CPTS 360: CPTS 360: Programming Assignment 4

Building a Web Proxy

Notes:

- This is an individual assignment.
- You should complete the assignment on a Linux environment (either a dedicated machine or VM). Using WSL (Windows Subsystem for Linux) on Windows is NOT recommended.
- Your submission will be tested on a Linux environment. You will NOT receive any points if your submitted program does not work on a Linux system.
- Start EARLY. The assignment may take more time than you anticipated.
- Read the entire document carefully before you start working on the assignment.
- The code and other answers you submit MUST be entirely your own work, and you are bound by the WSU Academic Integrity Policy (`https://www.communitystandards.wsu.edu/policies-and-reporting/academic-integrity-policy/`).
- You MAY consult with other students about the conceptualization of the tasks and the meaning of the questions, but you MUST NOT look at any part of someone else's solution or collaborate with anyone.
- You may consult published references or search online archives, provided that you appropriately cite them in your reports/programs.
- The use of artificial intelligence (AI)–generated texts/code snippets must be CLEARLY DISCLOSED, including the prompt used to generate the results and the corresponding outputs the tool(s) provide.

Good Luck!

1 Introduction

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

Proxies are useful for many purposes. Sometimes proxies are used in firewalls, so that browsers behind a firewall can only contact a server beyond the firewall via the proxy. Proxies can also act as anonymizers: by stripping requests of all identifying information, a proxy can make the browser anonymous to Web servers. Proxies can even be used to cache web objects by storing local copies of objects from servers then responding to future requests by reading them out of its cache rather than by communicating again with remote servers.

In this lab, you will write a simple HTTP proxy. You will set up the proxy to accept incoming connections, read and parse requests, forward requests to web servers, read the servers' responses, and forward those responses to the corresponding clients. The lab will involve learning about basic HTTP operation and how to use sockets to write programs that communicate over network connections.

2 Lab Tasks: Implementing a Web Proxy

The first step is implementing a basic sequential proxy that handles HTTP/1.0 GET requests. Other requests type, such as POST, are strictly optional.

When started, your proxy should listen for incoming connections on a port whose number will be specified on the command line. Once a connection is established, your proxy should read the entirety of the request from the client and parse the request. It should determine whether the client has sent a valid HTTP request; if so, it can then establish its own connection to the appropriate web server then request the object the client specified. Finally, your proxy should read the server's response and forward it to the client.

2.1 HTTP/1.0 GET requests

When an end user enters a URL such as `http://www.wsu.edu/eecs/index.html` into the address bar of a web browser, the browser will send an HTTP request to the proxy that begins with a line that might resemble the following:

```
GET http://www.wsu.edu/eecs/index.html HTTP/1.1
```

In that case, the proxy should parse the request into at least the following fields: the hostname, `www.wsu.edu`; and the path or query and everything following it, `/eecs/index.html`. That way, the proxy can determine that it should open a connection to `www.wsu.edu` and send an HTTP request of its own starting with a line of the following form:

```
GET /eecs/index.html HTTP/1.0
```

Note that all lines in an HTTP request end with a carriage return, `‘\r’`, followed by a newline, `‘\n’`. Also important is that every HTTP request is terminated by an empty line: `"\r\n"`.

You should notice in the above example that the web browser's request line ends with `HTTP/1.1`, while the proxy's request line ends with `HTTP/1.0`. Modern web browsers will generate HTTP/1.1 requests, but your proxy should handle them and forward them as HTTP/1.0 requests.

It is important to consider that HTTP requests, even just the subset of HTTP/1.0 GET requests, can be incredibly complicated. The textbook describes certain details of HTTP transactions, but you should refer to RFC 1945 for the complete HTTP/1.0 specification. Ideally your HTTP request parser will be fully robust according to the relevant sections of RFC 1945, except for one detail: while the specification allows for multiline request fields, your proxy is not required to properly handle them. Of course, your proxy should never prematurely abort due to a malformed request.

2.2 Request Headers

The important request headers for this lab are the `Host`, `User-Agent`, `Connection`, and `Proxy-Connection` headers:

- Always send a `Host` header. While this behavior is technically not sanctioned by the HTTP/1.0 specification, it is necessary to coax sensible responses out of certain Web servers, especially those that use virtual hosting.

  The Host header describes the hostname of the end server. For example, to access http://www.wsu.edu/eecs/index.html, your proxy would send the following header:

  ```
  Host: www.wsu.edu
  ```

  It is possible that web browsers will attach their own Host headers to their HTTP requests. If that is the case, your proxy should use the same Host header as the browser.

- You *may* choose to always send the following User-Agent header:

  ```
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:10.0.3)
      Gecko/20120305 Firefox/10.0.3
  ```

  The header is provided on two separate lines because it does not fit as a single line in the writeup, but your proxy should send the header as a single line.

  The User-Agent header identifies the client (in terms of parameters such as the operating system and browser), and web servers often use the identifying information to manipulate the content they serve. Sending this particular User-Agent: string may improve, in content and diversity, the material that you get back during simple telnet-style testing.

- Always send the following Connection header:

  ```
  Connection: close
  ```

- Always send the following Proxy-Connection header:

  ```
  Proxy-Connection: close
  ```

  The Connection and Proxy-Connection headers are used to specify whether a connection will be kept alive after the first request/response exchange is completed. It is perfectly acceptable (and suggested) to have your proxy open a new connection for each request. Specifying close as the value of these headers alerts web servers that your proxy intends to close connections after the first request/response exchange.

For your convenience, the values of the described User-Agent header is provided to you as a string constant in proxy.c.

Finally, if a browser sends any additional request headers as part of an HTTP request, your proxy should forward them unchanged.

2.3 Port Numbers

There are two significant classes of port numbers for this lab: HTTP request ports and your proxy's listening port.

The HTTP request port is an optional field in the URL of an HTTP request. That is, the URL may be of the form, http://www.wsu.edu:8080/eecs/index.html, in which case your proxy should connect to the host `www.wsu.edu` on port 8080 instead of the default HTTP port, which is port 80. Your proxy must properly function whether or not the port number is included in the URL.

The listening port is the port on which your proxy should listen for incoming connections. Your proxy should accept a command line argument specifying the listening port number for your proxy. For example, with the following command, your proxy should listen for connections on port 15213:

```
sh> ./proxy 15213
```

You may select any non-privileged listening port (greater than 1,024 and less than 65,536) as long as it is not used by other processes. Since each proxy must use a unique listening port and many people will simultaneously be working on each machine, the script `port-for-user.pl` is provided to help you pick your own personal port number. Use it to generate port number based on your user ID:

```
sh> ./port-for-user.pl cougs
cougs: 45806
```

The port, p, returned by `port-for-user.pl` is always an even number. So if you need an additional port number, say for the `Tiny server`, you can safely use ports p and $p+1$.

Please don't pick your own random port. If you do, you run the risk of interfering with another user.

2.4 Robustness

As always, you must deliver a program that is robust to errors and even malformed or malicious input. Servers are typically long-running processes, and web proxies are no exception. Think carefully about how long-running processes should react to different types of errors. For many kinds of errors, it is certainly inappropriate for your proxy to immediately exit.

Robustness implies other requirements as well, including invulnerability to error cases like segmentation faults and a lack of memory leaks and file descriptor leaks.

3 Testing and Debugging

We do not provide any sample inputs or a test program to test your implementation. You will have to come up with your own tests and perhaps even your own testing harness to help you debug your code and decide when you have a correct implementation. This is a valuable skill in the real world, where exact operating conditions are rarely known and reference solutions are often unavailable.

Fortunately, there are many tools you can use to debug and test your proxy. Be sure to exercise all code paths and test a representative set of inputs, including base cases, typical cases, and edge cases.

3.1 Tiny Web Server

The GitHub repository contains source code for the Tiny web server (discussed in class). The Tiny web server will be easy for you to modify as you see fit. It is also a reasonable starting point for your proxy code.

3.2 `telnet`

As in the class, you can use `telnet` to open a connection to your proxy and send it HTTP requests. For example, if Tiny is listening on port 15213, and proxy is listening on port 15214, then you can request a page from Tiny via your proxy using `telnet`:

```
sh> telnet 127.0.0.1 15214
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
GET http://localhost:15213/home.html HTTP/1.0

HTTP/1.0 200 OK
Server: Tiny Web Server
...
...
```

3.3 `curl`

You can use `curl` to generate HTTP requests to any server, including your own proxy. It is an extremely useful debugging tool. For example, if your proxy and Tiny are both running on the local machine (127.0.0.1), Tiny is listening on port 15213, and proxy is listening on port 15214, then you can request a page from Tiny via your proxy using the following `curl` command:

```
sh> curl -v --proxy http://localhost:15214 http://localhost:15213/home.html
* About to connect() to proxy localhost port 15214 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 15214 (#0)
> GET http://localhost:15213/home.html HTTP/1.1
> User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu)...
> Host: localhost:15213
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Server: Tiny Web Server
< Content-length: 120
< Content-type: text/html
<
<html>
<head><title>test</title></head>
<body>
<img align="middle" src="wsu.png">
CPTS360
</body>
</html>
* Closing connection #0
```

3.4 netcat

netcat, also known as nc, is a versatile network utility. You can use netcat just like telnet, to open connections to servers. Hence, imagining that your proxy were running on cougs using port 12345 you can do something like the following to manually test your proxy:

```
    sh> nc cougs.eecs.wsu.edu 12345
    GET http://www.wsu.edu/eecs/index.html HTTP/1.0

    HTTP/1.1 200 OK
    ...
```

In addition to being able to connect to Web servers, netcat can also operate as a server itself. With the following command, you can run netcat as a server listening on port 1235:

```
    sh> nc -l 1235
```

Once you have set up a netcat server, you can generate a request to a phony object on it through your proxy, and you will be able to inspect the exact request that your proxy sent to netcat.

For example, if a netcat server is running at port 1235 and the proxy is at port 1234, we can generate request to the server through our proxy using the following command:

```
sh> pkill proxy
sh> ./proxy 1234
sh> curl -v --proxy 127.0.0.1:1234 127.0.0.1:1235/$(date +"%T").dat
```

This generates request a (pseudo) .dat file indexed by the current time. You may expect the following output:

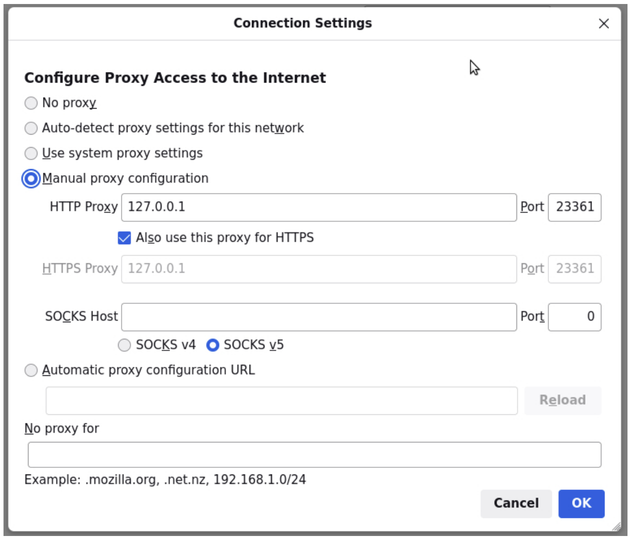
```
*   Trying 127.0.0.1:1234...
* Connected to 127.0.0.1 (127.0.0.1) port 1234 (#0)
> GET http://127.0.0.1:1235/10:14:54.dat HTTP/1.1
> Host: 127.0.0.1:1235
> User-Agent: curl/7.74.0
> Accept: */*
> Proxy-Connection: Keep-Alive
```

3.5 Web Browsers

Eventually, you should test your proxy using the *most recent version* of Mozilla Firefox. Visiting About Firefox will automatically update your browser to the most recent version.

To configure Firefox to work with a proxy, visit (also see Figure 1):

```
    Preferences>Advanced>Network>Settings
```



Connection Settings

Configure Proxy Access to 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 23361

☑ Also use this proxy for HTTPS

HTTPS Proxy 127.0.0.1 Port 23361

SOCKS Host Port 0

○ SOCKS v4 ● SOCKS v5

○ Automatic proxy configuration URL

No proxy for

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

Figure 1: Configuring Firefox with a proxy running at `localhost` (127.0.0.1) and port 23361.

It will be very exciting to see your proxy working through a real Web browser. Although the functionality of your proxy will be limited, you will notice that you are able to browse the vast majority of websites through your proxy.

Visit few websites, for example `www.google.com`, `www.bing.com`, and `www.wiki.com` and report which ones can be accessible with your proxy.

4 Hints

- Chapters 10-12 of the textbook (CSAPP) contain useful information on system-level I/O, network programming, HTTP protocols, and concurrent programming.
- RFC 1945 (`http://www.ietf.org/rfc/rfc1945.txt`) is the complete specification for the HTTP/1.0 protocol.
- As discussed in class, using standard I/O functions for socket input and output are a problem. Instead, we recommend that you use the Robust I/O (RIO) package, which is provided in the `csapp.c` file in the handout directory.
- The error-handling functions provided in `csapp.c` are not appropriate for your proxy because once a server begins accepting connections, it is not supposed to terminate. You'll need to modify them or write your own.
- You are free to add new files or modify the provided files any way you like. Of course, adding new files will require you to update the provided `Makefile`.
- Sometimes, calling `read` to receive bytes from a socket that has been prematurely closed will cause `read` to return `-1` with `errno` set to `ECONNRESET`. Your proxy should not terminate due to this error.
- Remember that not all content on the web is ASCII text. Much of the content on the web is binary data, such as images and video. Ensure that you account for binary data when selecting and using functions for network I/O.

- Forward all requests as HTTP/1.0 even if the original request was HTTP/1.1.

5 Deliverable

- Complete the implementation of the proxy.

  [90 Points]

  - The point split for your implementation is as follows (see below for details):
    - **80 points** for correctness and
    - **10 points** for good code organization, indentation, and proper comments.
- A PDF document containing a summary of how you implemented your proxy. **You should include your report in the GitHub repository.** See below for the formatting and content requirements.

  [10 Points]

  - Your report should be **no more than two pages**; use standard **letter paper size** (8½ by 11 inches) with **11 points Times New Roman font** and **1-inch margin**.
  - Your PDF filename should be `pa4_lastname.pdf`, where `lastname` is your surname.
  - Include your **full name** and **WSU ID** inside the report.
  - Include your report on the GitHub repository.
  - We will **deduct 5 points** if the filename/contents do not follow this convention and/or you use a different font/margin than that listed above.
  - Your report should include the answer to the following question with a **clear heading**.
    - **[5 Points]** While browsing using Firefox, you may find some URLs can not be accessible using your proxy. Why? Explain in detail.

5.1 Grading Notes

Our course staff will check the functional correctness of your code. See below for additional details.

What to Expect for Full Points

1. We may check for a communication between your proxy (running on 127.0.0.1) and Tiny web server using the `telnet` utility as demonstrated in the class. For example, we may check if we can initiate a GET request using `telnet`.
2. We may check for the correct workflow of the `curl` command as mentioned in Section 3.3.
3. We may check for the correct workflow of the `netcat` command as mentioned in Section 3.4.
4. We may check for the correct workflow of browsing websites using Firefox configured with your proxy and port (see Section 3.5). We may browse the following URLs: `www.google.com`, `www.bing.com`, and `www.wiki.com` (and a few others) and show which one is loading with your proxy.

Important Note

In our evaluation, we may randomly check for any utilities listed in Section 3. During your own testing, you should check whether your connection works for all the utilities (e.g., `telnet`, `curl`, `netcat`, and browser).

6 Submission Guidelines

Commit and push your work (source and report) on the GitHub Classroom Programming Assignment 4 repository. **Make sure you can successfully push commits on GitHub Classroom — *last-minute excuses will not be considered***. Paste the URL of your GitHub repo into the corresponding assignment section of Canvas. Check the course website for additional details.

Note: Your work on GitHub Classroom will not be considered for grading unless you submit the link to the GitHub repository URL on Canvas. Hence, **You will NOT receive any points if you fail to submit your URL on Canvas.** Submissions received after the due date will be graded based on the late submission policy as described in the course syllabus.

Acknowledgement. This assignment was initially developed by the team members of the course textbook (CSAPP) and has been customized for CPTS 360 by the instructor.