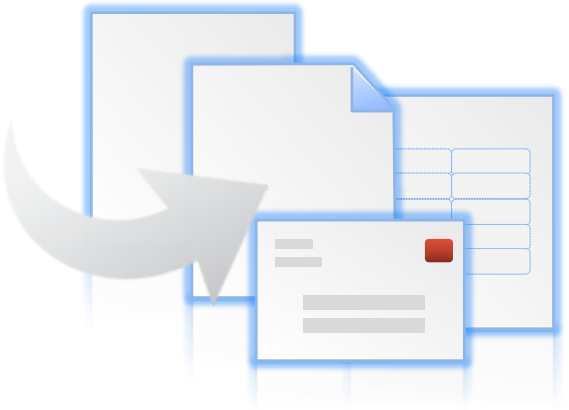
# NETWORK PROGRAMMING

**UNIT 3 ELEMENTARY TCP SOCKETS**

**BY**

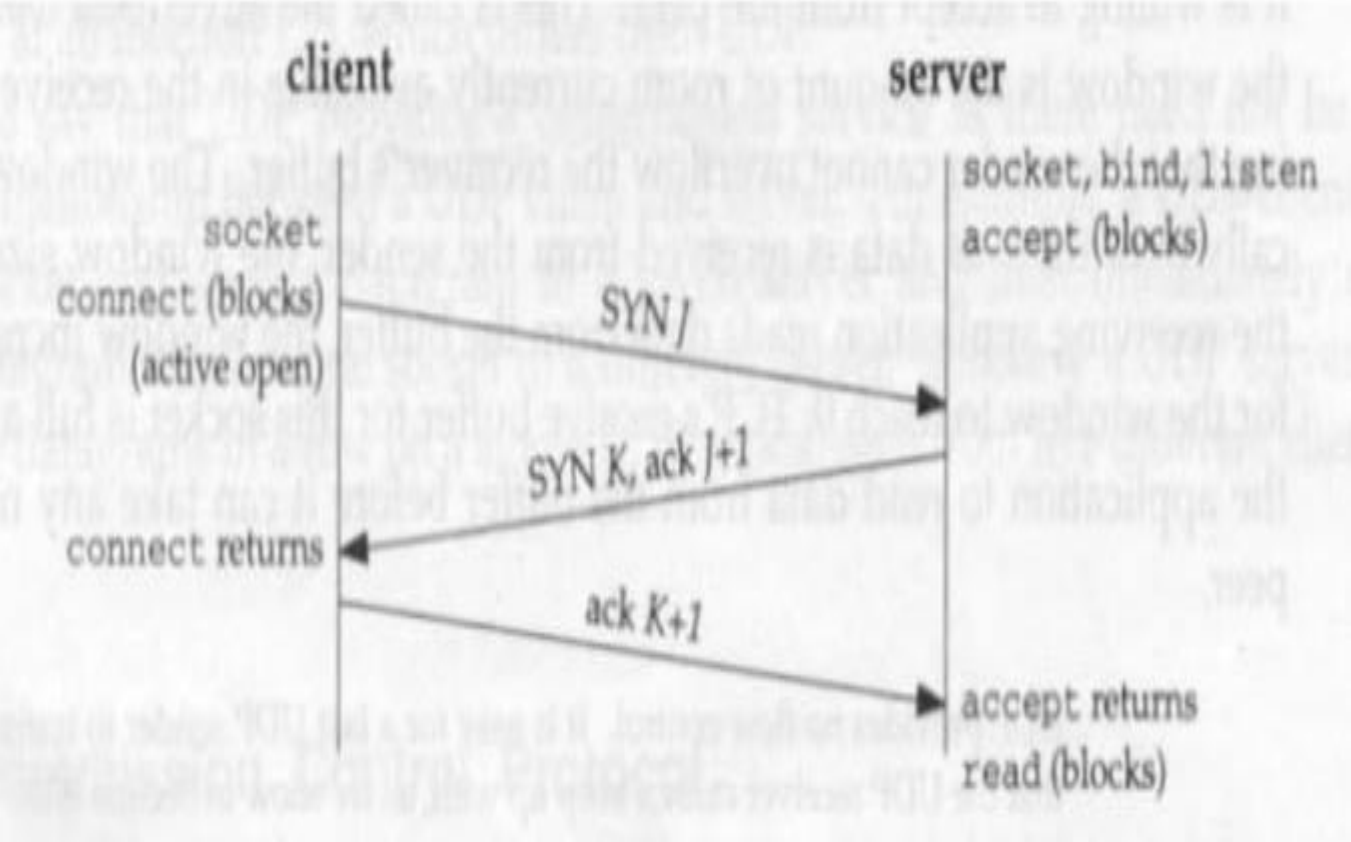
**MR.PRASAD SAWANT**



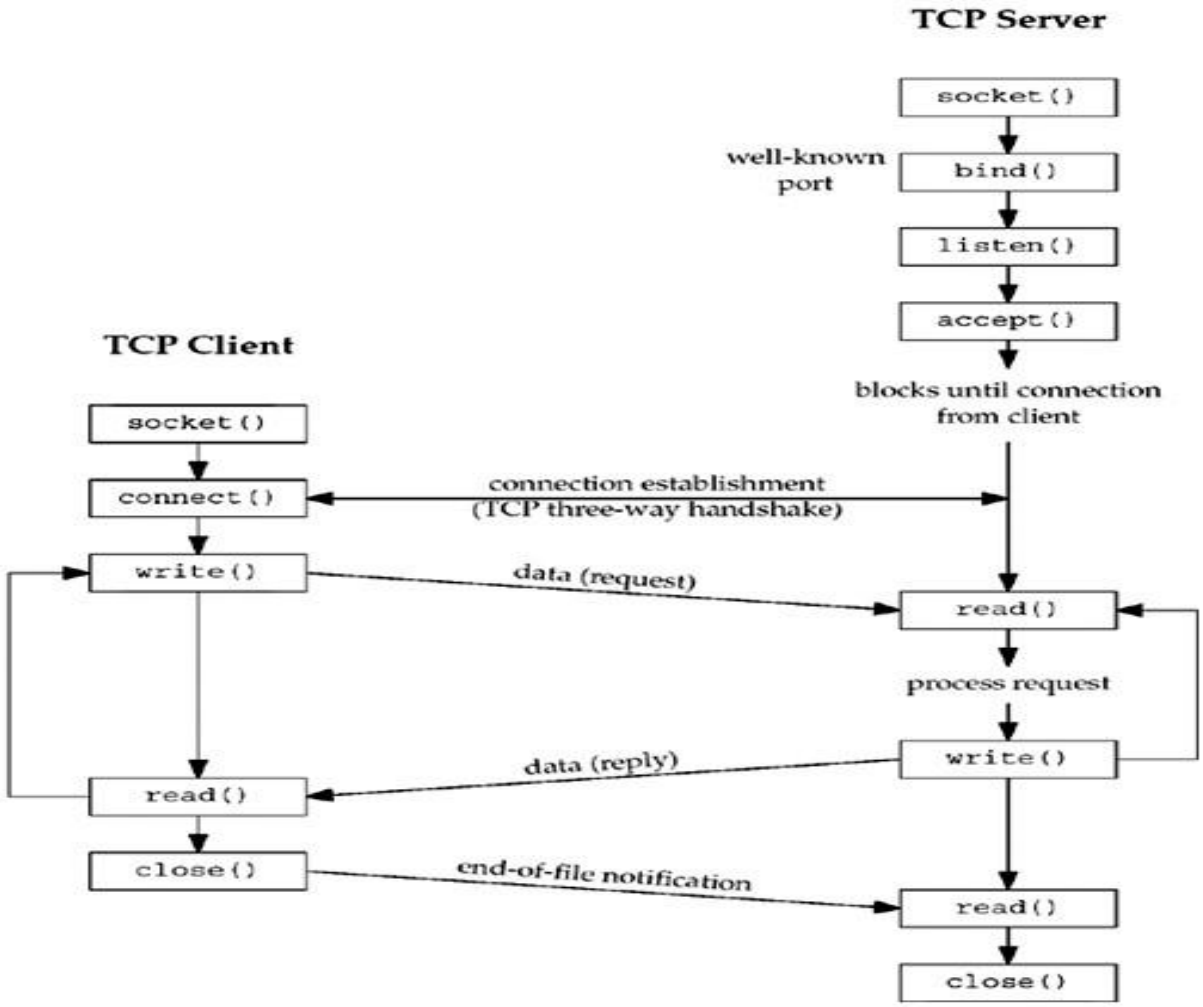
# ELEMENTARY TCP SOCKETS

1. *socket* function
2. *connect* function
3. *bind* function
4. *listen* function
5. *accept* function
6. *fork* and *exec* functions
7. Concurrent servers
8. *close* function
9. *getsockname* and *getpeername* functions

# TCP THREE-WAY HANDSHAKE



# Socket Functions For Elementary TCP Client/Server



# SOCKET FUNCTION

- To open a socket for performing network I/O

```
#include <sys/socket.h>
int socket (int family, int type, int protocol);
                returns: nonnegative descriptor if OK, -1 on error
```

| *family* | Description |
|---|---|
| AF_INET | IPv4 protocols |
| AF_INET6 | IPv6 protocols |
| AF_LOCAL | Unix domain protocols (Chapter 15) |
| AF_ROUTE | Routing sockets (Chapter 18) |
| AF_KEY | Key socket (Chapter 19) |

**Figure 4.3. *type* of socket for `socket` function.**

| *type* | Description |
|---|---|
| SOCK_STREAM | stream socket |
| SOCK_DGRAM | datagram socket |
| SOCK_SEQPACKET | sequenced packet socket |
| SOCK_RAW | raw socket |

**Figure 4.4. *protocol* of sockets for `AF_INET` or `AF_INET6`.**

| *Protocol* | Description |
|---|---|
| IPPROTO_TCP | TCP transport protocol |
| IPPROTO_UDP | UDP transport protocol |
| IPPROTO_SCTP | SCTP transport protocol |

# SOCKET FUNCTION

**Combinations of family and type for the socket function.**

| | AF-INET | AF_INET6 | AF_LOCAL | AF_ROUTE | AF_KEY |
|---|---|---|---|---|---|
| SOCK_STREAM | TCP | TCP | YES | | |
| SOCK_DGRAM | UDP | UDP | YES | | |
| SOCK_RAW | IPv4 | IpV6 | | YES | YES |

# The Connect() Function (1/3)

- Is used by a client to establish a connection with a server via a 3-way handshake

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

int connect(int sockfd,  const struct sockaddr *servaddr,
            socklen_t  addrlen);

                                    Returns: 0 if OK, -1 on error
```

  - *sockfd* is a socket descriptor returned by the socket() function
  - *servaddr* contains the IP address and port number of the server
  - *addrlen* has the length (in bytes) of the server socket address structure

# THE CONNECT() FUNCTION (2/3)

- This function returns only when the connection is established or an error occurs
- Some possible errors:
  - If the client TCP receives no response to its SYN segment, ETIMEDOUT is returned
    - The connection-establishment timer expires after 75 seconds
      - The client will resend SYN after 6 seconds later, and again another 24 seconds later. If no response is received after a total of 75 seconds, the error is returned

# THE CONNECT() FUNCTION (3/3)

- **Hard error**: RST received in response to client TCP's SYN (server not running)
  - returns ECONNREFUSED
- If an ICMP "destination unreachable" is received from an intermediate router, EHOSTUNREACH or ENETUNREACH is returned. This is a **soft error**
  - Upon receiving the first ICMP message, the client kernel will keep sending SYNs at the same time intervals as mentioned earlier, until after 75 seconds have elapsed (4.4BSD)

# The bind() Function (1/2)

- Assign a local protocol address to a socket

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

int bind(int sockfd,  const struct sockaddr *myaddr,
         socklen_t  addrlen);

                                  Returns: 0 if OK, -1 on error
```

  - *sockfd* is a socket descriptor returned by the socket() function
  - *myaddr* is a pointer to a protocol-specific address. With TCP, it has the IP address and port number of the server
  - *addrlen* has the length (in bytes) of the server socket address structure


# The bind() Function (2/2)

- IP address/Port number assignment :

| Process specifies | | Results |
|---|---|---|
| IP address | Port | |
| Wildcard | 0 | Kernel chooses IP address and port |
| Wildcard | Nonzero | Kernel chooses IP address, process specifies port |
| Local IP addr | 0 | Process specifies IP address, kernel chooses port |
| Local IP addr | Nonzero | Process specifies IP address and port |

  - Wildcard address: INADDR_ANY (IPv4), in6addr_any (IPv6)

```
struct sockaddr_in  servaddr;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

  - TCP servers typically bind their well-known port, and clients let the kernel choose an ephemeral port

# The listen() Function (1/2)

- Is used by a server to convert an unconnected socket to a passive socket

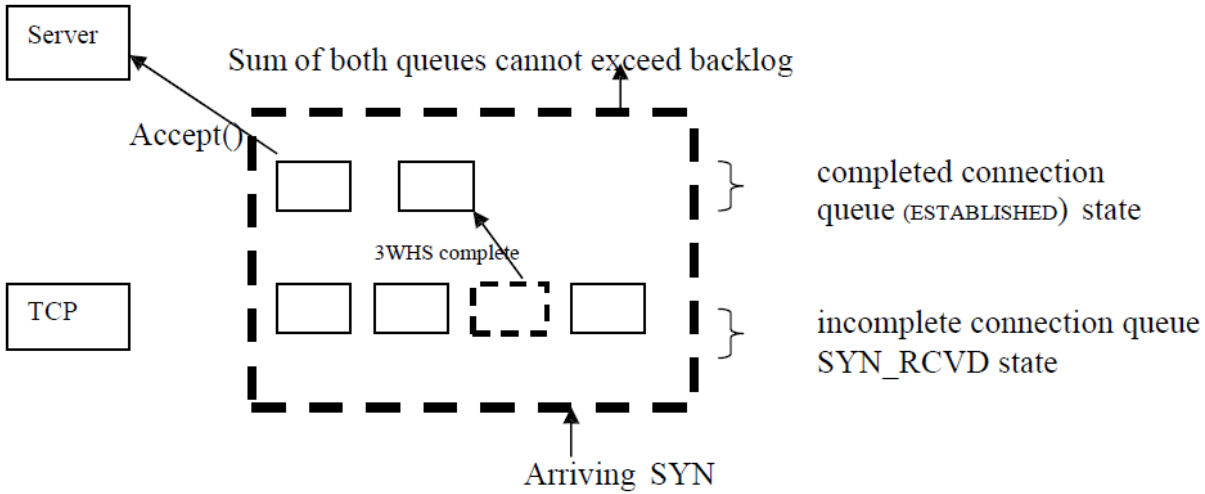
```
#include <sys/socket.h>

int listen(int sockfd,  int backlog);

                                    Returns: 0 if OK, -1 on error
```

  - *sockfd* is a socket descriptor returned by the socket() function
  - *backlog* specifies the maximum number of connections the kernel should queue for this socket

# The listen() Function (2/2)

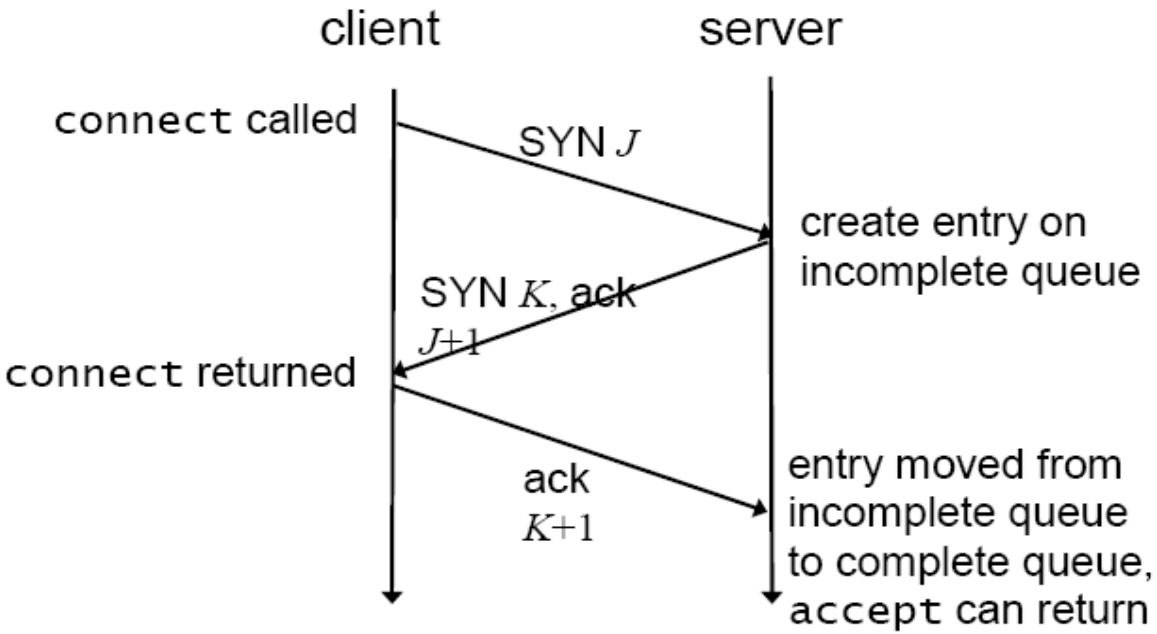
The kernel maintains two queues and the backlog is the sum of these two queues

- An **Incomplete Connection Queue**, which contains an entry for each SYN that has arrived from a client for which the server is awaiting completion of the TCP three way handshakes. These sockets are in the **SYN_RECD** state
- A **Completed Connection Queue** which contains an entry for each client with whom three handshakes has completed. These sockets are in the **ESTABLISHED** state.




The two queues maintained by TCP for a listening socket.

# THE TWO QUEUES



# The accept() Function (1/2)

- Is called by a server to return a new descriptor, created automatically by the kernel, for the connected socket

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

int accept(int sockfd,  struct sockaddr *cliaddr,
           socklen_t  *addrlen);

                Returns: non-negative descriptor if OK, -1 on error
```

  - *sockfd* is a socket descriptor returned by the socket() function
  - *cliaddr* contains the IP address and port number of the connected client (a value-result argument)
  - *addrlen* has the length (in bytes) of the returned client socket address structure (a value-result argument)
  - If the completed connection queue is empty, the process is put to sleep

# The accept() Function (2/2)

- The new socket descriptor returned by accept() is called a *connected socket*, whereas the one returned by socket() is called a *listening socket*
  - A given server usually creates only one *listening socket*. It exists for the lifetime of the server
  - A *connected socket* is created for each client connection that is accepted. It exists only for the duration of the connection
- Both *cliaddr* and *addrlen* may be set to the NULL pointer, if the server is not interested in knowing the identity of the client

# The UNIX fork() Function (1/2)

- Is used in UNIX to create a new process

```
#include <unistd.h>

pid_t fork(void);

    Returns: 0 in child, process ID of child in parent, -1 on error
```

  - *fork()* is called once, but returns
    - Once in the calling process, called the *parent*
    - Once in the newly created process, called the child
  - A parent may have more than 1 child process

## The UNIX fork() Function (2/2)

- All descriptors open in the parent before fork() are shared with the child after fork()
  - The connected socket is then shared between the parent and the child
- Two typical uses of fork() :
  - A process makes a copy of itself so that one copy can handle one operation, and the other copy does something else
    - This is typical for network servers
  - A process want to execute a new program by calling exec() in the child process
    - User commands in UNIX are typically handled this way
- fork() can be used to implement concurrent servers

# The UNIX exec() Function (1/3)

- Is used in UNIX to execute a program.
- Is a family name for six like functions virtually doing the same thing, only slightly different in syntax

```
#include <unistd.h>

int  execl(…), execv(…), execle(…), execve(…),
     execlp(…), execvp(…);

                    Returns: -1 on error, no return on success
```

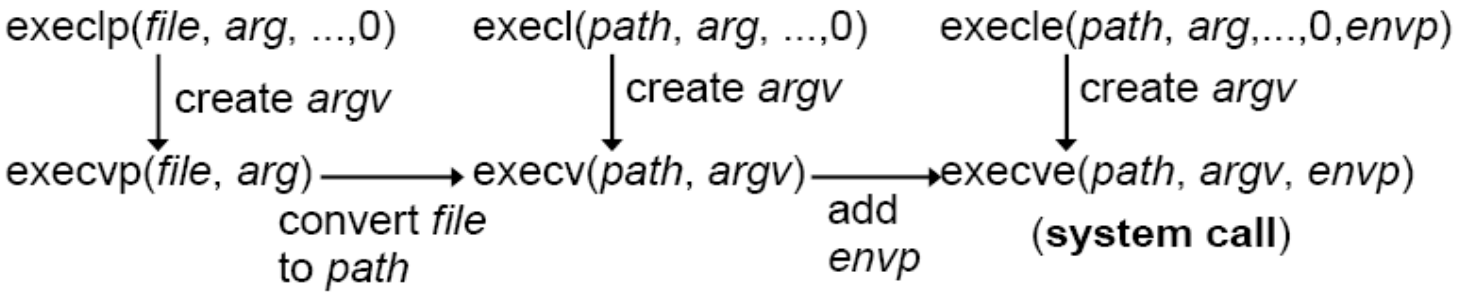
  - Descriptors open in the process before calling exec() normally remain open in the new program

# exec Function (2/3)

```
int execl (const char *pathname, const char *arg0, .... /* (char *) 0 */);

int execv (const char *pathname, char *const argv[ ]);

int execle (const char *pathname, const char *arg0, ... /* (char *)0,
        char *const envp[ ] */);

int execve (const char *pathname, char *const argv[ ], char *const envp[ ]);

int execlp (const char *filename, const char *arg0, ... /* (char *) 0 */);

int execvp (const char *filename, char *const argv[ ]);
```

All return: -1 on error, no return on success

# THE EXEC FAMILY (3/3)




- Meaning of different letters :

*l* : needs a list of arguments

*v* : needs an *argv[]* vector (*l* and *v* are mutually exclusive)

*e* : needs an *envp[]* array

*p* : needs the PATH variable to find the executable file

# close function

- The normal Unix close function is also used to close a socket and terminate a TCP connection.
-

```
#include <unistd.h>
int close (int sockfd);
```

Returns: 0 if OK, -1 on error



# *getsockname () and getpeername():*

These two functions return either the local protocol address associated with a socket (getsockname) or the foreign protocol address associated with a socket (getpeername).

```
#include <sys/socket.h>
int getsockname(int sockfd, struct sockaddr *localaddr, socklen_t *addrlen);
int getpeername(int sockfd, struct sockaddr *peeraddr, socklen_t *addrlen);
```

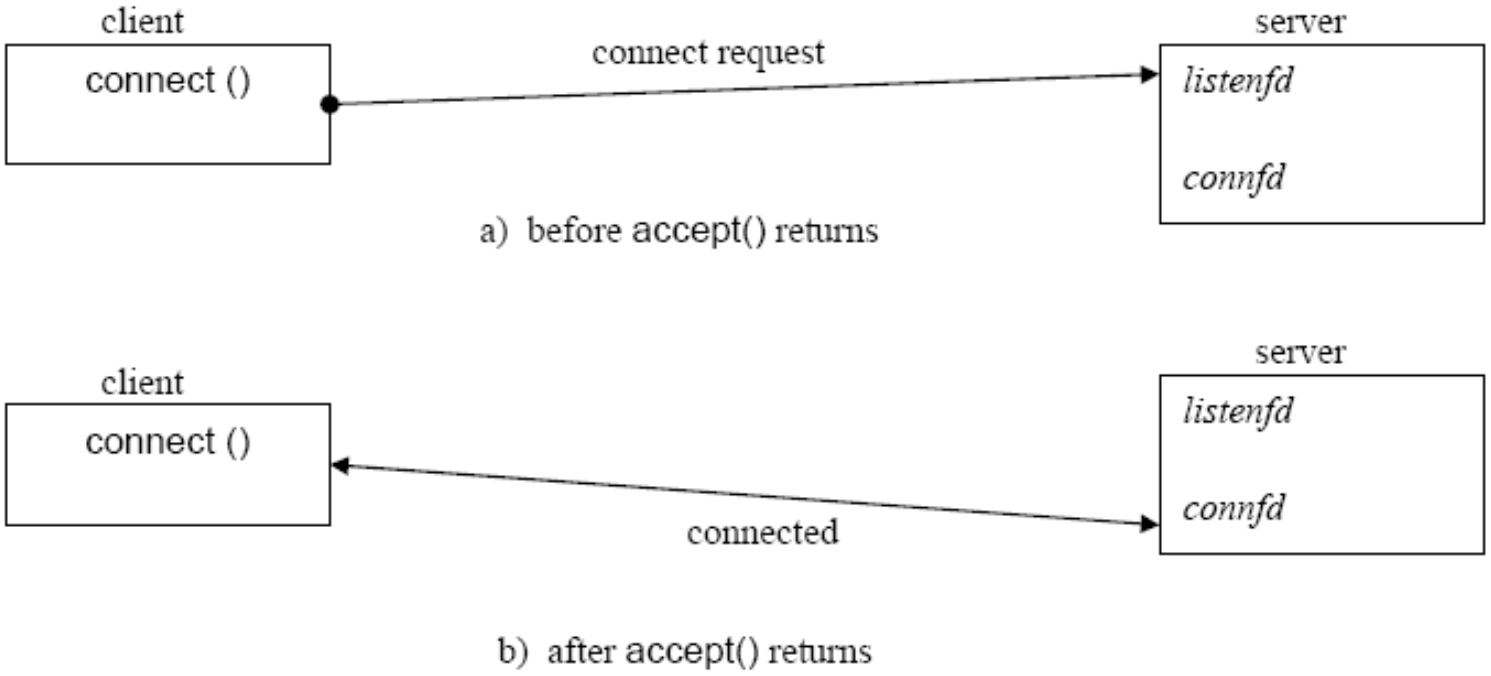
Both return: 0 if OK, -1 on error



# CONCURRENT SERVERS: OUTLINE

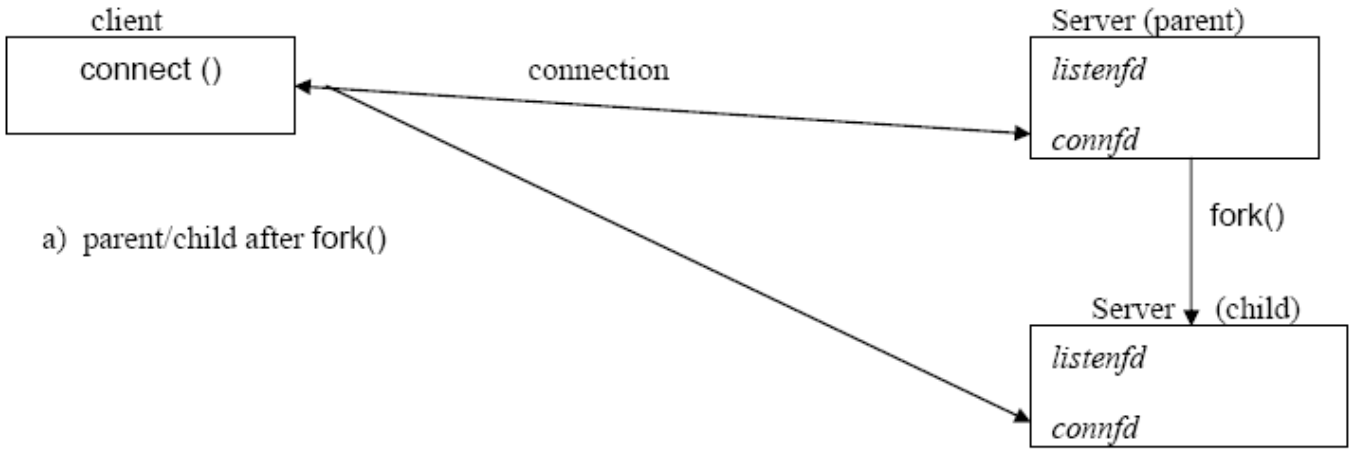
```
pid_t pid;
int listenfd, connfd;
listenfd = Socket (...);
/* fill in socket_in{} with server's well-known port */
Bind (listenfd, ...);
Listen (listenfd, LISTENQ);

for ( ; ; ){
        connfd = Accept (listenfd, ...); /* probably blocks */
        if ( (pid = Fork ( ) ) == 0) {
                Close (listenfd); /* child closes listening socket */
                doit (connfd); /* process the request */
                Close (connfd); /* done with this client */
                exit (0); /* child terminates */
        }
        Close (connfd); /* parent closes connected socket */
}
```

# Server/Client Connection Status (1/2)



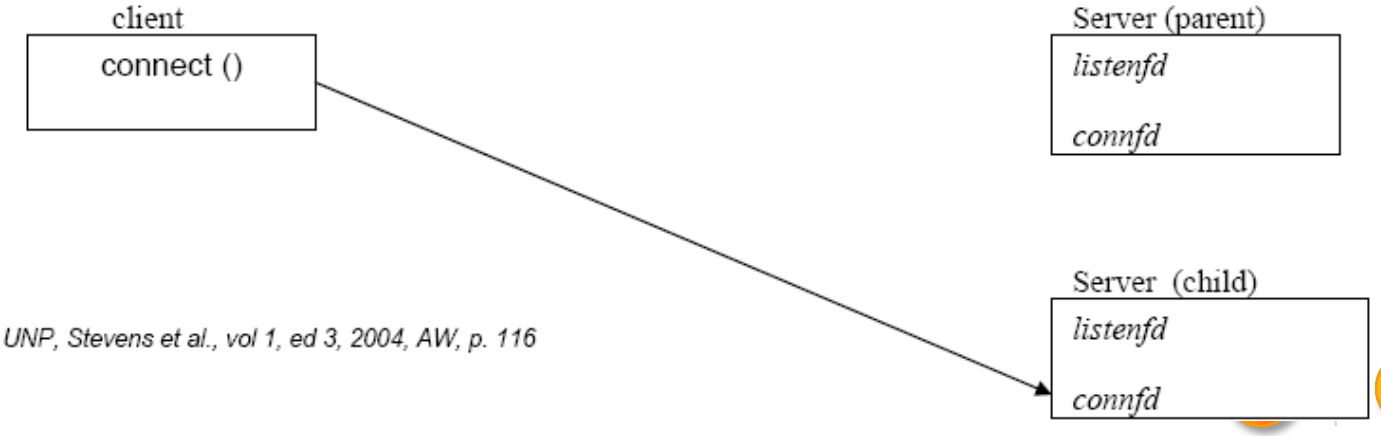

*Ref: UNP, Stevens et al., vol 1, ed 3, 2004, AW, p. 115*

# Server/Client Connection Status (2/2)




a) parent/child after fork()

b) parent/child after closing appropriate sockets




*Ref: UNP, Stevens et al., vol 1, ed 3, 2004, AW, p. 116*

# ASSIGNMENT # 3

1. Write a note on socket Function
2. Write a note on connect Function
3. Write a note on bind Function
4. Write a note on listen Function
5. Write a note on accept Function
6. What is purpose of fork () and exec ()
7. Describe close function
8. Describe getsockname and getpeername Functions
9. When Hard error and soft error occurs
10. What is diff. between complete connection queue and incomplete connection queue.
11. Dead line 2nd sept 2013