ECE 435 – Network Engineering
Lecture 5

Vince Weaver

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

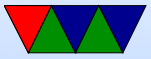
vincent.weaver@maine.edu

12:30pm, Barrows 125

30 January 2024

Announcements

- HW#1 will be graded soon
- HW#2 posted, chanted due date to Friday.
  This is possibly the most difficult assignment.

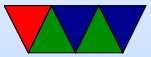


HW#1 Review – Notes

- Aside, why port 31337? (LEET speak)

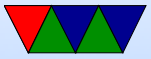
  `https://en.wikipedia.org/wiki/Leet`

- 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!



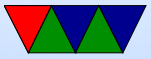
HW#1 Review – Writing Data

- 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.



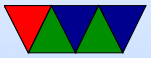
HW#1 Review – Specifications

- 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?

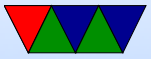


HW#1 Review – Something Cool

- 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

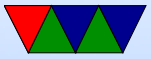


- This might not be obvious if you don't know what the port/address should look like (`netstat` or `ss` can help)



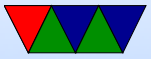
HW#1 Review – Questions

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



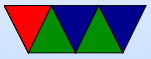
Homework #2 Notes – Connecting

- 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



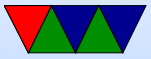
Homework #2 Notes – Common Issues

- 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.

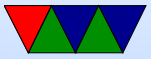


Homework #2 Notes – Debugging

- 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, try using `gdb` but that's sort of a

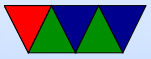


last-resort type thing



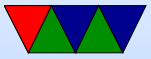
HW#2 Hints – Reading Request

- 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

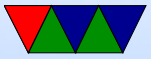


HW#2 Hints – Parsing the Request

- 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);
```

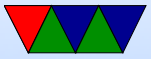
  - Look for `"GET "`
    Actually points to beginning of GET. How to skip ahead?
  - pointer+=4 is one way. (pointer math, ugh)
  - How to get to first space?



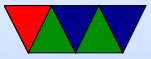
- `strtok(pointer," ");`
  Will split the string into chunks, put 0 at end.
- Also can do this manually;

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

- Be sure to strip off initial /, and if it's just / return index.html
- Do you need to handle spaces in the filename?



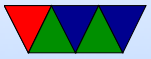
Thankfully no, URLs can't have spaces



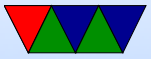
HW#2 Hints – Generating Response Headers

- 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);
```

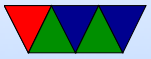


- `snprintf()` might be a bit safer as you can specify the max length of the string (to avoid overflowing)
- Try not to be too fancy with one gigantic `sprintf()` call as C can evaluate function parameters in arbitrary orders



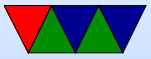
HW#2 Hints – General C annoyances

- 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.



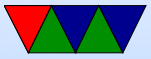
HW#2 Hints – Getting Size of File

- 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.



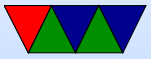
HW#2 Hints – Sending File Contents

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



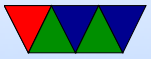
```
fd=open(filename,O_RDONLY);
if (fd<0) fprintf(stderr,"Error opening %s\n",filename);
while(1) {
    result=read(fd,buffer,256);
    if (result<=0) break;
    write(network_fd,buffer,result);
}
```

Be sure to close afterward.



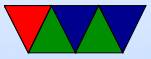
HW#2 Notes – Knowing Request is Done (part1)

- 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



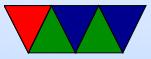
the last `read()` call blocks forever waiting then your program is stuck waiting and can't accomplish anything else

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

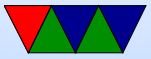


HW#2 Notes – Knowing Request is Done (part2)

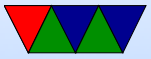
- Can you just assume each read() matches an exact write() from the cient?
  - No: TCP is a byte stream, you can't see packet boundaries and they might not correspond to the write() calls on the other side anyway
- Can you infer that there's more data based on the content being sent?
  - Yes, for example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

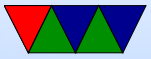


- Note in these cases you have to periodically poll the socket to check for input which might not be optimal
- You can use `poll()` or `select()` to be notified when a fd has data but that's complex
- You can also possibly set up multiple threads with pthreads or similar, with one thread handling the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



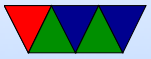
Remote Connections

- One of the first uses of networks was logging into remote computers
- In the old days computers were super expensive
- Often there'd be one big one in a central location and you could connect remotely with smaller, cheaper terminals



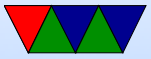
Historical – telnet

- log in to remote system
- (tcp port 23)
- everything (including passwords) sent in plain text
- client not much more complex than HW#1
- telnetd server providing connections, gives you a login via the Linux pty (pseudo-tty) interface



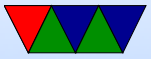
Historical – rsh/rlogin

- remote shell, remote login
- (tcp port 514)
- Didn't even need password, could configure to let you run commands on remote machine
- Security based if you had same username on both machines, assumption was getting root on a UNIX machine and connected to Ethernet was expensive/difficult



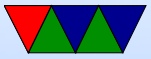
SSH secure shell (background)

- Encrypts a connection between machines
- tcp port 22
- can login, run commands, tunnel tcp/ip, tunnel X11, file transfer (scp, sftp)
- Large number of RFCs
- Version 1: 1995, originally freeware but became private
- Version 2: 2005, openBSD based on last free version
- For security reasons support for Version 1 essentially discontinued

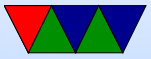


SSH (implementation)

- uses public-key cryptography
- transport layer: arranges initial key exchange, server authentication, key re-exchange
- user authentication layer: can have password, or can set up keys to allow passwordless, DSA or RSA key pairs
- connection layer: set up channels
- lots of encryption types supported, old ones being obsoleted as found wanting
- Various ssh servers/clients. openssh. dropbear

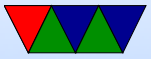


- Diffie-Helman key exchange (we'll talk about this later)



ssh downsides

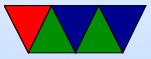
- Any downsides?
- Takes a lot of compute power. Not a problem for most modern machines.
- Maybe more of an issue on small embedded systems
- Does make it hard to put your 8-bit machine on the internet without a helper device
- Older protocols/keys expire which can make it hard to connect to older machines/operating-systems



ssh security

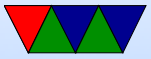
Brute forcing passwords is a major issue.

- password reuse from compromised machines
- Fail2ban
- Nonstandard port
- Port knocking
- Call asterisk for one-time pin?
- No-password (key only)
- Two-factor authentication (LCD keyfob)



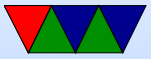
Alternatives to SSH?

- mosh



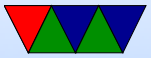
Encryption Background

- Most crypto papers involve Alice and Bob (maybe Eve)
- **Plaintext** is transformed by some sort of function parameterized by a "key" into **Ciphertext**.
  This is then transmitted. The other side then decrypts
- What can be kept secret? Security by obscurity? Kerckhoff's principle: "All algorithms must be public; only the keys are secret."
- Combination lock analogy. Longer the key, the harder it is to brute-force



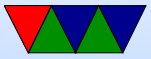
Encryption Types – Substitution

- Substitute each character for another with lookup table
- Decrypt by just doing the reverse
- Trivial Example: rot13 (Ceasar Cipher)
  - A-N, B-O, C-P, etc.
  - Weakness: English text easy to predict ('e' most common letter)
  - What about double-rot13?



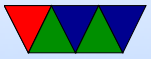
Encryption Types – Transposition

- transposition cipher, keep letters same, re-arrange order



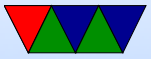
Encryption Types – One Time Pad

- Unbreakable
- Downsides: must have enough bits, cannot reuse, transporting.



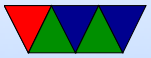
Secret Key Algorithm

- Key is secret
- How do you get it to the other person?
- How many keys do you need (ideally one per connection)



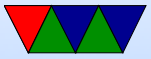
Symmetric Key Algorithms

- Use same key for encryption and decryption
- Block ciphers, take block of data and encrypt it to same size block (why in blocks?)
- P-box (permutation), S-box (substitution)
- shift/permute/xor
- *very* important that the key is picked randomly.



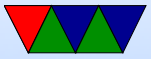
Symmetric Key Implementations – Historical (DES)

- DES – Data Encryption Standard
- From 1976
- 64 bit key (56-bits used)
- NSA had say on key size.
- 19 stages based on Key
- widely used until broken.
- Competition to break various sizes.



Symmetric Key Implementations – Historical (3DES)

- 3DES (running DES three times)
- encrypt/decrypt/encrypt with only two keys?
- Why? 112 bits seen as enough, also if set keys to same then it's same as single-DES (back compat)



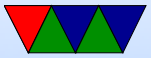
Symmetric Key Implementations – AES

- AES – Advanced Encryption Standard
  - replaces DES
  - NIST had a contest to find new standard
  - Rijndael won

    developed by two Belgian cryptographers Joan Daemen and Vincent Rijmen
  - NSA allows for classified data

    Intel chips have AES instructions

    Galois Field Theory (Gal-wah) interesting math guy



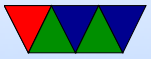
AES Encryption

1. Key Expansion

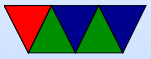
2. AddRound on initial key (add/xor on round key)

3. 9/11/13 rounds (depending on key size)

   (a) SubByte: non-linear substitution (w lookup table)
   (b) ShiftRows: transposition/row shift
   (c) MixColumns: mix columns (matrix multiply)
   (d) AddRound (xor again)

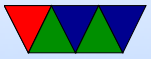


4. Final round: a,b,d again



AES Attacks

- In theory take billions of years to brute force
- "Attack" means finding some way to decode key faster than brute force
- Have been some but none really effective yet
- Side Channel Attacks are possible though



AES Performance

- Pentium Pro 200MHz: 11 MBits/s
- Modern Intel/AMD with AES in hardware, multiple GB/s