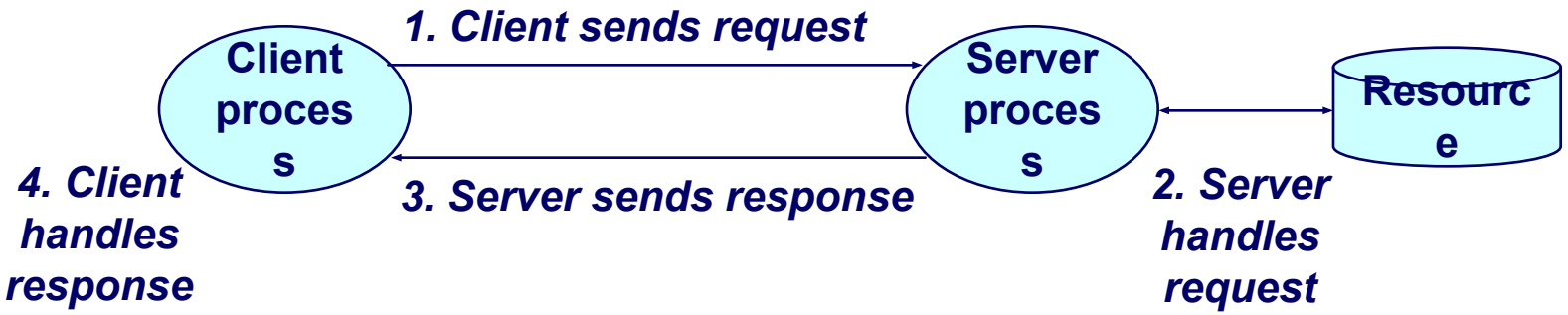
Socket Programming

Prof. H. B. Dave
eInfochips
Ahmedabad

A Client-Server Transaction

Every network application is based on the client-server model:

- A *server* process and one or more *client* processes
- Server manages some *resource*.
- Server provides *service* by manipulating resource for clients.



Note: clients and servers are processes running on hosts (can be the same or different hosts).

A Programmer's View of the Internet

1. Hosts are mapped to a set of 32-bit *IP addresses*.
   - 128.2.203.179

2. The set of IP addresses is mapped to a set of identifiers called Internet *domain names*.
   - 128.2.203.179 is mapped to www.cs.cmu.edu

3. A process on one Internet host can communicate with a process on another Internet host over a *connection*.

1. IP Addresses

32-bit IP addresses are stored in an *IP address struct*

- IP addresses are always stored in memory in network byte order (big-endian byte order)
- True in general for any integer transferred in a packet header from one machine to another.
  - E.g., the port number used to identify an Internet connection.

```
/* Internet address structure */
struct in_addr {
    unsigned int s_addr; /* network byte order (big-endian) */
};
```

Handy network byte-order conversion functions:

`htonl`: convert `long int` from host to network byte order.
`htons`: convert `short int` from host to network byte order.
`ntohl`: convert `long int` from network to host byte order.
`ntohs`: convert `short int` from network to host byte order.

2. Domain Naming System (DNS)

The Internet maintains a mapping between IP addresses and domain names in a huge worldwide distributed database called *DNS*.

- **Conceptually, programmers can view the DNS database as a collection of millions of *host entry structures*:**

```
/* DNS host entry structure */
struct hostent {
   char   *h_name;        /* official domain name of host */
   char   **h_aliases;    /* null-terminated array of domain names */
   int    h_addrtype;     /* host address type (AF_INET) */
   int    h_length;       /* length of an address, in bytes */
   char   **h_addr_list;  /* null-terminated array of in_addr structs */
};
```

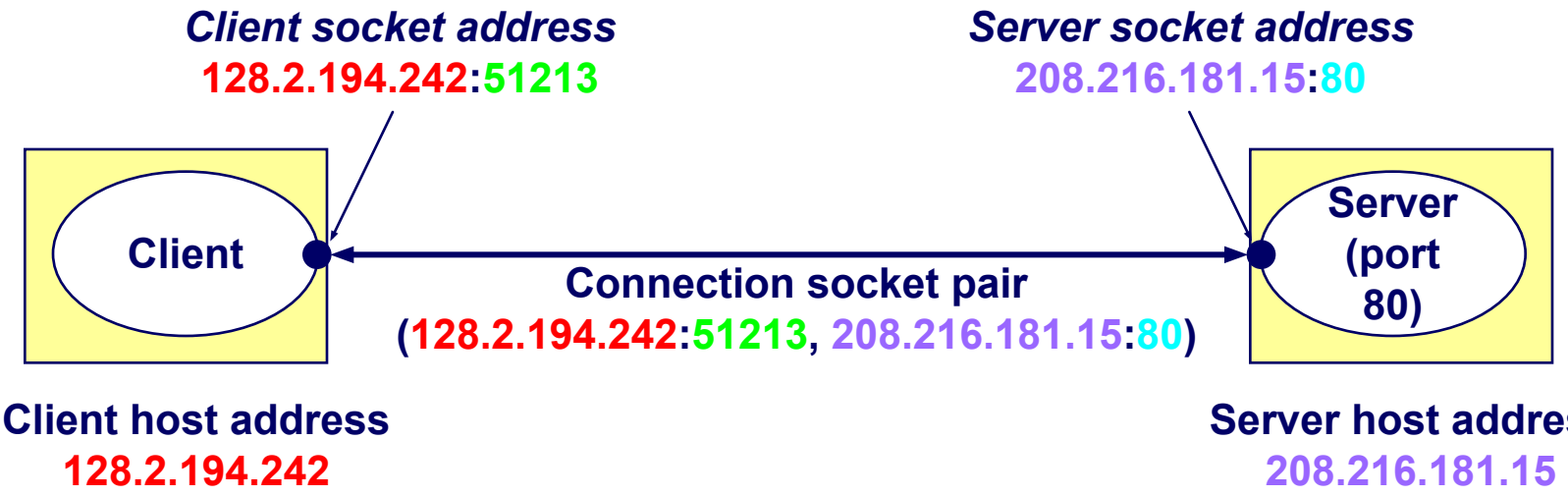
Functions for retrieving host entries from DNS:

- **`gethostbyname`: query key is a DNS domain name.**
- **`gethostbyaddr`: query key is an IP address.**

3. Internet Connections

Clients and servers communicate by sending streams of bytes over *connections*.

Connections are point-to-point, full-duplex (2-way communication), and reliable.



Note: 51213 is an ephemeral port allocated by the kernel

Note: 80 is a well-known port associated with Web servers

Clients

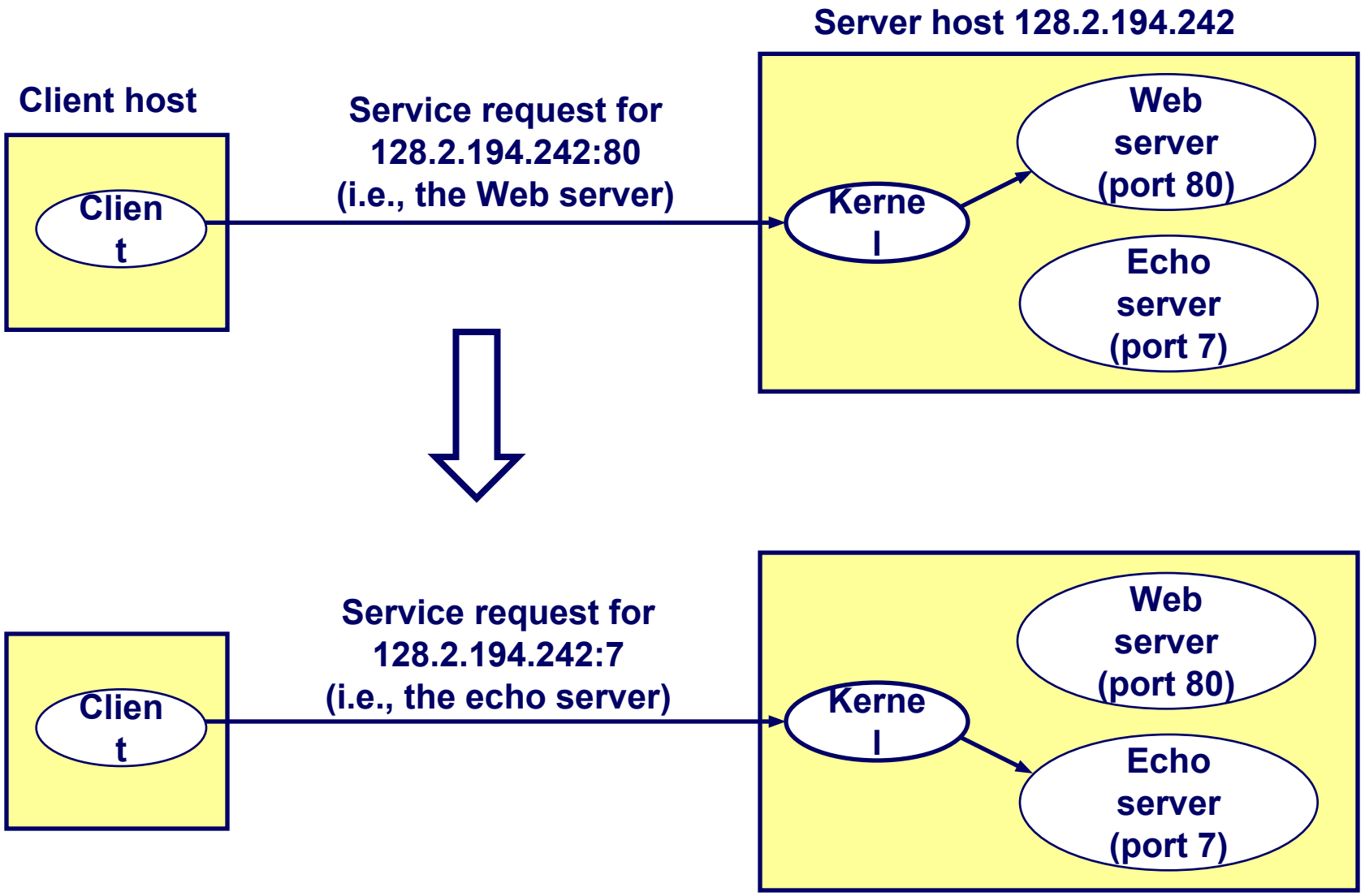
Examples of client programs

- Web browsers, `ftp`, `telnet`, `ssh`

How does a client find the server?

- The IP address in the server socket address identifies the host *(more precisely, an adapter on the host)*
- The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.
- Examples of well know ports
  - Port 7: Echo server
  - Port 23: Telnet server
  - Port 25: Mail server
  - Port 80: Web server

Using Ports to Identify Services



Servers

Servers are long-running processes (daemons).

- **Created at boot-time (typically) by the init process (process 1)**
- **Run continuously until the machine is turned off.**

Each server waits for requests to arrive on a well-known port associated with a particular service.

- **Port 7: echo server**
- **Port 23: telnet server**
- **Port 25: mail server**
- **Port 80: HTTP server**

A machine that runs a server process is also often referred to as a "server."

Server Examples

Web server (port 80)

- Resource: files/compute cycles (CGI programs)
- Service: retrieves files and runs CGI programs on behalf of the client

FTP server (20, 21)

- Resource: files
- Service: stores and retrieve files

See `/etc/services` for a comprehensive list of the services available on a Linux machine.

Telnet server (23)

- Resource: terminal
- Service: proxies a terminal on the server machine

Mail server (25)

- Resource: email "spool" file
- Service: stores mail messages in spool file

Sockets Interface

Created in the early 80's as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols.

Provides a user-level interface to the network.

Underlying basis for all Internet applications.

Based on client/server programming model.

1 Introduction

- **Abstraction: End of a communication link**
- **WAN, WLAN, LAN, within a system**
- **TCP/IP - Transport layer connection**
- **Compare it to audio-out socket of a DVD player connected to audio-in socket of the amplifier**
- **Flexibility of use leads to seeming comlexity in programming**

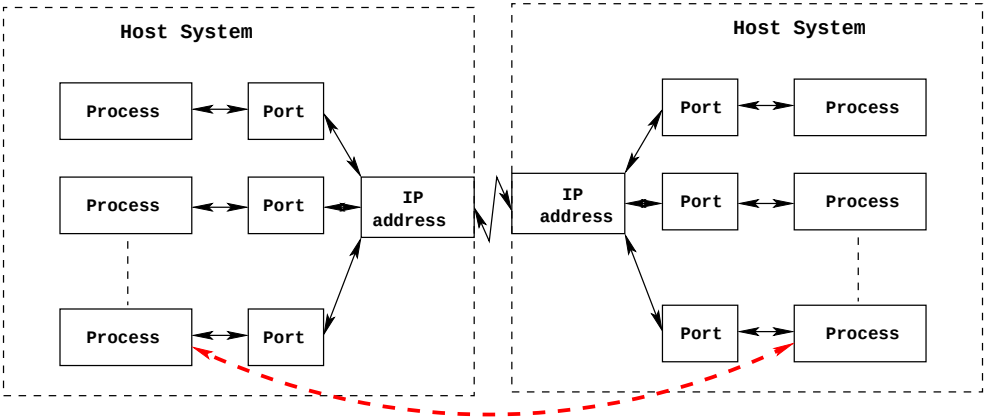


Figure 1 What we want

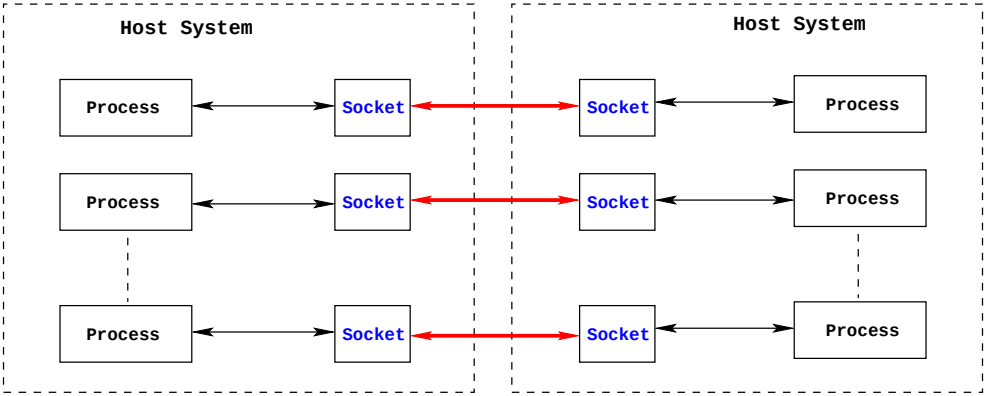


Figure 2 sockets are connection end-point abstraction

2 Background

- **IP address and ports**
- **port numbers: 1-65535**
  - **1 to 1023 - Well-known port numbers, assigned by IANA (Internet Assigned Numbers Authority)**
  - **1024 to 49151 - IANA registered port numbers**
  - **49152 to 65535 - IANA dynamic or provate (we can use for experimental purpose)**

- **Well-known ports are used by common ssytem utilities. E.g. 80 - Web server, 21,22 - FTP server, UDP 69 - TFTP,etc.**
- **The most important attribute of a socket is IP-address+Port-number**
- **Most used protocols: IPv4, TCP, UDP, SCTP (stream Control Trans Prot), ICMP (Internet Control Message Prot) and ARP (Address Resolution Prot)**
- **TCP - Transmission Control prot, also called connection-oriented protocol, compare it to a telephonic talk connection**

- **UDP - User Datagram Prot, also called conenctionless protocol, compare it to a Post-card sent/received**
- **Concurrent Servers usually use TCP, Iterative Servers use UDP.**

3 Some Standard Internet Services

Name	TCP port	UDP port
echo	7	7
discard	9	9
daytime	13	13
chargen	19	19
time	37	37

Table 1 Some Standard Internet Services

Sockets

What is a socket?

- To the kernel, a socket is an endpoint of communication.
- To an application, a socket is a file descriptor that lets the application read/write from/to the network.
  - Remember: All Unix I/O devices, including networks, are modeled as files.

Clients and servers communicate with each by reading from and writing to socket descriptors.

The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors.

Socket Address Structures

Generic socket address:

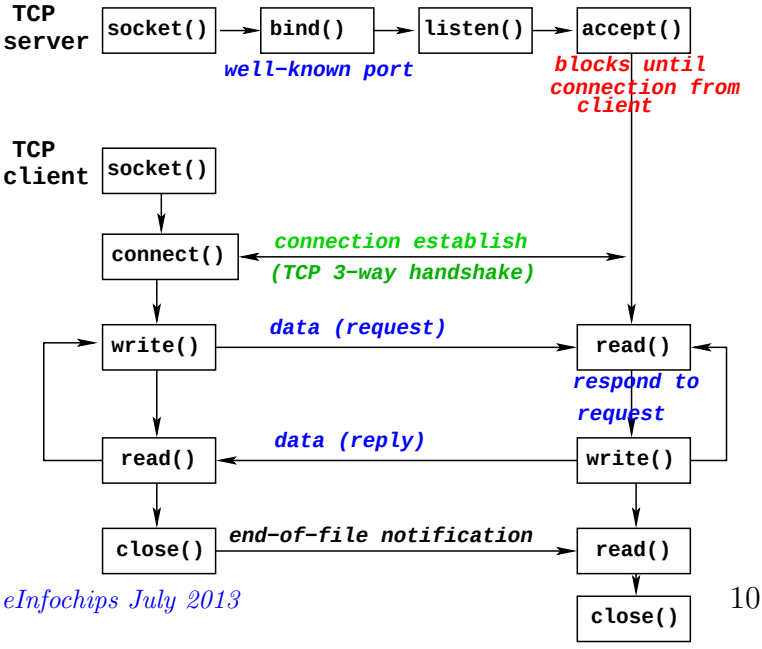
- For address arguments to `connect`, `bind`, and `accept`.
- Necessary only because C did not have generic (`void *`) pointers when the sockets interface was designed.

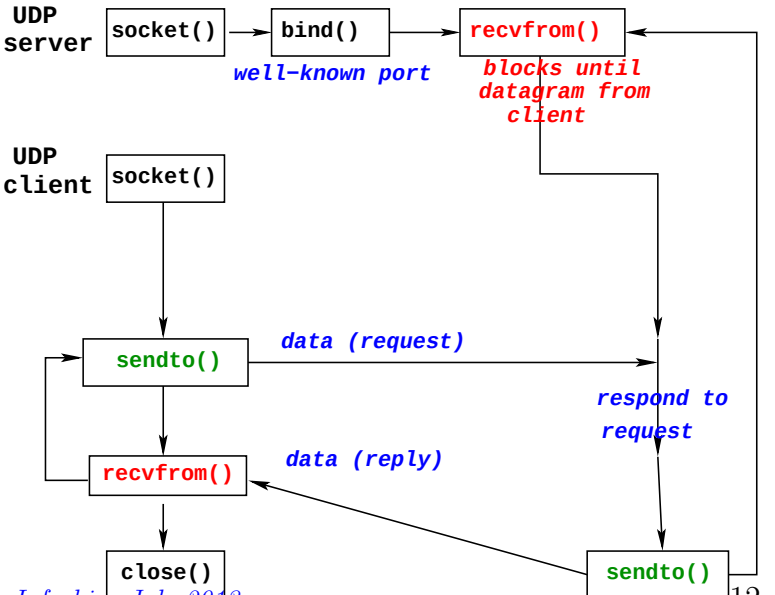
```
struct sockaddr {
  unsigned short  sa_family;    /* protocol family */
  char            sa_data[14];  /* address data.  */
};
```

Internet-specific socket address:

- Must cast (`sockaddr_in *`) to (`sockaddr *`) for `connect`, `bind` and `accept`

```
struct sockaddr_in  {
  unsigned short  sin_family;  /* address family (always AF_INET) */
  unsigned short  sin_port;    /* port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```


TCP server
socket()
bind()
listen()
accept()
well-known port
blocks until connection from client
TCP client
socket()
connect()
connection establish
(TCP 3-way handshake)
write()
data (request)
read()
respond to request
read()
data (reply)
write()
close()
end-of-file notification
read()
close()


UDP server
socket()
bind()
recvfrom()
well-known port
blocks until datagram from client
UDP client
socket()
sendto()
data (request)
respond to request
recvfrom()
data (reply)
close()
sendto()

4 Socket Functions

- `socket()` - **creates a socket**
- `bind()` - **binds a socket to a local socket address**
- `connect()` - **connects a socket to a remote socket address**
- `listen()` - **makes a socket a listening socket, new connections will be accepted**
- `accept()` - **used to get a new socket with a new incoming connection**

- `recvfrom()` - **receive data from a socket**
- `sendto()` - **send data over a socket**
- `read()` - **read data**
- `write()` - **write data**
- `close()` - **close a socket**
- `getsockname()` - **returns the local socket address**
- `getpeername()` - **returns the remote socket address**

- `getsockopt()` - **get socket layer or protocol options**
- `setsockopt()` - **set socket layer or protocol options**

4.1 socket()

Creates an endpoint for communication and returns a descriptor.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

domain - a communication domain, selects the protocol family which will be used for communication. The currently understood formats include:

Name	Purpose	Man page
AF_UNIX, AF_LOCAL	Local communication	unix(7)
AF_INET	IPv4 Internet protocols	ip(7)
AF_INET6	IPv6 Internet protocols	ipv6(7)
AF_NETLINK	Kernel user interface device	netlink(7)
AF_PACKET	Low level packet interface	packet(7)

type - the way communication takes place, values include

```
SOCK_STREAM - sequenced, reliable, two-way, connection-
based byte  streams
SOCK_DGRAM  - connectionless, unreliable messages
of a fixed maximum length
SOCK_RAW        Provides raw network protocol access
SOCK_NONBLOCK
```

`protocol` - specifies a particular protocol

4.2 Socket Data Structures

```
/* Internet address. */
struct in_addr {
    uint32_t s_addr; /* network byte order */
};

struct sockaddr_in {
    sa_family_t    sin_family; /*  AF_INET */
    in_port_t      sin_port;   /* network byte order
*/
    struct in_addr sin_addr;   /* internet address
*/
};
```

The actual structure passed for the addr argument will depend on the address family. The sockaddr structure is defined as something like:

```
struct sockaddr {
    sa_family_t sa_family;
    char        sa_data[14];
}
```

The only purpose of this structure is to cast the structure pointer passed in `addr` in order to avoid compiler warnings.

Network byte-order - Big-Endian

Functions are available to convert from the Host byte-order to network byte-order and vice-versa.

4.3 bind()

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr,
                     socklen_t addrlen);
```

The actual structure passed for the addr argument will depend on the address family.

4.4 connect()

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr,
                    socklen_t addrlen);
```

connects the socket referred to by the file descriptor `sockfd` to the address specified by `addr`.

4.5 listen()

```
#include <sys/types.h>
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

marks the socket referred to by `sockfd` as a *passive* socket - as a socket that will be used to accept incoming connection requests using `accept()`.

4.6 Acceptance of a connection

To accept connections, the following steps are performed:

1. A socket is created with `socket()`.
2. The socket is bound to a local address using `bind()`, so that other sockets may be `connect()`ed to it.
3. A willingness to accept incoming connections and a queue limit for incoming connections are specified with `listen()`.
4. Connections are accepted with `accept()`.

Echo Server: Main Loop

The server loops endlessly, waiting for connection requests, then reading input from the client, and echoing the input back to the client.

```
main() {

   /* create and configure the listening socket */

   while(1) {
      /* Accept(): wait for a connection request */
      /* echo(): read and echo input lines from client til EOF */
      /* Close(): close the connection */
   }
}
```

Echo Server: `accept`

`accept()` blocks waiting for a connection request.

```
int listenfd; /* listening descriptor */
int connfd;   /* connected descriptor */
struct sockaddr_in clientaddr;
int clientlen;

clientlen = sizeof(clientaddr);
connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
```

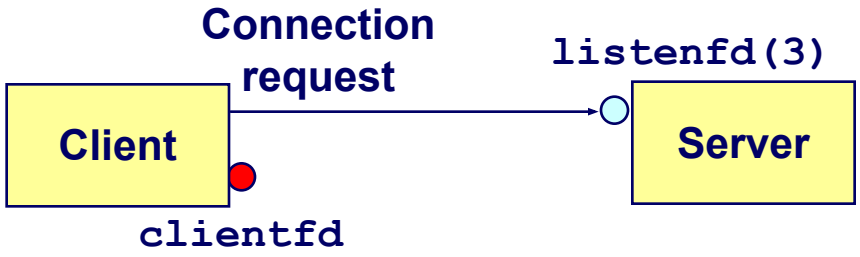
the same properties as the *listening descriptor* (`listenfd`)

- Returns when the connection between client and server is created and ready for I/O transfers.
- All I/O with the client will be done via the connected socket.

`accept` also fills in client's IP address.

Echo Server: `accept` Illustrated

1. Server blocks in `accept`, waiting for connection request on listening descriptor `listenfd`.



2. Client makes connection request by calling and blocking in `connect`.

3. Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`.

Connected vs. Listening Descriptors

Listening descriptor

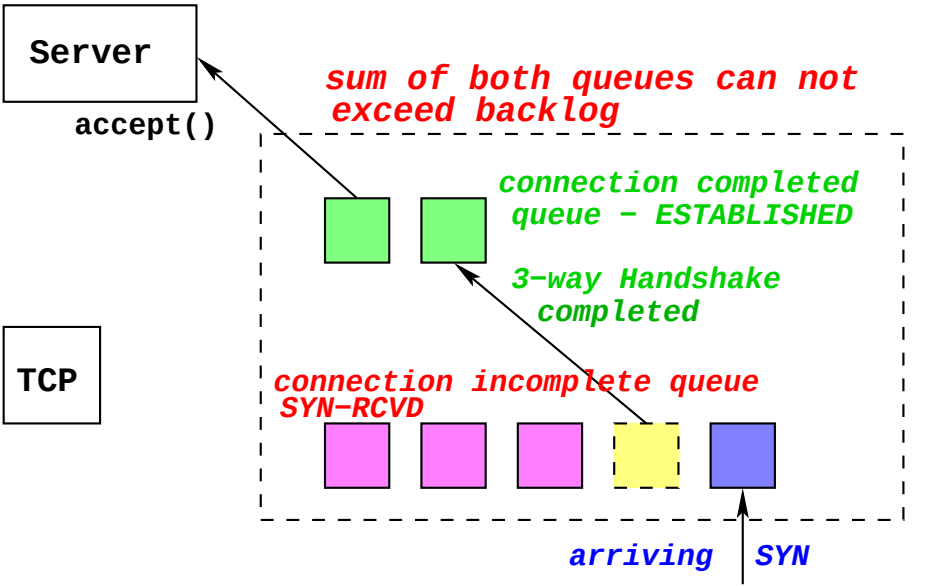
- End point for client connection requests.
- Created once and exists for lifetime of the server.

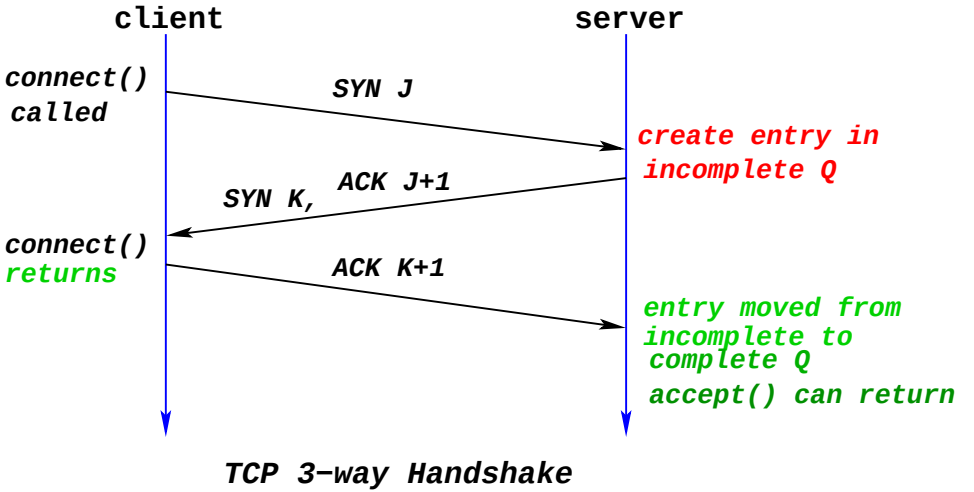
Connected descriptor

- End point of the connection between client and server.
- A new descriptor is created each time the server accepts a connection request from a client.
- Exists only as long as it takes to service client.

Why the distinction?

- Allows for concurrent servers that can communicate over many client connections simultaneously.
  - E.g., Each time we receive a new request, we fork a child to handle the request.


Server
accept()
sum of both queues can not exceed backlog
connection completed queue – ESTABLISHED
3–way Handshake completed
TCP
connection incomplete queue SYN–RCVD
arriving SYN



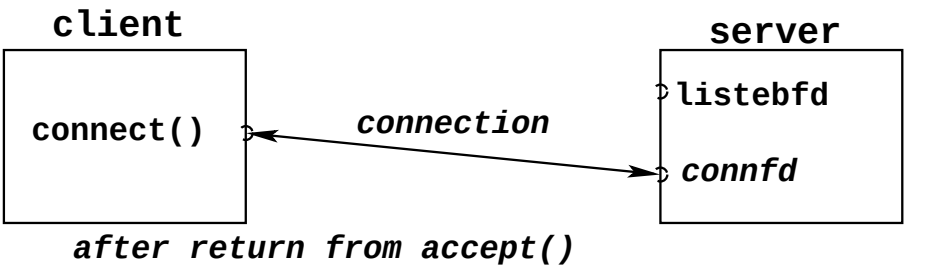
TCP 3−way Handshake

4.7 accept()

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t
*addrlen);
```

extracts the first connection request on the queue of pending connections for the listening socket `sockfd`, *creates a new connected socket*, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket `sockfd` is not affected.


client
server
connect()
connection request
listenfd
before call to accept() returns


client
server
connect()
connection
listebfd
connfd
after return from accept()

4.8 recvfrom()

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t recv(int sockfd, void *buf, size_t len,
int flags);

ssize_t recvfrom(int sockfd, void *buf, size_t len,
int flags,
                 struct sockaddr *src_addr, socklen_t
*addrlen);
```

```
ssize_t recvmsg(int sockfd, struct msghdr *msg,
int flags);
```

4.9 sendto()

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t send(int sockfd, const void *buf, size_t
len, int flags);

ssize_t sendto(int sockfd, const void *buf, size_t
len, int flags,
              const struct sockaddr *dest_addr, socklen_t
addrlen);

ssize_t sendmsg(int sockfd, const struct msghdr
*msg, int flags);
```

5 Demo Programs

- **Daytime Client: TCP, IPv4** `unpv13e/intro/daytimetcpcli.c`
- **Daytime Server: TCP, IPv4** `unpv13e/intro/daytimetcpsrv.c`
- **Echo Server: TCP, IPv4** `unpv13e/tcpcliserv/tcpserv01.c`
- **Echo Client: TCP, IPv4** `unpv13e/tcpcliserv/tcpcli01.c`
- **Echo Server: UDP, IPv4** `unpv13e/udpcliserv/udpserv01.c`
- **Echo Client: UDP, IPv4** `unpv13e/udpcliserv/udpcli01.c`
- **Daytime Client: TCP, IPv4** `unpv13e/names/daytimetcpcli1.c`

6 Some Useful Utilities for testing

- **NetCat** - `nc`
- `WireShark`
- `netstat`
- `yes` - **used to generate data**

Testing Servers Using `telnet`

The `telnet` program is invaluable for testing servers that transmit ASCII strings over Internet connections

- **Our simple echo server**
- **Web servers**
- **Mail servers**

Usage:

- **`unix> telnet <host> <portnumber>`**
- **Creates a connection with a server running on *`<host>`* and listening on port *`<portnumber>`*.**

Testing the Echo Server With

```
bass> echoserver 5000
server established connection with KITTYHAWK.CMCL (128.2.194.242)
server received 5 bytes: 123
server established connection with KITTYHAWK.CMCL (128.2.194.242)
server received 8 bytes: 456789


kittyhawk> telnet bass 5000
Trying 128.2.222.85...
Connected to BASS.CMCL.CS.CMU.EDU.
Escape character is '^]'.
123
123
Connection closed by foreign host.
kittyhawk> telnet bass 5000
Trying 128.2.222.85...
Connected to BASS.CMCL.CS.CMU.EDU.
Escape character is '^]'.
456789
456789
Connection closed by foreign host.
kittyhawk>
```

Running the Echo Client and Server

```
bass> echoserver 5000
server established connection with KITTYHAWK.CMCL (128.2.194.242)
server received 4 bytes: 123
server established connection with KITTYHAWK.CMCL (128.2.194.242)
server received 7 bytes: 456789
...


kittyhawk> echoclient bass 5000
Please enter msg: 123
Echo from server: 123


kittyhawk> echoclient bass 5000
Please enter msg: 456789
Echo from server: 456789
kittyhawk>
```

1 References

- **Richard Stevens et al, "Unix Network Programing" vol.1, 3rd Ed, Prentice-Hall**
- **Robbins and Robbins, "Prctical Unix Programing" Prentice-Hall**