Introduction to Socket Programming Part-I

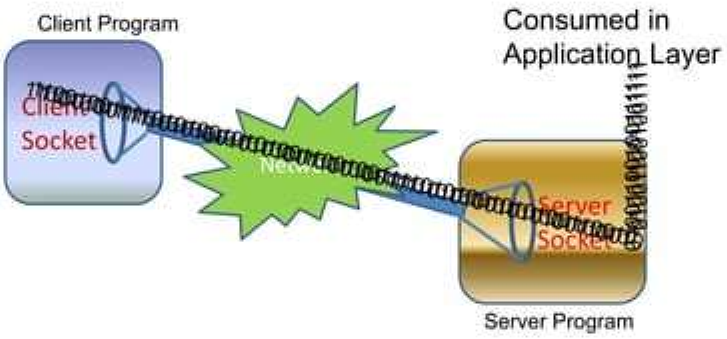
Ajit K Nayak, Ph.D.

Sockets!!!

- Is it like this ???

- Electrical Sockets, used to transfer Electrical power

Sockets in Network Programs



Network Sockets

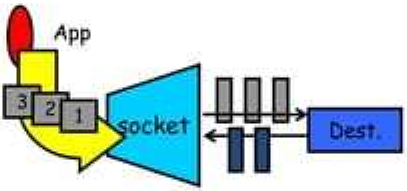
- The *socket* is the method for accomplishing communication among processes in a network.
- An interface between application and the network
- i.e. The application can send/receive data to/from the network via sockets
- In UNIX terminology you can say that it is also a file,
  - As a file is understood by a *file descriptor*
  - *Socket* is also recognized by a *socket descriptor* (a 16 bit integer)
- Sockets are always created in pairs and they are used to communicate with other sockets
- There are many kinds of sockets like Internet sockets, UNIX sockets, . . .

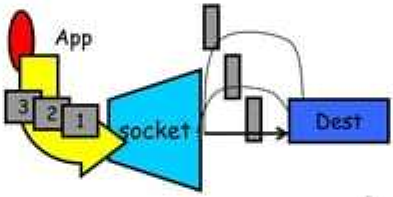
Socket API

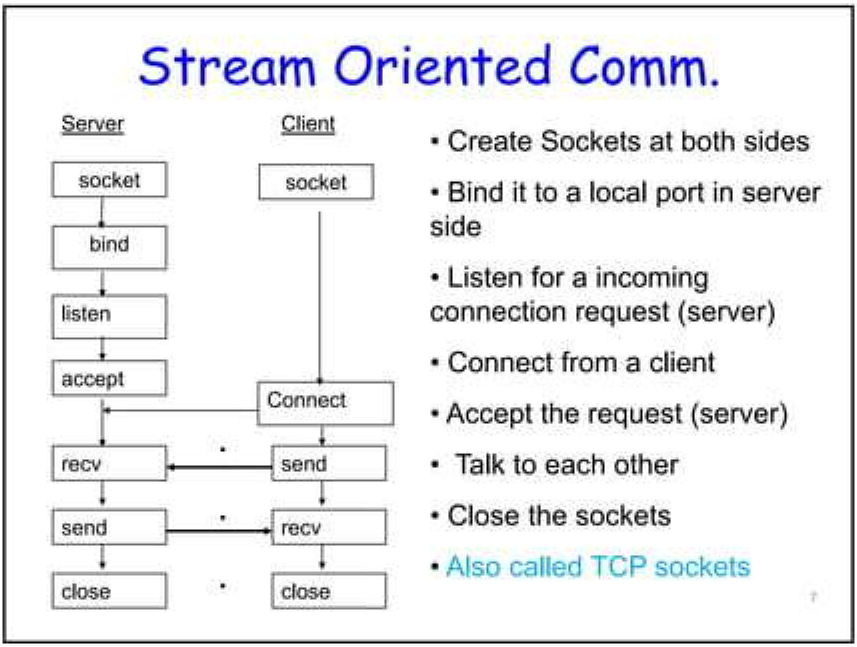
- We will use Socket API with C programming in Linux environment
- Socket API contains many data structures, functions and system calls which will help us to write good and efficient network programs
- Requirement is any PC loaded with Linux, C and Socket API
- It is also possible to program in other environments like windows and other UNIX systems!!! You need to explore

Essential types of Sockets

- SOCK_STREAM
  - TCP
  - reliable delivery
  - in-order guaranteed
  - connection-oriented
  - bidirectional
- SOCK_DGRAM
  - UDP
  - unreliable delivery
  - no order guarantee
  - no notion of "connection"
  - can send or receive







Stream Oriented Comm.

Server

socket

bind

listen

accept

recv

send

close

Client

socket

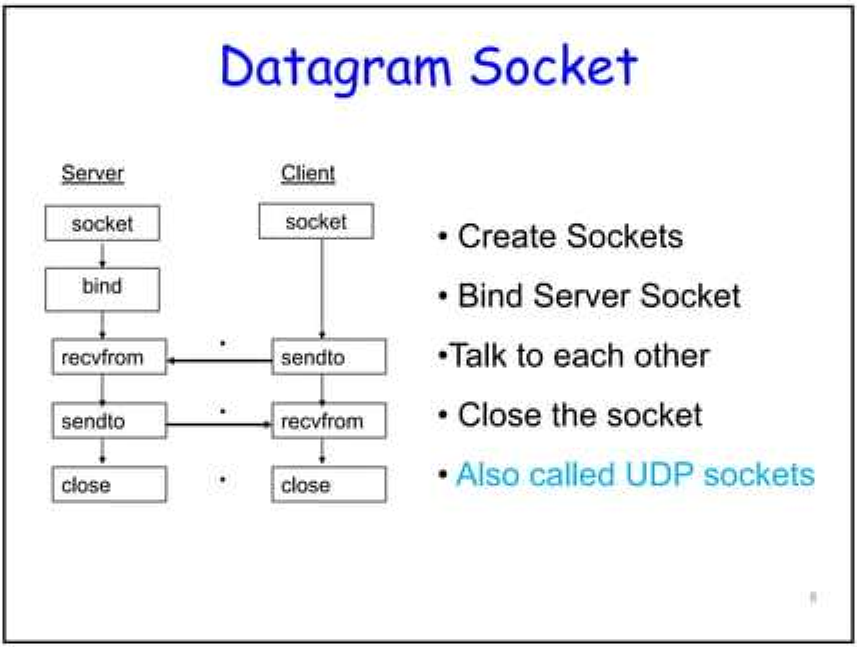
Connect

send

recv

close

- Create Sockets at both sides
- Bind it to a local port in server side
- Listen for a incoming connection request (server)
- Connect from a client
- Accept the request (server)
- Talk to each other
- Close the sockets
- Also called TCP sockets



Datagram Socket

Server

socket

bind

recvfrom

sendto

close

Client

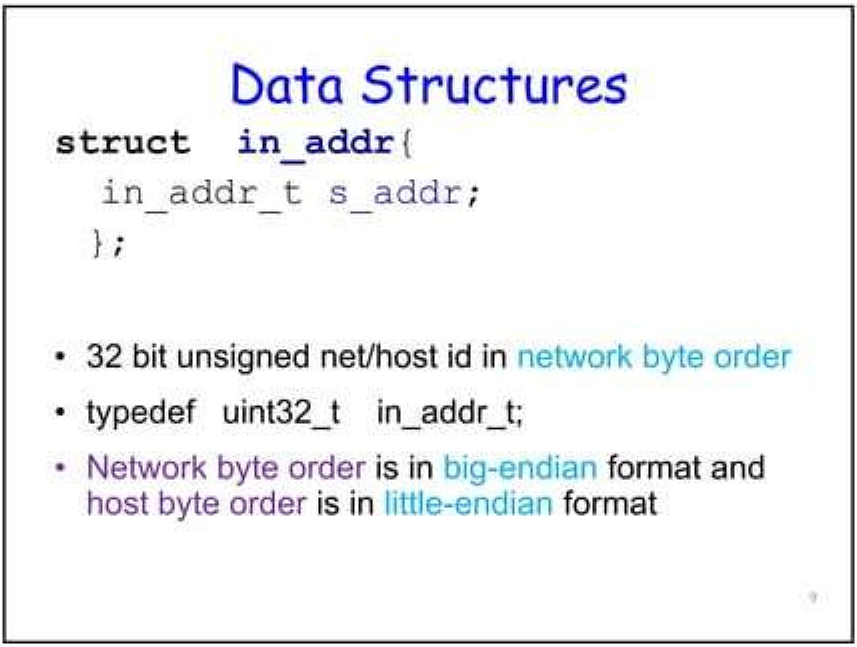
socket

sendto

recvfrom

close

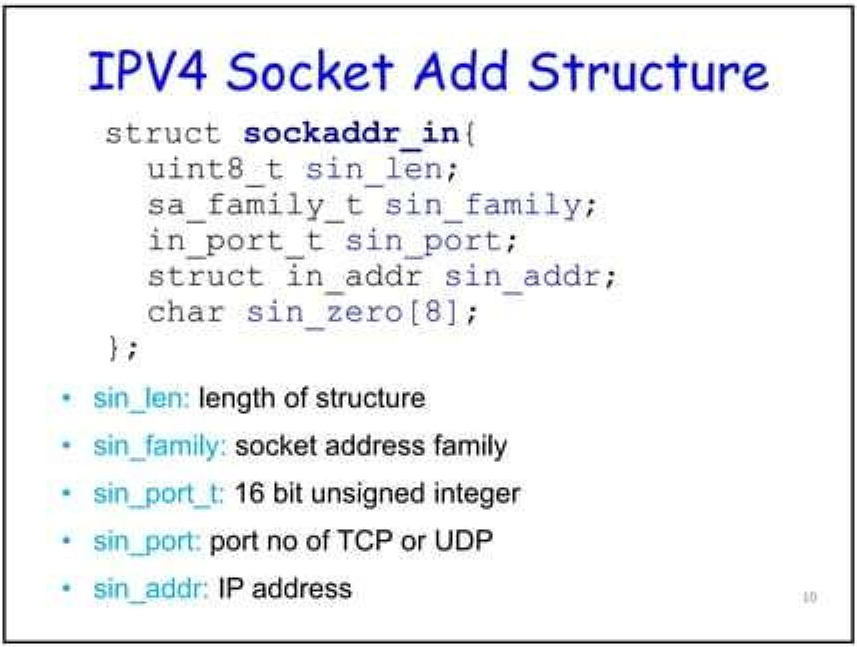
- Create Sockets
- Bind Server Socket
- Talk to each other
- Close the socket
- Also called UDP sockets



Data Structures

```
struct  in_addr{
   in_addr_t s_addr;
  };
```

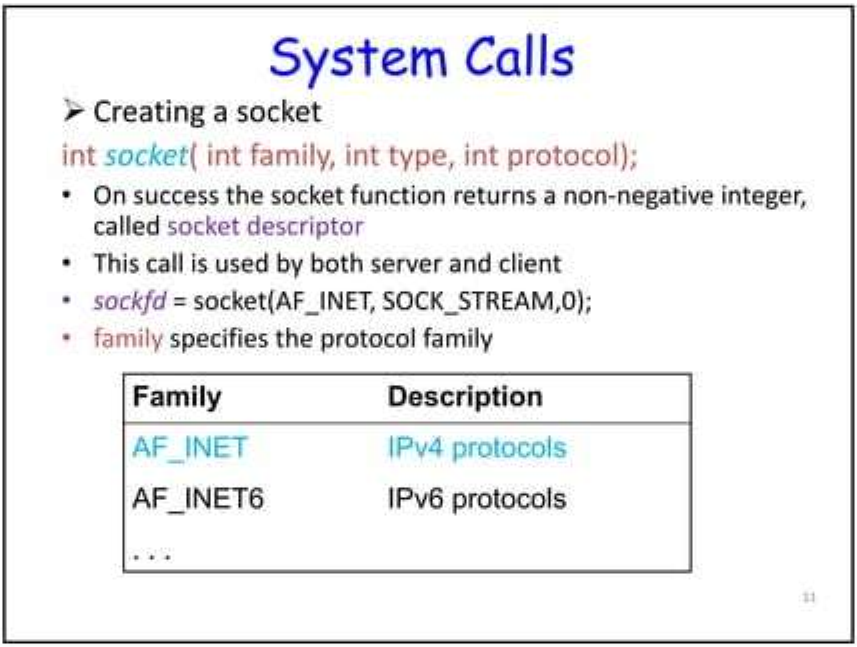
- 32 bit unsigned net/host id in network byte order
- typedef uint32_t in_addr_t;
- Network byte order is in big-endian format and host byte order is in little-endian format



IPV4 Socket Add Structure

```
struct sockaddr_in{
  uint8_t sin_len;
  sa_family_t sin_family;
  in_port_t sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
};
```

- sin_len: length of structure
- sin_family: socket address family
- sin_port_t: 16 bit unsigned integer
- sin_port: port no of TCP or UDP
- sin_addr: IP address



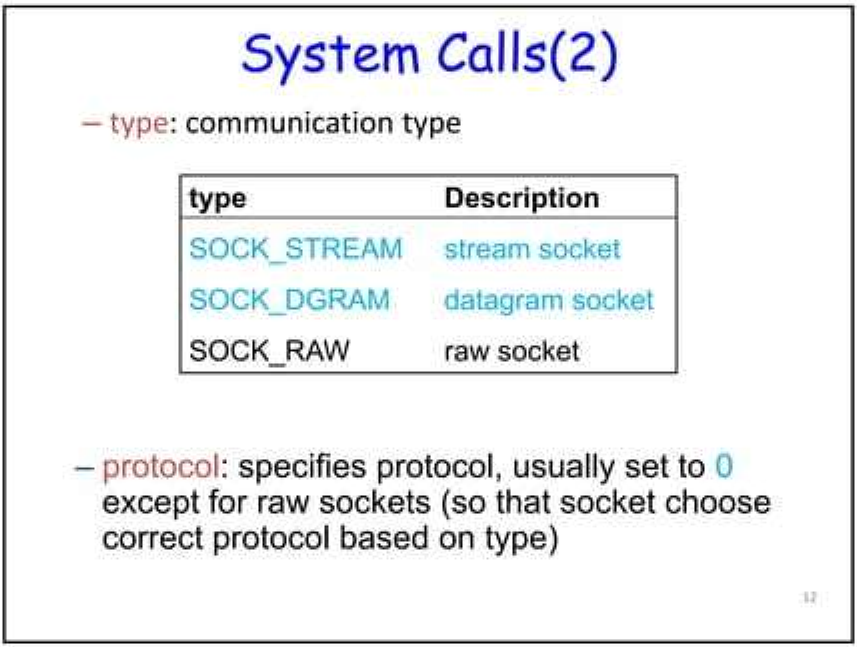
System Calls

➢ Creating a socket

int *socket*(int family, int type, int protocol);

- On success the socket function returns a non-negative integer, called socket descriptor
- This call is used by both server and client
- *sockfd* = socket(AF_INET, SOCK_STREAM,0);
- family specifies the protocol family

Family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
. . .	

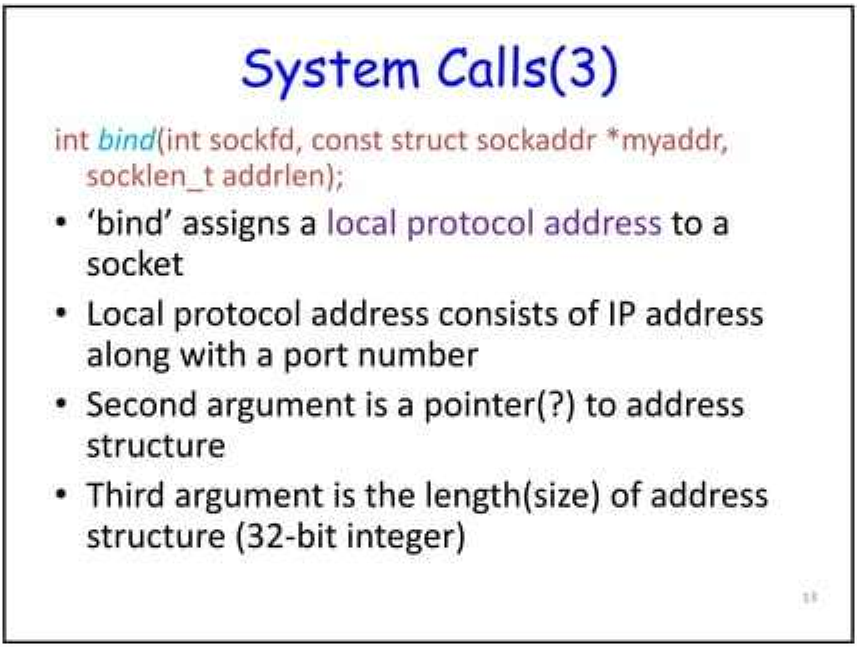


System Calls(2)

– type: communication type

type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_RAW	raw socket

– protocol: specifies protocol, usually set to 0 except for raw sockets (so that socket choose correct protocol based on type)

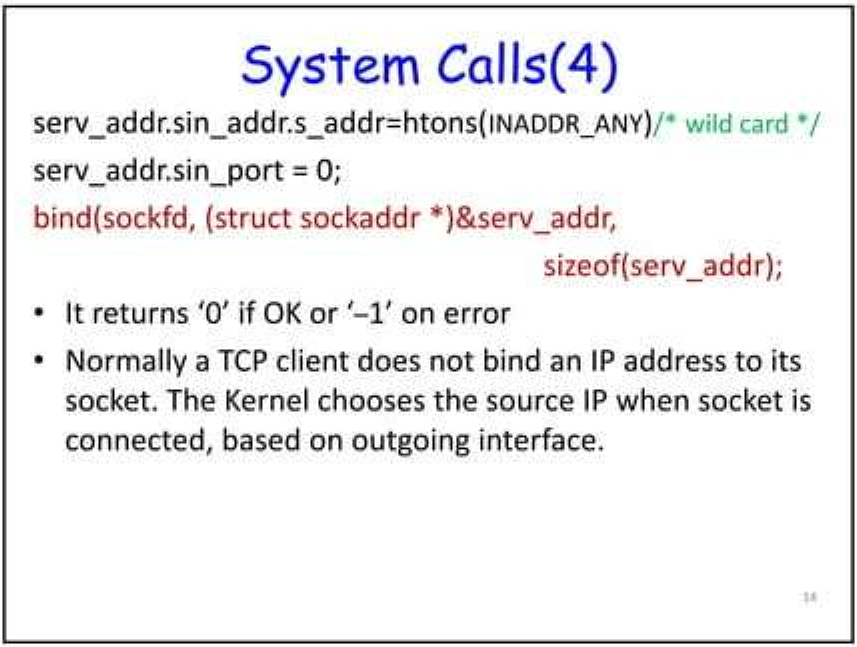


System Calls(3)

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

- 'bind' assigns a local protocol address to a socket
- Local protocol address consists of IP address along with a port number
- Second argument is a pointer(?) to address structure
- Third argument is the length(size) of address structure (32-bit integer)

13

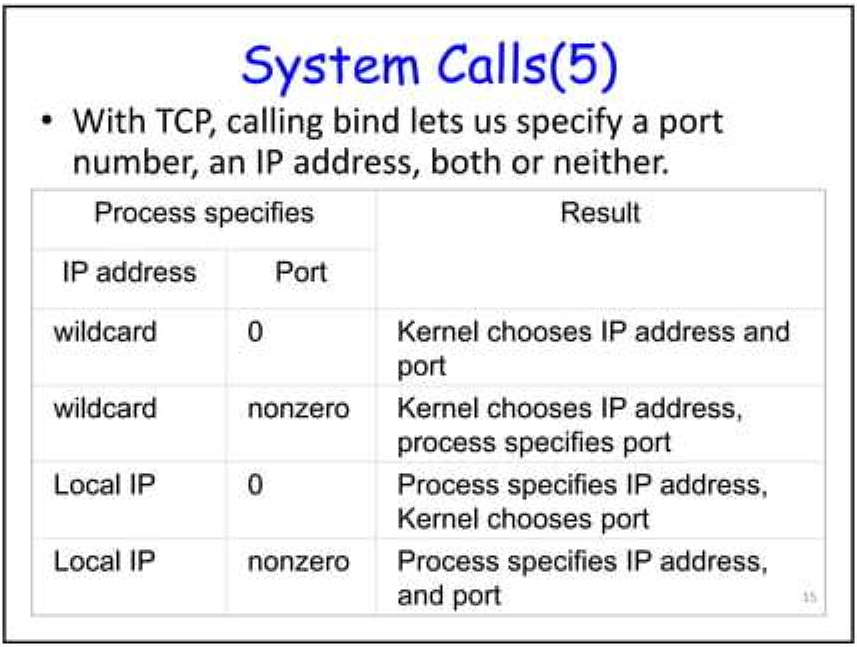


System Calls(4)

```
serv_addr.sin_addr.s_addr=htons(INADDR_ANY)/* wild card */
serv_addr.sin_port = 0;
bind(sockfd, (struct sockaddr *)&serv_addr,
                                        sizeof(serv_addr);
```

- It returns '0' if OK or '–1' on error
- Normally a TCP client does not bind an IP address to its socket. The Kernel chooses the source IP when socket is connected, based on outgoing interface.

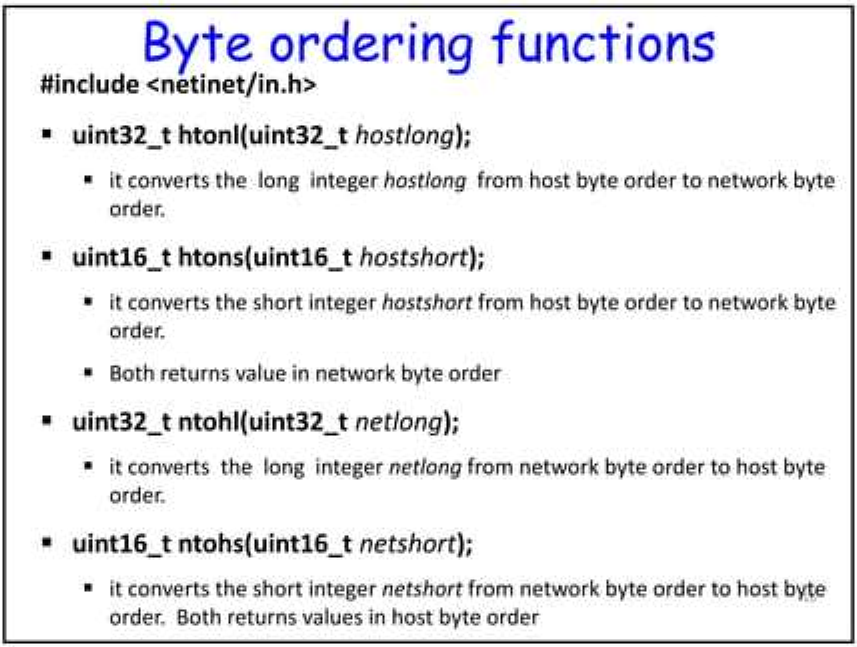
14



System Calls(5)

- With TCP, calling bind lets us specify a port number, an IP address, both or neither.

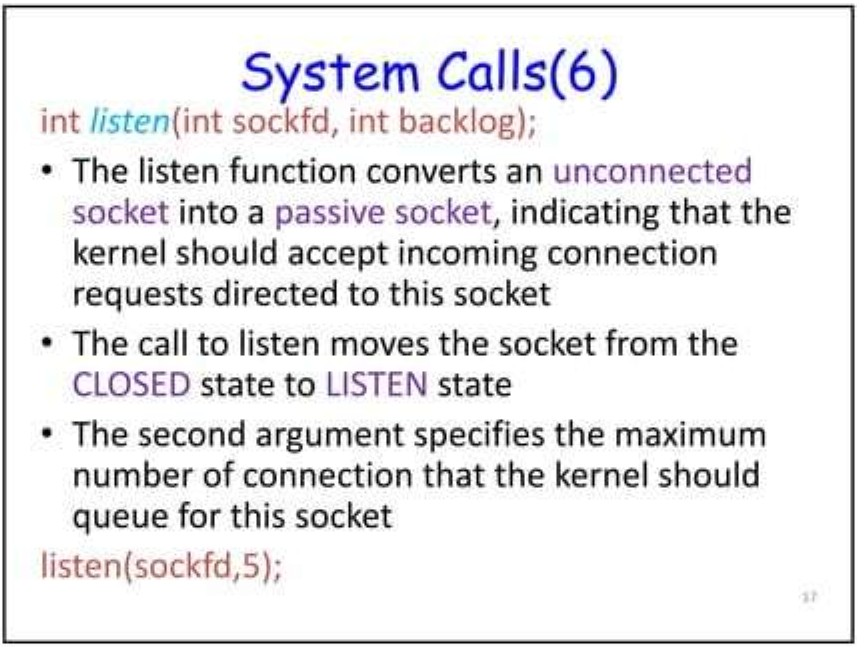
Process specifies		Result
IP address	Port	
wildcard	0	Kernel chooses IP address and port
wildcard	nonzero	Kernel chooses IP address, process specifies port
Local IP	0	Process specifies IP address, Kernel chooses port
Local IP	nonzero	Process specifies IP address, and port



Byte ordering functions

#include <netinet/in.h>

- **uint32_t htonl(uint32_t *hostlong*);**
  - it converts the long integer *hostlong* from host byte order to network byte order.
- **uint16_t htons(uint16_t *hostshort*);**
  - it converts the short integer *hostshort* from host byte order to network byte order.
  - Both returns value in network byte order
- **uint32_t ntohl(uint32_t *netlong*);**
  - it converts the long integer *netlong* from network byte order to host byte order.
- **uint16_t ntohs(uint16_t *netshort*);**
  - it converts the short integer *netshort* from network byte order to host byte order. Both returns values in host byte order



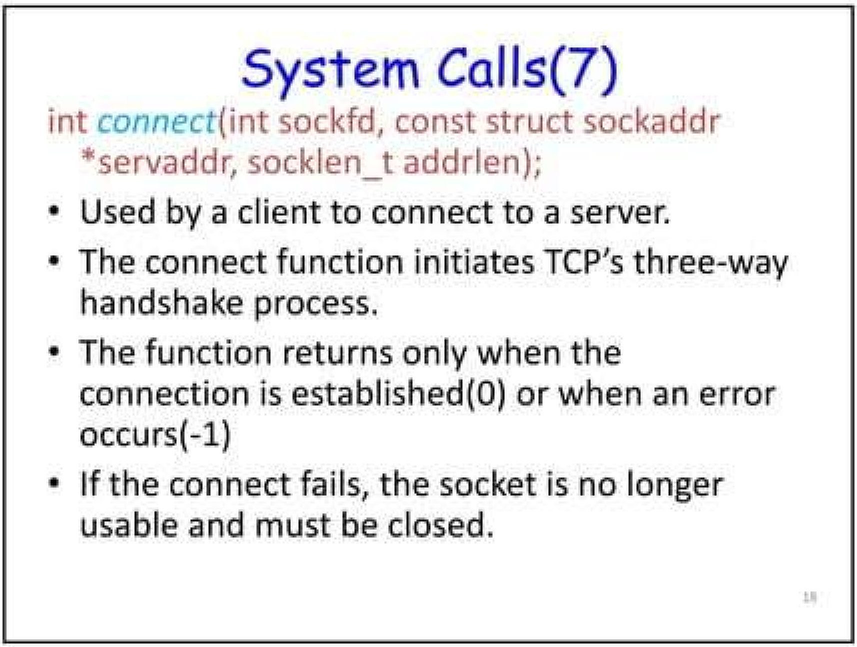
System Calls(6)

int *listen*(int sockfd, int backlog);

- The listen function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket
- The call to listen moves the socket from the CLOSED state to LISTEN state
- The second argument specifies the maximum number of connection that the kernel should queue for this socket

listen(sockfd,5);

17

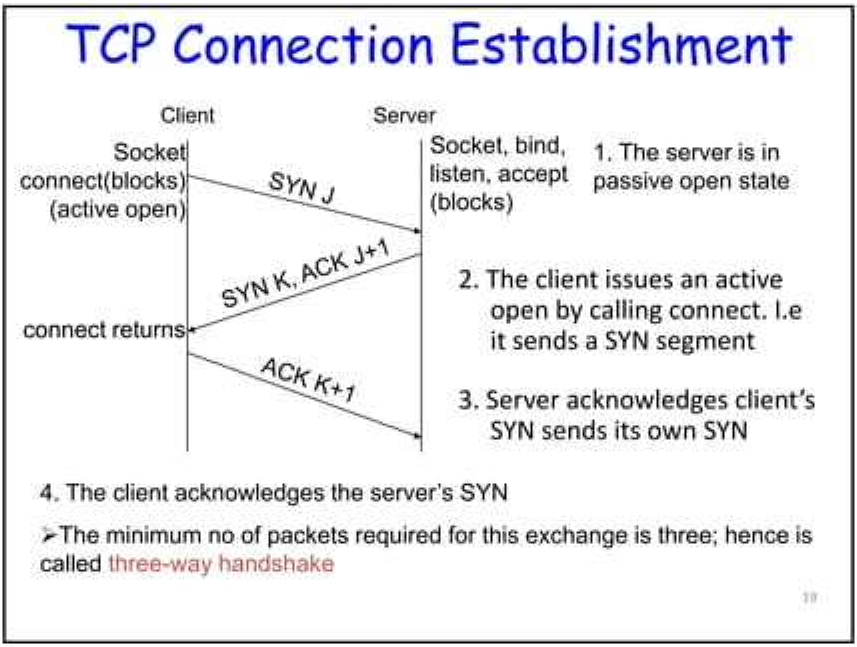


System Calls(7)

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

- Used by a client to connect to a server.
- The connect function initiates TCP's three-way handshake process.
- The function returns only when the connection is established(0) or when an error occurs(-1)
- If the connect fails, the socket is no longer usable and must be closed.

18



TCP Connection Establishment

1. The server is in passive open state

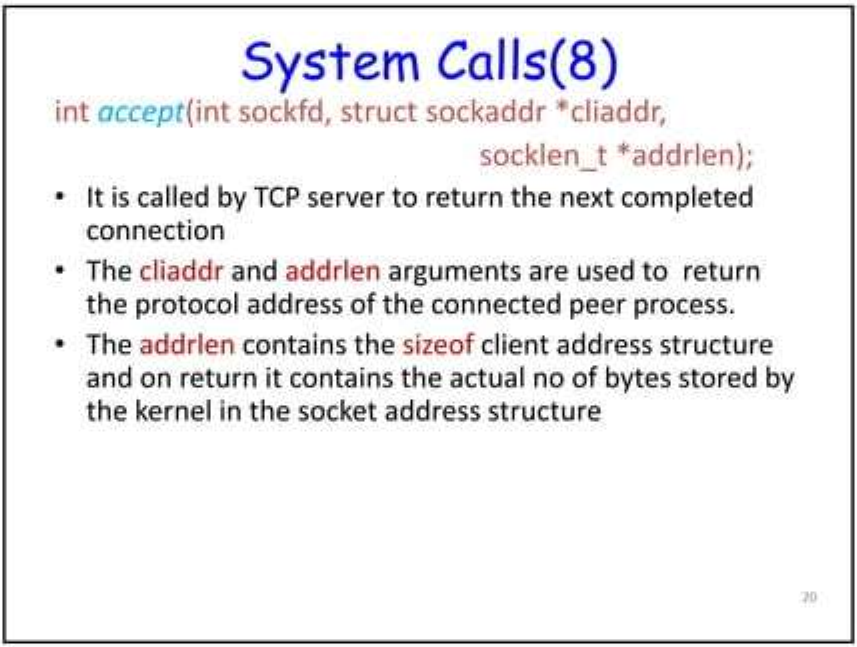
2. The client issues an active open by calling connect. I.e it sends a SYN segment

3. Server acknowledges client's SYN sends its own SYN

4. The client acknowledges the server's SYN

➢The minimum no of packets required for this exchange is three; hence is called three-way handshake

19

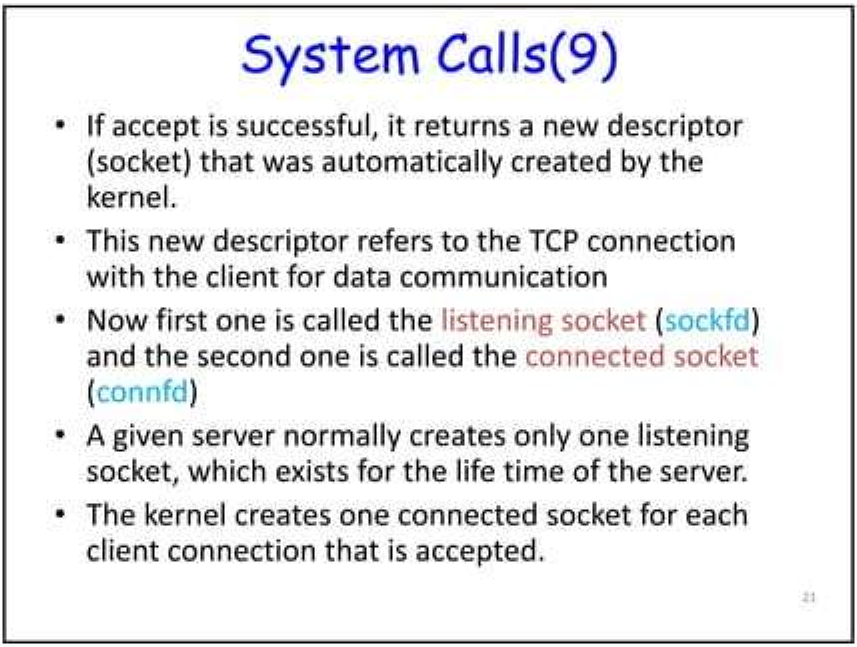


System Calls(8)

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

- It is called by TCP server to return the next completed connection
- The cliaddr and addrlen arguments are used to return the protocol address of the connected peer process.
- The addrlen contains the sizeof client address structure and on return it contains the actual no of bytes stored by the kernel in the socket address structure

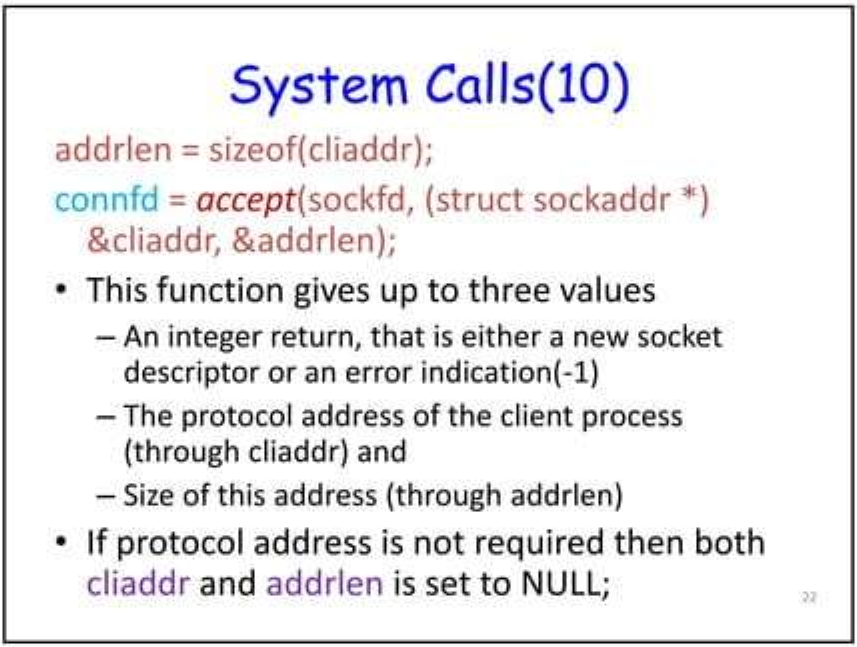
20



System Calls(9)

- If accept is successful, it returns a new descriptor (socket) that was automatically created by the kernel.
- This new descriptor refers to the TCP connection with the client for data communication
- Now first one is called the listening socket (sockfd) and the second one is called the connected socket (connfd)
- A given server normally creates only one listening socket, which exists for the life time of the server.
- The kernel creates one connected socket for each client connection that is accepted.

21

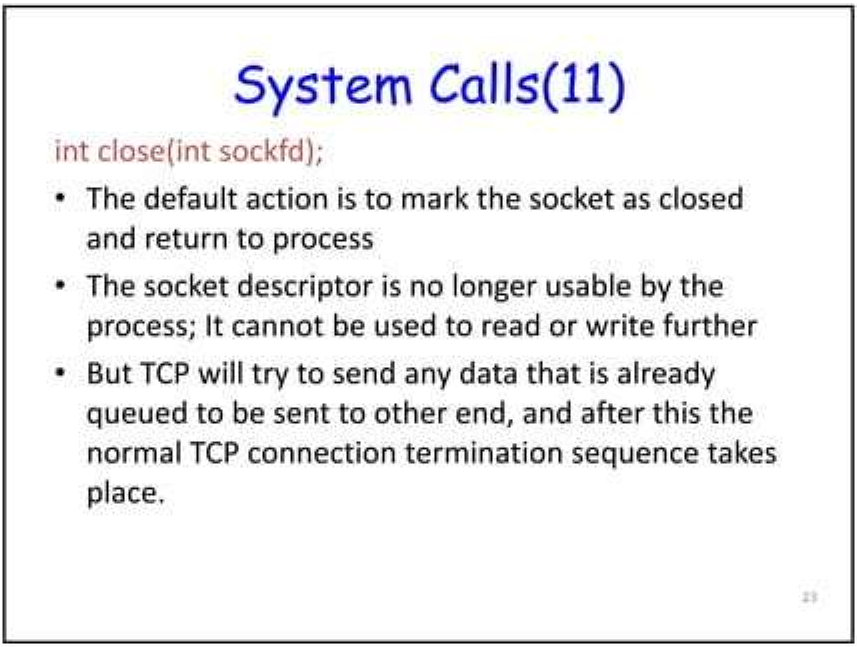


System Calls(10)

```
addrlen = sizeof(cliaddr);
connfd = accept(sockfd, (struct sockaddr *)
  &cliaddr, &addrlen);
```

- This function gives up to three values
  - An integer return, that is either a new socket descriptor or an error indication(-1)
  - The protocol address of the client process (through cliaddr) and
  - Size of this address (through addrlen)
- If protocol address is not required then both cliaddr and addrlen is set to NULL;

22

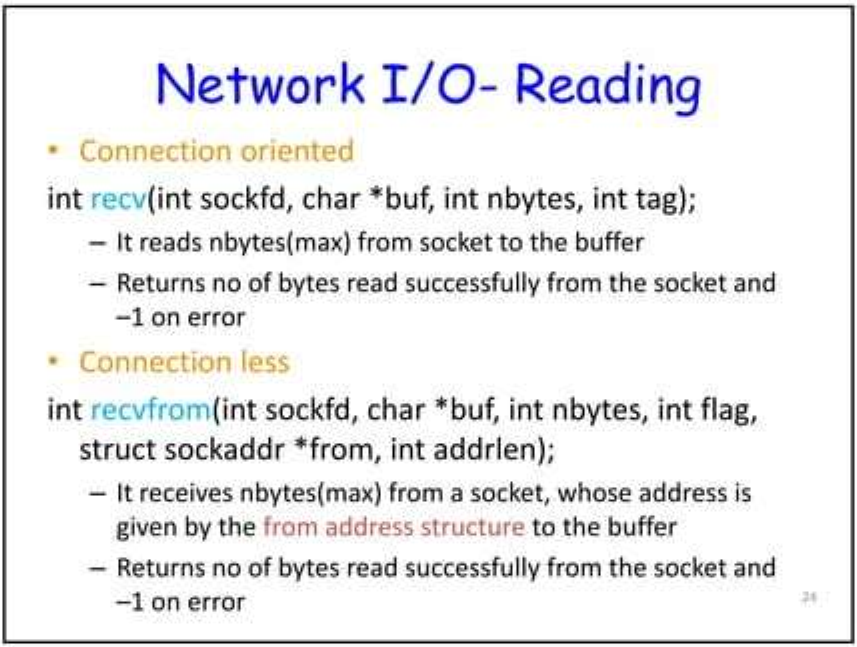


System Calls(11)

int close(int sockfd);

- The default action is to mark the socket as closed and return to process
- The socket descriptor is no longer usable by the process; It cannot be used to read or write further
- But TCP will try to send any data that is already queued to be sent to other end, and after this the normal TCP connection termination sequence takes place.

23



Network I/O- Reading

- Connection oriented

int recv(int sockfd, char *buf, int nbytes, int tag);

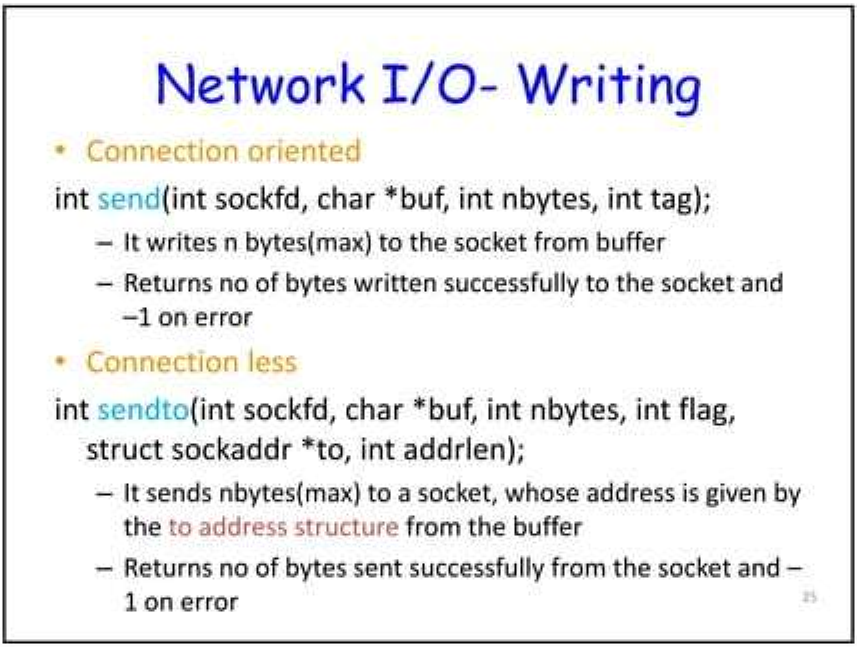
- It reads nbytes(max) from socket to the buffer
- Returns no of bytes read successfully from the socket and –1 on error

- Connection less

int recvfrom(int sockfd, char *buf, int nbytes, int flag, struct sockaddr *from, int addrlen);

- It receives nbytes(max) from a socket, whose address is given by the from address structure to the buffer
- Returns no of bytes read successfully from the socket and –1 on error

24



Network I/O- Writing

- Connection oriented

int send(int sockfd, char *buf, int nbytes, int tag);

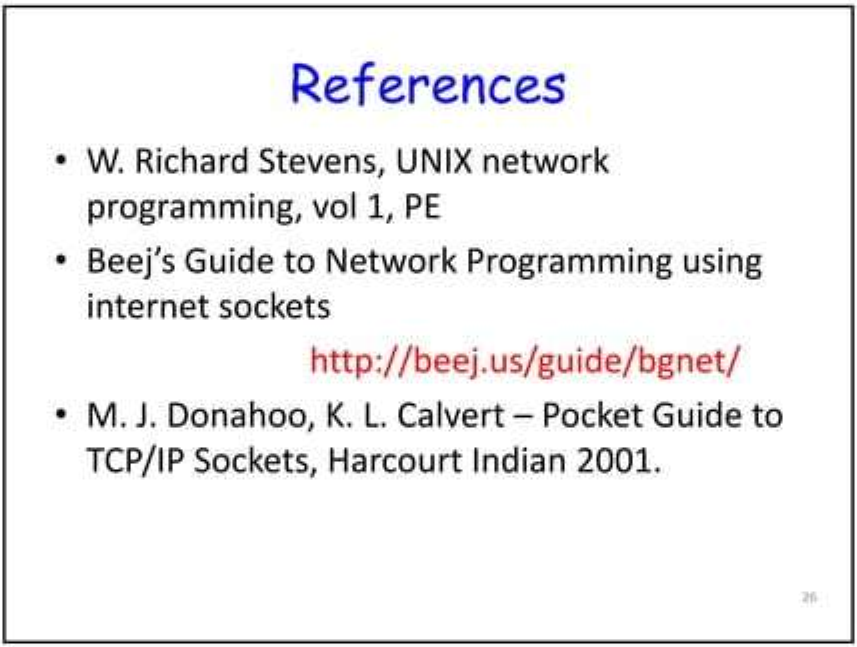
  - It writes n bytes(max) to the socket from buffer
  - Returns no of bytes written successfully to the socket and –1 on error

- Connection less

int sendto(int sockfd, char *buf, int nbytes, int flag, struct sockaddr *to, int addrlen);

  - It sends nbytes(max) to a socket, whose address is given by the to address structure from the buffer
  - Returns no of bytes sent successfully from the socket and –1 on error

25



References

- W. Richard Stevens, UNIX network programming, vol 1, PE
- Beej's Guide to Network Programming using internet sockets

  http://beej.us/guide/bgnet/
- M. J. Donahoo, K. L. Calvert – Pocket Guide to TCP/IP Sockets, Harcourt Indian 2001.

26

Thank You

27