How do processes communicate?

- Same system: shared memory, pipes, message queue
- Wired: serial, network
- Wireless: network, bluetooth, etc.

Serial Communication

http://www.easysw.com/~mike/serial/serial.html

serial communication: one bit at a time

baud rate: bits per second

Rs-232C: electrical standard for serial communication
mark/on/1: -3 to -12V
space/off/0: +3 to +12V
range 25 feet
also specifies pin assignments: incoming and outgoing data, timing, handshaking, etc.

in UNIX, all devices are represented as “device files” in the /dev directory.

serial ports on Linux are usually /dev/ttyS0 or something like that (tty = teletypewriter)

```c
// first, open the connection
int fd = open("/dev/ttyS0", O_RDWR);
if (fd == -1) uh oh!

// then configure it
struct termios options;
tcgetattr(fd, &options);
cfsetispeed(&options, 9600);
cfsetospeed(&options, 9600);
tcsetattr(fd, TCSANOW, &options);

// read data
char buf[100];
int chars_read = read(fd, &buf, 100);

// use fcntl(fd, F_SETFL, FNDELAY) if you don't want to wait for chars to appear

// write data
char *msg = "hello!\n";
int chars_written = write(fd, msg, strlen(msg));

// when you're done:
close(fd);
```
USB: Linux /dev/ttyUSB0; Mac: /dev/cu.usbserial-[driver]

Socket Programming

Low-level sending of bytes (as opposed to using more structured messages)

Apps communicate over ports that are managed by OS

Client/server model: one app (client) instantiates the connection with another (server) that is listening on a well-known port

Socket Programming in C

Server:
1. socket: create a socket descriptor
2. bind: associate socket with port number
3. listen: indicate that we're ready to accept connections
4. accept: block and wait for incoming connection

Client
1. gethostbyname: create a struct to represent the remote host
2. socket: create a socket and configure it with host info
3. connect: attempt to establish connection

Each can then send and recv bytes over the socket connection.

Socket Programming in Java

Server:

```java
ServerSocket server = new ServerSocket(port);
// wait here until connection comes in
Socket s = server.accept();
// now we have a Socket
InputStream in = s.getInputStream();
OutputStream out = s.getOutputStream();
```

Client:

```java
Socket s = new Socket(host, port);
// now we have a Socket
InputStream in = s.getInputStream();
OutputStream out = s.getOutputStream();
```

You can read input with a Scanner (pass InputStream to its constructor) and write output with a PrintWriter (pass OutputStream to its constructor) or any other classes that you prefer.
**Shared Memory**

Two processes on the same machine can ask the OS to create a shared segment of memory, to which both processes have pointers. One process (“server”) must first create the segment in the OS before the other (“client”) can use it.

**Server:**

```c
key_t key = 5678; // segment ID number
int id = shmget(key, SIZE, IPC_CREAT|0x666);
void *p = shmat(id, NULL, 0);
```

**Client:**

```c
key_t key = 5678; // must be same ID number!
int id = shmget(key, SIZE, 0x666);
void *p = shmat(id, NULL, 0);
```

**Pipes**

Direct channel from one process to another. Usually from a parent to a (forked) child.

```c
int fd[2]; // fd[0] = input, fd[1] = output
pipe(fd); // establish connections

if (fork() == 0) { // child
    char *msg = "hello!";
    write(fd[1], msg, strlen(msg)+1);
    ...
}
else { // parent
    char buf[80];
    read(fd[0], buf, 80);
    // now buf holds the message that was sent
```