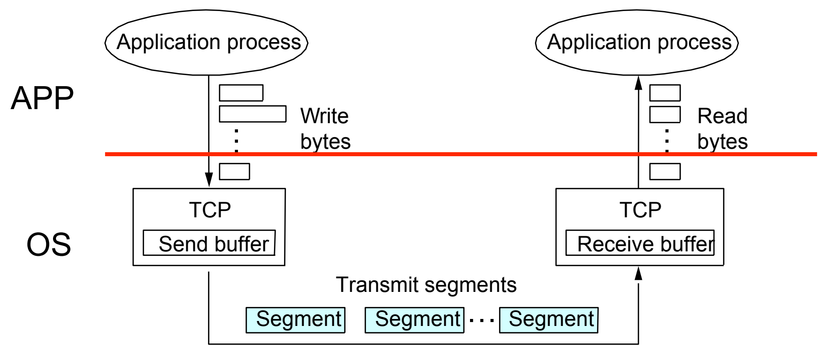
CSE/EE 461
Getting Started with Networking

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

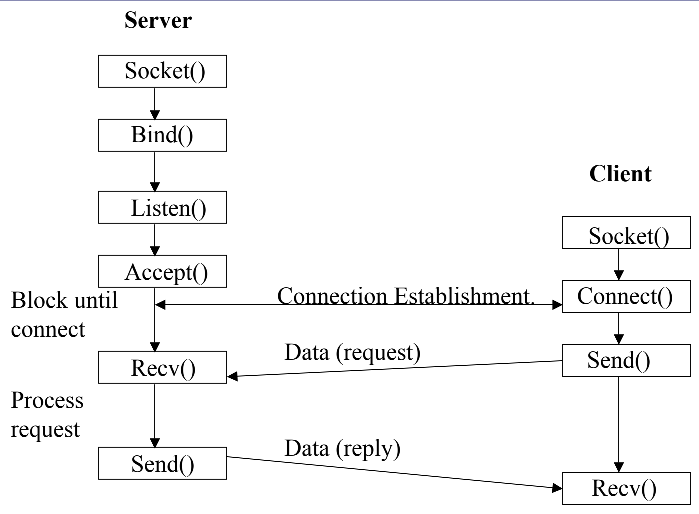
Berkeley Sockets

- 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 (possibly 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



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**

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

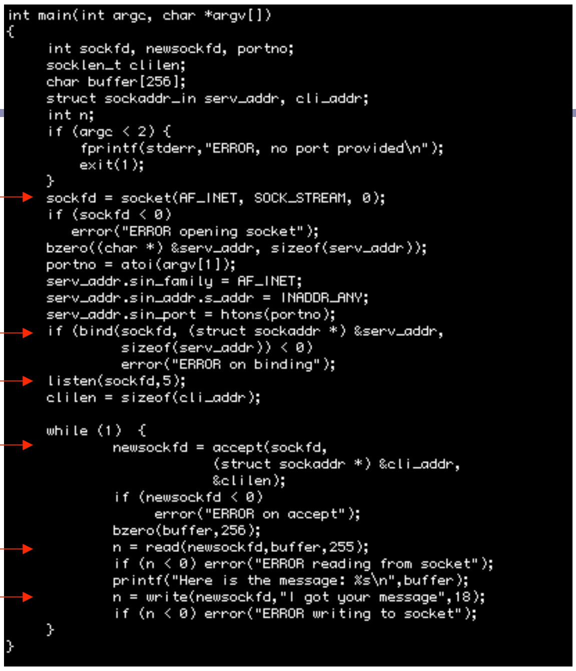
- After connection has been made, application uses send/recv to data
- int send(int socket, char *message, int msg_len, int flags)
  - Send specified message using specified socket
- int recv(int socket, char *buffer, int buf_len, int flags)
  - Receive message from specified socket into specified buffer
- 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 sendto/recvfrom
- 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

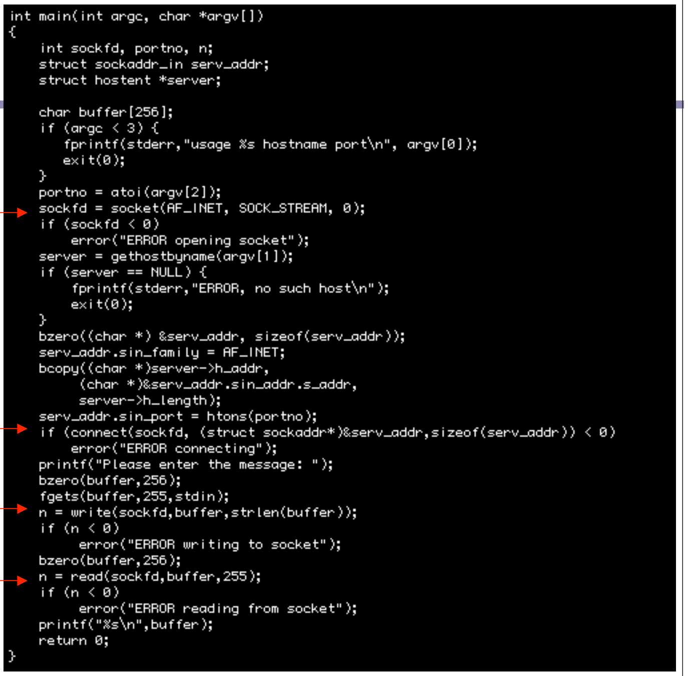
SERVER



```
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

```
dogmatix.dyn.cs.washington.edu arvind%   ./server 9998  &
[1]  736
dogmatix.dyn.cs.washington.edu arvind%   ./client localhost 9998
Please enter the message: This is a test
Here is the message: This is a test

I got your message
```

Run 2

```
dogmatix.dyn.cs.washington.edu arvind%   ./server 9999  &
[1]  736
dogmatix.dyn.cs.washington.edu arvind%   ./client dogmatix 9999
Please enter the message: This is a test
Here is the message: This is a test

I got your message
```

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!