CSE 333
Section 8
Client-Side Networking
Logistics

Due TODAY:
  Homework 3 (@9:00 pm)

Due Monday:
  Exercise 15 (@11:00 am)
Computer Networks: A 7-ish Layer Cake

- format/meaning of messages
- sending data end-to-end
- routing of packets across networks
- multiple computers on a local network
- bit encoding at signal level
Exercises 1 & 2
Exercises 1 & 2

- format/meaning of messages
- sending data end-to-end
- routing of packets across networks
- multiple computers on a local network
- bit encoding at signal level
Sockets

- Just a file descriptor for network communication
- Types of Sockets
  - Stream sockets (TCP)
  - Datagram sockets (UDP)
- Each socket is associated with a port number and an IP address
  - Stored in network byte order (big endian)

```
struct sockaddr_in:

<table>
<thead>
<tr>
<th>family</th>
<th>port</th>
<th>addr</th>
<th>zero</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

struct sockaddr_in6:

<table>
<thead>
<tr>
<th>fam</th>
<th>port</th>
<th>flow</th>
<th>addr</th>
<th>scope</th>
</tr>
</thead>
</table>
| 0    | 2    | 4    | 8    | 24    | 28
```
**Sockets**

**struct sockaddr** (pointer to this struct is used as parameter type in system calls)

| fam | ????
|-----|------

...  

**struct sockaddr_in** (IPv4)

<table>
<thead>
<tr>
<th>fam</th>
<th>port</th>
<th>addr</th>
<th>zero</th>
<th>16</th>
</tr>
</thead>
</table>

**struct sockaddr_in6** (IPv6)

<table>
<thead>
<tr>
<th>fam</th>
<th>port</th>
<th>flow</th>
<th>addr</th>
<th>scope</th>
<th>28</th>
</tr>
</thead>
</table>

**struct sockaddr_storage**

<table>
<thead>
<tr>
<th>fam</th>
</tr>
</thead>
</table>
Big Endian and Little Endian

- **Network Byte Order (Big Endian)**
  - The most significant byte is stored in the highest address

- **Host byte order**
  - Might be big or little endian, depending on the hardware

- **To convert between orderings, we can use**
  - `uint32_t htonl (uint32_t hostlong);`
  - `uint32_t ntohl (uint32_t hostlong);`
Exercise 3
1. ________ (hostname, servname, ___, ___)

specify lookup hints
1. `getaddrinfo()`

- Performs a **DNS Lookup** for a hostname

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

- Performs a **DNS Lookup** for a hostname
- Use “hints” to specify constraints (**struct addrinfo**)  
- Get back a linked list of **struct addrinfo** results

```c
int getaddrinfo(const char *hostname,
                const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```
1. getaddrinfo() - Interpreting Results

```
struct addrinfo {
    int ai_flags; // additional flags
    int ai_family; // AF_INET, AF_INET6, AF_UNSPEC
    int ai_socktype; // SOCK_STREAM, SOCK_DGRAM, 0
    int ai_protocol; // IPPROTO_TCP, IPPROTO_UDP, 0
    size_t ai_addrlen; // length of socket addr in bytes
    struct sockaddr* ai_addr; // pointer to socket addr
    char* ai_canonname; // canonical name
    struct addrinfo* ai_next; // can form a linked list
};
```

- ai_addr points to a struct sockaddr describing the socket address
1. getaddrinfo() - Interpreting Results

With a `struct sockaddr*`:

- The field `sa_family` describes if it is IPv4 or IPv6
- Cast to `struct sockaddr_in*` (v4) or `struct sockaddr_in6*` (v6) to access/modify specific fields
- Create a `struct sockaddr_storage` to make a space big enough for either

![Diagram](image-url)
2. extract fields from result (IPv4 vs IPv6)

(______, type, protocol)
2. socket()

- Creates a “raw” socket, ready to be bound
- Returns file descriptor (sockfd) on success, -1 on failure

```c
int socket(int domain,       // AF_INET, AF_INET6
    int type,              // SOCK_STREAM (TCP)
    int protocol);         // 0
```
3.

- *int (sa_family)*
- *extract fields from result (IPv4 vs IPv6)*

2. *socket(domain, type, protocol)*

- *int (sockfd)*

3. *________(____,______, addrlen)*

[Diagram of socket operations with arrows connecting the steps]
3. connect()

- Connects an available socket to a specified address
- Returns 0 on success, -1 on failure
3. connect()

- Connects an available socket to a specified address
- Returns 0 on success, -1 on failure

```
int connect (int sockfd, // from 2
const struct sockaddr *serv_addr, // from 1
socklen_t addrlen); // size of serv_addr
```

```
extract fields from result
(IPv4 vs IPv6)
```

```
struct sockaddr_storage* to sockaddr*
```

```
int (sa_family)
```

```
socket(domain, type, protocol)
```

```
connect(sockfd, serv_addr, addrlen)
```

```
Cast sockaddr_storage* to sockaddr*!
```
4. read/write and 5. close

- Thanks to the file descriptor abstraction, use as normal!
- read from and write to a buffer, the OS will take care of sending/receiving data across the network
- Make sure to close the fd afterward