ECE 435 – Network Engineering
Lecture 5

Vince Weaver

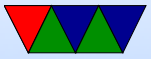
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

31 January 2025

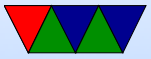
Announcements

- HW#1 was due.
- HW#2 will be posted. Write a mini-webserver.



Web-servers

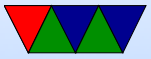
- famously netcraft had a list (meme netcraft reports BSD is dying)
- NCSA was first popular one
- Apache ("a patchy" version of NCSA) took over
- Microsoft IIS
- Other companies like Sun/Netscape/SGI
- nginx ("engine-x")
  Designed to be faster than apache (Apache has lots of RAM overhead)



Solve c10k problem (having 10k concurrent socket connections at once)

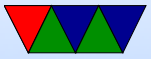
Now there's the c10M problem

- lighthttpd ("lightly")

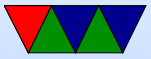


simple web server

- Listen on port 80
- Accept a TCP connection
- Get name of file requested
- Read file from disk
- Return to client
- Release TCP connection
- How do we make this faster?
  - Cache things so not limited by disk
    (also cache in browser so not limited by network)

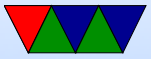


- Make server multithreaded



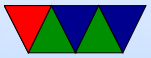
http

- HyperText Transfer Protocol
  RFC 2068 (1997), RFC 2616 (1999), RFC 7230 (2016)
- Make ASCII request, get a MIME-like response
- Connect with TCP socket
- Plain text request, followed by text headers
- Expects carriage returns in addition to linefeeds
- Influences from e-mail servers

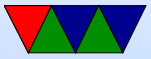


http Commands

- GET *filename* HTTP/1.1
  get file
- HEAD
  get header (can check timestamp. why? see if cache up to date)
- PUT
  send a file
- POST
  append to a file (send form data)

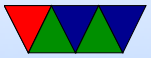


- DELETE
  remove file (not used much)
- TRACE
  debugging
- CONNECT, OPTIONS



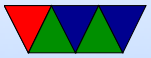
http three digit status codes

- 1xx – informational – not used much
- 2xx – Success – 200 = page is OK
- 3xx – Redirect – 303 = page moved
- 4xx – Client Error – 403 = forbidden, 404 = not found
- 5xx – Server Error – 500 = internal, 503 = try again



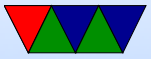
Example http request from browser

```
GET / HTTP/1.1
Host: 471-pi3:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/109.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```



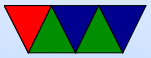
Selected http request headers (included after GET)

- Host: server you are requesting
  Can configure browser to open up helper util for this (for example, run Office if it's a word file)
- User-Agent (browser info). Can you lie? Can you leak info?
- Accept-*: type of documents can accept, compression, character set
- Authorization: if you need special permissions/login



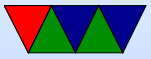
- Referer [sic] URL that referred to here
- Cookie: deals with cookies

  Statelessness – how do you remember setting, logins, shopping cart, etc. “cookies”. Expire. Can be misused.
- If-Modified-Since – caching



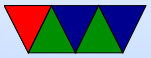
Example http response

```
HTTP/1.1 200 OK\r\n
Date: Fri, 26 Jan 2024 04:56:25 GMT\r\n
Server: ECE435\r\n
Last-Modified: Sun, 26 Mar 2017 04:31:47 GMT\r\n
Content-Length: 64\r\n
Content-Type: text/html\r\n
\r\n
<html><head><title>Test</title></head>
<body>test</body></html>
```



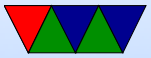
Selected http response headers

- Content-Encoding,Language,Length,Type
- Last-Modified: helps with caching
- Location: used when redirecting
- Accept-Ranges: partial downloads (downloading a large file, interrupted, can restart where left off)
- Content-Length: length of file being sent
- Content-Type: type of data
- Date: current date
- Server: Name of webserver (should you report this?)

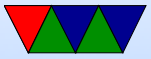


HW#2

- Can use existing server code, will connect to it with any web-browser
- Listen on port 8080 (why not 80?)
- Once browser connects, read entire request into buffer (more proper way to dynamically allocate memory?)
- Ignore most of the headers, mostly want to parse the GET request
- Generate headers for response
- Send header and file back to browser over socket

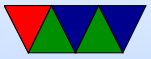


- Handle a few corner cases, like 404 errors



HW#2 Hints

- Get the header printing first, then worry about correctness of headers (dates, length))



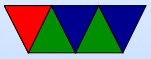
HW#2 – Parsing for filename

- Know how to search for a string and point to location after it?
  - Find a string and point to beginning of it.

```
char *pointer;
pointer=strstr(haystack,needle);
```

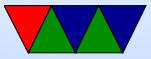
  - Look for `"GET "`

    Actually points to beginning of GET. How to skip ahead?
  - pointer+=4 is one way. (pointer math, ugh)
  - How to get to first space?



- `strtok(pointer," ");`
  Will split the string into chunks, put 0 at end.
- Also can do this manually;

```
pointer2=pointer;
while(*pointer) {
   if (pointer==' ') {
      *pointer=0;
      break;
   }
   pointer++;
}
printf("%s\n",pointer2);
```

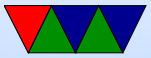


HW#2 – Constructing the Headers

- Know how to construct a string on the fly? `strcat()`, `sprintf()`
  `strcpy()` first bit in.
  `strcat()` additional strings.
  If you want formatting you can do things like

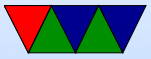
```
sprintf(temp_string,"File size=%d\r\n",filesize);
strcat(out_string,temp_string);
```

  Create big enough buffer.



HW#2 – Calculating Content-length

- How to find size of a file?
- Can read it in, and count. Note: don't use strlen() for this as a binary file might have zeros in it
- Might be better to use `stat()` (`man stat.2`) need .2 (or `man -a`) as there's a command line tool called stat that comes ip first.

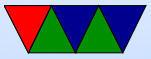


HW#2 – Reading/Writing File

- How to read/write file. There are a large number of ways to do this. `open()/read()/write()/close` `fopen()/fread/fwrite/fclose` (careful! Buffered! And maybe need fdopen() to print to file descriptor).

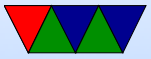
```
fd=open(filename,O_RDONLY);
if (fd<0) fprintf(stderr,"Error opening %s\n",filename);
while(1) {
    result=read(fd,buffer,256);
    if (result<=0) break;
    write(network_fd,buffer,result);
}
```

  Be sure to close afterward.



HW#2 – Getting Filetype

- Easiest way is calculating based on extension
- Take filename, look for . and compare after it
- Can use strstr() again, but think of corner cases
  What if multiple dots? What if no dots?



HW#2 – To Be Continued

There will be some further notes on HW#2 code next lecture