ECE435: Network Engineering – Homework 2
http, HTML, webserver

Due: Thursday 2 February 2023, 3:30pm

1. **Write a simple webserver** (7 pts)

   The first part of this homework is to create a sockets program that acts as a simple webserver.

   (a) Download the code from:
   `https://web.eece.maine.edu/~vweaver/classes/ece435/ece435_hw2_code.tar.gz`

   (b) Unpack the files:
   `tar -xzvf ece435_hw2_code.tar.gz`

   (c) Build the C files:
   ```
   cd ece435_hw2_code
   make
   ```

   (d) You can use your code from HW#1 as a basis, or you can use the provided `webserver.c` file. If you use your own code, just be sure it is copied over `webserver.c`

   (e) Modify the code so it listens on port 8080.

   (f) When a connection comes in, read the entire incoming request from the file descriptor into a char buffer. You can ignore most of the request; the important thing your code will need to do is find the line that has the http `GET` request in it.

   Note, for this assignment, feel free to use a large buffer to store the incoming header, and simply error out if it is not large enough. The "proper" way would be to use `malloc()` and friends to autosize the buffer, but that is pretty advanced low-level C and has only caused pain in past years when students tried to implement it.

   (g) Find the line with `GET` in it and obtain the filename that is being requested (it follows immediately after GET). The `strstr()` function may be of use here. Be sure to drop any leading "/" too so we can serve files out of the current directory.

   (h) Open the requested file. If it exists, open the file and write it out to the file descriptor. You can use `open()` and `read()`. Or you can use the buffered `fopen()` and such, but in that case you may want to read up on the `fdopen()` function.

   (i) You will also need to send proper http headers (See below)

      i. Date – send the date, use the format as shown below. You can look into the `time()`, `gmtime()`, and `strftime()` functions.

      ii. Last-modified time: You can find the last-modified time for a file as well as the size of the file (for `Content-Length:`) using the `stat()` function.

      iii. Content-Type you can base this on the extension of the requested file. Support the following common MIME types: .html = "text/html" .txt = "text/plain" .png = "image/png" .jpg = "image/jpeg"

iv. You will want to write out this buffer containing the header data to the socket filedescriptor. One way to do this is to use `sprintf()` to print everything to a buffer, then use `write()` to write it out.

(j) Put a blank like to indicate the end of the header.
Note that the http specification calls for carriage-returns and linefeeds (\r\n), not just linefeeds (\n).

(k) Finally send the contents of the file. It is probably best to just do this in a loop, loading in a chunk from disk, writing that chunk to the socket, and repeating until the file is done. This is much simpler than trying to read the whole thing into memory then sending it all one transaction.

Also be sure you can send binary files (which might have zeros in them) so you can't rely on `strlen()` or the `printf()` functions to print the file contents.

A sample header is shown below:

```
HTTP/1.1 200 OK\r\n
Date: Fri, 08 Sep 2017 04:56:25 GMT\r\n
Server: ECE435\r\n
Last-Modified: Fri, 08 Sep 2017 04:31:47 GMT\r\n
Content-Length: 100\r\n
Content-Type: text/html\r\n
\r\n
```

This should be immediately followed by the file contents.

(l) If the requested filename is not found, return a 404 not found error.

```
HTTP/1.1 404 Not Found\r\n
Date: Fri, 08 Sep 2017 04:56:25 GMT\r\n
Server: ECE435\r\n
Content-Length: 100\r\n
Connection: close\r\n
Content-Type: text/html; charset=iso-8859-1\r\n
\r\n
<html><head>\r\n
<title>404 Not Found</title>\r\n
</head><body>\r\n
<h1>Not Found</h1>\r\n
<p>The requested URL was not found on the server.<br />\r\n
</p>\r\n
</body></html>\r\n
```

(m) Log all requests received to stdout (just use `printf()`). Print the name of each file requested.

(n) Be sure to comment your code!

(o) To test, start the server and connect from a web browser on the same machine to
`http://localhost:8080/test.html`
and the webpage should appear.

You can use any browser and it should work.

Note, some browsers (notably chromium) will try to work around buggy servers, so be sure you are sending valid data even if it views properly in your browser.

If you're remotely logging into a Pi or similar you can install and use the text-mode `lynx` browser to test. You can also debug headers using `wget -S` to make sure your server is sending what you expect.

If you have your test system on the network you can possibly connect from a remote machine if you know the IP address. Sometimes things like firewalls can get in the way though.

(p) Issues you might encounter:

- If the browser gets your html and displays it, but keeps spinning, it's probably because you told it you were sending more in content length than you delivered. Be sure they match up!
- Be sure to drop the leading / in a URL when working out the filename to open.
- Be sure you handle a url with no file specified like http://localhost:8080/
  Usually most webservers default to returning a file called index.html in that case.
- Be sure you don't crash if there's an extremely long or unusual URL.

2. **Answer the following questions in the README file** (2pts)

(a) You use a browser to try to connect to a webserver.

i. The webserver returns error code 404. What is wrong?

ii. The webserver returns error code 418. What is wrong?

iii. The webserver returns error code 451. What is wrong?

(b) You use telnet to connect to a web server, and this is the top part of the results you get:

```
telnet www.maine.edu 80
Trying 130.111.28.85...
Connected to lv-o-wpc.its.maine.edu.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.maine.edu

HTTP/1.1 302 Moved Temporarily
Server: nginx/1.16.1
Date: Sat, 06 Feb 2021 01:18:33 GMT
Content-Type: text/html
Content-Length: 145
Connection: keep-alive
Location: https://www.maine.edu/

<html>
<head><title>302 Found</title></head>
<body>
<center><h1>302 Found</h1></center>
<hr><center>nginx/1.16.1</center>
</body>
</html>
```

i. What web-server was the maine.edu website running?

ii. Why was the webserver redirecting instead of giving us the requested web page?

3. **Something cool** (1pt)

   Modify the provided test.html to be more interesting. Do one or or more of the following:

   (a) Change the title to "ECE435 Homework 2"

   (b) Add a size-1 header, centered, saying "ECE435 Homework 2"

   (c) Put your name, as a size-2 header, centered.

   (d) Have a horizontal rule (horizontal line)

   (e) Have a clickable link to a website of your choice (although, keep it safe for work please). Have some surrounding text that briefly describes it, and bolds or italicizes at least one word.

   (f) Add colors

   (g) Add a table

   (h) Add an ordered list

   (i) Go overboard and have Javascript

4. **Submit your work**

   - Please edit the README file to include your name.
     Also put your answers to the questions there.
   - Run `make submit` which will create a `hw2_submit.tar.gz` file containing `README`, `Makefile`, `webserver.c` and `test.html`.
     You can verify the contents with `tar -tzvf hw2_submit.tar.gz`
   - e-mail the `hw2_submit.tar.gz` file to me (vincent.weaver@maine.edu) by the homework deadline. Be sure to send the proper file!