CS 39006: Assignment 6

Implement a Simple HTTP Proxy Server

Assignment Date: 27th February 2020

Deadline: 5th March, 2020

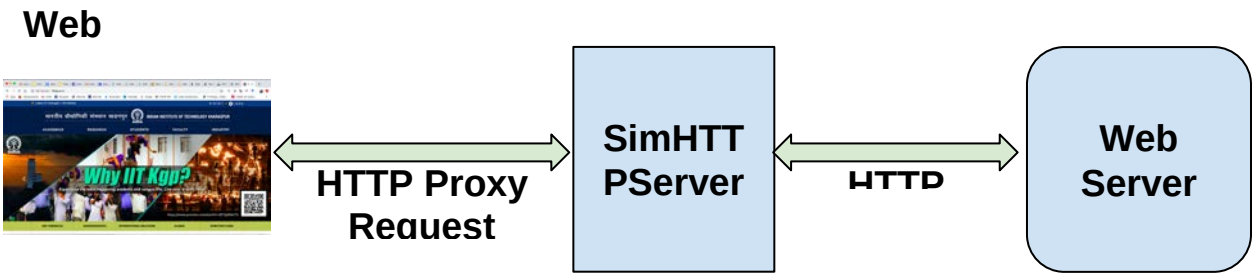
Objective:

The objective of this assignment is to implement a simple HTTP proxy server where incoming connection requests from the browser are first parsed to read the the HTTP headers and then redirected to the destination server. The implementation will help you to understand the functionality of (a) an HTTP proxy, (b) structure of HTTP requests and the headers (c) parsing protocol headers.

Problem Statement:

This assignment is an extension of Assignment 5, where we'll look into the protocol headers to parse the header fields. In this assignment, you have to implement a simple HTTP proxy server over TCP, which parses the incoming HTTP packets and forwards them to the intended destination.

In this assignment you have to build a simple HTTP proxy server which will accept the incoming requests from the browsers, parse the HTTP request headers, and forward the request along with any payload to the target destination. Then the proxy server will receive the HTTP response and forward it back to the browser. We assume that the HTTP proxy server listens to the incoming connection over TCP port 8080.



You can run the simple proxy server as a command line application with the format as follows.

```
$ ./SimHTTPProxy <listen_port>
```

The `listen_port` is the port where the proxy server listens for the incoming connections. In the above example, it is port 8080.

For every connection accepted by the proxy server, it should print a message "`Connection accepted from <client_IP>:<client_Port>`". You can close the proxy server any time by typing "`exit`" command over the console.

A typical runtime display at the console when running the proxy server will be as follows.

```
$./SimHTTPProxy 8080
Connection accepted from 10.5.20.111:52145
GET http://cse.iitkgp.ac.in/ HTTP/1.1, Host: cse.iitkgp.ac.in, Port: 80, Path: /
Connection accepted from 10.5.20.111:52168
GET http://cse.iitkgp.ac.in/support/jquery-3.3.1.min.js HTTP/1.1, Host:
cse.iitkgp.ac.in, Port: 80, Path: /support/jquery-3.3.1.min.js

exit
$
```

Note that the `exit` command above is a command line input that closes the connection.

Here are a few guidelines that you should follow during the implementation.

1. You should not use the `fork()` system call[1]. We want a single-process implementation to make the proxy lightweight. You have to multiplex between three tasks -- (i) accept new incoming connections, (ii) receive data from an incoming connection and forward them to the outgoing connection, (iii) receive command line input from `STDIN` (for the `exit` command). Use the `select()` system call to multiplex between these three tasks.

2. Although you are using the `select()` system call here, you do not know a priori how much data is going to be received over an incoming connection from the browser. Therefore, you cannot estimate how many `recv()` calls you have to execute to receive the complete data. As the `recv()` call is a blocking call, the process will get blocked if there is no data available on an incoming socket. To alleviate this problem, you've to use non-blocking I/O in your program. You can make a socket non-blocking through the `fcntl()` system call as follows: `fcntl(socketfd, F_SETFL, O_NONBLOCK)`

   where `socketfd` is the socket file descriptor that you want to use for non-blocking I/O. You have to be careful in handling the error values returned by the `recv()` call over a non-blocking socket. Check the error numbers to properly handle them in your code.

3. Setting `O_NONBLOCK` will also cause send() calls to become non-blocking. Therefore the error values returned by the send calls need to be handled to ensure that the entire data is sent. For this, an extra buffer may be used to keep the data until it is completely sent. You also need to pass the `socketfd` to the `select` a call to find out when buffer is ready.

[1] You are not allowed to use a fork or any kind of thread. Although you are allowed to handle signals, you are not allowed to use it to get a thread-like environment.

4. You need to careful about the `socketfds` you are going to put to the select call. For example, if you put a `socketfd` which have enough buffer to send some data in `writefd` list of the `select` system call, select call will return immediately without blocking causing unnecessary cpu-spin. Similarly, if you put a `socketfd` which has some data to be read, to the readfd list, the select call will return without waiting. You need to avoid such cases.

5. In Assignment 5, the simple proxy server has redirected every incoming connection to the institute proxy server. However, in this assignment, the HTTP proxy redirects the connection to the target destination, not the institute proxy. For example, if the HTTP request is for http://10.5.20.130:8000/, then the request is forwarded to the IITKGP web server, whereas if the destination of the request is www.google.com, the request is forwarded to Google web server. Therefore, the HTTP proxy needs to parse the HTTP header of the incoming request to know the target server. When the proxy server receives a new incoming request from a browser, it cannot immediately open a TCP connection to the target web server since the destination Host and Port are not known at this point. Therefore, after accepting the connection, the proxy server must first receive the HTTP request headers. NOTE, the entire HTTP request might not be received in a single recv() call, as a result, it must be stored in a temporary buffer. The end of the request headers is marked by two newlines ("\r\n\r\n"). After the headers are received they need to be parsed.

6. Parse the headers to obtain (a) the HTTP request method (eg. GET, POST). (b) the Host address (c) the Port (d) the Path (eg. /about). For our simple HTTP proxy server, we will only handle GET and POST requests and drop other HTTP requests.

7. After the parsing is complete, open a TCP connection to the obtained Host and Port, and forward the entire HTTP request (the headers as well as the payload) to the destination host.

8. The response from the target web server is sent back to the browser as it is, without any processing.

9. Your code should be compatible with a browser. If you are running the proxy server on a machine having the IP address <proxy_ip> and over port <proxy_port>, then you should configure your browser with these IP and port addresses. You should be able to access any website over this proxy server from within the institute.

10. For this assignment, we will provide a virtual machine to each group. You have to run your code (the proxy server) in the virtual machine for testing as well as submission.

Tips and Hints:

1. You need to map each browser socket to each web server socket. Also, you need to keep track if a particular browser socket's corresponding server socket is already created or not. If it is not created, then it indicates that the request headers were not completely received. If it is already created then you just need to forward any data that is being received.
2. For parsing headers, you may use `strstr` and `strtok` library functions.
3. The end of a HTTP Request header is marked with ("\r\n\r\n"), and then follows the HTTP Request payload.

Submission Instruction:

You should write one C program - `simHTTPProxy.c` containing the proxy server program. Compress this file in a zip file with name <roll number1>_<roll number2>_Assignment6.zip. DO NOT submit any additional files. Check that you DO NOT have any hidden file in your submission.

Upload this compressed folder to the Moodle course page by the deadline (5th March, 2020).

You also need to run your http proxy server in your VM at the port 8080. It will be used during evaluation.