CS 118: Computer Network Fundamentals - Project 1

Jair Hinojosa, Kyle Romero
904771873, 204747283

Date Due: April 27, 2018

Introduction

The purpose of this project is to design and develop a Web server that is able to parse and respond to HTTP file requests. From this project, we gained a better understanding of the HTTP protocol and general Web communications between a client and server.

Design Overview

To start the server executable, one must specify the port number to be used to listen for HTTP requests on the command line. On start-up, our server uses these arguments to configure the TCP socket that is used to setup the server connection. That being said, our high-level design begins by configuring a socket for the TCP protocol.

A loop is used to handle all the requests made to server. Once a request is made by a client, typically a browser or another terminal, the HTTP request is printed to the console and is parsed by a variety of helper functions in order to serve the request. For example, the function getFilename() is used to obtain the filename from the message and the function getFiletype() is used to extract and analyze the file extension. Once this information is determined based on the HTTP request, the server attempts to open the filename requested. If the filename is not found, a HTTP response with a 404 Error and a HTML file displaying a 404 Error is presented to the client. If the filename is found, the file is opened and the sendResponse function is invoked.

The sendResponse() function compiles the HTTP response header using the information that was extracted from the HTTP request and the files in the server's working directory. The textbook was used to determine how we should format our HTTP response. Once compiled, the response header is sent to the client. Then, the sendResponse() function invokes the sendFile() function to send the body of the HTTP response, the requested file, in fixed-size chunks. After the sendResponse() function returns, the connection is closed and the server waits for another request.

Difficulties Faced

We faced a variety of issues while dealing with HTTP requests. For example, we faced an issue when we downloaded an HTML file from a random Wikipedia page to test on our server. Whenever our client requested that HTML file, our server would hang and the client would freeze. For quite some time, we thought that there was a bug in our server implementation. After an hour or so, we determined that the HTML file contained internal PHP requests that were being passed on to the server. Since our HTTP server was not designed to handle PHP requests, this was the root of our problems.

Furthermore, we faced a few challenges in getting our server to handle requests from both types of clients—a web browser and a terminal. We initially designed our server to handle HTTP requests from a web browser and then adapted to include support for 'curl'. This design choice caused issues with handling spaces in the filename because the web browser adds '%20' characters where spaces are in an HTTP request. The 'curl' command does not do this and so

we had to make changes to some of our helper functions in order to make our server more robust.

Manual for Compiling and Running Our Program

1 - Our project submission should be named: 204747283.tar.gz. In order to untar this file, one should enter the following command in the directory where the tarball is saved:

```
tar -zxvf 204747283.tar.gz
```

2 - Compilation of our program is handled by our Makefile. One is able to compile our program by typing the following command:

```
make
```

This should create an executable file named 'server'.

3 - Running our program can be done by entering a command of the following form:

```
./server <port number>
```

For example, to use port number 1234 the following command is used

```
./server 1234
```

4 - At this point the server will be listening to connections on the specified port number and client requests can be made using the 'curl' command or a web browser.

You can press Ctrl-C to shutdown the server.

Sample Outputs

For all sample outputs, we have assumed that the user has downloaded our submission tarball, and completed the steps above (i.e. the server should be running on some port number).

For all of the following tests, we used port number 1234.

1 - Small Binary File

For this test, we found a small binary file that we called 'small_binary'. First, we ensured that our server was still running and then opened another terminal window and entered the following commands:

```
curl http://localhost:1234/small_binary > test
diff test small_binary
```

The second command prints nothing, confirming that the test was successful. The following is the output for our server:

```
GET /small_binary HTTP/1.1
Host: localhost:1234
User-Agent: curl/7.47.0
Accept: */*
```

small_binary
HTTP/1.1 200 OK
Connection: close
Date: Fri,27 Apr 2018 20:48:24 GMT
Server: Ubuntu
Last-Modified: Fri,27 Apr 2018 19:00:12 GMT
Content-Length: 264749
Content-Type: application/octet-stream

2 - Large Binary File

For this test, we found a large binary file that we called 'large_binary'. First, we ensured that our server was still running and then opened another terminal window and entered the following commands:

curl http://localhost:1234/large_binary > test
diff test large_binary

The second command prints nothing, confirming that the test was successful. The following is the output for our server:

GET /large_binary HTTP/1.1
Host: localhost:1234
User-Agent: curl/7.47.0
Accept: */*

large_binary
HTTP/1.1 200 OK
Connection: close
Date: Fri,27 Apr 2018 20:54:49 GMT
Server: Ubuntu
Last-Modified: Fri,27 Apr 2018 19:02:38 GMT
Content-Length: 104857600
Content-Type: application/octet-stream

3 - HTML File on Browser

For this test, we found an HTML file that we called 'test.html'. First, we ensured that our server was still running and then opened a web browser. Then we entered

'http://localhost:1234/test.html' as the URL. And then we ensured that the browser displayed the HTML webpage properly. The following is the output from our server:

GET /test.html HTTP/1.1
Host: localhost:1234
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: __utma=111872281.362423834.1524361094.1524550849.1524591074.4;
__utmz=111872281.1524361094.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)
Connection: keep-alive
Upgrade-Insecure-Requests: 1

test.html
HTTP/1.1 200 OK
Connection: close
Date: Fri,27 Apr 2018 23:55:06 GMT
Server: Ubuntu
Last-Modified: Fri,27 Apr 2018 19:10:30 GMT
Content-Length: 156
Content-Type: text/html

4 - Image on Browser

For this test, we found a JPEG file that we called 'test.html.jpeg'. First, we ensured that our server was still running and then opened a web browser. Then we entered 'http://localhost:1234/test.html.jpeg' as the URL. Then, we ensured that the browser displayed the JPEG properly. The following is the output from our server:

GET /test.html.jpeg HTTP/1.1
Host: localhost:1234
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: __utma=111872281.362423834.1524361094.1524550849.1524591074.4;
__utmz=111872281.1524361094.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)
Connection: keep-alive

```
Upgrade-Insecure-Requests: 1


test.html.jpeg
HTTP/1.1 200 OK
Connection: close
Date: Fri,27 Apr 2018 23:57:32 GMT
Server: Ubuntu
Last-Modified: Fri,27 Apr 2018 19:05:50 GMT
Content-Length: 5667
Content-Type: image/jpeg
```

5 - Filename with Space in Browser

For this test, we found an HTML file that we called 'spaces spaces.htm'. First, we ensured that our server was still running and then opened a web browser. Then we entered 'http://localhost:1234/spaces spaces.htm' into the URL line. Then we ensured that the browser displayed the HTML properly. The following is the output from our server:

```
GET /spaces%20spaces.htm HTTP/1.1
Host: localhost:1234
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: __utma=111872281.362423834.1524361094.1524550849.1524591074.4;
__utmz=111872281.1524361094.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)
Connection: keep-alive
Upgrade-Insecure-Requests: 1


spaces spaces.htm
HTTP/1.1 200 OK
Connection: close
Date: Fri,27 Apr 2018 23:59:41 GMT
Server: Ubuntu
Last-Modified: Fri,27 Apr 2018 19:06:05 GMT
Content-Length: 115
Content-Type: text/html
```