CS 359: Assignment 8

Implement a Simple Application Proxy Server

Assignment Date: 22-03-2021

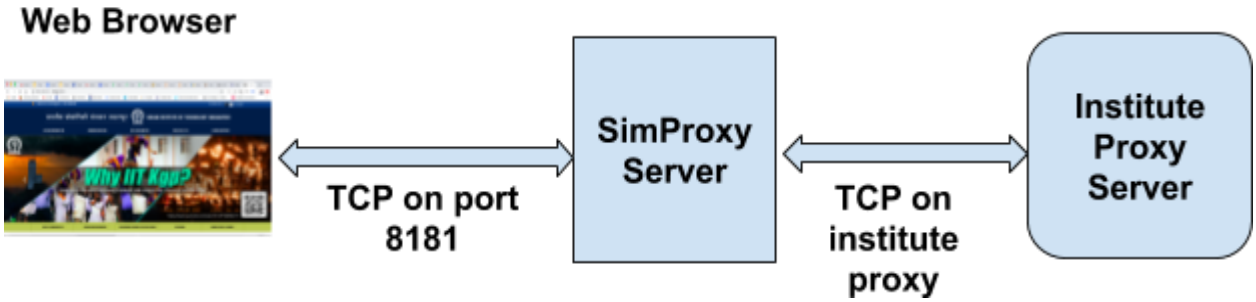
Deadline: 29-03-2021

Objective:

The objective of this assignment is to implement a simple proxy server over TCP where incoming connection requests from the browser are redirected to the institute proxy through a local server over TCP. The implementation will help you to understand the functionality of (a) an application proxy, (b) the `select()` system call used for servicing multiple requests over different sockets, and the (c) `fctl()` system call used for non blocking I/O.

Problem Statement:

In this assignment, you have to implement a simple proxy server over TCP. Typically, a HTTP/HTTPS proxy server parses the incoming HTTP/HTTPS packets and forwards them to the intended destination. In this simple proxy server, you don't have to parse the application data packets; whatever the simple proxy server receives in its incoming connection, it will forward the same to our institute proxy server. So, the system architecture of the simple proxy server is as follows. We assume that the simple proxy server listens the incoming connection over TCP port 8181.



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

```
$ ./SimProxy <listen_port> <institute_proxy_IP> <institute_proxy_port>
```

The `listen_port` is the port where the proxy server listens for the incoming connections. In the above example, it is port 8181.

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 by typing "`exit`" command over the console.

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

```
$./SimProxy 8181 172.16.2.30 8080
Proxy running on port 8181. Forwarding all connections to 172.16.2.30:8080
Connection accepted from 172.16.1.1:21344
Connection accepted from 10.134.2.33:51123
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. 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, we'll use non-blocking I/O in our 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. 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.

Submission Instruction:

You should write one C program - `proxy.c` containing the proxy server program. Compress this file in a zip file with name <roll number1>_<roll number2>_Assignment8.zip.