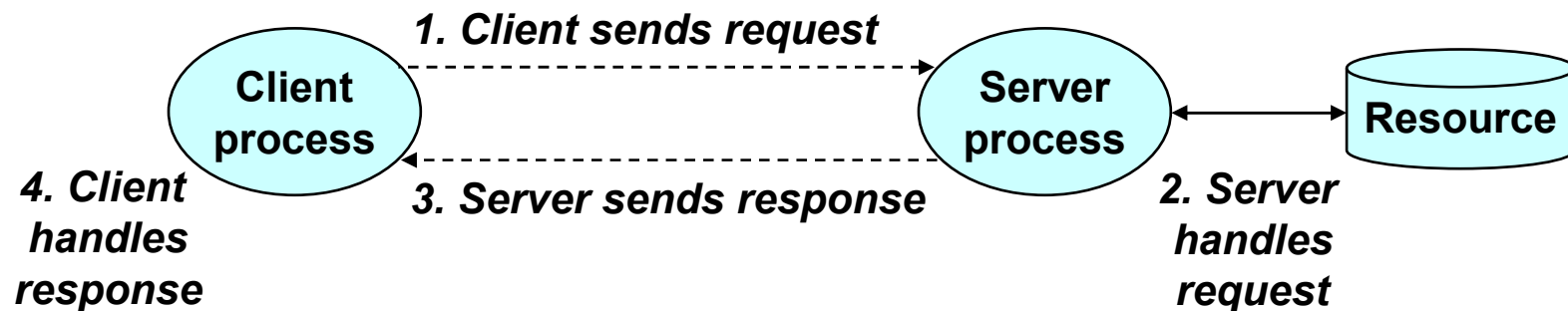


# Socket Programming

18-345 Fall 2008

# A Client-Server Transaction

- Most network applications are based on the client-server model:
  - A server process and one or more of client processes.
  - Server manages some resource. (ex. Data base)
  - Server provides service by manipulating resource for clients.



**Note: clients and servers are processes running on hosts (can be the same or different hosts).**

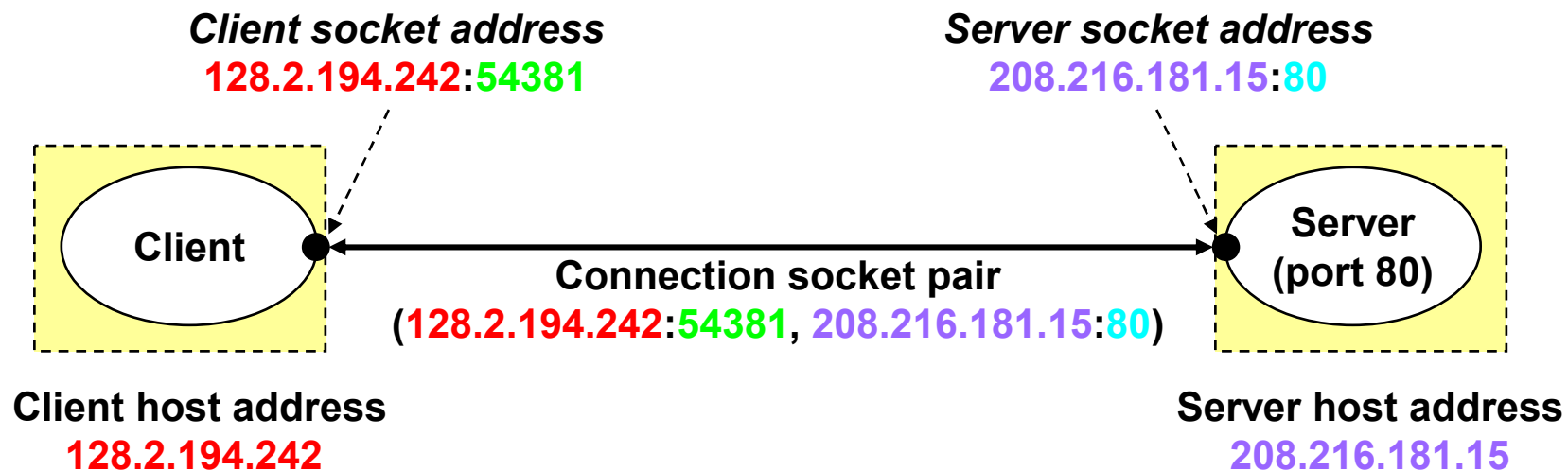
# The Socket Interface

- What is a **socket**?
  - A descriptor that lets an application read/write from/to the network
  - Similar abstraction for network I/O as file I/O
  - Clients and servers communicate by reading/writing from/to socket descriptors



# Internet Connections

- Clients and servers are communicate by sending streams of bytes over connections.
- Socket address is identified by an **IPaddress:port** pair.
- A **port** is a 16-bit unsigned integer identifying a process (ephemeral, not physical port.)
- Ports below 1024 are reserved for well-known services (http:80, ftp:21, ssh:22, telnet:23)
- Ports 1024-49151 are registered ports managed by **IANA** (<http://www.iana.org/assignments/port-numbers>)
- Dynamic/Private ports are those from 49152-65535



*Note: 54381 is an ephemeral port allocated by the kernel*

*Note: 80 is a well-known port associated with Web servers*

# Key data structures

## IP Address

```
/* Internet address structure */
struct in_addr {
    unsigned int s_addr; /* network byte order (big-endian) */
};
```

- 32-bit IP addresses are stored in an *IP address struct* in `<netinet/in.h>`
  - IP addresses are always stored in memory in **network byte order** (big-endian byte order)
  - True in general for any integer transferred in a packet header from one machine to another.
    - E.g., the port number used to identify an Internet connection.
- Handy network byte-order conversion functions:
  - **htonl**: convert long int from host to network byte order.
  - **htons**: convert short int from host to network byte order.
  - **ntohl**: convert long int from network to host byte order.
  - **ntohs**: convert short int from network to host byte order.

**You will have to use these functions in the project.**

# Byte-Ordering

- Consider a hexadecimal **4A3B2C1D** at address 100. the bytes could be stored within the address range 100 through 103 in the following order:
- Big-endian (“big end first”)**

	100	101	102	103	
...	4A	3B	2C	1D	...

- the **most significant byte (msb)**, 4A) is stored at the lowest address.
- Used by Motorola/SPARC and **network devices**.

- Little-endian (“little end first”)**

	100	101	102	103	
...	1D	2C	3B	4A	...

- The **least significant byte** (lsb, 1D) is stored at the lowest address
  - Used by Intel x86, DEC VAX
- Endianness does *not* denote what the value *ends* with when stored in memory, but rather *which end* it begins with.

# Key data structures

## Internet Socket Address

```
struct sockaddr_in {
    unsigned short  sin_family; /* address family (always AF_INET) */
    unsigned short  sin_port;   /* port num in network byte order */
    struct in_addr  sin_addr;   /* IP addr in network byte order */
    unsigned char   sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```

- Also defined in `<netinet/in.h>`
- IP address and port number must be stored in network byte order.

**Generic socket address:** used in connect, bind

```
struct sockaddr {
    unsigned short  sa_family; /* protocol family */
    char           sa_data[14]; /* address data. */
};
```

# Key data structures

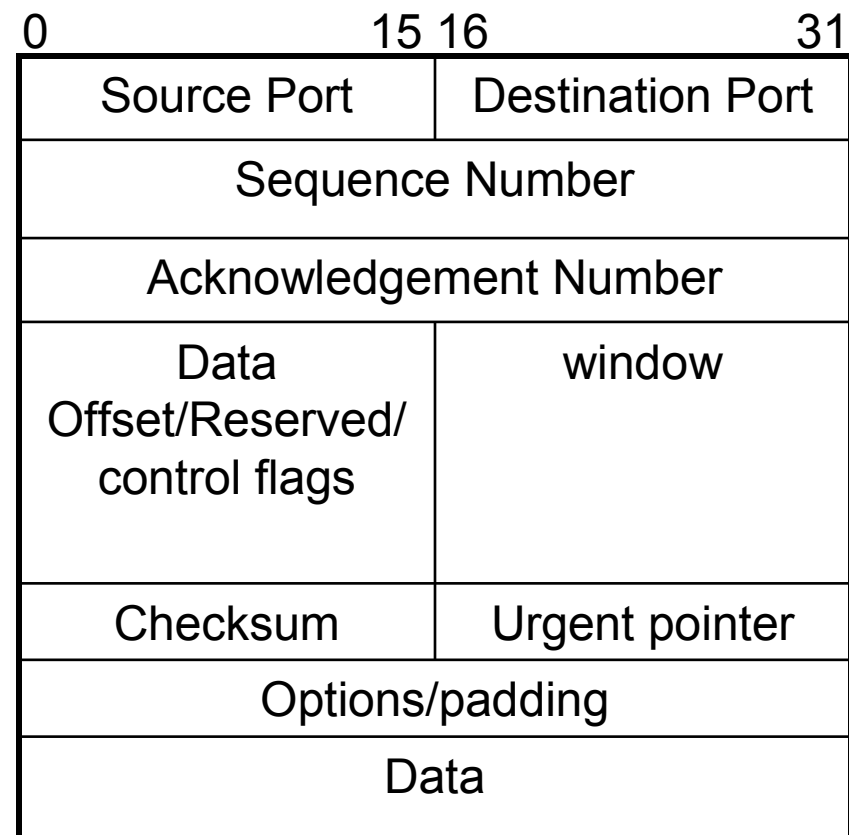
## DNS host entry

```
/* DNS host entry structure */
struct hostent {
    char    *h_name;          /* official domain name of host */
    char    **h_aliases;     /* null-terminated array of alias domain */
    int     h_addrtype;      /* host address type (AF_INET) */
    int     h_length;        /* length of an address, in bytes */
    char    **h_addr_list;   /* null-terminated array of in_addr structs */
};
#define h_addr h_addr_list[0] /* first entry in h_addr_list */
```

- This structure is defined in `<netdb.h>`
- The Internet maintains a mapping between IP addresses and domain names in a huge worldwide distributed database called *DNS*.
  - Conceptually, programmers can view the DNS database as a collection of millions of *host entry structures*:
- Functions for retrieving host entries from DNS:
  - `gethostbyname()` : query key is a DNS domain name.
  - `gethostbyaddr()` : query key is an IP address.

# Transmission Control Protocol (TCP)

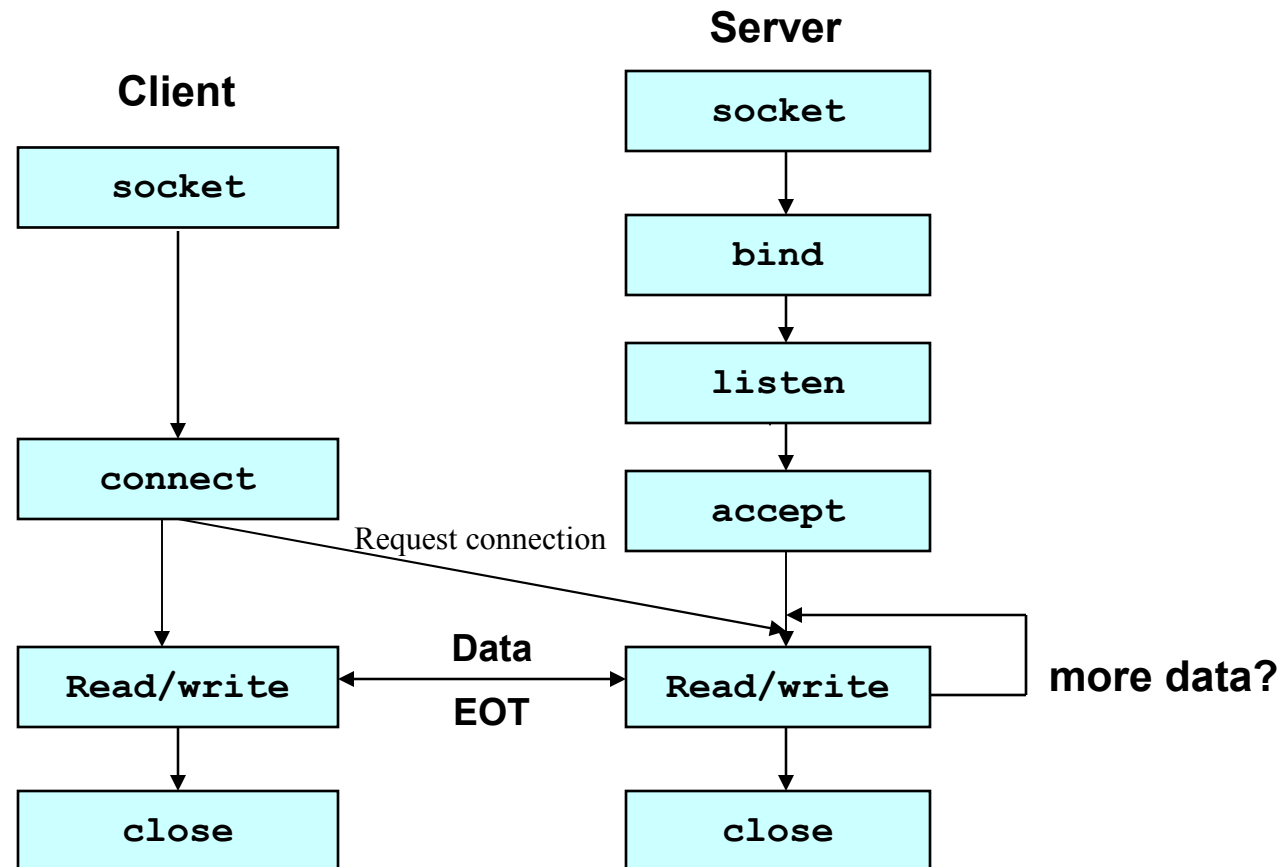
- Allow networked computers to reliable transfer of **byte stream**
- Connection-oriented provides error recovery and flow control
- Data received in correct order
- More information on RFC793



**TCP Packet Format**

# Programming TCP Socket

## A Simple Control Flow



# Client: Creating a Socket

```
#include <sys/types.h>
#include <sys/socket.h>
sd = socket(pf, type, protocol);
```

- **sd** is an integer socket descriptor or -1 if failed.
- **pf** specifies protocol family. Use **AF\_INET**(Internet protocol)
- Type specifies type of communication
  - **SOCK\_STREAM** for reliable stream delivery service (TCP)
  - **SOCK\_DGRAM** for connectionless datagram service (UDP)
- Protocol specifies a particular protocol in the family. Just use 0.
- Example:

```
clientfd = socket(AF_INET, SOCK_STREAM, 0);
```

# Client: Obtaining a Server's Address

```
#include <netdb.h>
hp = gethostbyname(hostname) ;
```

- **hostname** is a string of host name or an IPv4 address in standard dot notation.
- Returns a pointer to **struct hostent**
- Use **hp** to fill in a server address structure **struct sockaddr\_in**
- Example: getting address to **pegasus.ece.cmu.edu:12345**

```
struct sockaddr_in server;
struct hostent *hp;
hp = gethostbyname("pegasus.ece.cmu.edu") ;
bcopy(hp->h_addr, &(server.sin_addr), hp->h_length);
server.sin_family = AF_INET;
server.sin_port = htons(12345);
```

# Client: Creating a Connection

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sd, const struct sockaddr *to,
            int tolen );
```

- **sd** is an integer socket descriptor
- **to** is a pointer to the socket address structure specifying the destination address
- **tolen** is the size of the socket address structure (use `sizeof(struct sockaddr_in)`)

# Client: Closing a Socket

```
#include <unistd.h>  
close (socketfd) ;
```

- **socketfd** is the integer socket descriptor
- Must terminate both ends of a socket

# Bare minimum TCP Client

```
#include <...>
int main(int argc, char **argv) {
    int sd;
    struct sockaddr_in server;
    struct hostent *hp;

    sd = socket(AF_INET, SOCK_STREAM, 0);

    server.sin_family = AF_INET;
    server.sin_port = htons(12345);
    hp = gethostbyname(argv[1]);
    bcopy(hp->h_addr, &(server.sin_addr), hp->h_length);

    for (;;) {
        connect(sd, (struct sockaddr *) &server, sizeof(server));

        write(sd, "Hi" ,2);
        sleep(2);
    }
    close(sd);
}
```

# Client: Sending Datagrams

```
#include <sys/types.h>
#include <sys/socket.h>
int  sendto(int sd, const void *msg, size_t len, int flags,
            const struct sockaddr *to, socklen_t tolen);
```

- **sd** is an integer socket descriptor
- **msg** is a pointer to the buffer to be sent
- **len** is the buffer's size
- **flags** could be set to specify sending option (usually set to zero)
- **to** is a pointer to the socket address structure specifying the destination address
- **tolen** is the size of the socket address structure (use `sizeof(struct sockaddr_in)`)

# Bare minimum UDP Client

```
#include <...>
int main(int argc, char **argv) {
    int sd;
    struct sockaddr_in server;
    struct hostent *hp;

    sd = socket(AF_INET, SOCK_DGRAM, 0);

    server.sin_family = AF_INET;
    server.sin_port = htons(12345);
    hp = gethostbyname(argv[1]);
    bcopy(hp->h_addr, &(server.sin_addr), hp->h_length);

    for (;;) {
        sendto(sd, "HI", 2, 0, (struct sockaddr *) &server, sizeof(server));
        sleep(2);
    }
    close(sd);
}
```

# Server: Binding a Local Address

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

## **bind(socket, localaddr, addrlen)**

- Associates a socket to a machine
- **socket** is the integer socket descriptor
- **localaddr** is the struct `sockaddr_in`
- **addrlen** is the size of the socket address structure (use `sizeof(struct sockaddr_in)`)

```
int sd;
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(12345);

sd = socket(AF_INET, SOCK_DGRAM, 0);
bind(sd, (struct sockaddr *)&server, sizeof(server));
```

Tell the system to  
fill in IP address of  
local machine

# Server: Accepting Connection

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

- **Parameters:**

- **sd** is an integer socket descriptor
- **backlog** max number of connection requests system will queue while waiting for server to accept
- **addr** pointer to addr struct
- **AddrLen** pointer to integer that specifies space in addr struct

# Bare minimum TCP server

```
#include <...>
int main() {
    int n, sd, new_sd, client_len;
    struct sockaddr_in server, client;
    char buf[512];

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(12345);

    sd = socket(AF_INET, SOCK_STREAM, 0);

    bind(sd, (struct sockaddr *)&server, sizeof(server));
    listen(sd, 5);

    for(;;){
        client_len = sizeof(client);
        new_sd = accept(sd, (struct sockaddr *)&client, &client_len);
        n = read(new_sd, buf, sizeof(buf), 0);
        buf[n] = '\\0';
        printf("Received: %s\\n", buf);

        close(new_sd);
    }
    close(sd);
    return 0;
}
```

# Server: Receiving Datagrams

- Possible socket I/O system calls:
  - `int recv(int sd, void *buf, size_t len, int flags);`
  - `int recvfrom(int sd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);`
- Parameters:
  - `sd` is an integer socket descriptor
  - `buf` contains the memory address to store the data
  - `len` specifies maximum size of bytes to read
  - `flags` specifies I/O option, usually zero
  - `from`, if not null, will be filled with sender's address (for UDP)
  - `fromlen` will be filled with size of sender's address structure
- These I/O functions return number of bytes received or -1 if error
- `recvfrom` will supply sender's address to the variable `from`

# Bare minimum UDP server

```
#include <...>
int main() {
    int n, sd;
    struct sockaddr_in server;
    char buf[512];

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(12345);

    sd = socket(AF_INET, SOCK_DGRAM, 0);

    bind(sd, (struct sockaddr *)&server, sizeof(server));

    for(;;){
        n = recv(sd, buf, sizeof(buf), 0);
        buf[n] = '\0';
        printf("Received: %s\n", buf);
    }
    close(sd);
    return 0;
}
```

# How does this relate to 213 I/O library

- Csapp.h functions use same functions described here
- They are essentially wrappers that check for certain errors
- DON'T USE csapp.h
  - We want to see what you can write
  - The functions don't handle all cases correctly, students who used them in the past had problems

# Project 1: Socket Programming

- Check out section 2.4 of your textbook (Leon-Garcia)
- The codes in this lecture are for demonstration purpose only
- Your “**real**” network code should be
  - **Correct**: works all the time
  - **Robust**: properly handles errors/exceptions
  - **Portable**: works across different systems (UNIX/LINUX)
- Some useful reference for C
  - Harbison, Steele; **C: A Reference Manual**; Prentice Hall 2002
  - Muldner; **C for java™ programmers**; Addison-Wesley 2000

# That's all!!!!

- References:
  - Bryant, O'Hallaron;  
**Computer Systems: A Programmer's Perspective**
  - Leon-Garcia, Widjaja; **Communication Networks**
  - Previous 18345 Lecture notes
  - ietf.org, **rfc768**, **rfc793**
  - Wikipedia.org
- The Ultimate Reference...
  - Unix/Linux man page 😊

# Resources

- Network Programming Tutorial
  - <http://www.ecst.csuchico.edu/~beej/guide/net/html/syscalls.html>
- Makefile
  - <http://oucsace.cs.ohiou.edu/~bhumphre/makefile.html>
  - <http://vergil.chemistry.gatech.edu/resources/programming/c-tutorial/make.html>
- Internet
  - <http://www.ietf.org/>
  - <http://www.iana.org>