */
```

Figure 5 An example of a HTTP response from web server

```
//HTTP Response proxy -> browser
HTTP/1.1 200 OK\r\n /* response-line */
Date: Tue, 10 Apr 2018 06:44:25 GMT\r\n
Server: Apache/2.4.7 (Ubuntu)\r\n
Last-Modified: Tue, 10 Apr 2018 06:24:32 GMT\r\n
ETag: "83393-56978935661aa-gzip"\r\n
Accept-Ranges: bytes\r\n
Vary: Accept-Encoding\r\n
Content-Encoding: gzip\r\n
```

```
Content-Type: text/html\r\n
Transfer-Encoding: chunked\r\n
Via: 1.1 ubuntu (ee323_proxy/1.0.0)\r\n
Connection: keep-alive\r\n
\r\n
[message-body] /* contains requested resources */
```

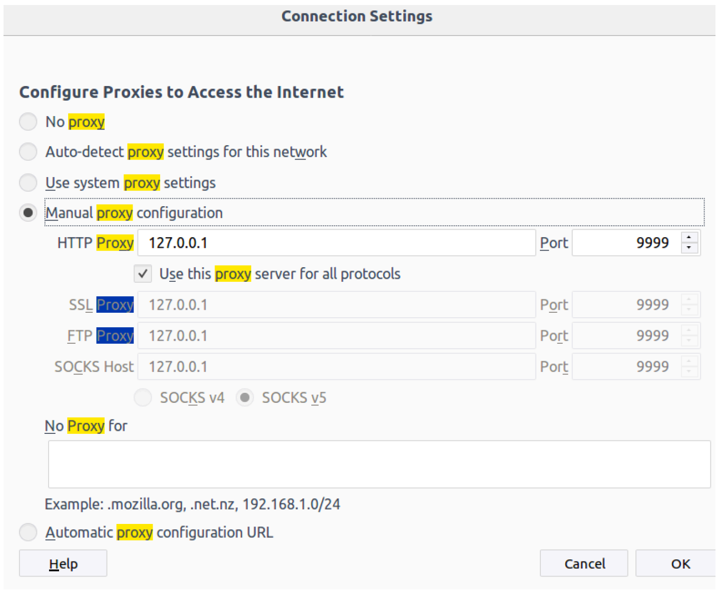
Figure 6 An example of a HTTP response from proxy

3. Operational (Test) Scenario

1) First, you need to run your proxy using the command "`./proxy 9999`"

1) Second, you need to set the proxy in the Firefox setting menu (click the right menu -> Preferences -> Search 'proxy' -> click 'Settings' of Network Proxy)

2) Next, click the "Manual proxy configuration" and set the HTTP Proxy as "127.0.0.1" and use the Port 9999.



Connection Settings

Configure Proxies to Access the Internet

- No proxy
- Auto-detect proxy settings for this network
- Use system proxy settings
- [✓ Manual proxy configuration]

HTTP Proxy 127.0.0.1 Port 9999

[✓] Use this proxy server for all protocols

SSL Proxy 127.0.0.1 Port 9999

FTP Proxy 127.0.0.1 Port 9999

SOCKS Host 127.0.0.1 Port 9999

SOCKS v4 [✓ SOCKS v5]

No Proxy for

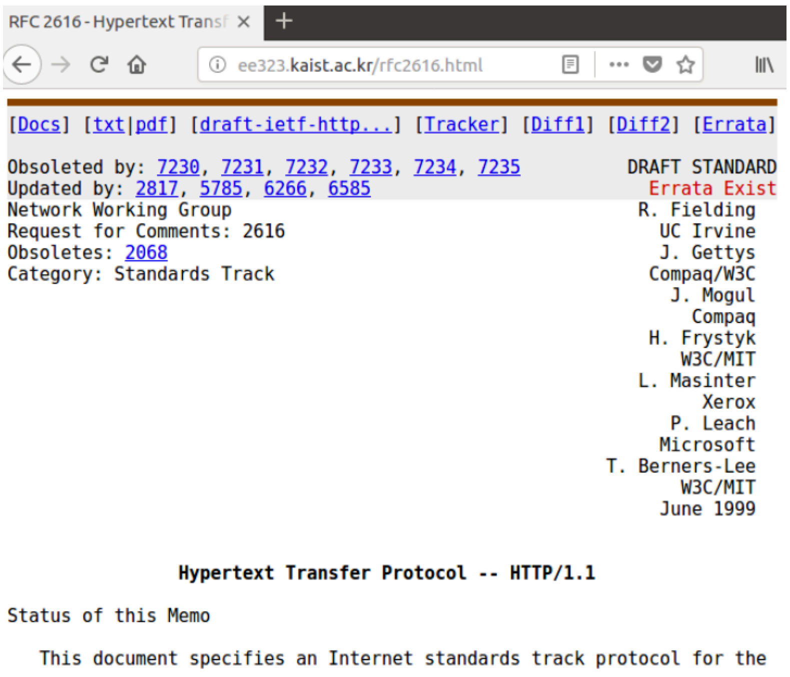
Example: .mozilla.org, .net.nz, 192.168.1.0/24

- Automatic proxy configuration URL

Figure 7 Proxy Configuration in Firefox

3) When you enter the URL (http://ee323.kaist.ac.kr/rfc2616.html) in the Firefox, the browser requests it to the configured proxy. The proxy retrieves it from the web server and delivers it to the browser.

4) Now, you should see the following screen, meaning that you successfully connected the web server and retrieved the content through your proxy.



RFC 2616 - Hypertext Transf ✕ +

ee323.kaist.ac.kr/rfc2616.html

```
[Docs] [txt|pdf] [draft-ietf-http...] [Tracker] [Diff1] [Diff2] [Errata]

Obsoleted by: 7230, 7231, 7232, 7233, 7234, 7235           DRAFT STANDARD
Updated by: 2817, 5785, 6266, 6585                           Errata Exist
Network Working Group                                          R. Fielding
Request for Comments: 2616                                       UC Irvine
Obsoletes: 2068                                                  J. Gettys
Category: Standards Track                                       Compaq/W3C
                                                                  J. Mogul
                                                                    Compaq
                                                                H. Frystyk
                                                                   W3C/MIT
                                                               L. Masinter
                                                                     Xerox
                                                                  P. Leach
                                                                 Microsoft
                                                            T. Berners-Lee
                                                                   W3C/MIT
                                                                 June 1999


                Hypertext Transfer Protocol -- HTTP/1.1

Status of this Memo

   This document specifies an Internet standards track protocol for the
```

Figure 8 An example of the result screen

4. Instructions for Submission

- You will be submitting one taball file to the KLMS website.
- Create a tarball (*.tar) with all the source codes, README, and executables.
- Tarball structure for program assignment 1:

1) Create a folder called "`p1`"
2) Put your source code "`proxy.c`" in the folder named "`p1/src`"
3) Put your executable "`proxy`" in the folder named "`p1/bin`"
4) Put your "`Makefile`" in the root path - "`p1/`
   A. Make sure your `Makefile` correctly complies your source code
5) `readme.txt` file goes to the root path - "`p1/`"
   A. NO README, NO POINTS; let us know how to run your program (be concise, it is not related your score even though you have long readme)
6) `install.sh` goes to the root path as well – "`p1/`" (optional)
   A. But, if your program needs a third-party library, you should provide a script "install.sh"
7) Then, compress the entire "`p1`" folder into a single tarball; the name will be **"`p1_yourIDnumber.tar`"**
8) Write the **concise** comments in the source code to understand your program.

! IMPORTANT 1: Please strictly follow the above structure and name policy; if not, TAs will not evaluate your program.

! IMPORTANT 2: Please make sure that there is no compile and execution error. TAs will not give you any score in case of the problems.

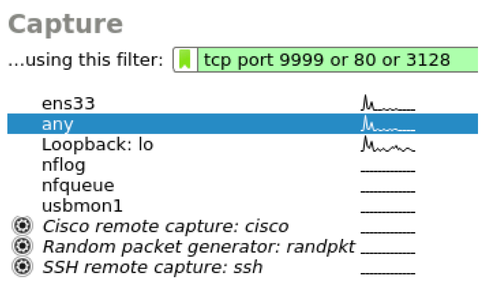
5. Evaluation (can be changed when we evaluate your program later) – 100 pts

1) Does your proxy receive a HTTP GET request from Firefox? (10 pts)
2) Does your proxy modify the request correctly? (30 pts)
3) Does your proxy send the request to the web server (i.e., ee323.kaist.ac.kr)? (10 pts)
4) Does your proxy receive the HTTP response from the web server? (10 pts)
5) Does your proxy send the response to the Firefox correctly? (30 pts)
6) Can TAs see the received web page (`rfc2616.html`) in the Firefox? (10 pts)
7) Can the proxy cache the requested file? I mean, can you receive the requested file in the second try without contacting the web server? (Optional; but you can get bonus points if you implement it, use any method that you want) (20 pts)

6. Top tips

Here are top tips for you guys who are not familiar with network programming.

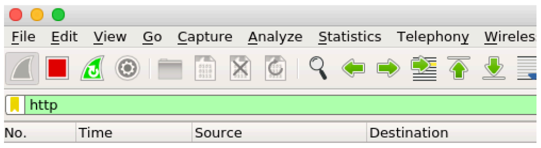
- Use Wireshark with packet filter options, to see only HTTP related packets
  - e.g.,



Capture

...using this filter: tcp port 9999 or 80 or 3128

```
ens33
any
Loopback: lo
nflog
nfqueue
usbmon1
Cisco remote capture: cisco
Random packet generator: randpkt
SSH remote capture: ssh
```



http

```
No.     Time        Source          Destination
```

- (In case of C) Use GDB, which is the most famous debugger for C program
  - Compile your program with -g option (e.g., gcc -g -o proxy proxy.c)
  - Execute GDB with "`gdb ./proxy`"
  - When you encounter trouble, use the following commands:
    - Type "`b [function or line_number]`" where you want to know the problem and type "`run`"
    - "`n`" is for executing next line in your source code
    - "`disp [variable_name]`" enables you to see a value of variable continuously
    - "`print [variable_name]`" temporarily displays a value of the variable
    - For more information, read [2]
- Use `squid` first, to see what happens when we use a real proxy
  - `sudo apt-get install squid`
  - After installation with the above command, the squid is running as Linux daemon, listening port 3128

- Configure Firefox to use proxy information as `127.0.0.1:3128`
- Enter "`ee323.kaist.ac.kr/rfc2616.html`", and see packets with Wireshark
- (In case of C) Use string related functions (Type `man string`)

7. Test Environment

- Language: C or Java
- Test O/S: **Ubuntu 16.04 LTS 64bit**
    - http://releases.ubuntu.com/16.04.4/ubuntu-16.04.4-desktop-amd64.iso
- If you need a VM, download using the following link.
    - http://ee323.kaist.ac.kr/ee323.ova
    - username: ee323, password: ee323
    - Import the VM using VMware or VirtualBox
- **NOTE: We will not consider your compilation and execution problems due to the different OS versions.**

8. Due Date

- 11:59 PM, 8 May, 2018 (Tuesday)

- The website automatically marks the time of the last submission/modification. (The website often becomes unavailable, so please make sure that you submit your assignment early)

- 10% late penalty per day

- Hard deadline: 11:59 PM, 15 May. 2018 (Tuesday)

* We will not receive additional submissions after the hard deadline.

* Exceptions: documented medical/personal emergency.

- Please DO NOT e-mail Prof. Shin or TAs to submit your assignments.

9. Plagiarism

- You can discuss with your colleagues, but you should turn in your own code

* We will run plagiarism detection program on source code

* "Copy and paste" codes will get severely penalized

* If detected, 0 points for all assignments (both providers and consumers)

10. Questions?

- Please use KLMS Q&A board to share your questions (and to save TA's time... but you can still email to TAs)
    - Jinwoo Kim (jinwoo.kim@kaist.ac.kr)
    - Hyeonseong Jo (hsjjo@kaist.ac.kr)
    - Junsik Seo (js0780@kaist.ac.kr)

10. References

[1] RFC2616, “Hypertext Transfer Protocol – HTTP/1.1”, https://tools.ietf.org/html/rfc2616

[2] “GDB Tutorial A Walkthrough with Examples”,
https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf