ICS 451: Today's plan

- Connection-oriented and connectionless
- Sockets API
  - Python
  - C
  - Windows
- C programming reminders

Connections vs Connectionless

- TCP expects a program to **connect** before sending or receiving any data
- At the end of the communication, the connection should be **shut down** (**closed**)
- With UDP, connections are not used:
  - each **send** or **receive** operation takes an address as parameter

Sockets API

- The socket is created by calling **socket**
- Client then calls **connect**, then **send/recv**, then **close**, **shutdown**, or both
- Server calls **listen**, then, in a loop
  - **accept**, then **recv/send**, then **close**, **shutdown**, or both
- **getaddrinfo** converts domain names to IP addresses

Sockaddrs

- An Address consists of

  (Address Family, IP Number, Port Number)
- Address Family is AF_INET for IPv4

  AF_INET6 for IPv6
- IP number can be given manually, or from domain name

  getaddrinfo converts DNs to IP numbers
- Port number you must know somehow

  usually 80 or 8080 for web servers

Using Socket Addresses

- **connect** requires an address
- **accept** returns the client's address

  (see server code in assignment 1 for ways of using **listen** and **accept**)
- For connectionless communications:

  sendto requires an address

  recvfrom returns the sender's address

Socket API in C: socket

- Every function except **socket** takes a socket as its first argument
  - A socket is the integer returned by **socket**

```
/* create a socket of a given type/protocol */

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

- Type is **SOCK_STREAM** (TCP) or **SOCK_DGRAM** (UDP)
- Protocol may be **0**, or **IPPROTO_TCP** (6) or **IPPROTO_UDP** (17)

Socket API in C: addresses

- An address is represented by a pointer to a generic **struct sockaddr**
- This must point to memory containing a specific address: a **struct sockaddr_in** or a **struct sockaddr_in6**:

```
struct sockaddr_in sin;
struct sockaddr * sap =
        (struct sockaddr *) (&sin);
```

- or created with **getaddrinfo**
- sap->sa_family may be AF_INET or AF_INET6

Socket API in C: connect

- **connect** creates a new connection

```
/* for addrlen use sizeof (sin) or ai_addrlen */

int connect(int sockfd, struct sockaddr *serv,
            socklen_t addrlen);
```

- The local IP and port number are selected automatically.
- Returns 0 for success, -1 for error
  - your code should check!

Socket API in C: server calls

- **listen** makes a socket a server socket

```
int listen(int sockfd, int queue);
```

  - Use a small value such as 5 for the queue size

- Bind selects a local port

  - do check that the return value is 0

```
sin.sin_family = AF_INET;

sin.sin_port = htons (server_port_number);

sin.sin_addr = INADDR_ANY;

if (bind(sockfd, (struct sockaddr *) (&sin),

        sizeof (sin)) != 0) perror ("bind");
```

Socket API in C: binding with IPv6

```
struct sockaddr_in6 sin6;

sin6.sin6_family = AF_INET6;

sin6.sin6_port = htons (server_port_number);

sin6.sin6_addr = in6addr_any;

if (bind(sockfd, (struct sockaddr *) (&sin6),
         sizeof (sin6)) != 0) perror ("bind v6");
```

Socket API in C: server accept

- **accept** creates new sockets from server socket

```
int accept(int sockfd, struct sockaddr * peer,
           socklen_t * addrlen);
```

- The return value (if >= 0) can be used with **send** and **recv**
- The memory that peer points to is filled with the peer's address, up to *addrlen bytes
- Then *addrlen is set to the size of the address

Socket API in C: close

- **close** closes a server or client socket

```
int close (int fd);
```

- **shutdown** can be used to declare that we will stop reading from or writing to a socket

```
int shutdown (int sockfd, int how);
```

- how is **SHUT_RD** or **SHUT_WR** (**SHUT_RDWR** is equivalent to **close**)

Socket API in C: send/recv

- **send** sends a buffer on a connected socket

```
int send (int fd, char * buffer, int len, int f);
```

  - Returns the number of bytes sent
  - Flags (f) will normally be 0

- **recv** receives data from a connected socket

```
int recv (int fd, char * buffer, int len, int f);
```

  - Returns the number of bytes received (<= len)
    - or 0 if the socket was closed
  - Flags (f) will normally be 0

Socket API in C: sendto/recvfrom

- Used on unconnected (UDP) sockets

```
int sendto (int fd, char* buffer, int len, int f,
            struct sockaddr* to, socklen_t alen);
```

  - Similar to **send**, but takes address
- **recv** receives data from a connected socket

```
int recvfrom(int fd, char* buffer, int len, int f,
             struct sockaddr* a, socklen_t* alen);
```

  - Address and alen filled in as for accept

Sockets API on Windows

- Windows requires a call to **WSAStartup** prior to using the sockets API
  - Parameters are the version of the latest Winsock DLL that is acceptable, and a data structure to be filled with information about the implementation in use
- Must use **closesocket** instead of **close**
- May call **WSACleanup** after finishing
- May have to `#include <sys/socket.h>`

C programming reminders

- Character arrays have a fixed size, cannot grow or shrink
- Pointers have to point somewhere
  - Where are your pointers pointing?
- Use strncpy in preference to strcpy
- Strings are null-terminated
  - data received from the network usually is not
- A null character is different from a null pointer!

Exercise: spot the bugs

```
char * a = NULL;
*a = 'x';
char b [1000];
a = b + 10;
strncpy (a, “foo”, sizeof (a));
int n = recv (s, a, strlen (a), 0);
a [n] = '\0';
return a;
```