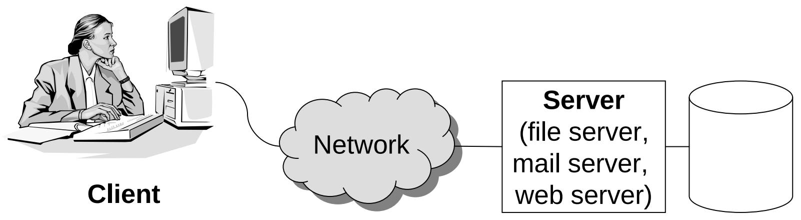
Client-Server Model



Client

Web browser
Emailer
Applications

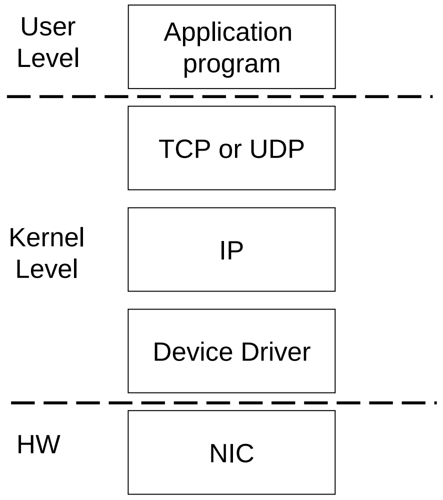
Active participant
Initiate contacts

Passive participant
Waiting to be contacted

Message Passing

- Mechanism to pass data between two processes
  - Sender sends a message from its memory
  - Receiver receives the message and places it into its memory
- Message passing is like using a telephone
  - Caller
  - Receiver

Network Subsystem



Socket API

Reliable data stream
or unreliable data grams

Routes through the internet

Transmit or receive on LAN

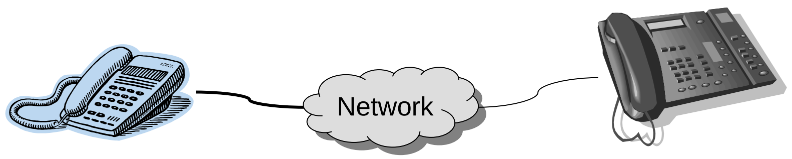
Network interface card

Names and Addresses

- Host name
  - like a post office name; e.g., **`www.cs.princeton.edu`**
- Host address
  - like a zip code; e.g., 128.112.92.191
- Port number
  - like a mailbox; e.g., 0-64k

Socket

- Socket abstraction
  - An end-point of network connection
  - Treat like a file descriptor
- Conceptually like a telephone
  - Connect to the end of a phone plug
  - You can speak to it and listen to it



Steps for Client and Server

Client

- Create a socket with the socket() system call
- Connect the socket to the address of the server using the connect() system call
- Send and receive data, using write() and read() system calls or send() and recv() system calls

Server

- Create a socket with the socket() system call
- Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the listen() system call
- Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.
- Send and receive data

client.c (part 1)

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

#define SERVER_PORT 5432
#define MAX_LINE 256

int
main(int argc, char *argv[])
{
    FILE *fp;
    struct hostent *hp;
    struct sockaddr_in sin;
    char *host;
    char buf[MAX_LINE];
    int s;
    int len;

    if (argc == 2) {
        host = argv[1];
    } else {
        fprintf(stderr, "usage: client host\n");
        exit(1);
    }

    /* translate host name into peer's IP address */
    hp = gethostbyname(host);
    if (hp == NULL) {
        fprintf(stderr, "client: unknown host: %s\n", host);
    }
```

client.c (part 2)

```
    /* build address data structure */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
    sin.sin_port = htons(SERVER_PORT);

    /* active open */
    if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("client: socket");
        exit(1);
    }
    if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        perror("client: connect");
        close(s);
        exit(1);
    }

    /* main loop: get and send lines of text */
    while (fgets(buf, sizeof(buf), stdin)) {
        buf[MAX_LINE-1] = 0;
        len = strlen(buf) + 1;
        send(s, buf, len, 0);
    }
}
```

server.c (part 1)

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

#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE    256

int
main()
{
    struct sockaddr_in sin;
    char buf[MAX_LINE];
    int len;
    int serverSocket, clientSocket;

    /* build address data structure */
    bzero((char *)&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(SERVER_PORT);

    /* setup passive open */
    if ((serverSocket = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        perror("server: socket");
        exit(1);
    }
```

server.c (part 2)

```
if ((bind(serverSocket, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("server: bind");
    exit(1);
}
listen(serverSocket, MAX_PENDING);

/* wait for connection, then receive and print text */
while (1) {
    if ((clientSocket = accept(serverSocket,
                         (struct sockaddr *)&sin, &len)) < 0) {
        perror("server: accept");
        exit(1);
    }
    while (len = recv(clientSocket, buf, sizeof(buf), 0)) {
        fputs(buf, stdout);
    }
    close(clientSocket);
}
```

Creating A Socket (Install A Phone)

```
if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("client: socket");
    exit(1);
}
```

- Creating a socket
  ```
  #include <sys/types.h>
  #include <sys/socket.h>
  int socket(int domain, int type, int protocol)
  ```
  - Domain: PF_INET (Internet), PF_UNIX (local)
  - Type: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
  - Protocol: 0 usually for IP (see /etc/protocols for details)
- Like installing a phone
  - Need to what services you want
    - Local or long distance
    - Voice or data
    - Which company do you want to use

Connecting To A Socket

client.c (part 2)

```
/* build address data structure */
bzero((char *)&sin, sizeof(sin));
sin.sin_family = AF_INET;
bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
sin.sin_port = htons(SERVER_PORT);

/* active open */
if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("client: socket");
    exit(1);
}
if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("client: connect");
    close(s);
    exit(1);
}
```

- Active open a socket (like dialing a phone number)
  ```
  int connect(int socket,
              struct sockaddr *addr,
              int addr_len)
  ```

Binding A Socket

```
if ((serverSocket = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("server: socket");
    exit(1);
}

if ((bind(serverSocket, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("server: bind");
    exit(1);
}
```

server.c

- Need to give the created socket an address to listen to (like getting a phone number)

```
int bind(int socket,
         struct sockaddr *addr,
         int addr_len)
```

  – Passive open on a server

Specifying Queued Connections

```
if ((serverSocket = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("server: socket");
    exit(1);
}

if ((bind(serverSocket, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("server: bind");
    exit(1);
}
listen(serverSocket, MAX_PENDING);
```

server.c

- Queue connection requests (like "call waiting")

```
int listen(int socket, int backlog)
```

  – Set up the maximum number of requests that will be queued before being denied (usually the max is 5)

Accepting A Socket

```
if ((bind(serverSocket, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("server: bind");
    exit(1);
}
listen(serverSocket, MAX_PENDING);

/* wait for connection, then receive and print text */
while (1) {
    if ((clientSocket = accept(serverSocket,
                        (struct sockaddr *)&sin, &len)) < 0) {
        perror("server: accept");
        exit(1);
    }
    while (len = recv(clientSocket, buf, sizeof(buf), 0)) {
        fputs(buf, stdout);
    }
    close(clientSocket);
}
```

server.c

- Wait for a call to a socket (picking up a phone when it rings)

```
int accept(int socket,
           struct sockaddr *addr,
           int addr_len)
```

  – Return a socket which is connected to the caller
  – Typically blocks until the client connects to the socket

Sending Data

```
if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("client: connect");
    close(s);
    exit(1);
}

/* main loop: get and send lines of text */
while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = 0;
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
}
```

client.c

- Sending a message

```
int send(int socket, char *buf, int blen, int flags)
```

Receiving Data

```
/* wait for connection, then receive and print text */
while (1) {
    if ((clientSocket = accept(serverSocket,
                              (struct sockaddr *)&sin, &len)) < 0) {
        perror("server: accept");
        exit(1);
    }
    while (len = recv(clientSocket, buf, sizeof(buf), 0)) {
        fputs(buf, stdout);
    }
    close(clientSocket);
}
```

server.c

- Receiving a message
  `int recv(int socket, char *buf, int blen, int flags)`

Close A Socket

```
/* wait for connection, then receive and print text */
while (1) {
    if ((clientSocket = accept(serverSocket,
                              (struct sockaddr *)&sin, &len)) < 0) {
        perror("server: accept");
        exit(1);
    }
    while (len = recv(clientSocket, buf, sizeof(buf), 0)) {
        fputs(buf, stdout);
    }
    close(clientSocket);
}
```

- Done with a socket (like hanging up the phone)

  `close(int socket)`

- Treat it just like a file descriptor

Summary

- Pipes
  - Process communication on the same machine
  - Connecting processes with stdin and stdout
- Messages
  - Process communication across machines
  - Socket is a common communication channels
  - They are built on top of basic communication mechanisms