15-213

"The Class That Gives CMU Its Zip!"

Introduction to Computer Systems

Taerim Kim
April 9, 2012

Topics:

- **malloclab**
- **Networking and Unix network I/O**
- **proxylab**

Reorientation

Monday, April 9

- Today

Thursday, April 12

- malloclab due
- proxylab out

Thursday, April 26

- proxylab due

Friday, May 11

- Final exam

malloclab

Why do I get a segmentation fault?

- You have access to a powerful interactive debugger
- What does your heap checker look like?

Why does the driver complain about running out of memory?

- Your allocator uses way too much memory
- Common pitfalls
  - The relationship between `malloc` and `free` is broken
  - Your allocator leaks blocks

Why does the driver complain about garbled bytes?

- Your allocator writes to payload areas of allocated blocks
- Check your pointer arithmetic

malloclab

How do I improve my score?

- That depends... There is often a trade-off between throughput and utilization
- Improving throughput:
  - Do you have segregated free lists?
  - Does your allocator perform redundant computation?
  - Have you tried `inline`-ing functions?
  - Have you tried loosening the search policy?
- Improving utilization:
  - Does your allocator coalesce adjacent free blocks?
  - Does your allocator split large blocks?
  - Have you tried tightening up the search policy?

Your problems

Let's talk about them

Networking

“Apartment building” analogy of computer networking

- Apartment building represents a computer
- Each resident represents a process
- Apartment building has a unique address and possibly a name, like a computer has an IP address and a hostname
  - Oakwood Apartments, 15213 Maple Ave
  - unix1.andrew.cmu.edu, 128.2.13.133
- Each resident uses a unique apartment number, like port numbers on a computer
  - Alice lives in #251; Bob lives in #410
  - SSH uses port 22 by default; HTTP uses port 80 by default
- Apartment buildings are connected using a “series of tubes”

Protocols

Two important Internet protocols: TCP and UDP

TCP

- Think of it like making a phone call
- Connection-based
- Reliable (you know if something went wrong)
- Error correction

UDP

- Think of it like sending a letter
- Not connection-based
- Unreliable (you don't know if there was success or failure)

Okay, great—but what do I do?

Use POSIX sockets to manipulate network I/O!

- Ubiquitous programming model
  - Send stuff, receive stuff...
  - Transmission details are opaque
- Generic functionality
  - Communication on the Internet
  - Inter-process communication (same host)

Just a suite of functions that use file descriptors

- Just like regular file descriptors you know and love
- As usual, reading and writing data is (almost) as simple as calling `read` and `write`

Sockets API

socket (both clients and servers)

- Create a file descriptor for network communication
- One socket can be used for two-way communication

bind (servers)

- Associate a socket with an IP address and port number

listen (servers)

- Wait for an incoming TCP connection

accept (servers)

- Accept an incoming TCP connection
- Return a descriptor for the accepted connection

Sockets API

connect (clients)

- Attempt to connect to specified IP address and port number

read (both clients and servers)

- Read bytes from socket

write (both clients and servers)

- Write bytes to socket

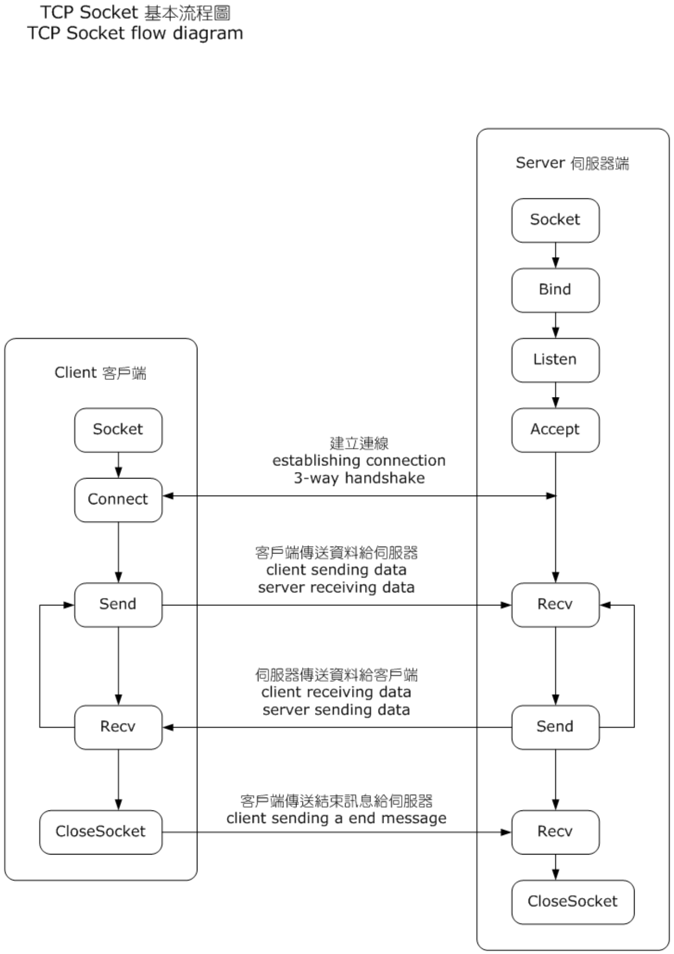
close (both clients and servers)

- Close the file descriptor (just like always)
- Important for TCP servers—close open connection on socket

Sockets API

Image shamelessly stolen from Wikipedia

Man page for `listen` also has details



proxylab

Write a web proxy

- Proxy server
- Multi-threaded
- Caching

proxylab

Proxy server

- An intermediary between a client and a server
- A proxy server is both a server and a client
  - Server to clients making HTTP requests (often web browsers)
  - Client to web servers to which requests are made
- Specifically, a proxy server for HTTP/1.0 GET requests

Typical operation

- Client connects to proxy and makes request
  - `GET http://www.google.com/index.html HTTP/1.0`
- Proxy connects to `www.google.com`; requests `index.html`
- `www.google.com` responds to proxy with `index.html`
- Proxy responds to client with `index.html`

proxylab

Multi-threaded proxy server

- Handle multiple simultaneous connections concurrently
- Simple model: spawn a new thread for each request
  - Alternative model: create a pool of worker threads
  - Do whatever you want as long as there is true concurrency

Multiplexing

- Lots of servers don't do real concurrency
- They do something called multiplexing using functions like `select`
- Don't do multiplexing for proxylab; we want to see real concurrent operation

proxylab

Multi-threaded caching proxy server

- Cache web objects in memory
- Whenever there is a cache hit, serve object from memory instead of retrieving again from server
- Must handle synchronization of concurrent access
- Must implement LRU eviction

proxylab logistics

Partners

- You may (and should) work in groups of two
- Less work for you (probably—cf. *The Mythical Man-Month*)
- ...Less work for us

Writeup

- It will be new this semester
- Please read it...carefully

Demos

- No compulsory demos this semester
- May optionally sign up for a quick, 15-minute demo anyway

In next week's episode...

malloclab

- What you did well
- What you did poorly

proxylab

- Introduction to threads
- Helpful tools
- Ideas for testing