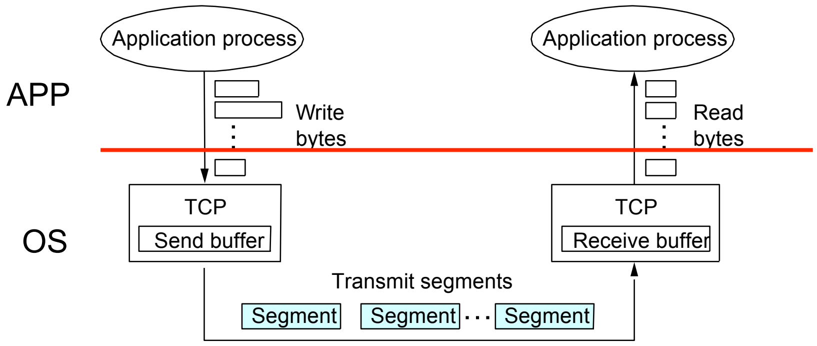
CSE/EE 461
Getting Started with Networking

Basic Concepts

- A PROCESS is an executing program somewhere.
  - Eg, "./a.out"
- A MESSAGE contains information sent by one PROCESS to ANOTHER
  - Eg, "please get www.cs.washington.edu/index.html"
- A COMMUNICATIONS ENDPOINT is the name of some source or destination of a message
  - Host: www.cs.washington.edu, Port: 80
- A PROTOCOL is the SET-OF-RULES governing the transmission of MESSAGES
  - Protocol: TCP/IP
- A MESSAGING-API is the programming interface used by PROCESSES to send/receive MESSAGES
- Typically,
  - OS implements the PARTS IN RED
  - Application provides/consumes the MESSAGES.

Example: TCP Delivery



The API

Unix SOCKETS

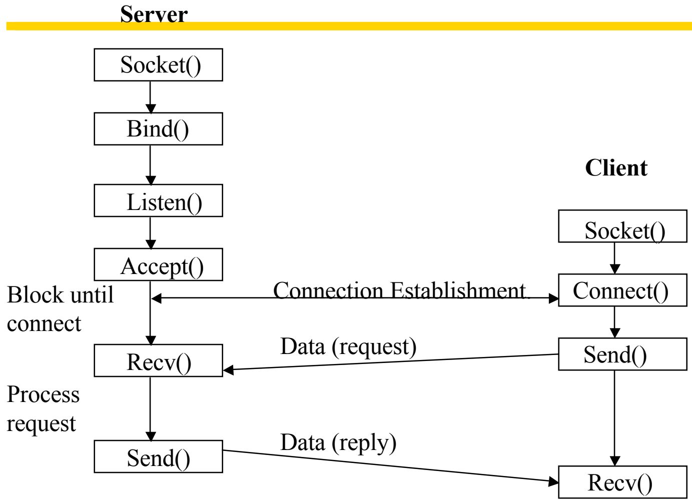
Berkeley Sockets

- Networking protocols are implemented as part of the OS
  - The networking API exported by most OS's is the *socket interface*
  - Originally provided by BSD 4.1c ~1982.
- The principal abstraction is a socket
  - Point at which an application attaches to the network
  - Defines operations for creating connections, attaching to network, sending/receiving data, closing.
- Two primary protocols used
  - Reliable Connections (TCP)
    - Like a telephone
  - Unreliable Datagrams (UDP)
    - Like postcards

The Client/Server Paradigm

- A **Server** is a long lived process that LISTENS in at some well-known COMMUNICATIONS-ENDPOINT
  - Awaiting a new request
  - Satisfy the new request
  - Send a response
  - Do it again
- A **Client** is a short lived process that makes requests on Servers.
  - Format a message containing the request
  - Send the message to the Server
  - Await the response
  - Process the response
- Classic Example:
  - WWW
    - Web Servers (Apache, IIS, etc)
    - Web Clients (IE, Safari, Firefox)
  - Clients CONNECT to SERVERS by means of an OS API

Client/Server Connection API



Server

Socket()

Bind()

Listen()

Accept()

Block until connect

Recv()

Process request

Send()

Client

Socket()

Connect()

Send()

Recv()

Connection Establishment

Data (request)

Data (reply)

7

Structure

- Server
  - Make a “rendezvous socket” on which to accept requests
    - **socket**
  - Associate an “address” with that socket so that others can submit requests
    - **bind**
  - Ready the socket for requests
    - **listen**
  - Await a request on the rendezvous socket
    - **accept**
      - Creates a SECOND socket
  - Read the request (from the SECOND socket)
    - **read**
  - Do the request
    - **XX**
  - Send the response
    - **write**
- Client
  - Make a local “socket” on which to send requests to the rendezvous address
    - **socket**
  - Connect to the rendezvous address by means of the local socket
    - **connect**
  - Send the request
    - **write**
  - Await the response
    - **read**

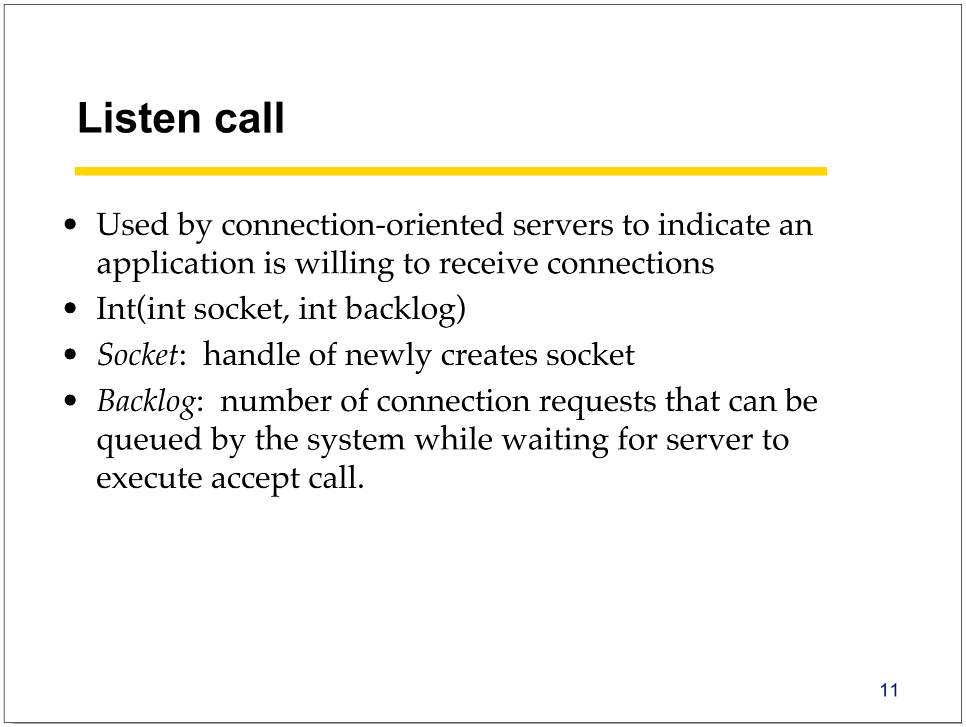
8

Socket call

- Means by which an application attached to the network
  - #include <sys/socket.h>…
- int socket(int family, int type, int protocol)
- *Family*: address family (protocol family)
  - AF_UNIX, AF_INET, AF_NS, AF_IMPLINK
- *Type*: semantics of communication
  - SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
  - Not all combinations of family and type are valid
- *Protocol*: Usually set to 0 but can be set to specific value.
  - Family and type usually imply the protocol
- Return value is a *handle* for new socket

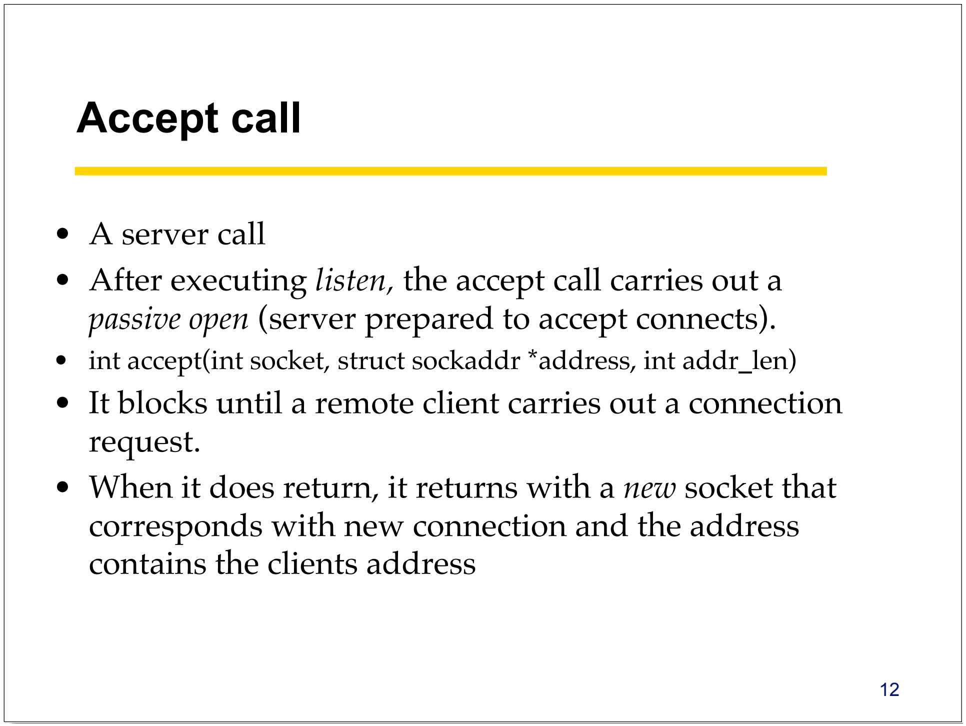
Bind call

- Typically a server call
- Binds a newly created socket to the specified address
  - *int bind(int socket, struct sockaddr *address, int addr_len)*
- *Socket*: newly created socket handle
- *Address*: data structure of address of *local* system
  - IP address (host identifier) and port number (endpoint on identified host)
- SOCKET and PORT are not the same concept
  - Socket: "widget" that a process uses to manipulate its endpoint
  - Port: hostwide name of a communication's endpoint
  - Address: hostname.port pair
  - For comparison:
    - Socket == file descriptor
    - port == file name,
    - address == network file name



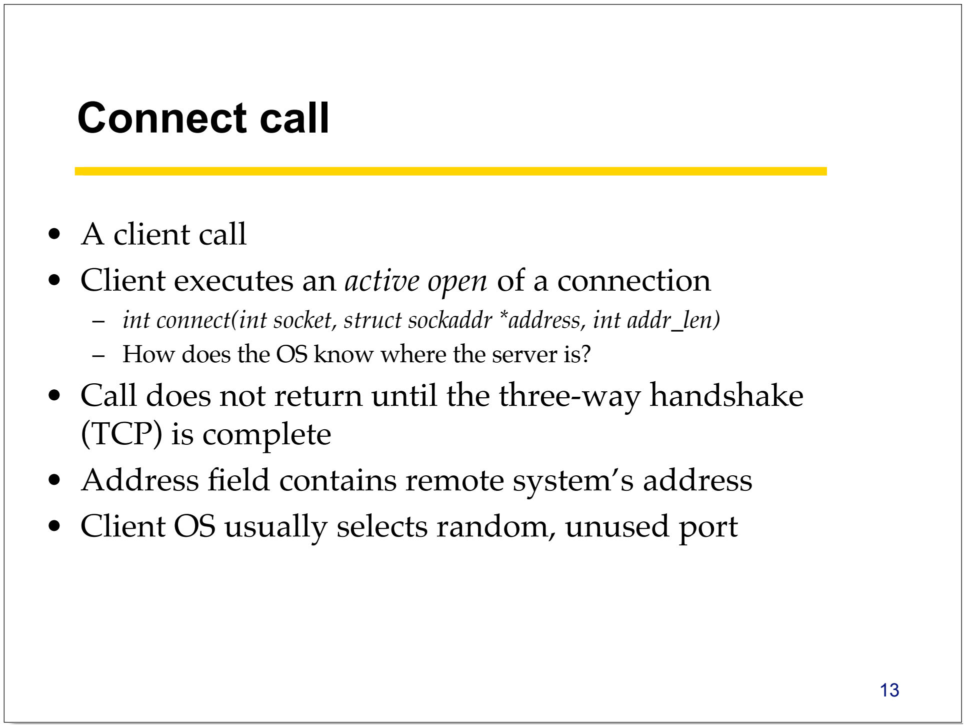
Listen call

- Used by connection-oriented servers to indicate an application is willing to receive connections
- Int(int socket, int backlog)
- *Socket*: handle of newly creates socket
- *Backlog*: number of connection requests that can be queued by the system while waiting for server to execute accept call.



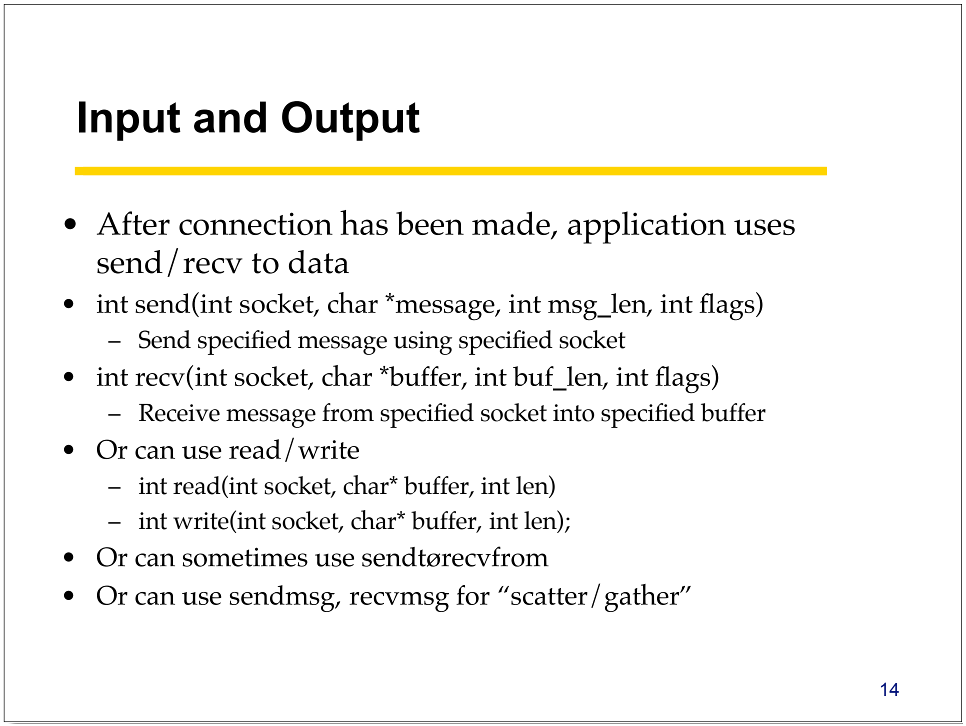
Accept call

- A server call
- After executing *listen*, the accept call carries out a *passive open* (server prepared to accept connects).
- int accept(int socket, struct sockaddr *address, int addr_len)
- It blocks until a remote client carries out a connection request.
- When it does return, it returns with a *new* socket that corresponds with new connection and the address contains the clients address



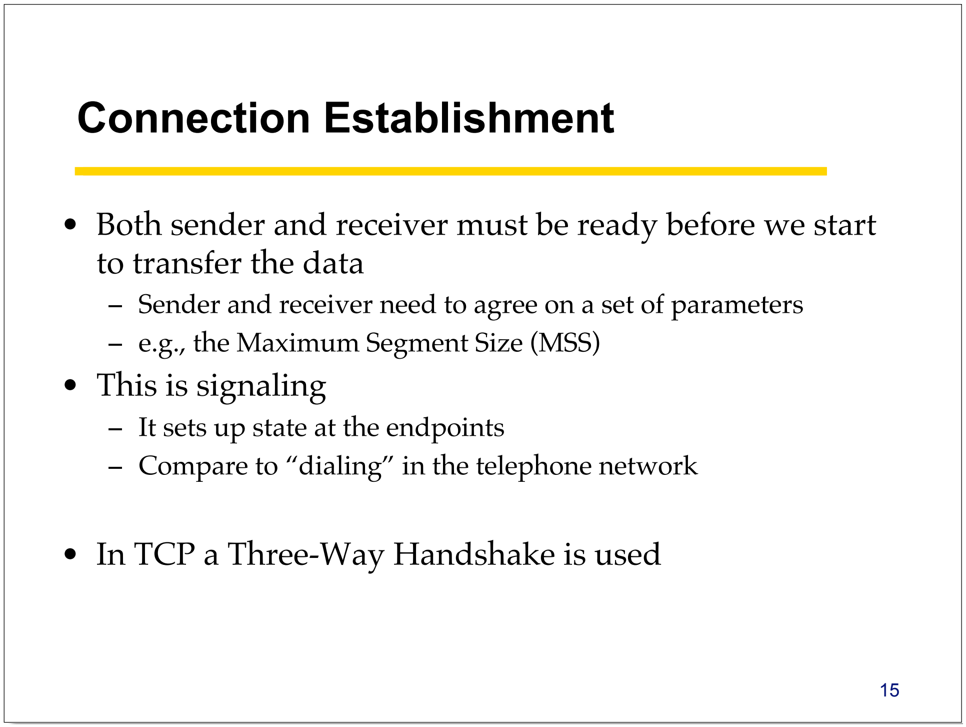
Connect call

- A client call
- Client executes an *active open* of a connection
  - *int connect(int socket, struct sockaddr *address, int addr_len)*
  - How does the OS know where the server is?
- Call does not return until the three-way handshake (TCP) is complete
- Address field contains remote system's address
- Client OS usually selects random, unused port



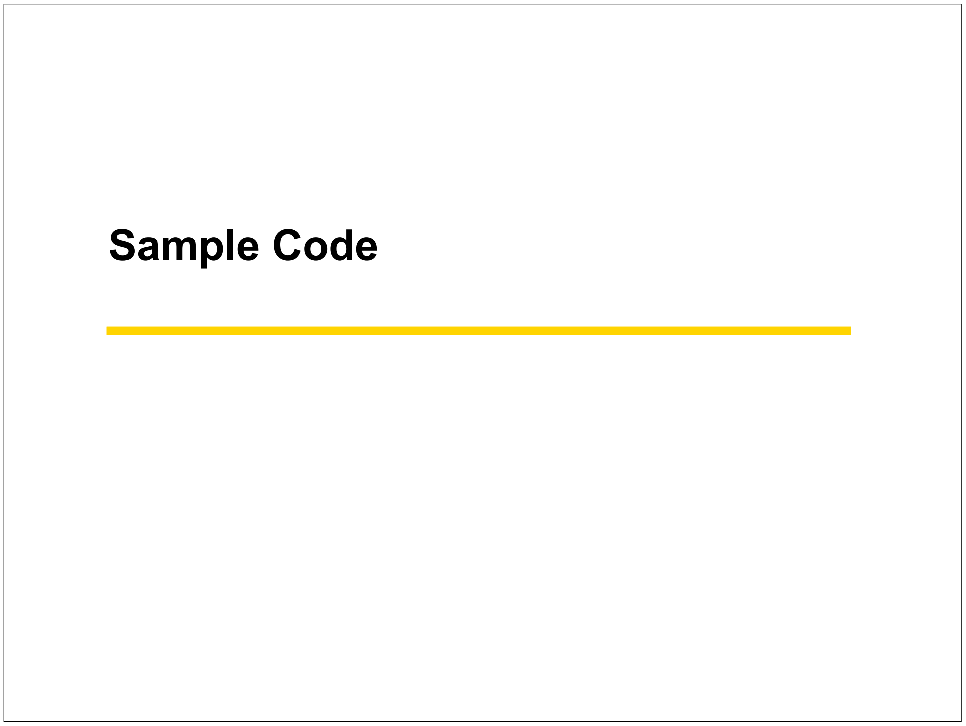
Input and Output

- After connection has been made, application uses send/recv to data
- int send(int socket, char *message, int msg_len, int flags)
  - Send specified message using specified socket
- int recv(int socket, char *buffer, int buf_len, int flags)
  - Receive message from specified socket into specified buffer
- Or can use read/write
  - int read(int socket, char* buffer, int len)
  - int write(int socket, char* buffer, int len);
- Or can sometimes use sendtørecvfrom
- Or can use sendmsg, recvmsg for "scatter/gather"



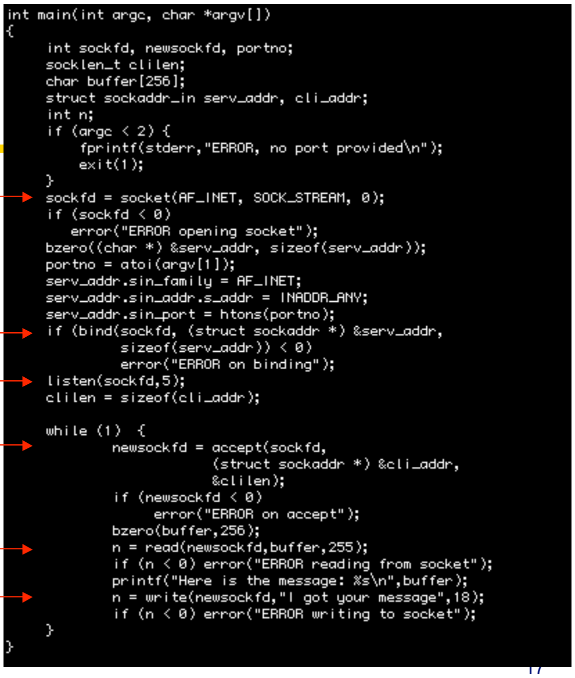
Connection Establishment

- Both sender and receiver must be ready before we start to transfer the data
  - Sender and receiver need to agree on a set of parameters
  - e.g., the Maximum Segment Size (MSS)
- This is signaling
  - It sets up state at the endpoints
  - Compare to "dialing" in the telephone network
- In TCP a Three-Way Handshake is used



Sample Code

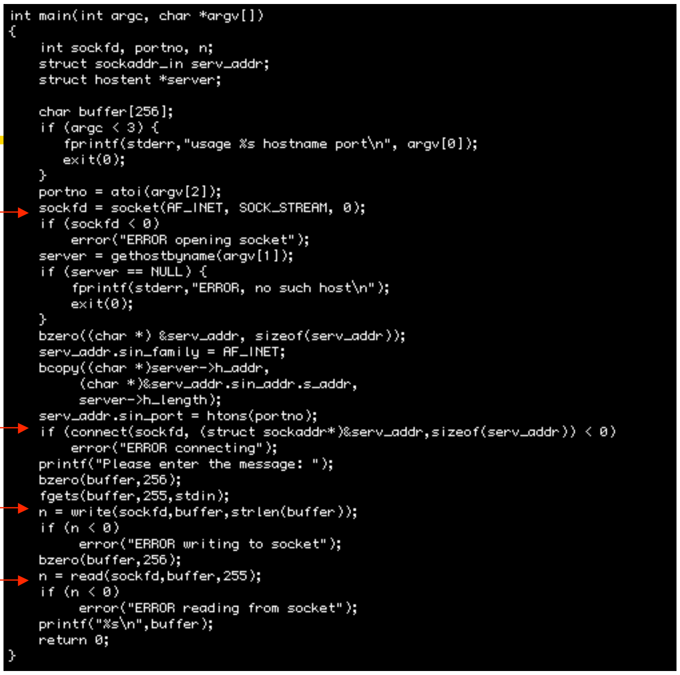
SERVER



```
int main(int argc, char *argv[])
{
     int sockfd, newsockfd, portno;
     socklen_t clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;
     int n;
     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1);
     }
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0)
        error("ERROR opening socket");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0)
              error("ERROR on binding");
     listen(sockfd,5);
     clilen = sizeof(cli_addr);

     while (1)  {
             newsockfd = accept(sockfd,
                         (struct sockaddr *) &cli_addr,
                         &clilen);
             if (newsockfd < 0)
                  error("ERROR on accept");
             bzero(buffer,256);
             n = read(newsockfd,buffer,255);
             if (n < 0) error("ERROR reading from socket");
             printf("Here is the message: %s\n",buffer);
             n = write(newsockfd,"I got your message",18);
             if (n < 0) error("ERROR writing to socket");
     }
}
```

CLIENT

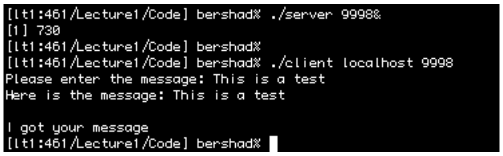


```
int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd, (struct sockaddr*)&serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
    if (n < 0)
         error("ERROR writing to socket");
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    if (n < 0)
         error("ERROR reading from socket");
    printf("%s\n",buffer);
    return 0;
}
```

Running it…

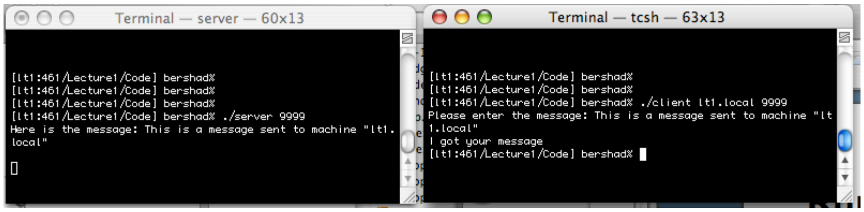
Run 1



```
[lt1:461/Lecture1/Code] bershad% ./server 9998&
[1] 730
[lt1:461/Lecture1/Code] bershad%
[lt1:461/Lecture1/Code] bershad% ./client localhost 9998
Please enter the message: This is a test
Here is the message: This is a test

I got your message
[lt1:461/Lecture1/Code] bershad%
```

Run 2



```
[lt1:461/Lecture1/Code] bershad%
[lt1:461/Lecture1/Code] bershad%
[lt1:461/Lecture1/Code] bershad%
[lt1:461/Lecture1/Code] bershad% ./server 9999
Here is the message: This is a message sent to machine "lt1.
local"

```

```
[lt1:461/Lecture1/Code] bershad%
[lt1:461/Lecture1/Code] bershad%
[lt1:461/Lecture1/Code] bershad% ./client lt1.local 9999
Please enter the message: This is a message sent to machine "lt
1.local"
I got your message
[lt1:461/Lecture1/Code] bershad%
```

How are these two runs different?

Observing Communication

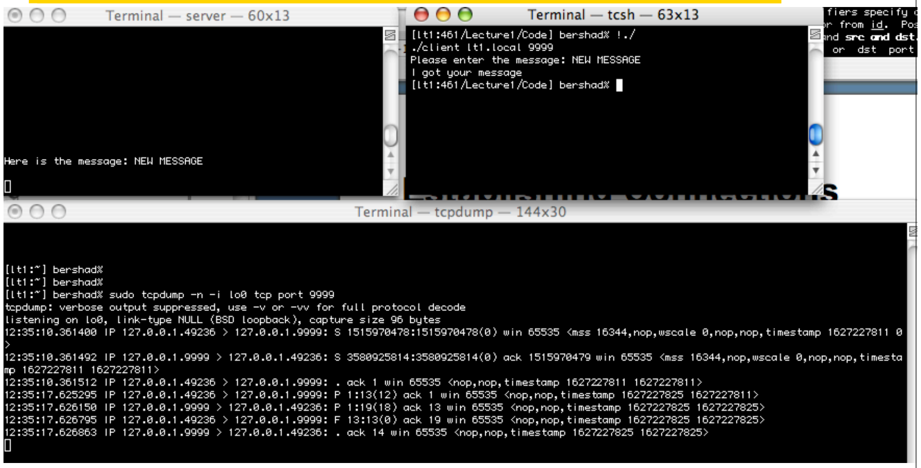
Messages are sent via NETWORK INTERFACES

eg, "lo0", "en0"

The tcpdump program allows us to observe network traffic.

"*man tcpdump*" for more information!

Establishing Connections



Terminal — server — 60x13

```
Here is the message: NEW MESSAGE
```

Terminal — tcsh — 63x13

```
[lt1:461/Lecture1/Code] bershad% !./
./client lt1.local 9999
Please enter the message: NEW MESSAGE
I got your message
[lt1:461/Lecture1/Code] bershad%
```

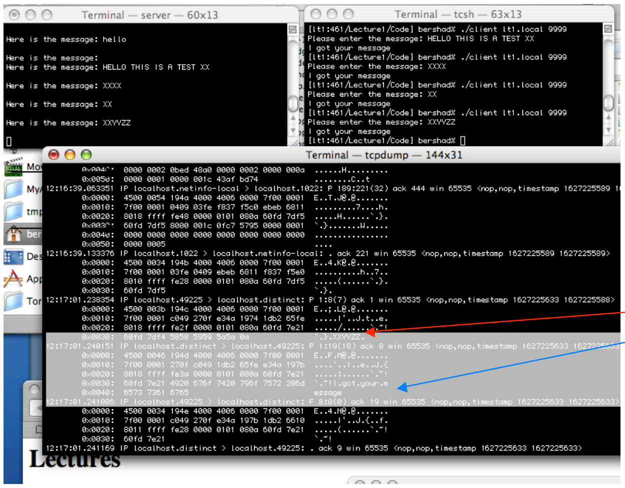
Terminal — tcpdump — 144x30

```
[lt1:~] bershad%
[lt1:~] bershad%
[lt1:~] bershad% sudo tcpdump -n -i lo0 tcp port 9999
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo0, link-type NULL (BSD loopback), capture size 96 bytes
12:35:10.361400 IP 127.0.0.1.49236 > 127.0.0.1.9999: S 1515970478:1515970478(0) win 65535 <mss 16344,nop,wscale 0,nop,nop,timestamp 1627227811 0
>
12:35:10.361492 IP 127.0.0.1.9999 > 127.0.0.1.49236: S 3580925814:3580925814(0) ack 1515970479 win 65535 <mss 16344,nop,wscale 0,nop,nop,timesta
mp 1627227811 1627227811>
12:35:10.361512 IP 127.0.0.1.49236 > 127.0.0.1.9999: . ack 1 win 65535 <nop,nop,timestamp 1627227811 1627227811>
12:35:17.625295 IP 127.0.0.1.49236 > 127.0.0.1.9999: P 1:13(12) ack 1 win 65535 <nop,nop,timestamp 1627227825 1627227811>
12:35:17.626150 IP 127.0.0.1.9999 > 127.0.0.1.49236: P 1:19(18) ack 13 win 65535 <nop,nop,timestamp 1627227825 1627227825>
12:35:17.626795 IP 127.0.0.1.49236 > 127.0.0.1.9999: F 13:13(0) ack 19 win 65535 <nop,nop,timestamp 1627227825 1627227825>
12:35:17.626863 IP 127.0.0.1.9999 > 127.0.0.1.49236: . ack 14 win 65535 <nop,nop,timestamp 1627227825 1627227825>
```

Each line is a network message
sent between the processes. What is this "conversation" saying?

21

Protocol vs. Message



Terminal — server — 60x13

```
Here is the message: hello

Here is the message:
Here is the message: HELLO THIS IS A TEST XX

Here is the message: XXXX

Here is the message: XX

Here is the message: XXYYZZ
```

Terminal — tcsh — 63x13

```
[lt1:461/Lecture1/Code] bershad% ./client lt1.local 9999
Please enter the message: HELLO THIS IS A TEST XX
I got your message
[lt1:461/Lecture1/Code] bershad% ./client lt1.local 9999
Please enter the message: XXXX
I got your message
[lt1:461/Lecture1/Code] bershad% ./client lt1.local 9999
Please enter the message: XX
I got your message
[lt1:461/Lecture1/Code] bershad% ./client lt1.local 9999
Please enter the message: XXYYZZ
I got your message
[lt1:461/Lecture1/Code] bershad%
```

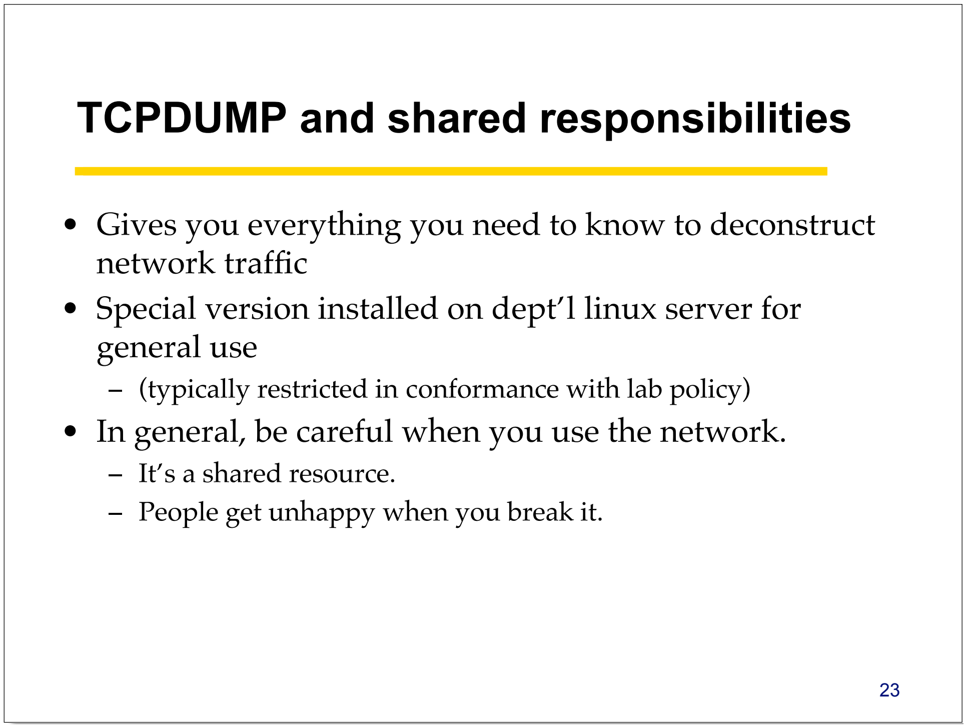
Terminal — tcpdump — 144x31

```
        0x0040:  0000 0002 0bed 48a0 0000 0002 0000 000a  ......H.........
        0x0050:  0000 0001 0000 001c 43af bd74            ........C..t
12:16:39.063351 IP localhost.netinfo-local > localhost.1022: P 189:221(32) ack 444 win 65535 <nop,nop,timestamp 1627225589 1
        0x0000:  4500 0054 194a 4000 4006 0000 7f00 0001  E..T.J@.@.......
        0x0010:  7f00 0001 0409 03fe f837 f5c0 ebeb 6811  .........7....h.
        0x0020:  8018 ffff fe48 0000 0101 080a 60fd 7df5  .....H......`.}.
        0x0030:  60fd 7df5 8000 001c 0fc7 5795 0000 0001  `.}.......W.....
        0x0040:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0050:  0000 0005                                ....
12:16:39.133376 IP localhost.1022 > localhost.netinfo-local: . ack 221 win 65535 <nop,nop,timestamp 1627225589 1627225589>
        0x0000:  4500 0034 194b 4000 4006 0000 7f00 0001  E..4.K@.@.......
        0x0010:  7f00 0001 03fe 0409 ebeb 6811 f837 f5e0  ..........h..7..
        0x0020:  8010 ffff fe28 0000 0101 080a 60fd 7df5  .....(......`.}.
        0x0030:  60fd 7df5                                `.}.
12:17:01.238354 IP localhost.49225 > localhost.distinct: P 1:8(7) ack 1 win 65535 <nop,nop,timestamp 1627225633 1627225588>
        0x0000:  4500 003b 194c 4000 4006 0000 7f00 0001  E..;.L@.@.......
        0x0010:  7f00 0001 c049 270f e34a 1974 1db2 65fe  .....I'..J.t..e.
        0x0020:  8018 ffff fe2f 0000 0101 080a 60fd 7e21  ...../......`.~!
        0x0030:  60fd 7df4 5858 5959 5a5a 0a              `.}.XXYYZZ.
12:17:01.240151 IP localhost.distinct > localhost.49225: P 1:19(18) ack 8 win 65535 <nop,nop,timestamp 1627225633 162722
        0x0000:  4500 0046 194d 4000 4006 0000 7f00 0001  E..F.M@.@.......
        0x0010:  7f00 0001 270f c049 1db2 65fe e34a 197b  ....'..I..e..J.{
        0x0020:  8018 ffff fe3a 0000 0101 080a 60fd 7e21  .....:......`.~!
        0x0030:  60fd 7e21 4920 676f 7420 796f 7572 206d  `.~!I.got.your.m
        0x0040:  6573 7361 6765                           essage
12:17:01.241086 IP localhost.49225 > localhost.distinct: F 8:8(0) ack 19 win 65535 <nop,nop,timestamp 1627225633 1627225633>
        0x0000:  4500 0034 194e 4000 4006 0000 7f00 0001  E..4.N@.@.......
        0x0010:  7f00 0001 c049 270f e34a 197b 1db2 6610  .....I'..J.{..f.
        0x0020:  8011 ffff fe28 0000 0101 080a 60fd 7e21  .....(......`.~!
        0x0030:  60fd 7e21                                `.~!
12:17:01.241169 IP localhost.distinct > localhost.49225: . ack 9 win 65535 <nop,nop,timestamp 1627225633 1627225633>
```

tcpdump -i lo0 -x

The data itself

22



TCPDUMP and shared responsibilities

- Gives you everything you need to know to deconstruct network traffic
- Special version installed on dept'l linux server for general use
  - (typically restricted in conformance with lab policy)
- In general, be careful when you use the network.
  - It's a shared resource.
  - People get unhappy when you break it.