RECITATION 10: NETWORKING

15-213 M12

Rick Benua

Malloc Lab

- Due Wednesday

Proxy Lab

- Out this week
- Due Tuesday 8/7
- You may work with a partner
  - Sign up on Autolab
  - Use ##213 to meet people!
  - You may use the minimum number of grace days either one has remaining
  - The lab is doable on your own if necessary.
- Two (somewhat) separate challenges
  - Learning the C/UNIX networking interfaces
  - Writing a concurrent cache (next week's lectures / recitation)

Network Stack

- Multiple levels of abstraction
- Hardware – cables / 802.11 / 3G / whatever
- Link layer – Ethernet – Provides local-area network
- Internet layer – IP – Joins local networks into wide-area network
- Transport layer – TCP / UDP – Establishes reliable communication between hosts
- Application layer – HTTP / FTP / many others – Communicates in a format specific to the applications involved (e.g. web server / web browser, two BitTorrent clients, etc)

Network Stack – What you need

- Your program's functionality should care about the Application layer.
- Access transport layer functions (connecting to hosts, listening for connections, looking up DNS entries, sending / receiving data) using system calls
- Lower abstraction layers are implemented in libraries, kernel code, device drivers, or hardware.
- UNIX "sockets" interface provides a file-like model for interacting with network connections – connections can be bound to file descriptors for use with read(), write(), etc.

The Sockets Interface - Basics

- Create sockets with socket()
  - Returns a file descriptor pointing to the socket
- Connect to a remote host with connect()
- OR assign it a specific local address with bind()
- Mark it as a listening process with listen()
- listen for connections on a specified port with accept()
  - accept() blocks the process until it receives a connection, then returns a second file descriptor representing the connection, leaving the original one free to listen again

The Sockets Interface - Details

- Use gethostbyname() or gethostbyaddr() (deprecated) or getaddrinfo() to get an address to pass to connect()
- csapp.c provides open_clientfd() and open_listenfd() functions
  - open_clientfd() does socket() -> connect(); socket ready for read/write
  - open_listenfd() does socket() -> bind() -> listen(); socket ready for accept()

The Application Layer - HTTP

- Your proxy is a little bit odd in this respect, because of the simplified nature of the assignment.
- You need to accept HTTP/1.1 requests and send HTTP/1.0 requests.
  - Functionally, they won’t differ for you, but remember to get the version numbers right
- HTTP is an ASCII protocol – commands are issued in text
- You need only handle GET requests, but there are other types.
  - GET asks a server for a resource
  - POST submits data – included in the body
  - PUT submits a resource to upload
  - Lots more

The Application Layer – HTTP Requests

- HTTP requests have three parts – sent sequentially, separated by CRLF (“\r\n”)
- Request line specifies the location of a resource, the type of the request, and the protocol version
  - GET /path/to/resource HTTP/1.0
- Request headers – one per line, format Name:Value
  - User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:13.0) Gecko/20100101 Firefox/13.0.1
  - (line wrapped to fit screen, but one line in the protocol)
  - Specify information related to the request
- Body – Any data can follow the request
  - Not used for GET, but contains form data for POST

The Application Layer – HTTP Responses

- On receiving a request, the server sends back a response
- Contains a numeric status, some headers, and the requested information
- HTTP/1.0 200 OK
  <html><head>...
- Or, a numeric error code:
- HTTP/1.0 404 NOT FOUND