CSS 503 Lecture # 16 2019-05-22

Administrivia

OSSF

IPv4 - 32-bit addresses
IPv6 - 128-bit addresses

DNS system:
- distributed hierarchical database
- maps domain names to IP addresses

network OS
- networking applications

distributed OS
- resource allocation across entire datacenter
- distributed lock service

sockets: "high-level" abstraction
- once network connection is established, can use read/write system calls
- socket descriptor: specialized file descriptor

Endianness (byte order)

```
#include <arpa/inet.h>
```

htons } short (16 bits)
ntohs }

htonl } long (32 bits)
ntohl }

IP Address Manipulation

- text format

  32-bit number
  - host byte order (little endian on Intel)
  - network byte order (big endian)

#include
  - sys/socket.h
  - netinet/in.h
  - arpa/inet.h

```
typedef uint32_t in_addr_t
struct in_addr {
    in_addr_t s_addr;
}
```

inet_aton(char* cp, struct in_addr* inp)

  String rep (12.34.56.78) → 32-bit int

inet_ntoa

  number to dotted string

Network Programming API

- better-designed API (hypothetical):

  namespace-oriented

  e.g.
  - open("/net/127.45.67.89/80")
  - or /net/<IP>/tcp/<port>
  - or like URI (Uniform Resource Identifier)
    http://host:port/path/...

- still need to translate that to wire format (protocol)

- text-based (URI): easier on eyes, harder on computer
  - good tradeoff these days
    (programmers are expensive)

- but we're all about looking at the nitty gritty here

- Given hostname, get IP address:
  library call
    - looks up system config files
      - /etc/hosts
        /etc/host.conf
        /etc/nsswitch.conf

- obsolete: struct hostent * gethostbyname(char *name)
  pointer to 4 bytes (IP address)

  getaddrinfo
    - more complex API
    - returns struct addrinfo *

- Once name lookup is complete, host name is not used further
    - all connections via IP address

- higher-level protocol: http
  http://bulk.com/ } => 12.34.56.78
  http://thor.com/
  => "virtual" host field in http

- IP address manipulation

IPv4 address object (32-bits):
- typedef uint32_t in_addr_t
- struct in_addr {
    in_addr_t s_addr;
  }

inet_aton(char *cp, struct in_addr *inp)
- Parse IP address string (dot notation)

inet_ntoa
- convert address to string rep

Network Programming API

e.g. X.25 – {niche legacy apps}

not just for IP (although)

- Socket: (Unix) abstraction for all networking
  - also used by Windows
  - AKA Berkeley Sockets
  - mostly TCP/UDP
    - Unix-domain sockets (IPC)
    - X.25 (obsolete) packet switching

- create socket: socket(2) system call
  - returns socket descriptor
    - small integer, same namespace as file descriptors
  - once the socket is set up: may use read/write/close system calls
  - may also use send/recv
    - better control over message-oriented protocols
    - e.g. UDP
  - additional system calls to set up connections

⇒ acts like a file, but not completely transparent
  - different "constructor"

Unix Socket

```
#include
    <sys/types.h>
    <sys/socket.h>
```

socket (int domain, int type, int protocol)

domain
- AF_UNIX (yet another IPC)
- AF_INET (internet protocol - IP)

type
- SOCK_RAW (raw IP packets)
- SOCK_DATAGRAM (UDP)
- SOCK_STREAM (TCP)

protocol
- 0 for AF_INET

* freshly-allocated socket object
  → not (yet) tied to connection
  (no associated port)

Socket API (cont.)

- 2 operating modes
  passive (server)
  active (client)
  => difference between client & server:
  who initiates the transaction

Server:
  tcp
    bind() - associate socket with port #
    listen() - specify operating mode
    accept()
      - returns new socket descriptor
      for each connection
  udp
    bind()
    recvfrom()
      ↳ fd, buf, size,
      flags, sockaddr, addrlen

Socket API (cont.)

client

- typ (optional: bind) → associate local port #
  otherwise, port is assigned by system
  - connect()
- udp
  - bind() (optional, recommended)
  - sendto()
  - or { connect()
    send()

bind()

- associates socket & address (host: port)

```
int bind (int sd,
          const struct sockaddr *addr,
          socklen_t addrlen)
```

- sockaddr: essentially a void type
  - Protocol-specific address structs are cast to sockaddr *

local address

Socket API (cont.)

#include <sys/types.h>
#include <sys/socket.h>
} the usual suspects

int accept (int sockfd,
struct sockaddr * addr,
socklen_t * addrlen)
↑ pointer

out parameters:
- filled in with remote address
- addr may be null
- don't care

- returns new fd for communication
original sockfd remains open
for additional connections

- listen(2) - set size of backlog

- may have multiple concurrent
connections (less common with web page
requests all go to port 80)

Addressing

text: host : port

text processing
- expensive
- but only have to do it once

address data structure
struct sockaddr

But: addressing is dependent on the protocol
- Internet address: struct sockaddr_in
⇝ overlay: 1st field is family (AF_INET)

sin_family	AF_INET
sin_port	
sin_addr	struct sin_addr { uint32 s_addr; }

⇒ must construct IP Address object
& cast to sockaddr

Socket API (cont.)

internet address (TCP/IP or UDP/IP):

use struct sockaddr_in

sin_family = AF_INET (tag field)

sin_addr.s_addr = IP address 32 bits

sin_port = port 16 bits

host, port are in network byte order

=> use htonl
htons

- for server, use s_addr =
htonl (INADDR_ANY)

use any network interface a machine
→ computer may have multiple NICs

connect (int sd,
const struct sock_addr * addr,
socklen_t addrlen)

remote address

use port = 0 for randomly-assigned unused port
- ephemeral ports

UDP - User Datagram Protocol

Client:

Socket()
bind() (if response is needed)
sendto()
recvfrom()

or

connect()
send
recv

Server:

Socket()
bind()
recvfrom()
sendto()

- Connectionless
- packets may get lost, arrive out of order
- multicast

Applications
- TFTP/BOOTP
- NFS
- DNS
- media/ streaming

either you don't care if some data is lost or you have your own homebrew error recovery scheme

UDP (cont.)

Client (connectionless)

- create socket
- sendto(sd
  msg
  len
  flags
  addr
  addr len)

RTFM

⇒ live coding

- sockets & concurrency
  - threads
  - socket fd stays open across fork/exec
    - unless you explicitly set close-on-exec
    - like any file descriptor