Project #2: Building an HTTP Proxy

EE323 Spring 2021

ee323@nmsl.kaist.ac.kr

The Ultimate Guide

Primary Project Document

http://bit.ly/ee323-proj2-2021

This slide is based on the document above.

Please refer to this document first if there is any question.

Still ongoing project - we need your help and participation!

You can view and comment on the document directly, so please participate.

(hey, it's a rich source of participation points!)

Objectives

- Help you learn about
  - The HyperText Transfer Protocol (HTTP) v1.0
  - How proxy works
- You will implement **a simple web proxy** that passes requests and data between a web client and a web server.
- You will get an opportunity to configure your web browser to use your personal proxy as a web proxy.

Background: HTTP

- HyperText Transfer Protocol (HTTP)
  - The protocol used for communication on the web
  - Defines how
    - The web browser **requests** resources from a web browser
    - The server **responds**
- In this project, we only deal with HTTP version 1.0
  - Refer to: RFC 1945 https://www.ietf.org/rfc/rfc1945.txt
- Common basic format of request/response messages
  - An initial line
  - Zero or more header lines
  - A blank line (CRLF)
  - An optional message body

Request Message	Response Message
GET /nmsl/ee323.txt HTTP/1.0 Accept: text/* Accept-Language: en	HTTP/1.0 200 OK Content-type: text/plain Content-length: 12 Hello World!

Background: Common HTTP Transactions

1. A client **creates a connection** to the server.
2. The client issues **a request** by sending lines of text to the server.
   - An HTTP method: GET, POST, PUT, etc.
   - A request URI: URL, etc.
   - The protocol version: HTTP/1.0, etc.
   - The message body of the initial request is typically empty.
3. The server sends **a response** message
   - The protocol version: HTTP/1.0, etc.
   - A response status code: 200, 404, etc.
   - A reason phrase: OK, Not Found, etc.
   - Data requested by the client if request was successful
4. Connection closed

Hands-on Experience with telnet

Telnet is an application-layer client-server protocol that can be used to open a command line on a remote computer, typically a server.

1. Open a Unix prompt or a Linux terminal
2. Type “telnet www.google.com 80”
   a. It opens a TCP connection to the server at www.google.com listening on port 80
   b. 80 is the default HTTP port number.
3. Type “GET / HTTP/1.0”
4. Hit the enter twice and see what’s happening

Hands-on Experience with telnet

```
(base) yewon@chris:~$ telnet www.google.com 80
Trying 172.217.25.4...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.0 200 OK
Date: Fri, 26 Feb 2021 05:44:08 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2021-02-26-05; expires=Sun, 28-Mar-2021 05:44:08 GMT; path=/; domain=.google.com; Secure
Set-Cookie: NID=210=Zqr4zugbEPm8NN8hXIdgSo1GB3vhpw01bsNky_dfBLaDr0YDEqEpvdZS6NZ1eczqmUDLX-vB3BmIBFRg7YqhQPvwDqtMCocZFgjG
2WPn5aU1Srk0ZHoasCpBilmkf9UYuE8Vmjxpi3qjjXkyrISxTRl-EjqnhAeT4-J87FVbN9M; expires=Sat, 28-Aug-2021 05:44:08 GMT; path=/;
domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="ko"><head><meta content="text/html; charset
```

HTTP Proxy

Communication between the client and the server (without proxy)

Communication with proxy

Tasks

Build a basic web proxy capable of

1. Accepting HTTP requests from clients,
2. Sending requests to / receiving responses from remote servers,
3. Returning data to clients.

Additionally, your proxy should:

A. Serve multiple clients,
B. Work in real web browsers (Firefox in this project),
C. Support HTTP redirection functionality.

Requirements

Implementation restrictions

- Complete the project in C programming language
- Use C standard libraries & Linux system calls only
  - No other 3rd party libraries
- Should be compiled and run without errors from the Haedong Lounge machine

Include Makefile

- We do not provide any skeleton codes
- $ make all should generate an executable binary file: proxy
- Proxy should take as its first argument a port to listen on.
  - e.g., ./proxy 12345
  - Do not use a hard-coded port number.

Write a report

- Briefly explain your implementation. **No more than 3 pages. PDF only.**

Step 0. Socket Programming

Category	Function Name	Description
Parsing addresses	inet_addr	Convert a dotted quad IP address (e.g., 36.56.0.150) into a 32-bit address.
	gethostbyname	Convert a hostname (e.g., argus.Stanford.edu) into a 32-bit IP address.
	getservbyname	Find the port number associated with a particular service, such as FTP.
Setting up a connection	socket	Get a descriptor to a socket of the given type.
	connect	Connect to a peer on a given socket.
	getsockname	Get the local address of a socket.
Creating a server socket	bind	Assign an address to a socket.
	listen	Tell a socket to listen for incoming connections.
	accept	Accept an incoming connection.
Communicating over the connection	read/write	Read and write data to a socket descriptor.
	htons, htonl / ntohs, ntohl	Convert between host and network byte orders (and vice versa) for 16 and 32-bit values.

Step 1-1. Starting Your Proxy

- Establish a socket connection to **listen** for incoming connections.
  - Your proxy should listen on the port specified from the command line.
  - Wait for incoming client connections.
- Once a client is connected, the proxy should read data from the client and check if the HTTP request is **properly formatted**.
- An invalid request from the client should be answered with an **appropriate error code.**
  - **400 Bad Request** when the request from a client does not have the “Host” header field
  - **400 Bad Request** when it gets an invalid HTTP requests.
    (e.g., methods other than “GET”, different HTTP versions, etc.)
  - **503 Service Unavailable** when the “Host” header fields are invalid.
    (i. Invalid hostname, ii. Invalid IP address)

Step 1-2. Parse the URL

- Once your proxy sees a valid HTTP request, parse the requested URL.
- The proxy needs at most three pieces of information
  1. The requested **host**
  2. The requested **port**
  3. The requested **path**

strtok()	Breaks string into a series of tokens
strcmp() / strncmp()	Compares two strings
strlen()	Calculates the length of a string
strchr()	Searches for the first occurrence of a character in a string
strncpy() / strcpy() / memcpy()	Copies a string

Step 2. Get Data from the Remote Server

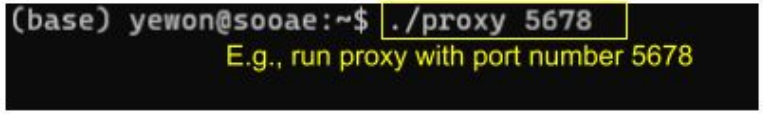
- After parsing the URL, make a connection to the requested host.
  - Use the **appropriate remote port**, or the **default of 80** if none is specified
- The proxy then sends the HTTP request (received from the client) to the remote server.

Step 3. Transfer Response to the Client

- After receiving the response from the remote server, your proxy should transfer the response message to the client via the appropriate socket.
- Close the connection once the transaction is complete.

Testing Your Proxy

- Run your client with the following command:
  - **./proxy <port>**
  - Port: the port number that the proxy should listen on.



- Open the second terminal, and try requesting a page using telnet.

- If your proxy is working correctly, the headers and HTML of the Google homepage should be displayed on your terminal screen.

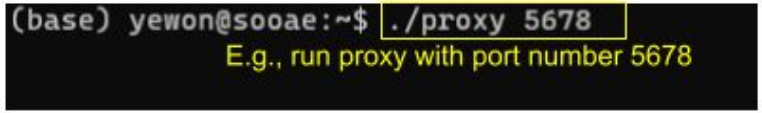
Task A. Serving Multiple Clients

- Your proxy should be able to **receive requests from multiple clients.**
- When multiple clients try to send requests to the proxy at the same time, do not block incoming requests and handle them simultaneously.
  - As you did in project 1

Task B. Working in a Real Web Browser

You will configure **a Firefox web browser** to use your proxy.

1. Turn on your proxy with a port number on the Haedong Lounge machine or your local machine.

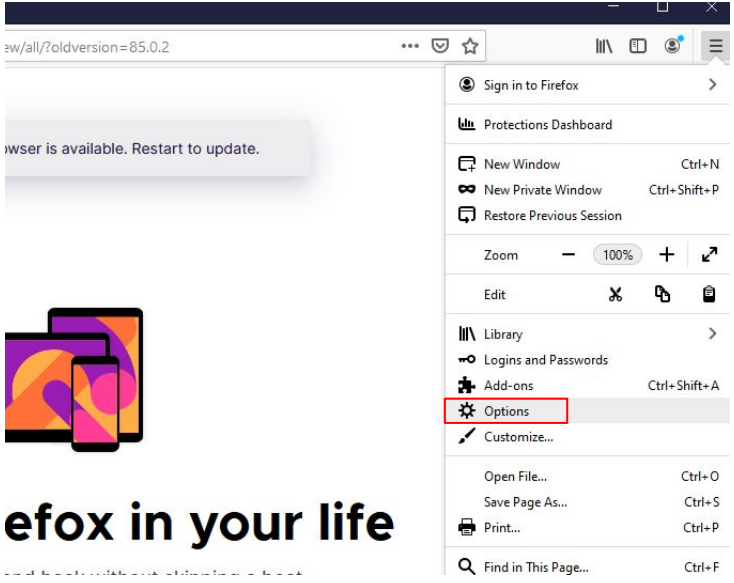


2. Set your web browser to to user your proxy with the port number that your proxy is listening on.
   - See next slides!

Task B. Working in a Real Web Browser

Testing environment: Firefox Version 86.0 (64-bit)

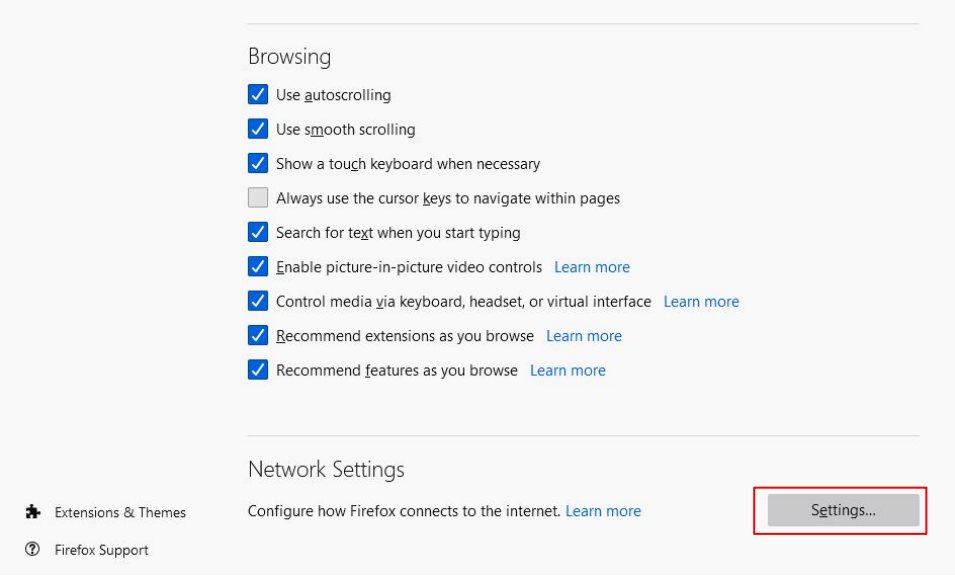
1. Select “Options” from the menu



Task B. Working in a Real Web Browser

Testing environment: Firefox Version 86.0 (64-bit)

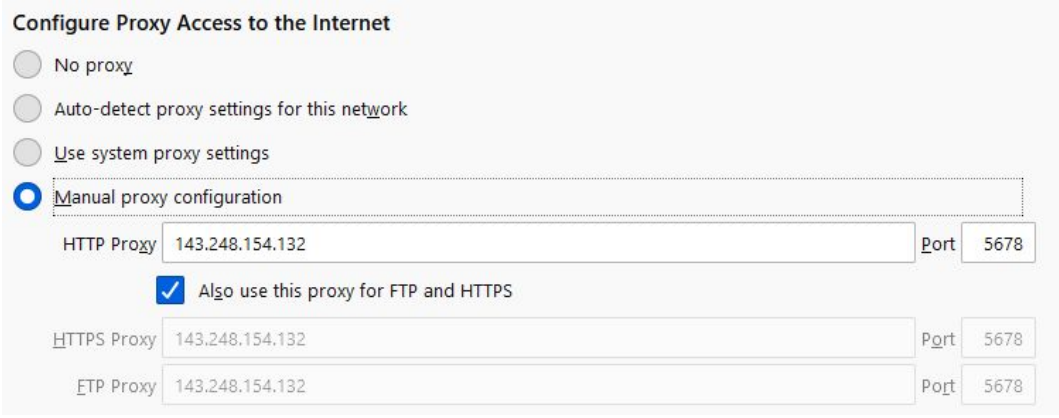
1. Select “Options” from the menu
2. Click the “Settings” button in the “Network Settings” section.



Task B. Working in a Real Web Browser

Testing environment: Firefox Version 86.0 (64-bit)

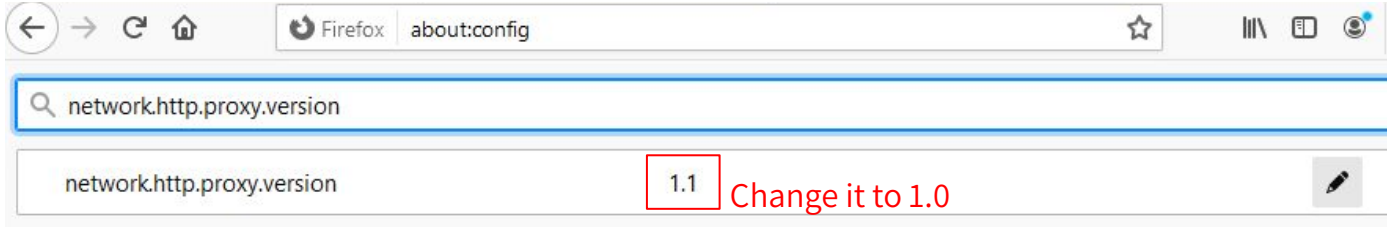
1. Select “Options” from the menu
2. Click the “Settings” button in the “Network Settings” section.
3. Select “Manual Proxy Configuration” from the options available.
    a. In the boxes, enter the host IP address and port where the proxy program is running.



Task B. Working in a Real Web Browser

Testing environment: Firefox Version 86.0 (64-bit)

1. Select “Options” from the menu
2. Click the “Settings” button in the “Network Settings” section.
3. Select “Manual Proxy Configuration” from the options available.
   a. In the boxes, enter the host IP address and port where the proxy program is running.
4. Type about:config in the address bar.
5. Search for “network.http.proxy.version” and set it to 1.0



Task B. Working in a Real Web Browser

Testing environment: Firefox Version 86.0 (64-bit)

1. Select “Options” from the menu
2. Click the “Settings” button in the “Network Settings” section.
3. Select “Manual Proxy Configuration” from the options available.
    a. In the boxes, enter the host IP address and port where the proxy program is running.
4. Type about:config in the address bar.
5. Search for “network.http.proxy.version” and set it to 1.0
6. Test with “http://example.com/”

Task C. Supporting HTTP Redirection Functionality

- Once the proxy gets blacklist pages as a standard input, block the request for those pages and redirect to the warning site (http://warning.or.kr).
  - e.g., ./proxy 5678 < blacklist.txt
- This means, the proxy **should not send requests** to the pages on the blacklist file.
  - Instead, send requests to the warning site, and return the response from the warning site to the client.
- The blacklist text file would look like below

```
http://www.google.com
http://ee.kaist.ac.kr
…
…
```

Figure 9. Example of blacklist.txt

Grading Criteria

This project is worth **4.5%** of your total grade.

1. (20%) Basic Functionality Test : see the next slide
2. (10%) Firefox Test : http://www.example.com/ in Firefox (refer to Task B)
3. (20%) Blacklist Test : Refer to Task C, hidden test cases
4. (40%) Error Handling Test : Hidden test cases
5. (10%) Multiple Client Supporting Test : Hidden test cases
6. (0%) Report: we will use it when there's any grading issue

Any violations will result in a penalty.

- Wrong makefile script
- Wrong file names
- Wrong report format
- etc.

Grading Criteria - Basic Functionality Test

We will check if your proxy works correctly with a small number of major web pages with this testing script.

- After downloading the script, give executable permission to the script:
  - $ chmod +x proxy_tester.py
- Run the script with the following format:
  - $ ./proxy_tester.py ./proxy <port>

Submission

Submit a zip file containing the following:

1. All of the source code for your proxy
2. A Makefile that builds your proxy
3. A report.pdf describing your code and the design decisions you made.

A name of the zip file should be:

- {StudentID}_{Name (in English)}_project2.zip
- e.g., 20211234_TaeckyungLee_project2.zip
-

Due March 30th, Tuesday, 11:55 PM (no late submission)

Submit Here: https://bit.ly/ee323-proj2-2021-submit