Networking and sockets
Goals for Today

- Overview of IP addresses
- Look at the IP address structures in C/C++
- Overview of DNS
- Write your own (short!) program to do the domain name – IP address translation
Networks
Network Addresses

- For IPv4, an IP address is a 4-byte tuple
  - e.g., 128.95.4.1 (80:5f:04:01 in hex)
- For IPv6, an IP address is a 16-byte tuple
  - e.g., 2d01:0db8:f188:0000:0000:0000:0000:1f33
  - 2d01:0db8:f188::1f33 in shorthand
IPv4 address structures

// Port numbers and addresses are in *network order*.

// A mostly-protocol-independent address structure.
struct sockaddr {
    short int sa_family; // Address family; AF_INET, AF_INET6
    char sa_data[14]; // 14 bytes of protocol address
};

// An IPv4 specific address structure.
struct sockaddr_in {
    short int sin_family; // Address family, AF_INET == IPv4
    unsigned short int sin_port; // Port number
    struct in_addr sin_addr; // Internet address
    unsigned char sin_zero[8]; // Same size as struct sockaddr
};

struct in_addr {
    uint32_t s_addr; // IPv4 address
};
IPv6 address structures

// A structure big enough to hold either IPv4 or IPv6 structures.

struct sockaddr_storage {
    sa_family_t ss_family; // address family
    // a bunch of padding; safe to ignore it.
    char __ss_pad1[_SS_PAD1SIZE];
    int64_t __ss_align;
    char __ss_pad2[_SS_PAD2SIZE];
};

// An IPv6 specific address structure.

struct sockaddr_in6 {
    u_int16_t sin6_family; // address family, AF_INET6
    u_int16_t sin6_port; // Port number
    u_int32_t sin6_flowinfo; // IPv6 flow information
    struct in6_addr sin6_addr; // IPv6 address
    u_int32_t sin6_scope_id; // Scope ID
};

struct in6_addr {
    unsigned char s6_addr[16]; // IPv6 address
};
Generating these structures

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

int main(int argc, char **argv) {
    struct sockaddr_in sa;  // IPv4
    struct sockaddr_in6 sa6; // IPv6

    // IPv4 string to sockaddr_in.
    inet_pton(AF_INET, "192.0.2.1", &(sa.sin_addr));

    // IPv6 string to sockaddr_in6.
    inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr));
    return EXIT_SUCCESS;
}
```
Generating these structures

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

int main(int argc, char **argv) {
    struct sockaddr_in6 sa6;    // IPv6
    char astring[INET6_ADDRSTRLEN]; // IPv6

    // IPv6 string to sockaddr_in6.
    inet_pton(AF_INET6, "2001:db8:63b3:1::3490", &(sa6.sin6_addr));

    // sockaddr_in6 to IPv6 string.
    inet_ntop(AF_INET6, &(sa6.sin6_addr), astring, INET6_ADDRSTRLEN);
    printf("%s\n", astring);
    return EXIT_SUCCESS;
}
```
DNS – Domain Name System/Service

- A hierarchical distributed naming system any resource connected to the Internet or a private network.
- Resolves queries for names into IP addresses.
- The sockets API lets you convert between the two.
- Is on the application layer on the Internet protocol suite.
DNS hierarchy

```
A B C ... M
com xxx uk ... org

google yahoo ... hulu

www mail docs ... finance

```

“.” -- root name servers
198.41.0.4 (a.root-servers.net)
192.228.79.201 (b.root-servers.net)
...
202.12.27.33 (m.root-servers.net)

“.com.” -- top-level domain server

```
gribble apache ... fsf

seattle www ...`
```
Resolving DNS names

- The POSIX way is to use `getaddrinfo()`.
- Set up a “hints” structure with constraints, e.g. IPv6, IPv4, or either.
- Tell `getaddrinfo()` which host and port you want resolved.
- Host - a string representation: DNS name or IP address
- `getaddrinfo()` gives you a list of results in an “addrinfo” struct.
getaddrinfo() and structures

```c
int getaddrinfo(const char *hostname, const char *servname, const struct addrinfo *hints, struct addrinfo **res);
```

// Hints and results take the same form. Hints are optional.
```c
struct addrinfo {
    int ai_flags; // Indicate options to the function
    int ai_family; // AF_INET, AF_INET6, or AF_UNSPEC
    int ai_socktype; // Socket type, (use SOCK_STREAM)
    int ai_protocol; // Protocol type
    size_t ai_addrlen; // INET_ADDRSTRLEN, INET6_ADDRSTRLEN
    char *ai_canonname; // canonical name for the host
    struct sockaddr *ai_addr; // Address (input to inet_ntop)
    struct addrinfo *ai_next; // Next element (It's a linked list)
};
```

// Converts an address from network format to presentation format
```c
const char *inet_ntop(int af, const void *restrict src, char *restrict dest, socklen_t size);
```
DNS lookup exercise

- Write a C++ program ‘dnsresolve.cc’ that:
  - Takes a hostname as an argument (e.g. google.com or cs.washington.edu)
  - Returns the IPv4 addresses associated with it.
  - Optional: Fix it so it retrieves both IPv4 and IPv6 addresses.

- **Steps (Recommended)**
  - Setup/initialize ‘hints’ and ‘results’ addrinfo structs (remember to free them later!).
  - Zero out everything in hints, and separately set the ai_family and ai_socktype.
  - Use getaddrinfo() to ask DNS for the IP addresses.
  - Use gai_strerror() to translate error codes for getaddrinfo() failures.
  - Cycle through returned addresses and use inet_ntop() to get a nice string out and print the results.
  - Free the addrinfo struct list returned by getaddrinfo() using freeaddrinfo().
  - For the extra credit part: Distinguish between IPv4 and IPv6 addresses while printing.