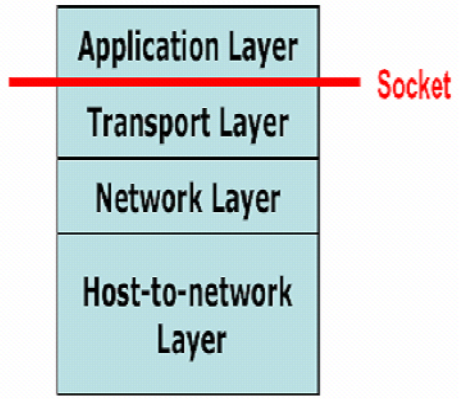
Socket Programming

COMPUTER NETWORKS 2015

What is Socket?



Networking's view:

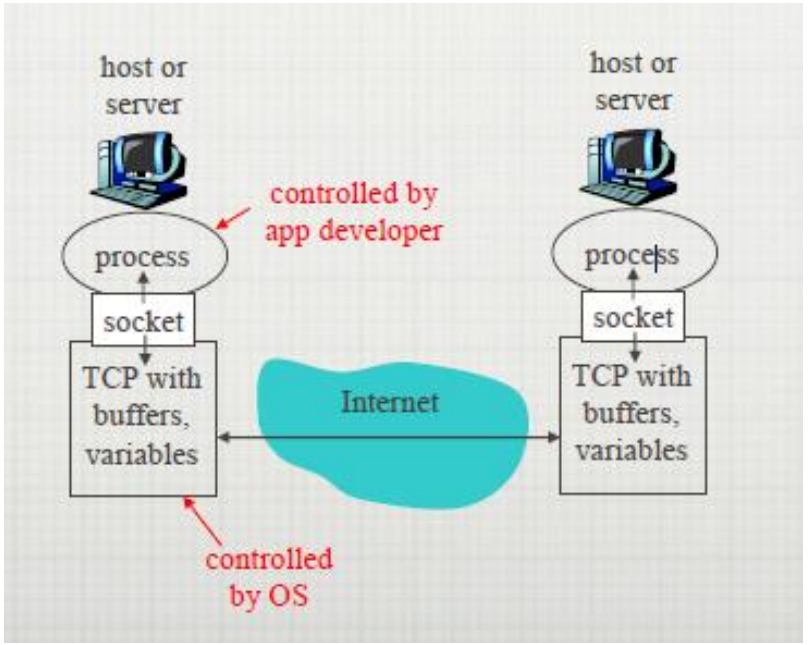
- Service access point of TCP/IP protocol stack.
- Provide communication between Application layer and Transport layer.

Programmer's view:

- A file descriptor.
- Allow applications to read/write data from/to the network.

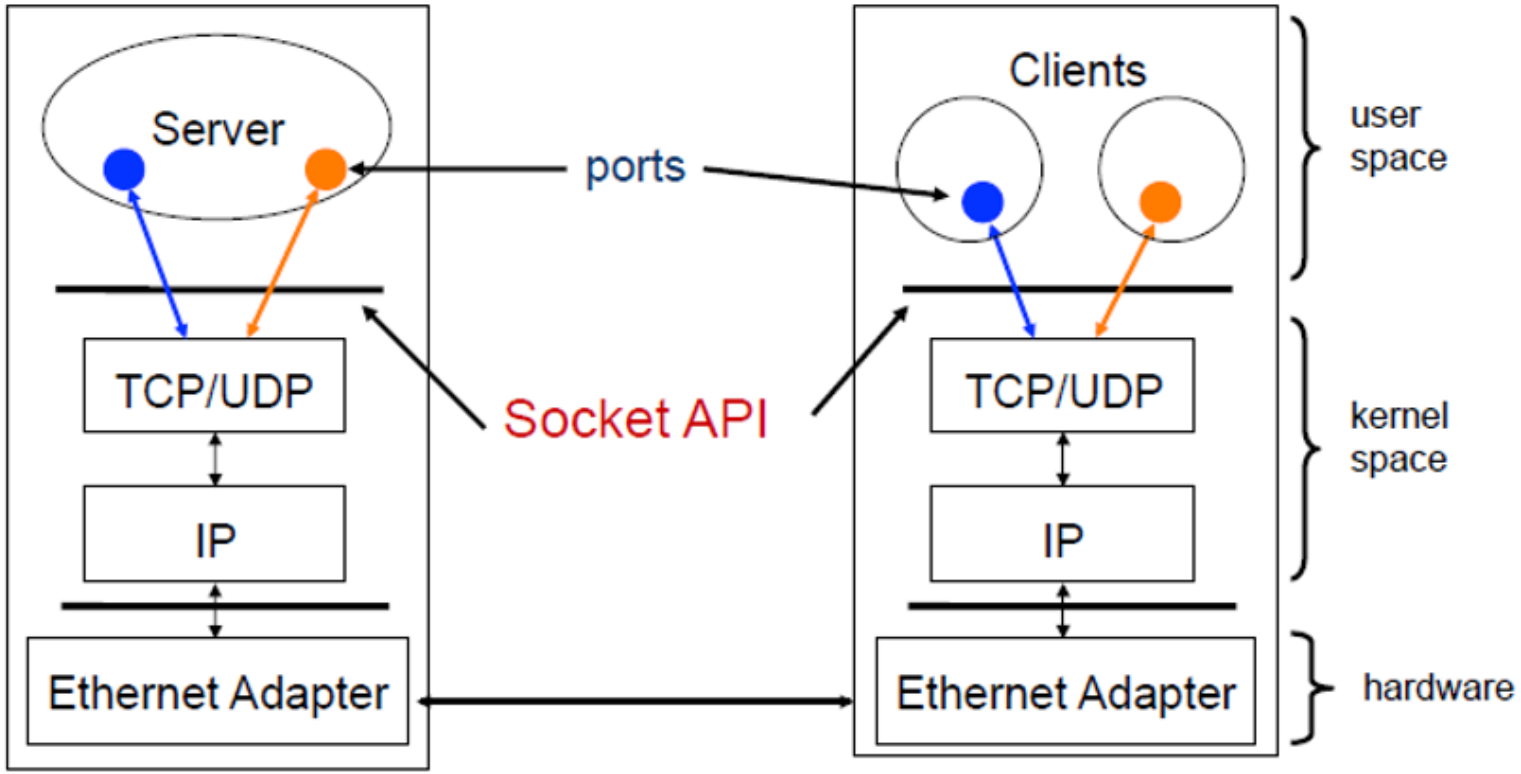
Once configured, the application can

- Send data to the socket.
- Receive data from the socket.

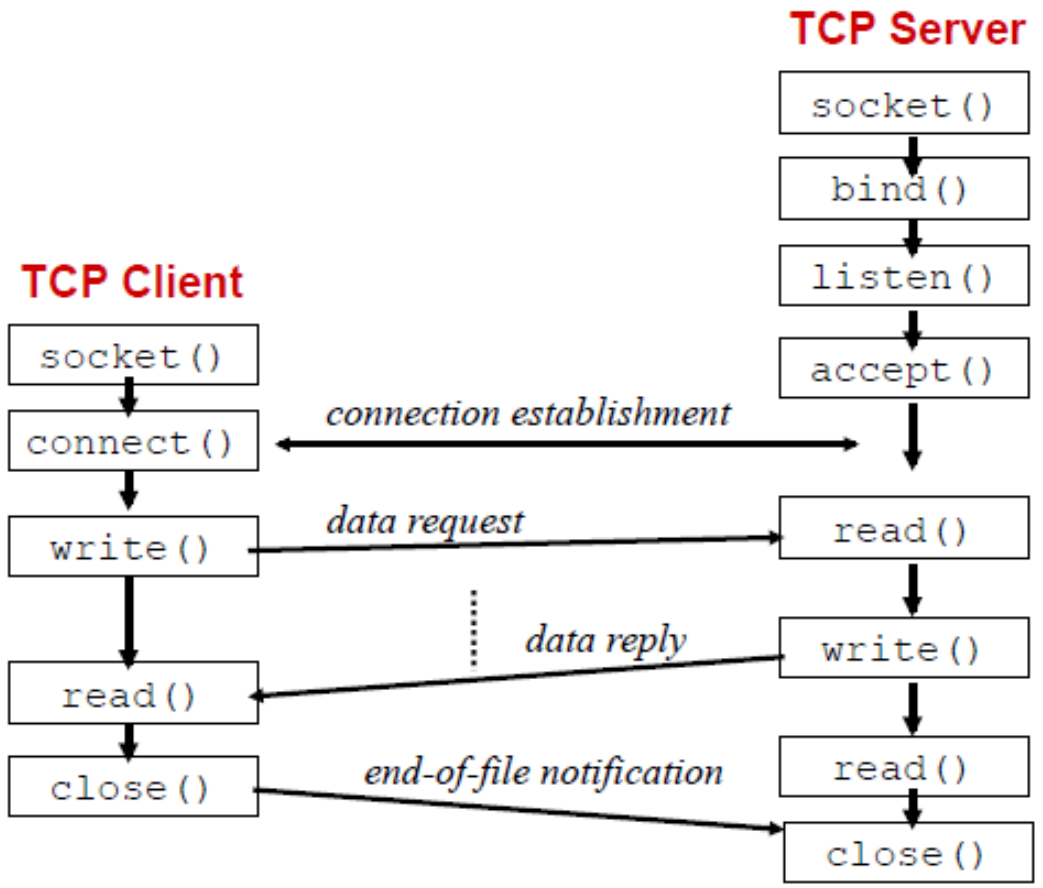


Socket API

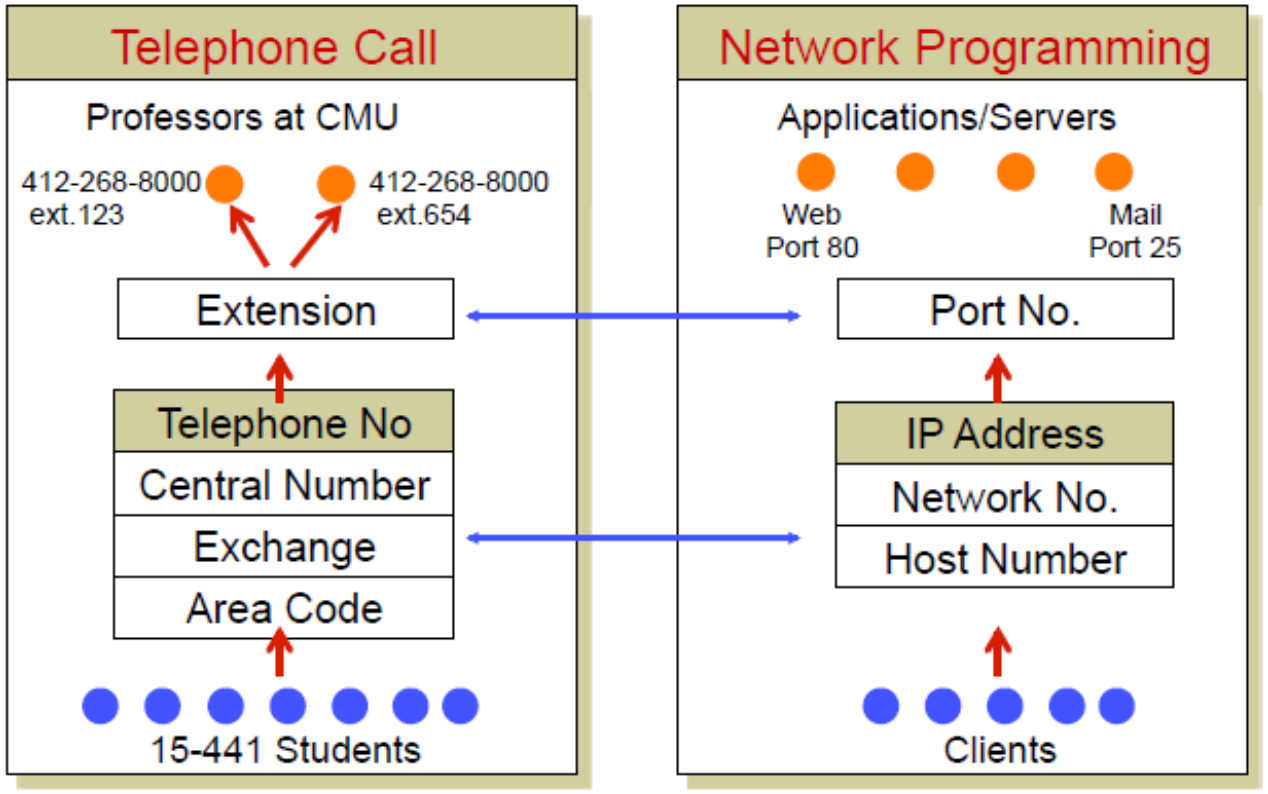
Server and Client exchange messages over the network through a common Socket API.



TCP Example


TCP Server
socket()
bind()
listen()
accept()
TCP Client
socket()
connect()
connection establishment
write()
data request
read()
write()
data reply
read()
close()
end-of-file notification
read()
close()

Network Addressing Analogy



Concept of Port Numbers

Port numbers are used to identify “entities” on a host.

Port numbers can be

- Well-known (port 0-1023)
- Dynamic or private (port 1024-65535)

Servers usually use well-known ports.

- Any client can identify the server/service.
- HTTP = 80, FTP = 21, Telnet = 23, ...

Clients usually use dynamic ports.

- Assigned by the kernel at runtime.

Names and Addresses

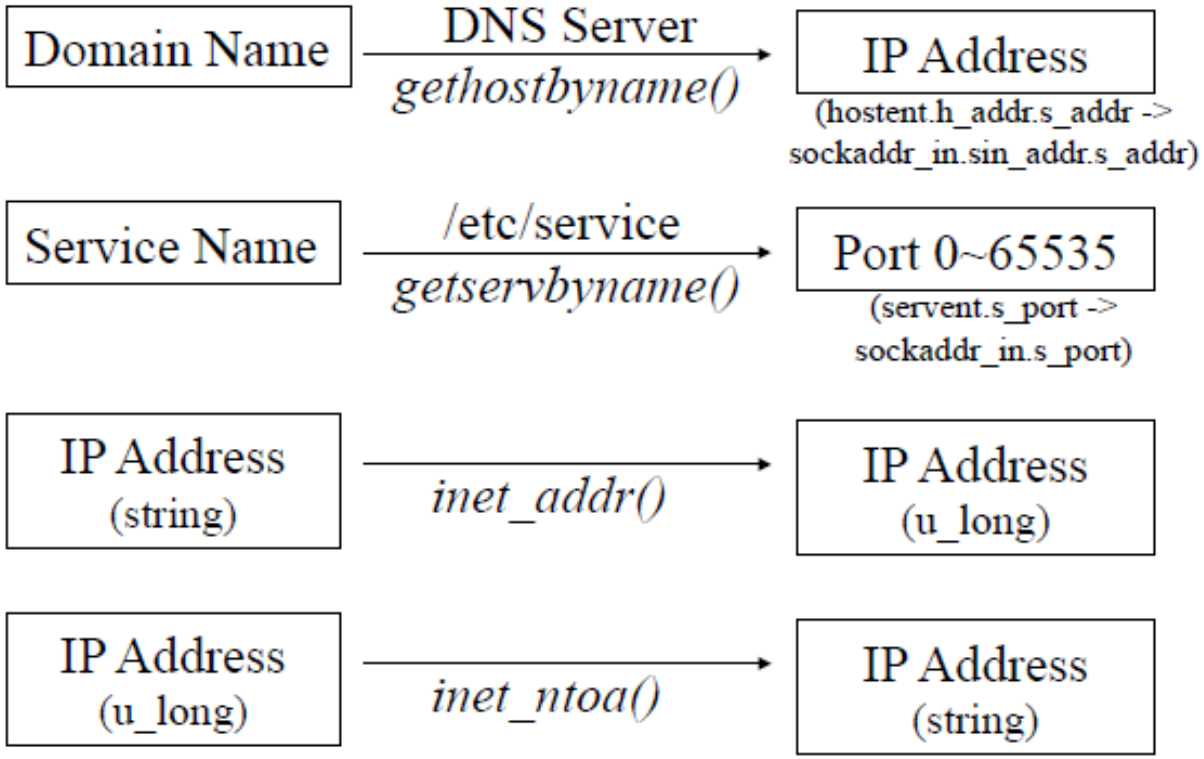
Each attachment point on Internet is given unique address.

- Based on location within network – like phone numbers.

Humans prefer to deal with names not addresses.

- DNS provides mapping of name to address.
- Name based on administrative ownership of host.

Address Translating Functions



Internet Addressing Data Structure

```
#include <netinet/in.h>

/* Internet address structure */
struct in_addr {
        u_long s_addr;          /* 32-bit IPv4 address */
};                              /* network byte ordered */

/* Socket address, Internet style. */
struct sockaddr_in {
      u_char  sin_family;       /* Address Family */
      u_short sin_port;         /* UDP or TCP Port# */
                                /* network byte ordered */
      struct in_addr sin_addr;        /* Internet Address */
      char    sin_zero[8];      /* unused */
};
```

sin_family = AF_INET selects Internet address family

Translating Names to Addresses

gethostbyname(): Given a hostname and return `hostent`.

gethostbyaddr(): Given host address and return `hostent`.

getservbyname(): Given a hostname and return `servent`. Used to get service description (typically port number).

```
#include <netdb.h>

struct hostent *hp; /*ptr to host info for remote*/
struct sockaddr_in peeraddr;
char *name = "www.cs.cmu.edu";

peeraddr.sin_family = AF_INET;
if((hp = gethostbyname(name))!=NULL)
{
        peeraddr.sin_addr.s_addr = ((struct in_addr*)(hp->h_addr))->s_addr;
        printf("Translate %s => %s\n", name, inet_ntoa(peeraddr.sin_addr));
}
```

Dealing with IP Addresses

IP Addresses are commonly written as strings (“128.2.35.50”), but programs deal with IP addresses as integers.

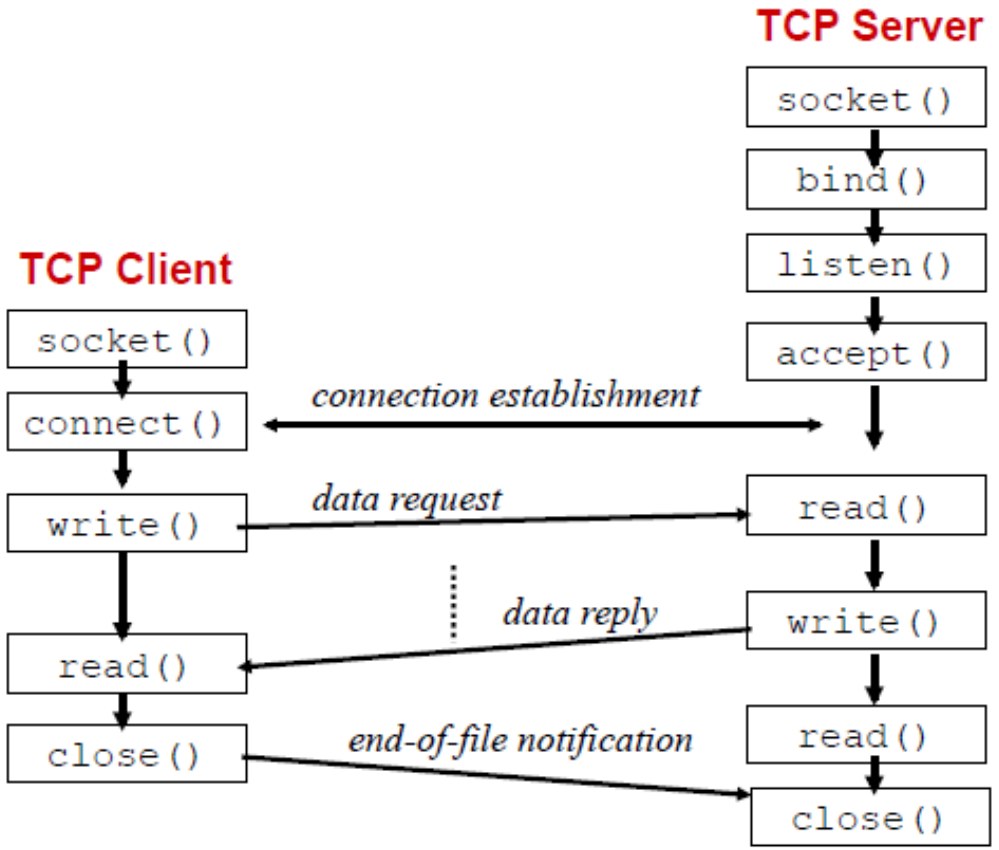
Converting strings to numerical address:

```
struct sockaddr_in srv;

srv.sin_addr.s_addr = inet_addr(“128.2.35.50”);
if(srv.sin_addr.s_addr == (in_addr_t) -1) {
      fprintf(stderr, "inet_addr failed!\n"); exit(1);
}
```

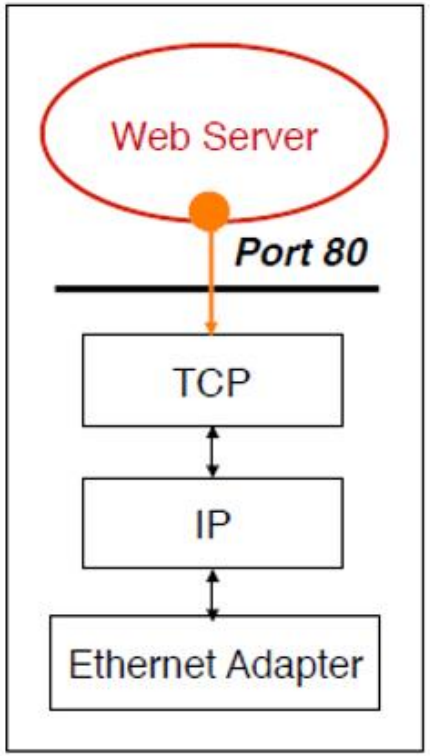
Converting a numerical address to a string:

```
struct sockaddr_in srv;
char *t = inet_ntoa(srv.sin_addr);
if(t == 0) {
      fprintf(stderr, “inet_ntoa failed!\n”); exit(1);
}
```

Recall TCP Client-Server Interaction



TCP Server



For example:

web server

What d(s a *web server* need to do so that a *web client* can connect to it?

Socket I/O: socket()

Since web traffic uses TCP, the web server must create a socket of type SOCK_STREAM

```
int fd;           /* socket descriptor */

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

socket returns an integer (**socket descriptor**)

- ***fd*** < 0 indicates that an error occurred

AF_INET associates a socket with the Internet protocol family

SOCK_STREAM selects the TCP protocol

Socket I/O: bind()

A ***socket*** can be bound to a ***port***

```
int fd;                              /* socket descriptor */
struct sockaddr_in srv;     /* used by bind() */

/* create the socket */

srv.sin_family = AF_INET; /* use the Internet addr family */

srv.sin_port = htons(80); /* bind socket 'fd' to port 80*/

/* bind: a client may connect to any of my addresses */
srv.sin_addr.s_addr = htonl(INADDR_ANY);

if(bind(fd, (struct sockaddr*) &srv, sizeof(srv)) < 0) {
       perror("bind"); exit(1);
}
```

Still not quite ready to communicate with a client...

Socket I/O: listen()

listen indicates that the server will accept a connection

```
int fd;                         /* socket descriptor */
struct sockaddr_in srv;         /* used by bind() */

/* 1) create the socket */
/* 2) bind the socket to a port */

if(listen(fd, 5) < 0) {
      perror("listen");
      exit(1);
}
```

Still not quite ready to communicate with a client...

Socket I/O: accept()

accept blocks waiting for a connection

```
int fd;                              /* socket descriptor */
struct sockaddr_in srv;             /* used by bind() */
struct sockaddr_in cli;             /* used by accept() */
int newfd;                          /* returned by accept() */
int cli_len = sizeof(cli); /* used by accept() */

/* 1) create the socket */
/* 2) bind the socket to a port */
/* 3) listen on the socket */

newfd = accept(fd, (struct sockaddr*) &cli, &cli_len);
if(newfd < 0) {
      perror("accept");   exit(1);
}
```

accept returns a new socket (***newfd***) with the same properties as the original socket (***fd***)

- ***newfd*** < 0 indicates that an error occurred

Socket I/O: accept() continued…

```
struct sockaddr_in cli;        /* used by accept() */
int newfd;                     /* returned by accept() */
int cli_len = sizeof(cli);     /* used by accept() */

newfd = accept(fd, (struct sockaddr*) &cli, &cli_len);
if(newfd < 0) {
      perror("accept");
      exit(1);
}
```

How does the server know which client it is?

- **cli.sin_addr.s_addr** contains the client's ***IP address***
- **cli.sin_port** contains the client's ***port number***

Now the server can exchange data with the client by

using ***read*** and ***write*** on the descriptor ***newfd***.

Why does ***accept*** need to return a new descriptor?

Socket I/O: read()

read can be used with a socket

***read* blocks waiting for data from the client but does not guarantee that sizeof(buf) is read**

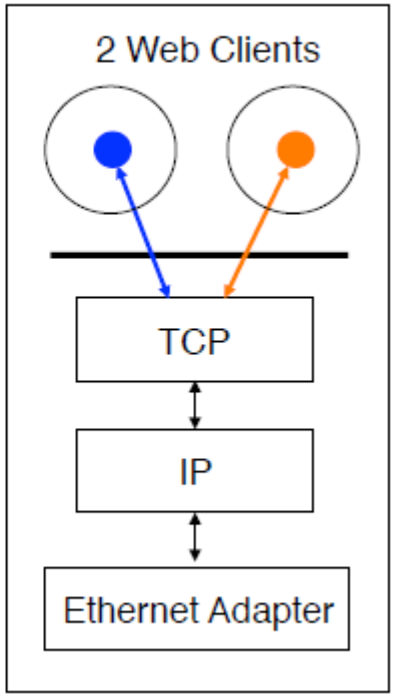
```
int fd;                                 /* socket descriptor */
char buf[512];                          /* used by read() */
int nbytes;                            /* used by read() */

/* 1) create the socket */
/* 2) bind the socket to a port */
/* 3) listen on the socket */
/* 4) accept the incoming connection */

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

TCP Client

For example:

web client

How does a *web client* connect to a *web server*?



Socket I/O: connect()

connect allows a client to connect to a server

```
int fd;                              /* socket descriptor */
struct sockaddr_in srv;              /* used by connect() */

/* create the socket */

/* connect: use the Internet address family */
srv.sin_family = AF_INET;

/* connect: socket 'fd' to port 80 */
srv.sin_port = htons(80);

/* connect: connect to IP Address "128.2.35.50" */
srv.sin_addr.s_addr = inet_addr("128.2.35.50");

if(connect(fd, (struct sockaddr*) &srv, sizeof(srv)) < 0) {
       perror("connect"); exit(1);
}
```

Socket I/O: write()

write can be used with a socket

```
int fd;                          /* socket descriptor */
struct sockaddr_in srv;          /* used by connect() */
char buf[512];                   /* used by write() */
int nbytes;                      /* used by write() */

/* 1) create the socket */
/* 2) connect() to the server */

/* Example: A client could "write" a request to a server
 */
if((nbytes = write(fd, buf, sizeof(buf))) < 0) {
      perror("write");
      exit(1);
}
```

Q & A