



Inter-process Communication & Distributed Computing

CIS*3110 Operating Systems - Winter 2011
Jason Ernst, University of Guelph



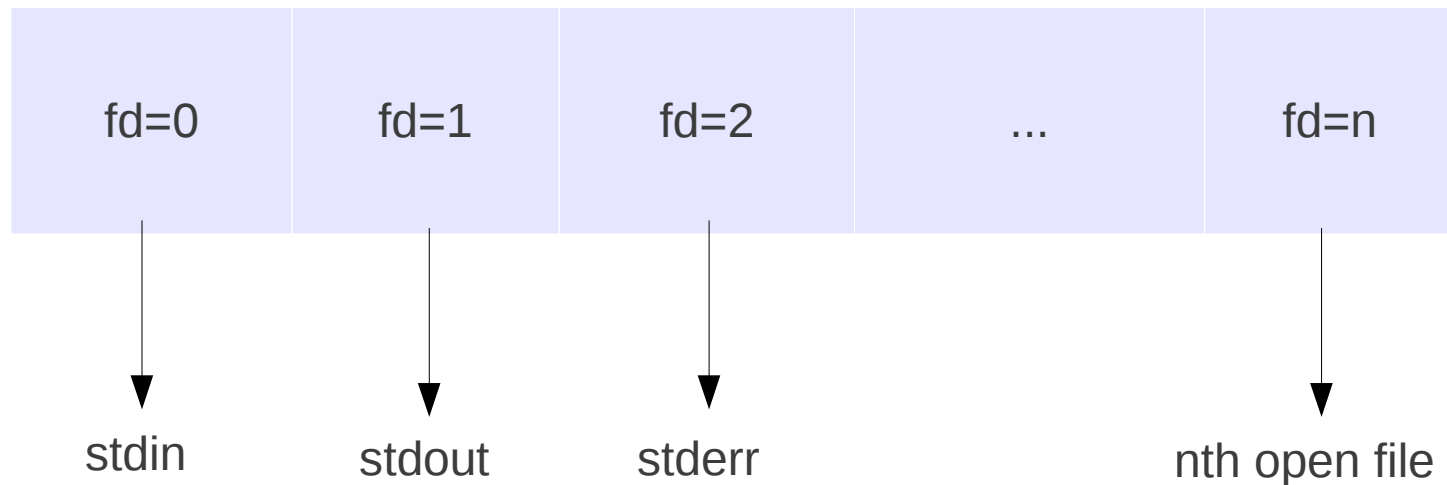


- Review of File Descriptors
- `socket()`
- `bind()`, `listen()`, `accept()` & `connect()`
- `select()` & `FD_set()`
- `open()`, `read()`, `write()`
- Other Considerations
- Summary & Further Reading



Review of File Descriptors

- From the last lab:
 - ♦ An integer index for an entry in the file descriptor table
 - ♦ Refer to files, directories, block or character devices, **sockets**, pipes





- Opens a file, returns a file descriptor, or error

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

```
#include <sys/stat.h>
```

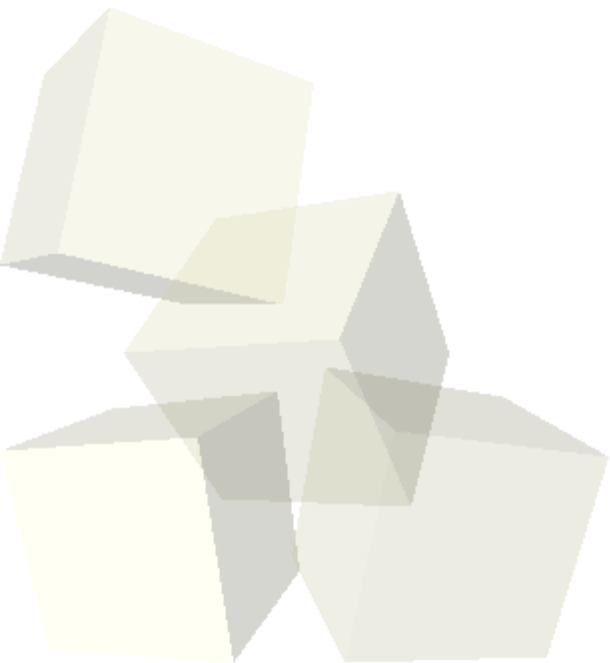
```
#include <fcntl.h>
```

```
int open(const char *pathname, int flags);
```



- Example which replaces stdin (file descriptor 0) with a file, and echos the contents to stdout
- Uses the dup2 function (more about this in man 2 dup2)

<http://www.uoguelph.ca/~jernst/cis3110/fd.c>



Useful Functions for IPC Over Networks

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connection
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

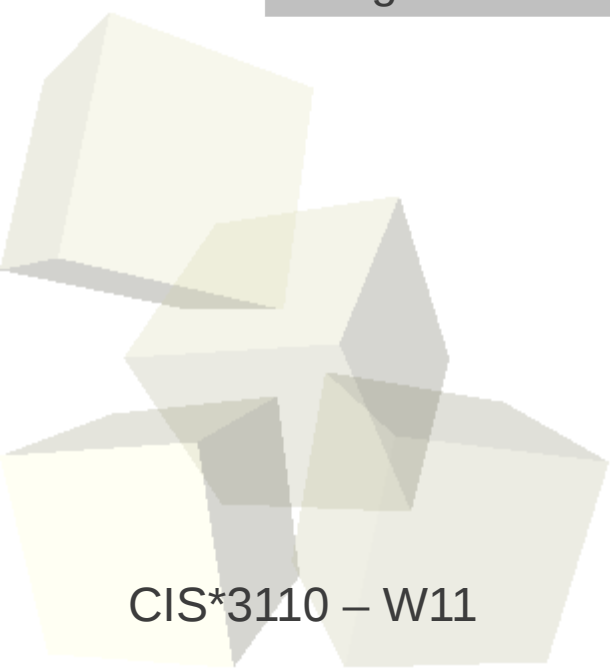
Figure 4-14 – Distributed Systems: Principles & Paradigms



A useful structure - sockaddr_in

struct sockaddr_in		
short int	sin_family	Address family, (is it local, Ipv4, Ipv6 etc?)
unsigned short int	sin_port	Port number (which process to map to)
struct in_addr	sin_addr	Internet address (127.0.0.1, 192.168..etc)
unsigned char	sin_zero[8]	Padding (to make same size as sockaddr)

Structure for IPv4 socket





A useful function - getaddrinfo()

- Can help fill in `sock_addr` structures instead of manually entering information such as IP addresses

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

- `int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);`
- example later...



■ Socket:

- An **endpoint** for **communication** (textbook)
- “A **pair** of process communicating over a network employ a **pair** of sockets – one for each process” (textbook)

Computer A



socket()

Computer B



socket()



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

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

(details to follow)



■ Socket domains

- ♦ AF_INET - IPv4 Internet communication
 - ♦ AF_INET6 – Ipv6 Internet communication
 - ♦ AF_UNIX, AF_LOCAL – local communication
 - ♦ Others but less likely to use
-
- ♦ Keep in mind, IPv4 is most common, but IPv6 should be used since IPv4 addresses are almost out!



■ Types of Sockets

- ◆ Reliable:
 - TCP - “Stream Socket” - `SOCK_STREAM`
 - Used for HTTP (web pages), SMTP (email), NFS (network file system), etc.
- ◆ Unreliable:
 - UDP - “Datagram Socket” - `SOCK_DGRAM`
 - Used in multimedia streaming, torrents, video games, etc.
- ◆ Raw & Other types:
 - Likely will not need to use these...



- For certain types of sockets only a single protocol exists – ex) `SOCK_DGRAM` uses UDP and `SOCK_STREAM` uses TCP
 - ◆ In this case we can leave `protocol = 0`
 - ◆ Otherwise, fill in with appropriate protocol number

socket() & getaddrinfo() - example

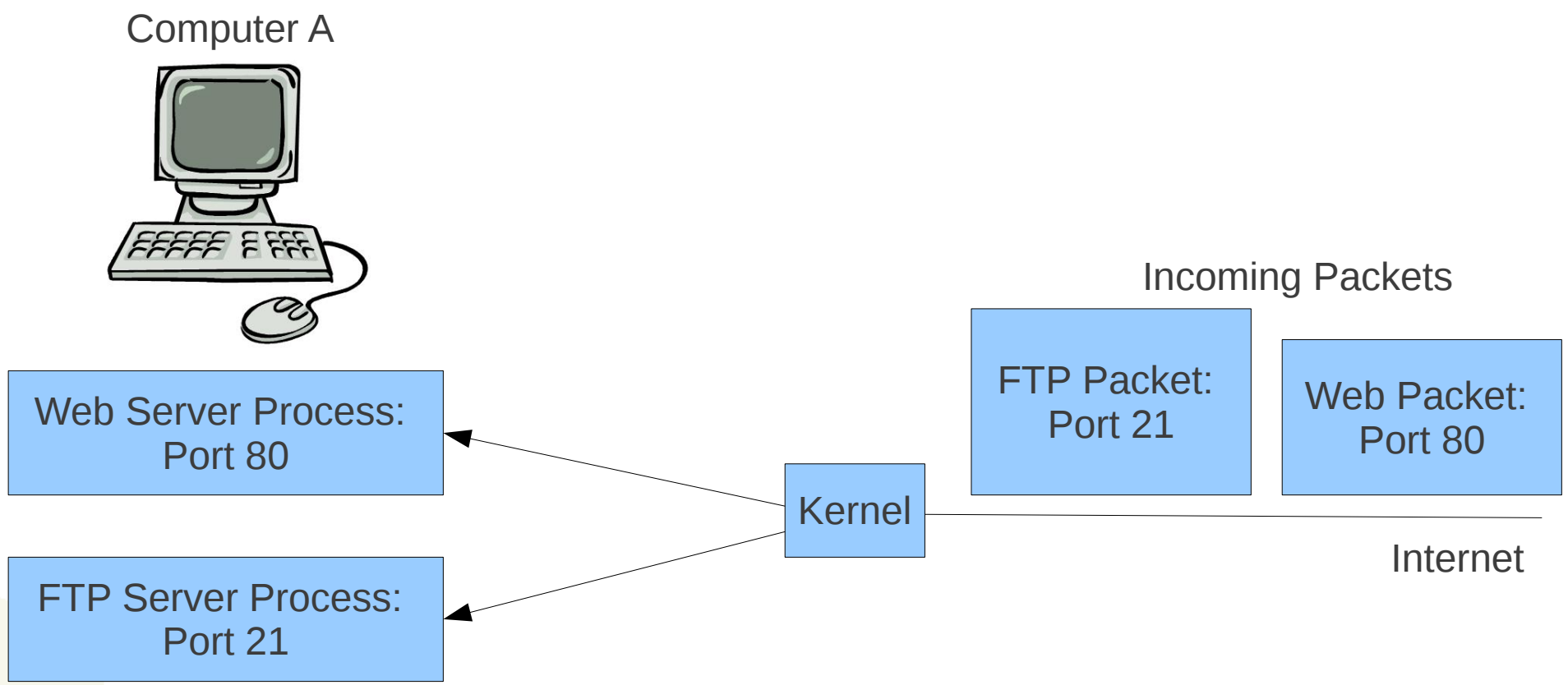
- showip.c is a quick example of how to use a socket and some of the benefits of using getaddrinfo
- The program simple shows the ip address for a hostname – ex) type in google.ca and it will find the ip address
- <http://www.uoguelph.ca/~jernst/cis3110/showip.c>



- Assigns a **unique port** (integer) to a process's socket descriptor (Beej's network guide)
 - ◆ Port allows the kernel to direct packets to the process they are intended for
 - ◆ Often called by the “server” in a “client-server” model
 - ◆ The client usually only need to call connect() .. more details on that later
 - ◆ This call will fail if the **port** is already used (unless it has been specifically set for re-use)



bind() - ports





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

```
int bind(int s, const struct sockaddr *name
         int namelen);
```

- **s** is the file descriptor of the socket
- **name** is the `sock_addr_in` structure
- **namelen** is the size of `name`
 - ◆ (or size of `sock_addr` struct)



listen() and accept()

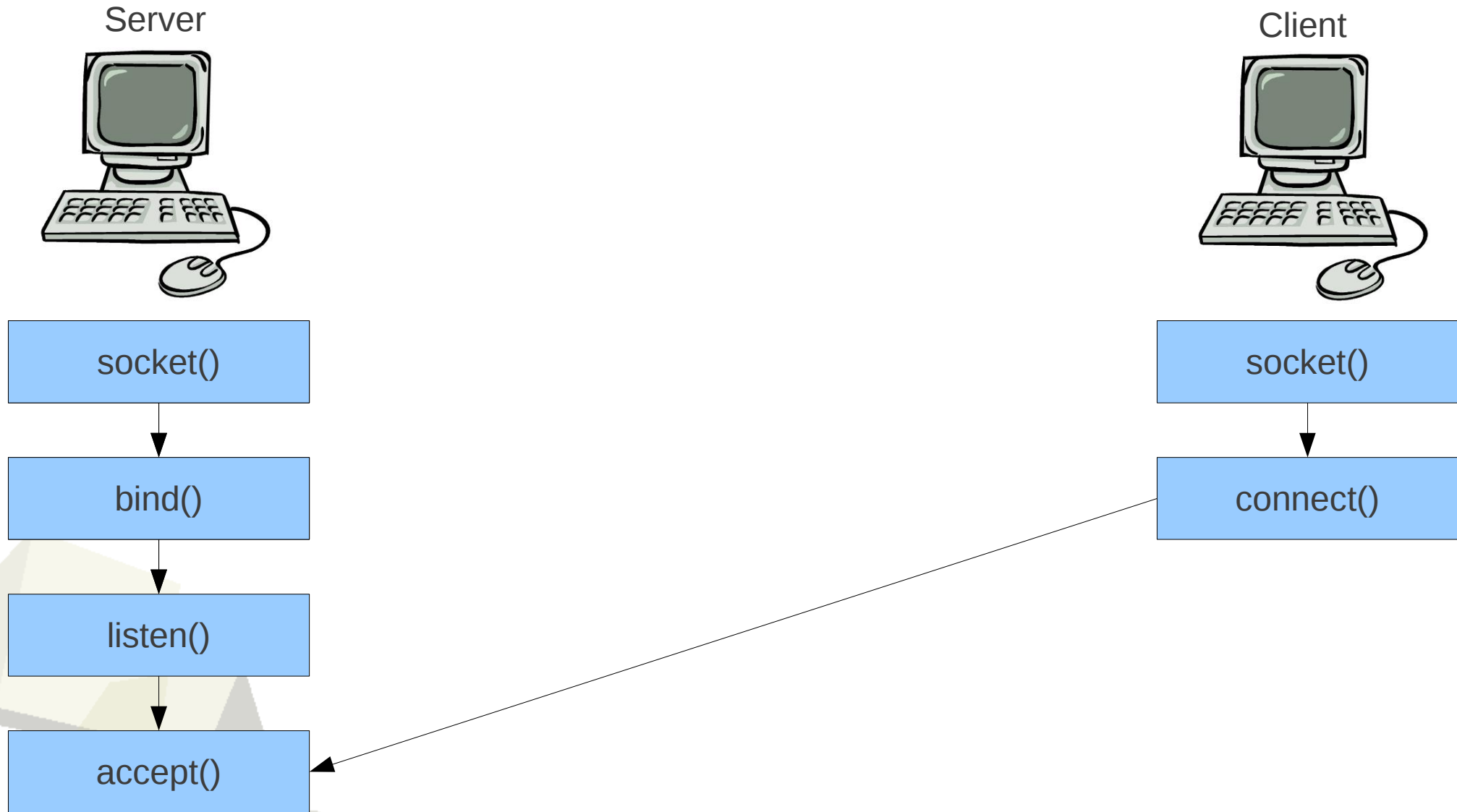
- In order to receive packets on a specific port, it is a two step process

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

```
int listen(int s, int backlog);  
int accept(int s, struct sockaddr *addr,  
           int *addrlen);
```



listen(), accept() & connect()





listen() & accept()

- listen() needs the socket descriptor and how many requests it should queue
- accept() needs the socket descriptor, and address information such as **port**, **type** of address (local, ipv4, ipv6 etc), & the **address to accept connections on** (may have more than one available on a computer)
- accept() **blocks** by default
 - ◆ Can make it non-blocking using **fcntl()**



- Called by the “client” to a “server” which has already called `listen()` and `accept()`
- Do not need to `bind()` on the “client”

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

```
int connect(int s, const struct sockaddr *name,
            int namelen);
```

- Need to know the socket descriptor, the **server** address information) that the client should connect to



Simple client / server example

- Simple echo server
 - ♦ receives string from client, prints to stdout
<http://www.uoguelph.ca/~jernst/cis3310/server.c>
- Simple echo client:
 - ♦ Sends string from stdin to server
<http://www.uoguelph.ca/~jernst/cis3110/client.c>
- Get ip address of server using ifconfig if using linux. Server takes port as argument, client takes port & ip address of server.



Simple pipe parent/child example

- Simple pipe child:
 - ♦ receives string from parent, prints to stdout
 - ♦ http://www.uoguelph.ca/~jernst/cis3110/pipe_child.c
- Simple pipe parent:
 - ♦ Sends string from stdin to child which prints it out
 - ♦ http://www.uoguelph.ca/~jernst/cis3110/pipe_parent.c
- Make sure to compile pipe_child.c with executable name pipe_child for this to work



- Allows monitoring of **sets of file descriptors** (or sockets) at the same time within the same process
 - ♦ (normally not possible because of blocking)
- Suppose we have a server listening to two sockets at once, or we want to be able to send and receive data at the same time
- One option is using `fork()` and having each socket in its own process, another option is `select`



```
#include <unistd.h>
#include <sys/types.h>
#include <bstring.h>
#include <sys/time.h>
```

```
int select (int nfd, fd_set *rfd, fd_set *wrfds,
            fd_set *exceptfds, struct timeval *timeout);
```

- nfd = highest numbered file descriptor + 1
- rfd, wrfd, exceptfds = pointer to a file descriptor set for read, write or exceptions to be watched
 - ◆ (more on this in a minute)
- timeout is upper bound on return time



- **FD_SET(fd, &fdset)**
 - ◆ Adds a file descriptor to a set

- **FD_CLR(fd, &fdset)**
 - ◆ Removes a file descriptor from a set

- **FD_ISSET(fd, &fdset)**
 - ◆ Returns true if the file descriptor is in the set

- **FD_ZERO(&fdset)**
 - ◆ Clears all entries from the set



- Simple example, waits 2.5 seconds for stdin input and then tells whether a string was entered or not
 - ◆ <http://www.uoguelph.ca/~jernst/cis3310/select.c>
- Compared with normal stdin input this does not block until input is received
- Similar technique can be used for non-blocking recv or read from sockets
- Can also be used with multiple socket I/O



- Attempts to read *count* bytes from file descriptor *fd* into the buffer
- May be a blocking function, unless the fd is set to non-blocking with **fcntl()**

```
#include <unistd.h>
```

```
ssize_t read(int fd, void * buf, size_t count);
```

- More details next week...



- Writes up to *count* bytes from the buffer to the file
- This may be less in several cases:
 - ◆ Insufficient space on the medium
 - ◆ Call interrupted by a signal
- On success returns # bytes sent
 - ◆ (0 indicates nothing sent)

```
#include <unistd.h>
```

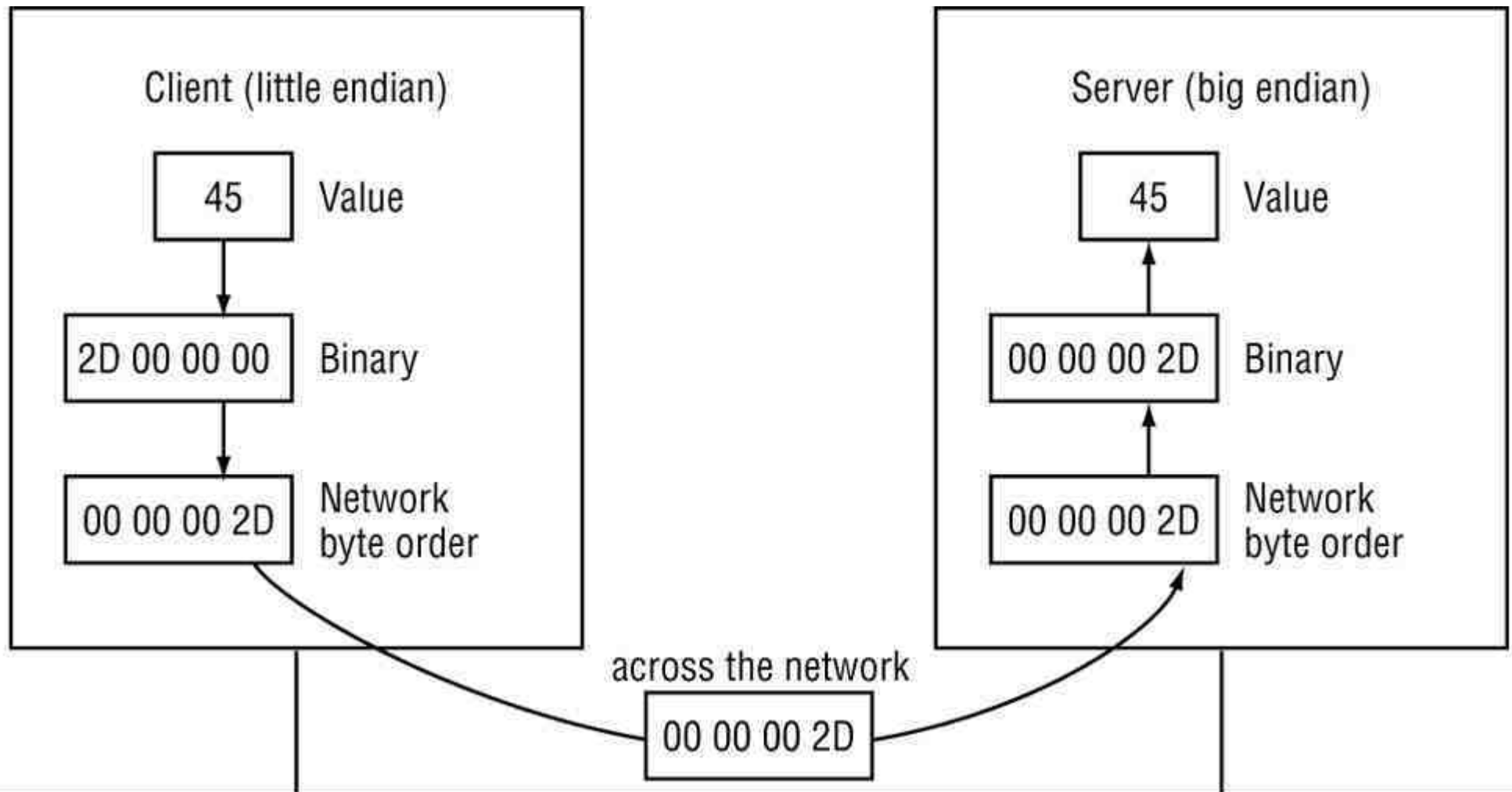
```
ssize_t write(int fd, const void *buf, size_t count);
```

- More details next week...



Some Considerations...

■ Network vs Host Byte Ordering:



Source: codeidol.com



■ Network vs Host Byte Ordering:

```
#include <sys/types.h>  
#include <netinet/in.h>
```

```
u_long  htonl(u_long hostlong);  
u_short htons(u_short hostshort);  
u_long  ntohl(u_long netlong);  
u_short ntohs(u_short netshort);
```

(host to network and network to host)



Some considerations...

- Network vs Host Byte Ordering
 - ◆ Important in things like the port in the `sock_addr` structures
- Return values:
 - ◆ Always check the return values of the functions from today's lab
 - ◆ Errors checking is especially important in network programming since the medium may be unreliable
 - (Think about wireless networks, or even wired networks which are congested)



Client-Server Models

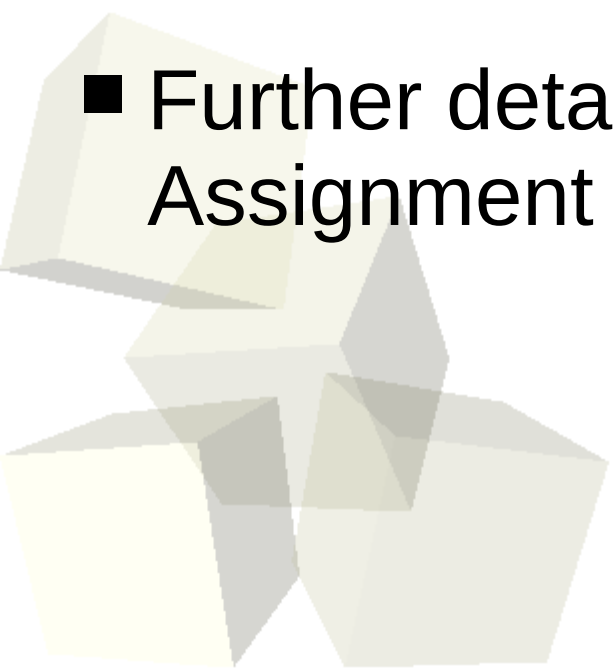
■ Four Models:

	synchronous	asynchronous
single process	no fork	“manual” i/o
multi-process	1 connection / fork	

Source: Prof. McCaughan's Notes



- Sending and Receiving details
 - ◆ read(), write(), send(), recv(), sendto(), recvfrom()
- Mechanics of various client server models
 - ◆ Iterative, Forking, Concurrent, etc.
- Examples & Implementations in detail
- Further details which may be useful on the 3rd Assignment





Summary & Further Reading

- Chapter 3.6 – Communication in Client-Server Systems (textbook)
- Kurose, J.F. & Ross, K.W. Computer Networking: A Top Down Approach Featuring the Internet
 - ◆ Chapter 2 – Application Layer
- Tannenbaum, A.S. & Steen, M.V. Distributed Systems: Principles and Paradigms
 - ◆ Chapter 4 – Message Oriented Communication
- Beej's Guide to Network Programming (free)
 - ◆ <http://beej.us/guide/bgnet/output/html/multipage/index.html>



- Assignment 2 regrades this week Tues & Thurs from 1-2 in Reynolds 001A

