

## Introduction to Unix Network Programming

Reference: Stevens Unix Network Programming

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzاهر, Kravets, Gupta

1

## Network Programming

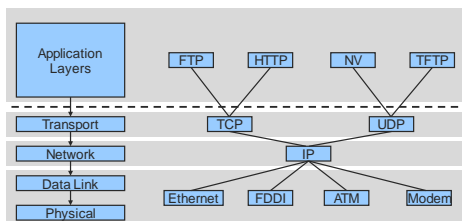
- Key Components:
  - Internet protocols
    - IP, TCP, UDP, etc
  - Sockets
    - API - application programming interface
- Why focus on the Internet?
  - Internet Protocol (IP)
    - IP is standard
    - allows a common namespace across most of Internet
    - reduces number of translations, which incur overhead
  - Sockets
    - reasonably simple and elegant, Unix interface

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzاهر, Kravets, Gupta

2

## Internet Protocols



CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzاهر, Kravets, Gupta

3

## Network Programming with Sockets

- Socket
  - Host-local, application-created, OS-controlled
  - Application process can both send and receive messages to/from another application process
- Sockets API
  - A transport layer service interface
    - Introduced in 1981 by BSD 4.1
    - Implemented as library and/or system calls
    - Similar interfaces to TCP and UDP
    - Also interface to IP (for super-user); "raw sockets"

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzاهر, Kravets, Gupta

4

## [ Beej's Guide ]

- How-to guide on network programming using Internet sockets, or "sockets programming"

<http://beej.us/guide/bgnet/>

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

5



## [ Outline ]

- Client-Server Model
- TCP Connection
- UDP Services
- Addresses and Data
- Sockets API
- Example

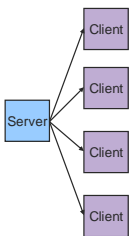
CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

6



## [ Client-Server Model ]



- Asymmetric Communication
  - Client sends requests
  - Server sends replies
- Server/Daemon
  - Well-known name
  - Waits for contact
  - Processes requests, sends replies
- Client
  - Initiates contact
  - Waits for response

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

7



## [ Client-Server Model ]

- Client contacts server
  - Server process must first be running
  - Server must have created socket that accepts client's contact
- Client: To initiate contact
  - Create client-local TCP socket
  - Specify IP address, port number of server process
  - When client creates socket: client TCP establishes connection to server TCP
- Server: When contacted by client
  - Create new socket for server process to communicate with client
  - Allows server to talk with multiple clients
  - Source port numbers used to distinguish clients

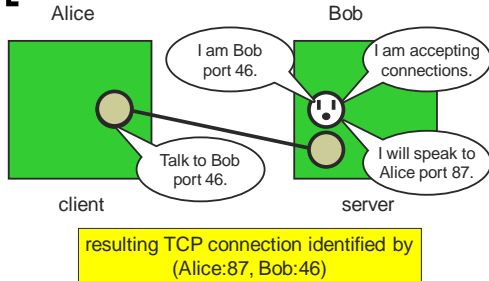
CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

8



## Example Client-Server Setup



CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

9



## Client-Server Model

- Service Model
  - Concurrent
    - Server processes multiple clients' requests simultaneously
  - Sequential
    - Server processes only one client's requests at a time
  - Hybrid
    - Server maintains multiple connections, but processes responses sequentially
- Client and server categories are not disjoint
  - A server can be a client of another server
  - A server can be a client of its own client
- Examples
  - Web
  - FTP
  - Telnet

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

10



## TCP Connections

- Transmission Control Protocol (TCP) Service
  - OSI Transport Layer
  - Service Model
    - Byte stream (interpreted by application)
    - 16-bit port space allows multiple connections on a single host
    - Connection-oriented
      - Set up connection before communicating
      - Tear down connection when done

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

11



## TCP Service

- Reliable Data Transfer
    - Guaranteed delivery
    - Exactly once if no catastrophic failures
  - Sequenced Data Transfer
    - In-order delivery
  - Regulated Data Flow
    - Monitors network and adjusts transmission appropriately
  - Data Transmission
    - Full-Duplex byte stream
- Telephone Call
    - Guaranteed delivery
    - In-order delivery
    - Connection-oriented
    - Setup connection followed by conversation

CS 241

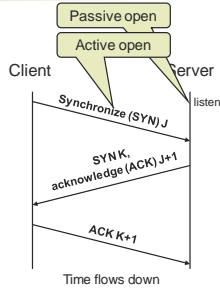
Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

12



## TCP Connection Establishment

- 3-Way Handshake
  - Sequence Numbers
    - J,K
  - Message Types
    - Synchronize (SYN)
    - Acknowledge (ACK)
  - Passive Open
    - Server listens for connection from client
  - Active Open
    - Client initiates connection to server



CS 241

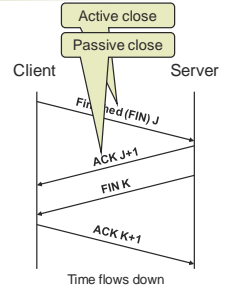
Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

13



## TCP Connection Termination

- Either client or server can initiate connection teardown
- Message Types
  - Finished (FIN)
  - Acknowledge (ACK)
- Active Close
  - Sends no more data
- Passive close
  - Accepts no more data



CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

14



## UDP Services

- User Datagram Protocol Service
  - OSI Transport Layer
  - Provides a thin layer over IP
  - 16-bit port space (distinct from TCP ports) allows multiple recipients on a single host

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

15



## UDP Services

- Unit of Transfer
  - Datagram (variable length packet)
- Unreliable
  - No guaranteed delivery
  - Drops packets silently
- Unordered
  - No guarantee of maintained order of delivery
- Unlimited Transmission
  - No flow control

- Postal Mail
  - Single mailbox to receive all letters
  - Unreliable
  - Not necessarily in-order
  - Letters sent independently
  - Must address each reply

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

16



## Addresses and Data

- Internet domain names
  - Human readable
  - Variable length
  - Ex: **sal.cs.uiuc.edu**
- IP addresses
  - Each attachment point on Internet is given unique address
  - Easily handled by routers/computers
  - Fixed length
  - Somewhat geographical
  - Ex: **128.174.252.217**

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

17



## Byte Ordering

- Big Endian vs. Little Endian
  - Little Endian (Intel, DEC):
    - Least significant byte of word is stored in the lowest memory address
  - Big Endian (Sun, SGI, HP):
    - Most significant byte of word is stored in the lowest memory address
  - Example: **128.2.194.95**

Big Endian	128	2	194	95
Little Endian	95	194	2	128

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

18



## Byte Ordering

- Big Endian vs. Little Endian
  - Little Endian (Intel, DEC):
    - Least significant byte of word is stored in the lowest memory address
  - Big Endian (Sun, SGI, HP):
    - Most significant byte of word is stored in the lowest memory address
  - Network Byte Order = Big Endian
    - Allows both sides to communicate
    - Must be used for some data (i.e. IP Addresses)
    - Good form for all binary data

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

19



## Byte Ordering Functions

- 16- and 32-bit conversion functions (for platform independence)
- Examples:

```
int m, n;
short int s, t;

m = ntohl (n) // net-to-host long (32-bit) translation
s = ntohs (t) // net-to-host short (16-bit) translation
n = htonl (m) // host-to-net long (32-bit) translation
t = htons (s) // host-to-net short (16-bit) translation
```

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

20



## Socket Address Structure

- IP address:
 

```
struct in_addr {
    in_addr_t s_addr;    /* 32-bit IP address */
};
```
- TCP or UDP address:
 

```
struct sockaddr_in {
    short sin_family;    /* e.g., AF_INET */
    ushort sin_port;    /* TCP/UDP port */
    struct in_addr;     /* IP address */
};
```
- all but `sin_family` in network byte order

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

21



## Structure: hostent

- The `hostent` data structure (from `/usr/include/netdb.h`)
  - Canonical domain name and aliases
  - List of addresses associated with machine
  - Also address type and length information

```
struct hostent {
    char* h_name;        /* official name of host */
    char** h_aliases;   /* NULL-terminated alias list */
    int h_addrtype;     /* address type (AF_INET) */
    int h_length;       /* length of addresses (4B) */
    char** h_addr_list; /* NULL-terminated address list */
#define h_addr h_addr_list[0] /* backward-compatibility */
};
```

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

22



## Address Access/Conversion Functions

- All binary values are network byte ordered
- ```
struct hostent* gethostbyname (const char*
    hostname);
```
- Translate English host name to IP address (uses DNS)
- ```
struct hostent* gethostbyaddr (const char*
    addr, size_t len, int family);
```
- Translate IP address to English host name (not secure)
- ```
char* inet_ntoa (struct in_addr inaddr);
```
- Translate IP address to ASCII dotted-decimal notation (e.g., "128.32.36.37"); not thread-safe

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

23



## Address Access/Conversion Functions

- ```
in_addr_t inet_addr (const char* strptr);
```
- Translate dotted-decimal notation to IP address; returns -1 on failure, thus cannot handle broadcast value "255.255.255.255"
- ```
int inet_aton (const char* strptr, struct
    in_addr inaddr);
```
- Translate dotted-decimal notation to IP address; returns 1 on success, 0 on failure
- ```
int gethostname (char* name, size_t
    namelen);
```
- Read host's name (use with `gethostbyname` to find local IP)

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

24



## Sockets API

- Basic Unix Concepts
- Creation and Setup
- Establishing a Connection (TCP)
- Sending and Receiving Data
- Tearing Down a Connection (TCP)
- Advanced Sockets

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

25



## Basic Unix Concepts

- Input/Output – I/O
  - Per-process table of I/O channels
  - Table entries describe files, sockets, devices, pipes, etc.
  - Unifies I/O interface
  - Table entry/index into table called “file descriptor”
- Error Model
  - Return value
    - 0 on success
    - -1 on failure
    - NULL on failure for routines returning pointers
  - `errno` variable

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

26



## Socket Creation and Setup

- Include file `<sys/socket.h>`
- Create a socket
  - `int socket (int family, int type, int protocol);`
  - Returns file descriptor or -1.
- Bind a socket to a local IP address and port number
  - `int bind (int sockfd, struct sockaddr* myaddr, int addrlen);`
- Put socket into passive state (wait for connections rather than initiate a connection).
  - `int listen (int sockfd, int backlog);`

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

27



## Functions: socket

- ```
int socket (int family, int type, int protocol);
```
- Create a socket.
    - Returns file descriptor or -1. Also sets `errno` on failure.
    - `family`: address family (namespace)
      - `AF_INET` for IPv4
      - other possibilities: `AF_INET6` (IPv6), `AF_UNIX` or `AF_LOCAL` (Unix socket), `AF_ROUTE` (routing)
    - `type`: style of communication
      - `SOCK_STREAM` for TCP (with `AF_INET`)
      - `SOCK_DGRAM` for UDP (with `AF_INET`)
    - `protocol`: protocol within family
      - typically 0

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

28



## Example: socket

```
int sockfd, new_fd; /* listen on sock_fd, new
                    connection on new_fd */
struct sockaddr_in my_addr; /* my address */
struct sockaddr_in their_addr; /* connector addr */
int sin_size;

if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

29



## Function: bind

```
int bind (int sockfd, struct sockaddr*
          myaddr, int addrlen);
```

- Bind a socket to a local IP address and port number
  - Returns 0 on success, -1 and sets `errno` on failure
  - `sockfd`: socket file descriptor (returned from `socket`)
  - `myaddr`: includes IP address and port number
    - IP address: set by kernel if value passed is `INADDR_ANY`, else set by caller
    - port number: set by kernel if value passed is 0, else set by caller
  - `addrlen`: length of address structure
    - `= sizeof (struct sockaddr_in)`

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

30



## Example: bind

```
my_addr.sin_family = AF_INET; /* host byte order */
my_addr.sin_port = htons(MYPORT); /* short, network
                                   byte order */
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
/* automatically fill with my IP */
bzero(&(my_addr.sin_zero), 8); /* zero struct */

if (bind(sockfd, (struct sockaddr *)&my_addr,
         sizeof(struct sockaddr)) == -1) {
    perror("bind");
    exit(1);
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

31



## TCP and UDP Ports

- Allocated and assigned by the Internet Assigned Numbers Authority
  - see RFC 1700

|             |                                                                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| 1-512       | <ul style="list-style-type: none"> <li>■ standard services (see <code>/etc/services</code>)</li> <li>■ super-user only</li> </ul>             |
| 513-1023    | <ul style="list-style-type: none"> <li>■ registered and controlled, also used for identity verification</li> <li>■ super-user only</li> </ul> |
| 1024-49151  | <ul style="list-style-type: none"> <li>■ registered services/ephemeral ports</li> </ul>                                                       |
| 49152-65535 | <ul style="list-style-type: none"> <li>■ private/ephemeral ports</li> </ul>                                                                   |

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

32



## Reserved Ports

| Keyword  | Decimal | Description | Keyword    | Decimal | Description |
|----------|---------|-------------|------------|---------|-------------|
|          | 0       | Reserved    | time       | 37      | tcp         |
|          | 0       | Reserved    | time       | 37      | udp         |
| tcpxmx   | 1       | tcp         | name       | 42      | tcp         |
| tcpxmx   | 1       | udp         | name       | 42      | udp         |
| echo     | 7       | tcp         | nameserver | 42      | tcp         |
| echo     | 7       | udp         | nameserver | 42      | udp         |
| sysstat  | 11      | tcp         | nicname    | 43      | tcp         |
| sysstat  | 11      | udp         | nicname    | 43      | udp         |
| daytime  | 13      | tcp         | domain     | 53      | tcp         |
| daytime  | 13      | udp         | domain     | 53      | udp         |
| qotid    | 17      | tcp         | whois++    | 63      | tcp         |
| qotid    | 17      | udp         | whois++    | 63      | udp         |
| chargen  | 19      | tcp         | gopher     | 70      | tcp         |
| chargen  | 19      | udp         | gopher     | 70      | udp         |
| ftp-data | 20      | tcp         | finger     | 79      | tcp         |
| ftp-data | 20      | udp         | finger     | 79      | udp         |
| ftp      | 21      | tcp         | finger     | 79      | tcp         |
| ftp      | 21      | udp         | finger     | 79      | udp         |
| ssh      | 22      | tcp         | http       | 80      | tcp         |
| sah      | 22      | udp         | http       | 80      | udp         |
| telnet   | 23      | tcp         | www        | 80      | tcp         |
| telnet   | 23      | udp         | www        | 80      | udp         |
| smtp     | 25      | tcp         | www-http   | 80      | tcp         |
| smtp     | 25      | udp         | www-http   | 80      | udp         |
|          |         |             | kerberos   | 88      | tcp         |
|          |         |             | kerberos   | 88      | udp         |

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzahr, Kravets, Gupta

33



## Functions: listen

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

- Put socket into passive state (wait for connections rather than initiate a connection)
  - Returns 0 on success, -1 and sets **errno** on failure
  - sockfd**: socket file descriptor (returned from **socket**)
  - backlog**: bound on length of unaccepted connection queue (connection backlog); kernel will cap, thus better to set high
  - Example:
 

```
if (listen(sockfd, BACKLOG) == -1) {
    perror("listen");
    exit(1);
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzahr, Kravets, Gupta

34



## Establishing a Connection

- Include file `<sys/socket.h>`

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

- Connect to another socket.

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

- Accept a new connection. Returns file descriptor or -1.

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzahr, Kravets, Gupta

35



## Functions: connect

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

- Connect to another socket.
  - Returns 0 on success, -1 and sets **errno** on failure
  - sockfd**: socket file descriptor (returned from **socket**)
  - servaddr**: IP address and port number of server
  - addrlen**: length of address structure
    - `= sizeof (struct sockaddr_in)`
- Can use with UDP to restrict incoming datagrams and to obtain asynchronous errors

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzahr, Kravets, Gupta

36



## Example: connect

```

their_addr.sin_family = AF_INET; /* interp'd by host */
their_addr.sin_port = htons (PORT);
their_addr.sin_addr = *((struct in_addr*)he->h_addr);

bzero (&(their_addr.sin_zero), 8);
/* zero rest of struct */
if (connect (sockfd, (struct sockaddr*)&their_addr,
            sizeof (struct sockaddr)) == -1) {
    perror ("connect");
    exit (1);
}

```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

37



## Functions: accept

```

int accept (int sockfd, struct sockaddr* cliaddr,
           int* addrlen);

```

- Block waiting for a new connection
  - Returns file descriptor or -1 and sets `errno` on failure
  - `sockfd`: socket file descriptor (returned from `socket`)
  - `cliaddr`: IP address and port number of client (returned from call)
  - `addrlen`: length of address structure = pointer to `int` set to `sizeof (struct sockaddr_in)`
- `addrlen` is a **value-result** argument
  - the caller passes the size of the address structure, the kernel returns the size of the client's address (the number of bytes written)

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

38



## Example: accept

```

sin_size = sizeof(struct sockaddr_in);
if ((new_fd = accept(sockfd, (struct sockaddr*)
                    &their_addr, &sin_size)) == -1) {
    perror ("accept");
    continue;
}

```

- How does the server know which client it is?
  - `their_addr.sin_addr` contains the client's IP address
  - `their_addr.port` contains the client's port number

```

printf ("server: got connection from %s\n",
        inet_ntoa (their_addr.sin_addr));

```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

39



## Functions: accept

- After `accept` returns a new socket descriptor, I/O can be done using `read()` and `write()`
- Why does `accept` need to return a new descriptor?

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

40



## [ Sending and Receiving Data ]

```
int write (int sockfd, char* buf, size_t
nbytes);
```

- Write data to a stream (TCP) or “connected” datagram (UDP) socket.
  - Returns number of bytes written or -1.

```
int read (int sockfd, char* buf, size_t
nbytes);
```

- Read data from a stream (TCP) or “connected” datagram (UDP) socket.
  - Returns number of bytes read or -1.

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

41



## [ Functions: write ]

```
int write (int sockfd, char* buf, size_t nbytes);
```

- Write data to a stream (TCP) or “connected” datagram (UDP) socket
  - Returns number of bytes written or -1 and sets **errno** on failure
  - **sockfd**: socket file descriptor (returned from **socket**)
  - **buf**: data buffer
  - **nbytes**: number of bytes to try to write
  - Example:

```
if ((w = write(fd, buf, sizeof(buf))) < 0) {
    perror("write");
    exit(1);
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

42



## [ Functions: write ]

```
int write (int sockfd, char* buf, size_t nbytes);
```

### ■ Notes

- **write** blocks waiting for data from the client
- **write** may not write all bytes asked for
  - Does not guarantee that **sizeof(buf)** is written
  - This is not an error
  - Simply continue writing to the device
- Some reasons for failure or partial writes
  - Process received interrupt or signal
  - Kernel resources unavailable (e.g., buffers)

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

43



## [ Example: written ]

```
/* Write "n" bytes to a descriptor */
ssize_t written(int fd, const void *ptr, size_t n) {
    size_t nleft;
    size_t nwritten;
    nleft = n;
    while (nleft > 0) {
        if ((nwritten = write(fd, ptr, nleft)) < 0) {
            if (nleft == n)
                return(-1); /* error, return -1 */
            else
                break; /* error, return amount written so far */
        }
        else if (nwritten == 0)
            break;
        nleft -= nwritten;
        ptr += nwritten;
    }
    return(n - nleft); /* return >= 0 */
}
```

**write** returned a potential error

0 bytes were written

Update number of bytes left to write and pointer into buffer

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

44



## Functions: read

```
int read (int sockfd, char* buf, size_t nbytes);
```

- Read data from a stream (TCP) or “connected” datagram (UDP) socket
  - Returns number of bytes read or -1, sets `errno` on failure
  - Returns 0 if socket closed
  - `sockfd`: socket file descriptor (returned from `socket`)
  - `buf`: data buffer
  - `nbytes`: number of bytes to try to read
  - Example
 

```
if((r = read(newfd, buf, sizeof(buf))) < 0) {
    perror("read"); exit(1);
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

45



## Functions: read

```
int read (int sockfd, char* buf, size_t nbytes);
```

- Notes
  - `read` blocks waiting for data from the client
  - `read` may return less than asked for
    - Does not guarantee that `sizeof(buf)` is read
    - This is not an error
    - Simply continue reading from the device

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

46



## Example: readn

```
/* Read "n" bytes from a descriptor */
ssize_t readn(int fd, void *ptr, size_t n) {
    size_t nleft;
    ssize_t nread;
    nleft = n;
    while (nleft > 0) {
        if ((nread = read(fd, ptr, nleft)) < 0) {
            if (nleft == n)
                return(-1); /* error, return -1 */
            else
                break; /* error, return amt read */
        }
        else
            if (nread == 0)
                break; /* EOF */
            nleft -= nread;
            ptr += nread;
        }
    return(n - nleft); /* return >= 0 */
}
```

`read` returned  
a potential error

0 bytes were  
read

Update number  
of bytes left to  
read and  
pointer into  
buffer

47



## Sending and Receiving Data

```
int send(int sockfd, const void * buf,
         size_t nbytes, int flags);
```

- Write data to a stream (TCP) or “connected” datagram (UDP) socket.
  - Returns number of bytes written or -1.

```
int recv(int sockfd, void *buf, size_t
         nbytes, int flags);
```

- Read data from a stream (TCP) or “connected” datagram (UDP) socket.
  - Returns number of bytes read or -1.

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

48



## Functions: send

```
int send(int sockfd, const void * buf, size_t
nbytes, int flags);
```

- Send data on a stream (TCP) or “connected” datagram (UDP) socket
  - Returns number of bytes written or -1 and sets `errno` on failure
  - `sockfd`: socket file descriptor (returned from `socket`)
  - `buf`: data buffer
  - `nbytes`: number of bytes to try to write
  - `flags`: control flags
    - `MSG_PEEK`: get data from the beginning of the receive queue without removing that data from the queue

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

49



## Functions: send

```
int send(int sockfd, const void * buf, size_t
nbytes, int flags);
```

- Example

```
len = strlen(msg);
bytes_sent = send(sockfd, msg, len, 0);
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

50



## Functions: recv

```
int recv(int sockfd, void *buf, size_t nbytes,
int flags);
```

- Read data from a stream (TCP) or “connected” datagram (UDP) socket
  - Returns number of bytes read or -1, sets `errno` on failure
  - Returns 0 if socket closed
  - `sockfd`: socket file descriptor (returned from `socket`)
  - `buf`: data buffer
  - `nbytes`: number of bytes to try to read
  - `flags`: see man page for details; typically use 0

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

51



## Functions: recv

```
int read (int sockfd, char* buf, size_t nbytes);
```

- Notes
  - `read` blocks waiting for data from the client but does not guarantee that `sizeof(buf)` is read
  - Example

```
if((x = read(newfd, buf, sizeof(buf))) < 0) {
    perror("read"); exit(1);
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

52



## Sending and Receiving Data

- Datagram sockets aren't connected to a remote host
  - What piece of information do we need to give before we send a packet?
  - The destination/source address!

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

53



## Sending and Receiving Data

```
int sendto (int sockfd, char* buf,
            size_t nbytes, int flags, struct
            sockaddr* destaddr, int addrlen);
```

- Send a datagram to another UDP socket.
  - Returns number of bytes written or -1.

```
int recvfrom (int sockfd, char* buf,
              size_t nbytes, int flags, struct
              sockaddr* srcaddr, int* addrlen);
```

- Read a datagram from a UDP socket.
  - Returns number of bytes read or -1.

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

54



## Functions: sendto

```
int sendto (int sockfd, char* buf, size_t nbytes,
            int flags, struct sockaddr* destaddr, int
            addrlen);
```

- Send a datagram to another UDP socket
  - Returns number of bytes written or -1 and sets `errno` on failure
  - `sockfd`: socket file descriptor (returned from `socket`)
  - `buf`: data buffer
  - `nbytes`: number of bytes to try to read
  - `flags`: see man page for details; typically use 0
  - `destaddr`: IP address and port number of destination socket
  - `addrlen`: length of address structure
    - = `sizeof (struct sockaddr_in)`

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

55



## Functions: sendto

```
int sendto (int sockfd, char* buf, size_t nbytes,
            int flags, struct sockaddr* destaddr, int
            addrlen);
```

- Example
 

```
n = sendto(sock, buf, sizeof(buf), 0, (struct
sockaddr *) &from, fromlen);
if (n < 0)
    perror("sendto");
exit(1);
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

56



## Functions: recvfrom

```
int recvfrom (int sockfd, char* buf, size_t
nbytes, int flags, struct sockaddr* srcaddr,
int* addrlen);
```

- Read a datagram from a UDP socket.
  - Returns number of bytes read (0 is valid) or -1 and sets `errno` on failure
  - `sockfd`: socket file descriptor (returned from `socket`)
  - `buf`: data buffer
  - `nbytes`: number of bytes to try to read
  - `flags`: see man page for details; typically use 0
  - `srcaddr`: IP address and port number of sending socket (returned from call)
  - `addrlen`: length of address structure = pointer to `int` set to `sizeof (struct sockaddr_in)`

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

57



## Functions: recvfrom

```
int recvfrom (int sockfd, char* buf, size_t
nbytes, int flags, struct sockaddr* srcaddr,
int* addrlen);
```

- Example
 

```
n = recvfrom(sock, buf, 1024, 0, (struct sockaddr
*) &from, &fromlen);
if (n < 0) {
    perror("recvfrom");
    exit(1);
}
```

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

58



## Tearing Down a Connection

```
int close (int sockfd);
```

- Close a socket.
  - Returns 0 on success, -1 and sets `errno` on failure.

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

- Force termination of communication across a socket in one or both directions.
  - Returns 0 on success, -1 and sets `errno` on failure.

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

59



## Functions: close

```
int close (int sockfd);
```

- Close a socket
  - Returns 0 on success, -1 and sets `errno` on failure
  - `sockfd`: socket file descriptor (returned from `socket`)
- Closes communication on socket in both directions
  - All data sent before `close` are delivered to other side (although this aspect can be overridden)
- After `close`, `sockfd` is not valid for reading or writing

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

60



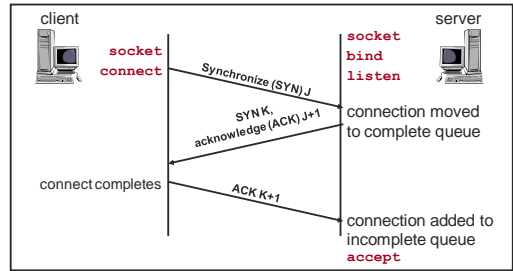
## Functions: shutdown

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

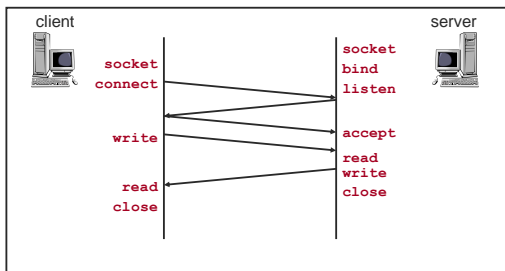
- Force termination of communication across a socket in one or both directions
  - Returns 0 on success, -1 and sets **errno** on failure
  - **sockfd**: socket file descriptor (returned from **socket**)
  - **howto**:
    - **SHUT\_RD** to stop reading
    - **SHUT\_WR** to stop writing
    - **SHUT\_RDWR** to stop both
- **shutdown** overrides the usual rules regarding duplicated sockets, in which TCP teardown does not occur until all copies have closed the socket



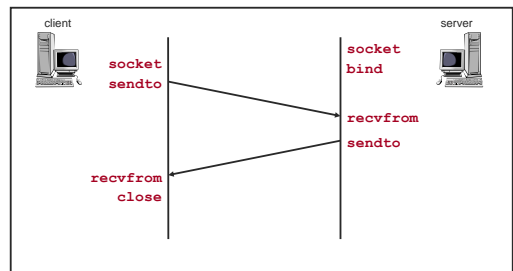
## TCP Connection Setup



## TCP Connection Example



## UDP Connection Example



## Examples

- Taken from Beej's Guide to Network Programming:  
<http://beej.us/guide/bgnet/>
- Structure
  - One server on a machine
  - Server handles multiple clients using `fork()`
- Basic routine
  - Server waits for a connection
  - `accept()` s the connection
  - `fork()` s a child process to handle the client

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

65



## Example: TCP

- Client-Server example using TCP
  - For each client
    - server forks new process to handle connection
    - sends "Hello, world"

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

66



## server

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#define PORT 3490 /* well-known port */
#define BACKLOG 10 /* how many pending
                   connections queue
                   will hold */
```

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

67



## server

```
main()
{
    int sockfd, new_fd; /* listen on sock_fd, new
                       connection on new_fd */
    struct sockaddr_in my_addr; /* my address */
    struct sockaddr_in their_addr; /* connector addr */
    int sin_size;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }
}
```

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

68



## [ server ]

```

my_addr.sin_family = AF_INET; /* host byte order */
my_addr.sin_port = htons(MYPORT); /* short, network
byte order */
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
/* automatically fill with my IP */
bzero(&my_addr.sin_zero, 8); /* zero struct */

if (bind(sockfd, (struct sockaddr *)&my_addr,
sizeof(struct sockaddr)) == -1) {
    perror("bind");
    exit(1);
}

```

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaker, Kravets, Gupta

69



## [ server ]

```

if (listen(sockfd, BACKLOG) == -1) {
    perror("listen");
    exit(1);
}

while(1) { /* main accept() loop */
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr*)
&their_addr, &sin_size)) == -1) {
        perror("accept");
        continue;
    }
    printf("server: got connection from %s\n",
inet_ntoa(their_addr.sin_addr));
}

```

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaker, Kravets, Gupta

70



## [ server ]

```

if (!fork()) { /* this is the child process */
    if (send(new_fd, "Hello, world!\n", 14, 0)
== -1)
        perror("send");
    close(new_fd);
    exit(0);
}
close(new_fd); /* parent doesn't need this */
/* clean up all child processes */
while(waitpid(-1, NULL, WNOHANG) > 0);
}

```

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaker, Kravets, Gupta

71



## [ client ]

```

#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#define PORT 3490 /* well-known port */
#define MAXDATASIZE 100 /* max number of bytes we
can get at once */

```

CS 241

Copyright ©: Nahrstedt, Angrave, Abdelzaker, Kravets, Gupta

72



## [ client ]

```
int main (int argc, char* argv[]){
    int sockfd, numbytes;
    char buf[MAXDATASIZE + 1];
    struct hostent* he;
    struct sockaddr_in their_addr;
    /* connector's address information */
    if (argc != 2) {
        fprintf (stderr, "usage: client hostname\n");
        exit (1);
    }
    if ((he = gethostbyname (argv[1])) == NULL) {
        /* get the host info */
        perror ("gethostbyname");
        exit (1);
    }
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

73



## [ client ]

```
if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) == -1) {
    perror ("socket");
    exit (1);
}

their_addr.sin_family = AF_INET; /* interper'd by host */
their_addr.sin_port = htons (PORT);
their_addr.sin_addr = *((struct in_addr*)he->h_addr);
bzero (&(their_addr.sin_zero), 8);
/* zero rest of struct */
if (connect (sockfd, (struct sockaddr*)&their_addr,
            sizeof (struct sockaddr)) == -1) {
    perror ("connect");
    exit (1);
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

74



## [ client ]

```
if ((numbytes = recv (sockfd, buf, MAXDATASIZE, 0))
    == -1) {
    perror ("recv");
    exit (1);
}

buf[numbytes] = '\0';
printf ("Received: %s", buf);
close (sockfd);
return 0;
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

75



## [ Example: UDP ]

- Client-Server example using UDP
  - For each client
    - **listener** sits on a machine waiting for an incoming packet on port 4950
    - **talker** sends a packet to that port, on the specified machine, that contains whatever the user enters on the command line

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

76



## [ listener ]

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define MYPORT "4950" // the port users will be connecting to
#define MAXBUFLen 100
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

77



## [ listener ]

```
// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa) {
    if (sa->sa_family == AF_INET) {
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }
    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

78



## [ listener ]

```
int main(void) {
    int sockfd, rv, numbytes;
    struct addrinfo hints, *servinfo, *p;
    struct sockaddr_storage their_addr;
    char buf[MAXBUFLen], s[INET6_ADDRSTRLEN];
    size_t addr_len;
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_flags = AI_PASSIVE; // use my IP

    if ((rv = getaddrinfo(NULL, MYPORT, &hints, &servinfo)) == -1) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

79



## [ listener ]

```
// loop through all results and bind to the first we can
for (p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
        p->ai_protocol)) == -1) {
        perror("listener: socket");
        continue;
    }

    if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("listener: bind");
        continue;
    }
    break;
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

80



## [ listener ]

```

if (p == NULL) {
    fprintf(stderr, "listener: failed to bind socket\n");
    return 2;
}
freeaddrinfo(servinfo);
printf("listener: waiting to recvfrom...\n");

addr_len = sizeof their_addr;
if ((numbytes = recvfrom(sockfd, buf, MAXBUFLEN-1, 0,
    (struct sockaddr *)&their_addr, &addr_len)) == -1) {
    perror("recvfrom");
    exit(1);
}

```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

81



## [ listener ]

```

printf("listener: got packet from %s\n",
    inet_ntop(their_addr.ss_family,
        get_in_addr((struct sockaddr *)&their_addr),
        s, sizeof s));
printf("listener: packet is %d bytes long\n", numbytes);
buf[numbytes] = '\0';
printf("listener: packet contains \"%s\"\n", buf);

close(sockfd);

return 0;
}

```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

82



## [ talker ]

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define SERVERPORT "4950" // port users will connect to

```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

83



## [ talker ]

```

int main(int argc, char *argv[]) {
    int sockfd, rv, numbytes;
    struct addrinfo hints, *servinfo, *p;

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;

    if ((rv = getaddrinfo(argv[1], SERVERPORT, &hints,
        &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }
}

```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

84



## [ talker ]

```
// loop through all the results and make a socket
for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
                       p->ai_protocol)) == -1) {
        perror("talker: socket");
        continue;
    }
    break;
}

if (p == NULL) {
    fprintf(stderr, "talker: failed to bind socket\n");
    return 2;
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

85



## [ talker ]

```
if ((numbytes = sendto(sockfd, argv[2], strlen(argv[2]), 0,
                      p->ai_addr, p->ai_addrlen)) == -1) {
    perror("talker: sendto");
    exit(1);
}

freeaddrinfo(servinfo);

printf("talker: sent %d bytes to %s\n", numbytes, argv[1]);
close(sockfd);

return 0;
}
```

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

86



## [ Connected Datagram Sockets ]

- **talker** calls `connect()` and specifies **listener**'s address
  - **talker** may only send to and receive from the address specified by `connect()`
  - Don't have to use `sendto()` and `recvfrom()`
  - Simply use `send()` and `recv()`

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

87



## [ Framing ]

- Goal
  - Framing messages on a byte stream
- Given a TCP message stream, how can we pass logical messages?
  - Note:
    - read may return partial or multiple messages
  - Questions:
    - How can we determine the end of a message?
- Hint
  - string storage in C and Pascal
  - format strings with `printf`

CS 241

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

88



## [ Framing Problem ]

- Approach
  - Think about the problem for a minute or two
  - Introduce yourself to 2-3 people near you (form groups of 3-4)
  - Discuss the problem and agree on a solution