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

Due: Thursday 1 February 2024, 12:30pm

1. **Write a simple webserver**

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

   (a) **Get the code building**

   - Download the code from:
     `https://web.eece.maine.edu/~vweaver/classes/ece435/ece435_hw2_code.tar.gz`
   - Unpack the files:
     `tar -xzvf ece435_hw2_code.tar.gz`
   - Build the C files:
     ```
     cd ece435_hw2_code
     make
     ```
   - The provided code is more or less the same code from HW#1. If you prefer you can use your code from HW#1 instead, just be sure it is copied over `webserver.c`

   (b) **Receive request from server, parse GET request (2pts)**

   - Modify the code so it listens on port 8080.
   - 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 (i.e. bump the size from 256 bytes to something larger like 4k or 8k) 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.
   - 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.
     Note that our "web root" that we are serving files from is the current directory so you will want to strip any leading "/" characters off the filename before trying to open it. Also detect if the filename is just a plain "/" and replace it with `index.html`

   (c) **Construct the proper http response header (2pts)**

   A sample header is shown below (note, don't actually print `\r\n` this is just showing you need both a carriage return and linefeed at the end of the line):

   ```
   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
```

- I recommend you try to generate this and print it to the screen to validate your code is generating the right values before sending it down the socket.
- There are various ways to generate this, the most common one people use is using `sprintf()` to create a string with the whole header, then later using `write()` to write the whole thing out the socket file descriptor at once.
- The first line should be the `HTTP/1.1` response with the status code.
- Next send the Date – use the format from the example. You can look into the `time()`, `gmtime()`, and `strftime()` functions.
- Last-modified time: You can use the `stat()` function to get this information about a file, you probably want the `st_mtime` field from that stat struct for this.
- Content-Length: it's critical to get this value right or the web-browser might get stuck. You can also get this value from `stat()` in the `st_size` field.
- Content-Type: support the following list, you can base this on the extension of the requested file. I'd suggest hard-coding "text/html" at first and adding support for the others once everything else is working.
  - .html = "text/html"
  - .txt = "text/plain"
  - .png = "image/png"
  - .jpg = "image/jpeg"
- Be sure after the header is done there's a blank line with just \r\n as this indicates the end of the header.

(d) **Send header and file data to the browser (2pt)**

- Write the header out to the socket file descriptor.
- Open the requested file (be sure to check for errors). You can use `open()` for this.

  If for some reason you prefer the `fopen()` class of functions for I/O instead, you can do that. You might have to look up how to use `fdopen()` in order to get a FILE pointer to your socket in this case.
- Write the contents of the open file to the file descriptor.

  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 in one transaction.
  Also your code should handle binary files (which might have zeros in them) so you can't rely on `strlen()` to calculate the size or `sprintf()` to output the file contents.

(e) **Handle Errors, Logging (1pt)**

- If the requested filename is not found, return a 404 Not Found response (see below)
- You will want to check for this before generating the full header. One good way to note a missing file is during the `stat()` call you'll need for the content-length, if the file doesn't exist it will fail, and you'll know to generate the 404 page instead.

```
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
```

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

(f) Be sure to comment your code!

(g) 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 see the actual headers being sent by your server be connecting to it with `wget -S`

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.

(h) 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 manually connect to a web server, and this is the results you get:

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

HTTP/1.1 302 Moved Temporarily
Server: nginx/1.20.1
Date: Fri, 26 Jan 2024 03:41:45 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/1.20.1</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!