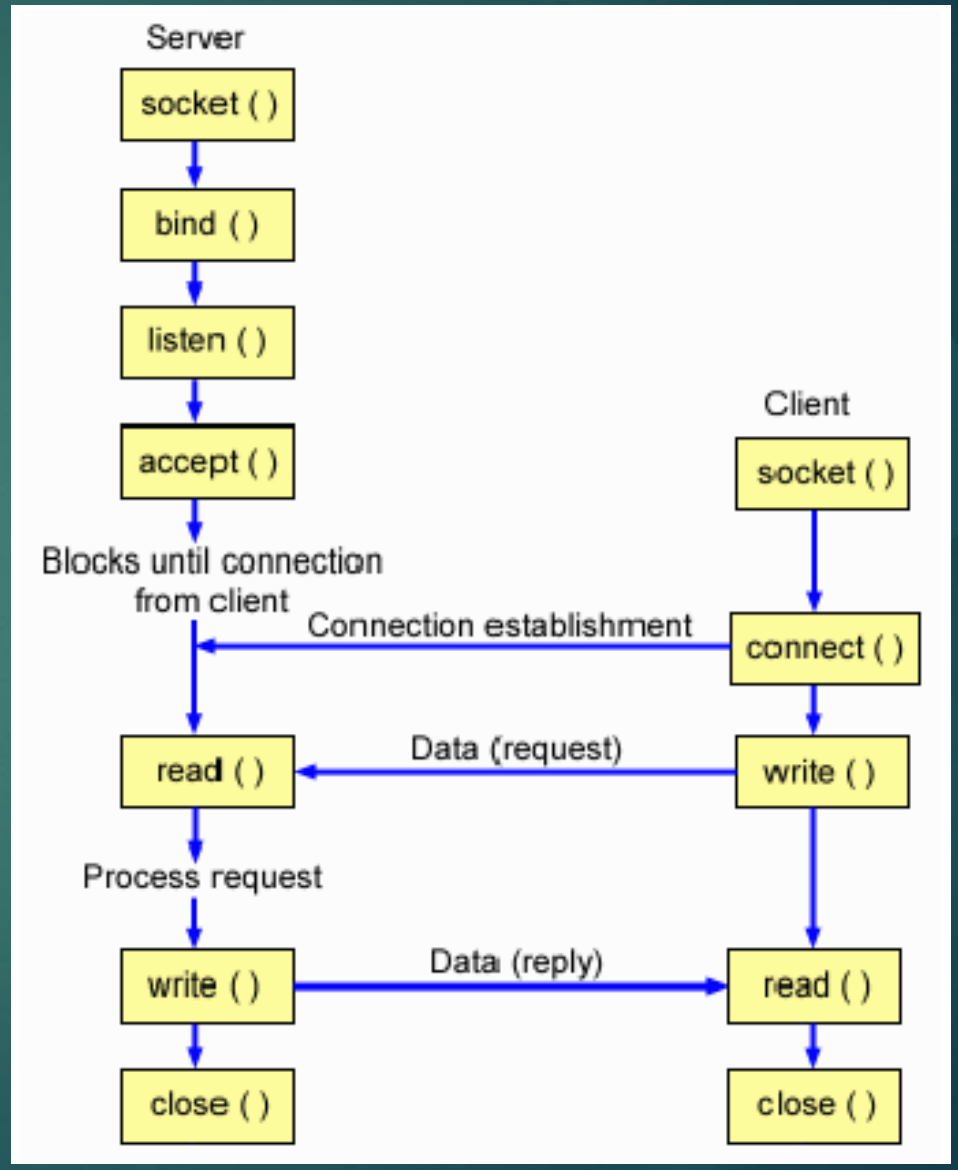
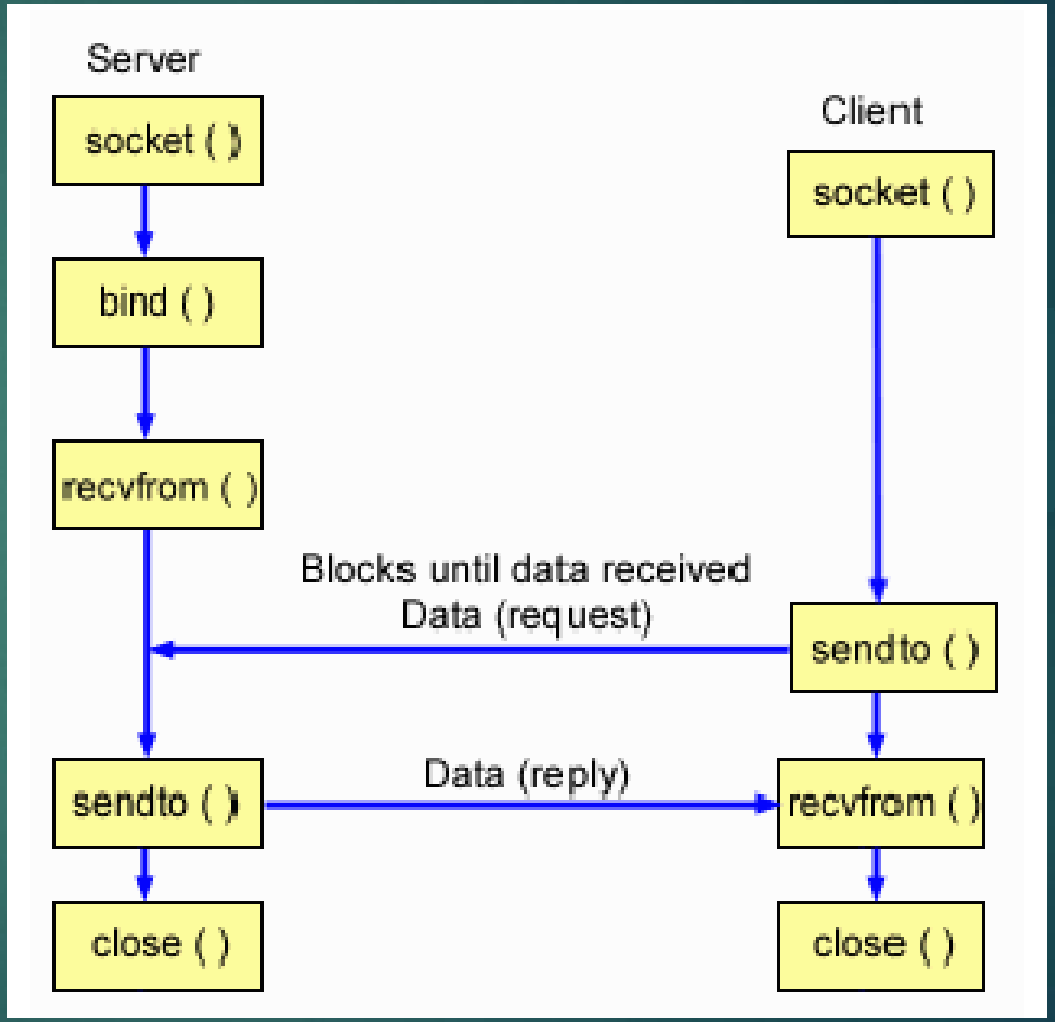
Elementary TCP Sockets

LECTURE 2

Connection Oriented Socket



Connectionless Socket



socket() – 1/2

- To perform I/O, first thing a process must do is call the socket function, specifying type of communication protocol desired

- Syntax

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

- Returns a small non negative integer value that will be the socket descriptor

socket() – 2/2

- family :- protocol family
  - AF_INET :- IPv4 protocols
  - AF_INET6 :- IPv6 protocols
- type :- one of the constants
  - SOCK_STREAM :- stream socket
  - SOCK_DGRAM :- datagram socket
- protocol :- set to 0 to choose respected socket type

bind() – 1/2

- Assigns a local protocol address to a socket
- Syntax

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

- Returns 0 if OK, -1 on error

bind() – 2/2

- sockfd :- socket descriptor returned by socket()
- myaddr :- pointer to struct sockaddr that contains information about IP address and port
- addrlen :- size of address structure i.e. sizeof(struct sockaddr)

connect() – 1/4

- Used by TCP to establish a connection with a TCP server
- Syntax

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

- Returns 0 if successful, else -1 on error

connect() – 2/4

- sockfd :- socket descriptor returned by socket function
- servaddr :- pointer to struct sockaddr that contains information on destination IP address and port
- addrlen :- pointers to socket address structure i.e. sizeof (struct sockaddr)

connect() – 3/4

- For TCP socket, connect function initiates TCP's 3-way handshake
- Function returns only when connection is established or error occurs
- Different error returns possible
  - **ETIMEOUT** - If a client TCP receives no response to its SYN segment
  - **ECONNREFUSED** - If server's response to client SYN's is an RST (reset – sent by TCP when something is wrong), indicates that no process is waiting for connections on the server host at the port specified.

connect() – 4/4

- Three conditions that generate RST
  - SYN arrives for a port that has no listening server
  - When TCP wants to abort an existing connection
  - When TCP receives a segment for a connection that does not exist

- **ENETUNREACH** or **EHOSTUNREACH** - If client's SYN elicits an ICMP destination unreachable from some intermediate router, called soft error.

listen() – 1/3

- Waits for incoming connections
- Syntax

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

- Returns -1 on error

listen() – 2/3

- sockfd :- descriptor of socket
- backlog :- no of connections allowed on incoming queue
- Need to call bind() before listen()

listen() – 3/3

- Called only by a TCP server and performs two actions
  1. Listen function converts an unconnected socket into a **passive** socket, indicating that kernel should accept incoming connection requests directed to this socket. In TCP, call to listen moves the socket from CLOSED state to LISTEN state
  2. Second argument (backlog) specifies number of maximum number of connections that kernel should queue for this socket

- **This function is called after both socket and bind functions and before accept function**

accept() – 1/5

- Gets the pending connection on the port you are listening
- Syntax

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

- Returns nonnegative descriptor (new socket file descriptor) if OK; else -1 on error

accept() – 2/5

- sockfd :- listening socket descriptor
- cliaddr :- info about incoming connection, pointer to local struct sockaddr_in
- addrlen :- size of structure

accept() – 3/5

- Called by a TCP server to return the next completed connection from the front of the completed connection queue
- If completed connection queue is empty, the process is put to sleep
- addrlen is a value-result argument before the call
- Set the integer value pointed to by *addrlen to the size of the socket address structure pointed to by cliaddr
- On return this integer value contains the actual no. of bytes stored by the kernel in the socket address structure

accept() – 4/5

- If accept is successful, its return value is a brand new descriptor that was automatically created by the kernel
- The new descriptor refers to the TCP connection with the client
- When discussing accept we call the first argument to accept the **listening socket** (descriptor created by socket and then used as the first argument to both bind and listen), and we call return value from accept the **connected socket**

accept() – 5/5

- A given server normally creates one listening socket, which then exists for lifetime of server
- Kernel then creates one connected socket for each client connection that is accepted
- When server is finished serving a given client, the connected socket is closed

close() – 1/2

- To close a socket and terminate a TCP connection

- Syntax

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

- Returns 0 if OK; else -1 on error

close() – 2/2

- Default action of close with a TCP socket is to mark the socket as closed and return to the process immediately
- Socket descriptor is no longer usable by the process
- It cannot be used as an argument to read or write

fork() – 1/3

- To create new process
- Syntax

```
#include <unistd.h>
pid_t fork(void);
```

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

fork() – 2/3

- **fork is called once but it returns twice**
  - Returns once in calling process (parent) with a return value that is process ID of newly created process (child)
  - Returns once in the child, with return value 0
- Hence *return value tells the process whether it is the parent or the child*

fork() – 3/3

- fork returns 0 in the child, instead of parent's process ID because a child has only one parent and it can obtain the parent's process ID by calling *getppid()*
- A parent can have any number of children and there is no way to obtain the process IDs of its children
- If parent wants to keep track of process IDs of all its children, it must record the return values from fork
- All descriptors open in parent before the call to fork are shared with the child after fork returns

Sample program


```
//filename: forktest.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
main()
{
   pid_t parent_pid, child_pid, my_pid;
   parent_pid = getpid();
   printf("Process %d about to fork", parent_pid);
   child_pid = fork();
   my_pid = getpid();
   if (child_pid == 0) //child
      printf("I am %d the child\n", my_pid);
```

```
   else if (child_pid>0) //parent
      printf (“I am %d the parent of %d\n”,my_pid, parent_pid);
   else  //error
      printf(“I am %d. fork failed with %d\n”,my_pid, child_pid);
}
```

Sample o/p:

```
Process 2857 about to fork
I am 2858 the child
Process 2857 about to fork
I am 2857 the parent of 2858
```

Concurrent Servers – 1/2

- Iterative server
  - Simple server
  - One server one client
  - Eg. Daytime server
- When a client request take longer to service, do not want to tie up single server with one client
- Handle multiple clients at the same time – concurrent server

Concurrent Servers – 2/2

- Simplest way to write a concurrent server is to **fork a child process to handle each client**
- When connection is established, accept() returns, server calls fork(), and child process services the client (on connfd i.e. connected socket) and parent process waits for another connection (on listenfd i.e. listening socket)
- Parent closes the connected socket since child handles new client

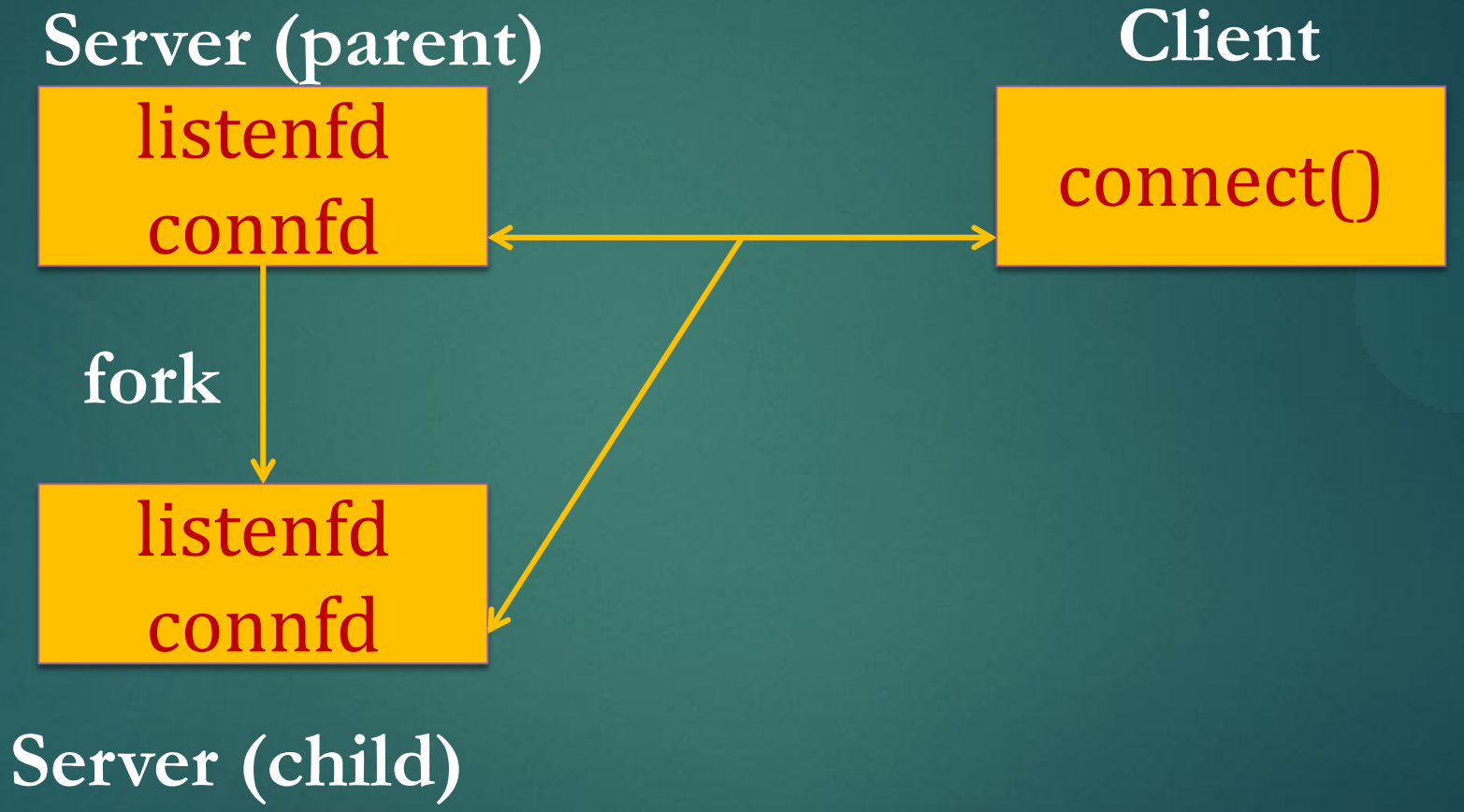
Diagrammatic representation – 1/4

(a) Before call to accept()

Diagrammatic representation – 2/4

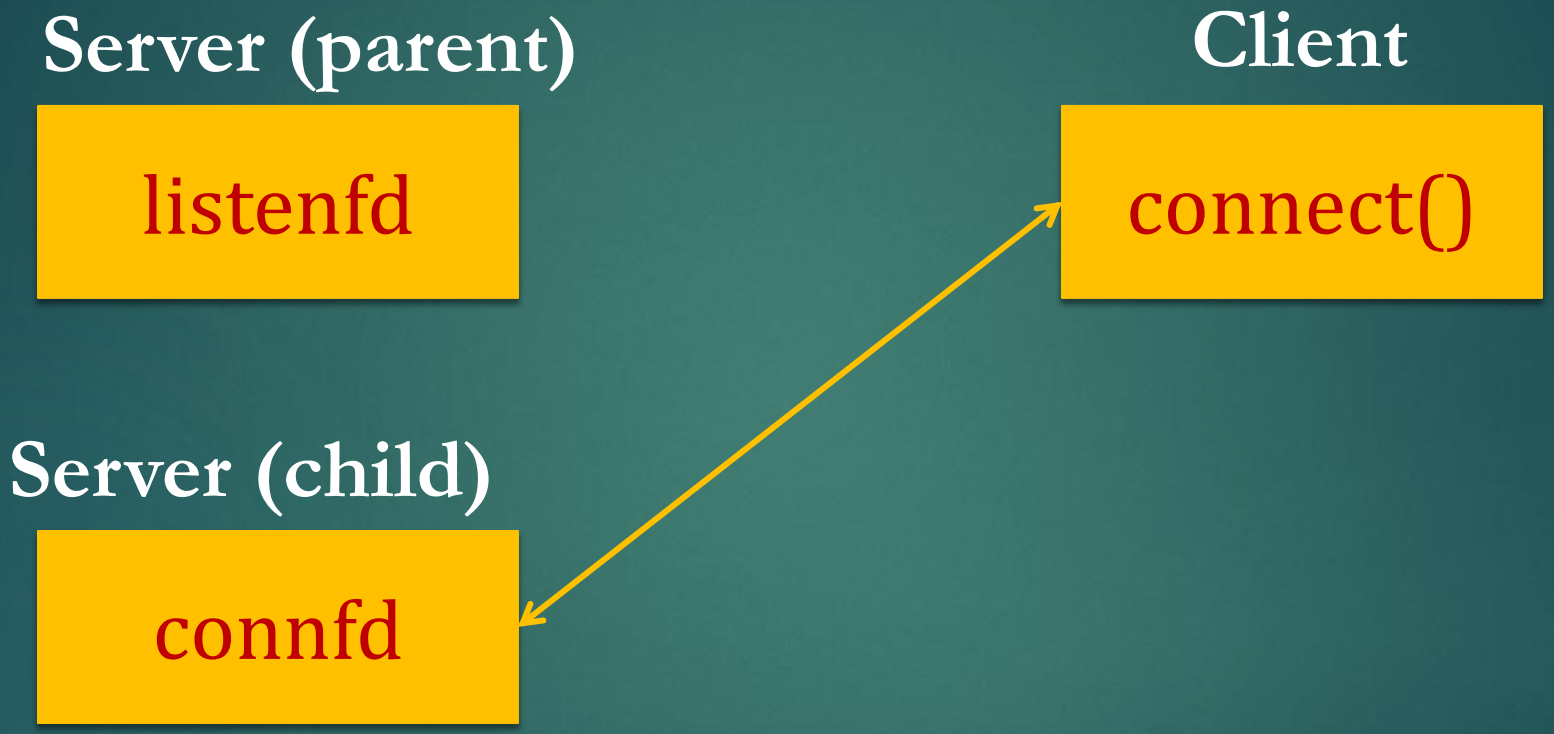
(b) After return from accept()

Diagrammatic representation – 3/4



(c) After fork() returns

Diagrammatic representation – 4/4



(d) After parent and child close appropriate sockets

Echo server client program