ECE 435 – Network Engineering Lecture 6

Vince Weaver

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

vincent.weaver@maine.edu

3 February 2025

Announcements

- HW#2 was posted. Write a mini-webserver.

HW#1 Review – Notes

- Aside, why port 31337? (LEET speak)

  `https://en.wikipedia.org/wiki/Leet`
- Don't ignore compiler warnings.
  What if `toupper()` not found?
  manpage. Need to include `ctype.h`
- Make sure your code doesn't segfault
- Comment your code!

HW#1 Review – Writing Data

- With write syscall, need to set the size to send back.
- If you always send size of BUFFER even if not full, it sends lots of useless zeros.
- You can use `strlen()` to get size of string (don't use `sizeof()`)
- Also if you got the data with a `read()` call, the return value of that is how many bytes that were read into the BUFFER.

HW#1 Review – Specifications

- When you type "bye" it would exit both sides.
  (bye by itself? cr/lf? byet?
- Postel's Law: strict what send, generous receive?
- Example of browser accepting herf instead of href? why could this be bad?

HW#1 Review – Something Cool

- Command line arguments
  - Don't interfere with default behavior (unexpected)
  - Is good to print expected command lines if there's an error, or have a help option
  - Can you just document it in the README? Sadly people don't always read documentation?
- Printing port/address
  - Biggest issue is forgetting to use `htons()` on the port and `htonl()` on address

- This might not be obvious if you don't know what the port/address should look like (`netstat` or `ss` can help)

HW#1 Review – Questions

- OSI reference model – was hoping for names not number
  - ○ Routing packets – network layer (3?)
  - ○ Bits and voltages – physical layer (1?)

    Not hardware layer

Homework #2 Notes – Connecting

- If connecting on same machine, can use localhost if over network, must use IP address.
- Can find this various ways (`ip addr` on Linux)
- Be aware depending on how your network is set up (firewalls, if behind NAT, etc) you might not be able to connect to your test machine remotely

Homework #2 Notes – Common Issues

- If browser confused, be sure you aren't sending extra zeros. strlen() is your friend
- If browser gets some data but then just spins waiting, be sure your Content-length field is set with the proper size Note it's the size of file you are sending, does not include header size.

Homework #2 Notes – Debugging

- A powerful tool is using
  `wget -S localhost:8080/test.html`
  which will show you the headers your server is sending and download the file so you can verify the contents. Note you might need to install the `wget` tool (easy to do on Linux, maybe more difficult elsewhere)
- The `strace` tool can also be useful as it can show you the bytes being sent by the various syscalls
- If getting segfaults, you might be stuck using `gdb`

HW#2 Hints – Reading Request

- First be sure you are getting the incoming header. Print it or use strace to verify.
- Some web-browsers might send really big requests, be sure getting it all
  - Use big enough buffer? 4096 bytes? How big?
  - How would a "proper" server do this?

    `malloc()`, `realloc()` if not big enough?

    Overkill for this homework. You can try this, but only if you know what you are doing. Goal of this assignment is a simple server not perfect server.

- Just use a bigger buffer if necessary and error if you get bigger, don't waste time chasing pointers/segfaults

HW#2 Hints – Parsing the Request

- 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);
```

- Be sure to strip off initial /, and if it's just / return index.html
- Do you need to handle spaces in the filename? Thankfully no, URLs can't have spaces

HW#2 Hints – Generating Response Headers

- Print to stdout to verify what sending, also can use lynx / wget.
- Know how to construct a string on the fly?
  - One way is to have empty string, than use `strcpy()` first bit in. `strcat()` additional strings.
  - Easier might be `sprintf()` If you want formatting you can do things like

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

- `snprintf()` might be a bit safer as you can specify the max length of the string (to avoid overflowing)
- Try not to be too fancy with one gigantic `sprintf()` call as C can evaluate function parameters in arbitrary orders

HW#2 Hints – General C annoyances

- When you use a char pointer to point into a string (as when using `strstr()` or `strtok()` remember what you have is a pointer, not a copy of the string you're pointing to. So if the buffer gets freed or re-used your pointer may suddenly point to something different.

HW#2 Hints – Getting Size of File

- Can read it in, and count.
- Or can use the stat (`man stat.2`) need .2 (or `man -a`) as there's a command line tool called stat that comes up first.

```
#include <sys/stat.h>
struct stat statbuf;

/* use stat() if have filename, fstat() if have file descriptor */
result=fstat(input_fd,&statbuf);
input_size=statbuf.st_size;
```

HW#2 Hints – Sending File Contents

- Reading file into buffer then writing to socket
  - ○ I don't recommend this as you have to dynamically handle different file sizes
  - ○ If you do this, don't use sprintf() with %s to print the contents. Won't work if 0 in file
- Reading/Writing in chunks
  - ○ open()/read()/write()/close

```
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);
    }
```

  - `fopen()/fread/fwrite/fclose` (careful! Buffered! And maybe need fdopen() to print to file descriptor).

- Be sure to close afterward.

HW#2 Notes – Knowing Request is Done (part1)

- This probably isn't needed for this assignment, but can be useful if you re-use code for your project
- When reading in data from a socket, you probably want to read in the entirety of a request even though it might be split across multiple reads (so `read()` in a `while(1)` loop)
- You might also want to read all you can and then have your client or server handle the request. However if

the last `read()` call blocks forever waiting then your program is stuck waiting and can't accomplish anything else

- Is there a way to have interactive programs that are also waiting for socket data?

HW#2 Notes – Knowing Request is Done (part2)

- Can you just assume each read() matches an exact write() from the cient?
  - No: TCP is a byte stream, you can't see packet boundaries and they might not correspond to the write() calls on the other side anyway
- Can you infer that there's more data based on the content being sent?
  - Yes, for example if the data read ends in a new-line it

could mean the transaction is done

  - Your protocol can contain info that lets you know how long things are (content-length), or have a signal (like the empty newline in http after headers) that let you know

- Can you have non-blocking read() calls?
  - You can set the fd to be non-blocking
  - The `recv()` call (unlike `read()` has some extra flags that can help. On Linux can pass `MSG_DONTWAIT` which will not-block and just return an error if no data is available

- Note in these cases you have to periodically poll the socket to check for input which might not be optimal
- You can use `poll()` or `select()` to be notified when a fd has data but that's complex
- You can also possibly set up multiple threads with pthreads or similar, with one thread handling the socket I/O

http 1.0

- RFC 1945 (1996)
- Single request / single response
- Each file/image requested was separate TCP connection

HTTP 1.1

- RFC 2068 (1997), RFC 2616 (1999)
- Introduced “Host” header to allow multiple web servers on same IP address
- Supports persistent connections, allowing multiple requests to happen with one TCP connection (lowering overhead).
- How do you know when to close? (timeout after 60s?)
- For improved performance, can you open multiple simultaneous connections? Common trick, but polite

to keep number low (less than 5?) instead? Yes, but frowned upon (server/network load)

HTTP/2

- 2015. RFC 7540 / 8740 / 9113
- `https://http2.github.io/faq/`
- Google push through, extension of their SPDY (speedy) Microsoft and Facebook giving feedback
- Why does google care about (relatively) small increases in web performance?
- Leaves a lot of high level things the same. Negotiate what level to use.

HTTP/2 decrease latency of rendering pages

- compress headers
- Server can push data the browser didn't request yet but it knows it will need (like images, etc)
- pipeline requests
  Send multiple requests without waiting for response
  good on high-latency links (FIFO on 1.1, new makes it asynchronous)
- multiplex multiple requests over one TCP connection

HTTP/2 Head of Line Blocking Problem

- line of packets held up by processing of first
- FIFO first requests
- waits until done until next, can't run in parallel
- Can still have issues if TCP packet gets lost

HTTP/2 Other notes

- Page load time 10-50% faster
- While can use w/o encryption, most browsers say will only do with encryption
- Criticism: was rushed through. Is way complex. Does own flow control (has own TCP inside of TCP) Re-implements transport layer at application layer
- Can check if your web-browser implements HTTP by going to `https://http2.golang.org/`

HTTP/2 Support

- Most browsers support it
- Wikipedia says in July 2023 36% of top websites using it
- Apache, nginx, lighthttpd, many other servers all support it

HTTP/3 or H3

- Standardized by RFC 9000 (QUIC) and 9114 (HTTP/3)
- As of 2024 supported by most web-browsers, 31% Top server
- Web-servers, supported by IIS and nginx, no Apache support yet (many use Litespeed which is proprietary but has apache compatible config)
- `https://blog.apnic.net/2023/09/25/why-http-3-is-eating-the-world/`

HTTP/3 and QUIC

- QUIC – runs sort of custom network congestion protocol in userspace over top of UDP
- HTTP/3 started as HTTP/2 over QUIC but has developed more
- QUIC is almost more of a TCP replacement
- Interface is no longer a sockets interface

HTTP/3 other

- HTTPS only
- Can handle better roaming around switching IP addresses w/o losing connection
- Also note it might not be possible to use self-signed certificates so you can only use http3 if approved by an authority

HTTP/3 Firefox issue 2022

- `https://hacks.mozilla.org/2022/02/retrospective-and-technical-details-on-the-recent-firefox-outag`
- Firefox stopped responding worldwide because of a bug in their HTTP/3 stack made their telemetry break a few weeks ago
- The fact that they let the telemetry break the browser is a whole other concerning tale
- But it turns out recent firefox has HTTP/3 set to automatic, and will use it if found, and google has been rolling out HTTP/3

- Part of the bug is http headers are supposed to be case-insensitive, and HTTP/2, HTTP/3 suggests they should be all lowercase, which can break your parser if you don't expect it
- Postel's Law in action?

Do you need a browser? (old)

```
telnet www.maine.edu 80
GET / HTTP/1.1
Host: www.maine.edu
(enter)(enter)
control-]
close
```

Do you need a browser? (https)

```
openssl s_client -connect www.maine.edu:443
GET / HTTP/1.1
Host: www.maine.edu
(enter)(enter)
```

Do you need a browser? (HTTP2)

```
openssl s_client -connect http2.akamai.com:443
GET / HTTP/1.1
Host: http2.akamai.com
```

Does not work.

See http://www.chmod777self.com/2013/07/http2-status-update.html

But need to first send a binary SETTINGS frame.

```
50 52 49 20 2a 20 48 54 54 50
2f 32 2e 30 0d 0a 0d 0a 53 4d
```

```
0d 0a 0d 0a 00 00 04 00 00 00
00 00
```

Then HEADERS frame, then compressed HEADERS.

Response is compressed HEADERS and DATA frames.

How simple can a server be?

- My Apple II webserver project

  `http://www.deater.net/weave/vmwprod/apple2_eth/`

High-Level WWW Concerns

Compression

- Even with 1.1 could use deflate compression
- CRIME attack, could figure out encryption things by seeing how well values compressed (?)
- Because of this http compression is usually disabled
- http2 HPACK special compression to be resistant

What if Server Overloaded?

- Slashdot effect (modern: HackerNews?)
  Too many machines connecting at once, can crash or cause DoS
- caching/proxy – squid
- Server farms / clusters
- Content Delivery Network
  - akami, mirroring content at nearby ISP level instead of one large server
  - cloudflare, similar, also provide DoS mitigation

- What if active attack? Can you block things?
- Recently: AI causing DoS like attacks as they try to scan entire internet for content

Web Security

- SSL – Secure Socket Layer
- Replaced by TLS (Transport Layer Security)
- Port 443 for https (we'll talk about soon)
- Public key encryption.

Do you need Encryption?

- Big push for “https everywhere”
- For personal data, banking info, e-commerce, secret sites
- What if for harmless / regular sites?

  One reason is to avoid man-in-the-middle attacks where someone in between could insert HTML (ads, malware, etc)

Https challenges

- Can be expensive to move to https
- Requires static IP, no multi-hosting?
- Need to buy certificate, can be expensive (though free otions like "Let's Encrypt"
- Certificates expire, from 2 years to 90 days and even pushes to make it shorter! Huge hassle

Authentication

- How do you know the site you connect to is the one you want?
- Little green padlock in corner
- https combines authentication with encryption
- self-signed certificates should be fine for non-critical sites, but browsers make a fuss if you use them

Web Privacy

- Cookies
- Cross-device tracing
- Browser Fingerprinting

Setting Up a Web-server

- Apache
- Easy to do, more difficult to secure

Web Seach

- Web-bots index the web. robots.txt file
  In 2025 huge challenge as AI bots scanning web for content, often ignoring robots.txt
- Altavista, Hotbot, Excite, Inktomi, etc.
- Curated search like Yahoo (people organize links rather than automatically search)
- Google (1996 some machine in Stanford, 1997-1998)
- MSN search 1999, rebranded Microsoft Bing 2009