pselect adds a sixth argument, a pointer to a signal mask. This allows the program to disable the delivery of certain signals, test some global variables that are set by the handlers for these now disabled signals & then call pselect telling it to reset the signal mask.

17) Why I/O & I/O multiplexing is needed in Socket programming? Explain various I/O models in Unix System?

→ I/O is needed in socket programming because it allows bi-direction communication between client & server.

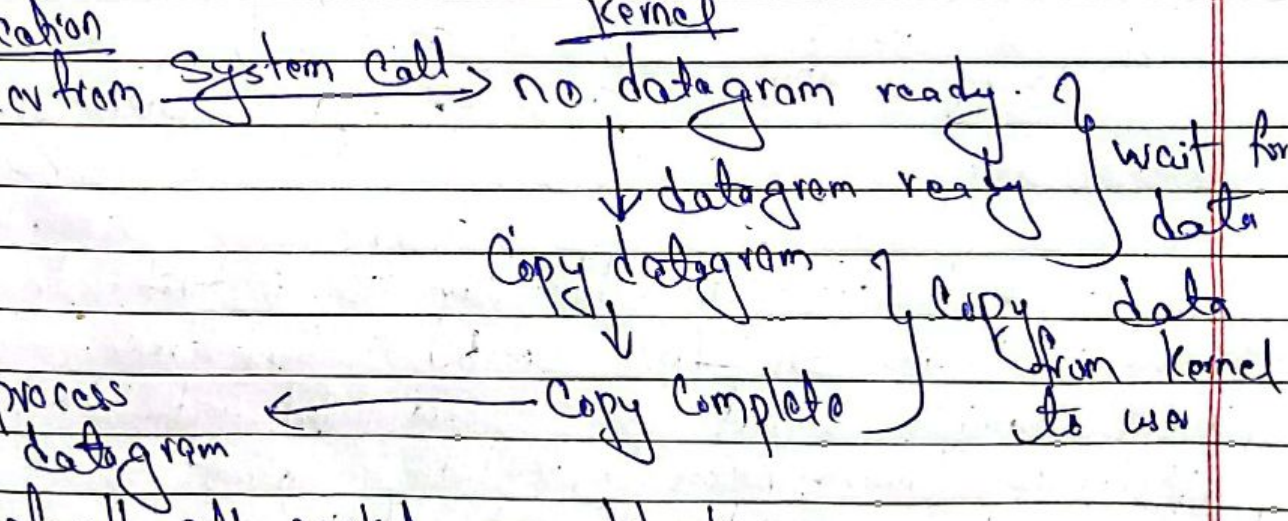
→ I/O multiplexing is needed in socket programming to tell the kernel that we want to be notified if one or more I/O conditions are ready (i.e. inputs is ready to be read or the descriptor is capable of taking more output). this capability is provided by the select and poll functions.

→ When a client is handling multiple descriptor. I/O multiplexing is used.

→ Various I/O models are :-

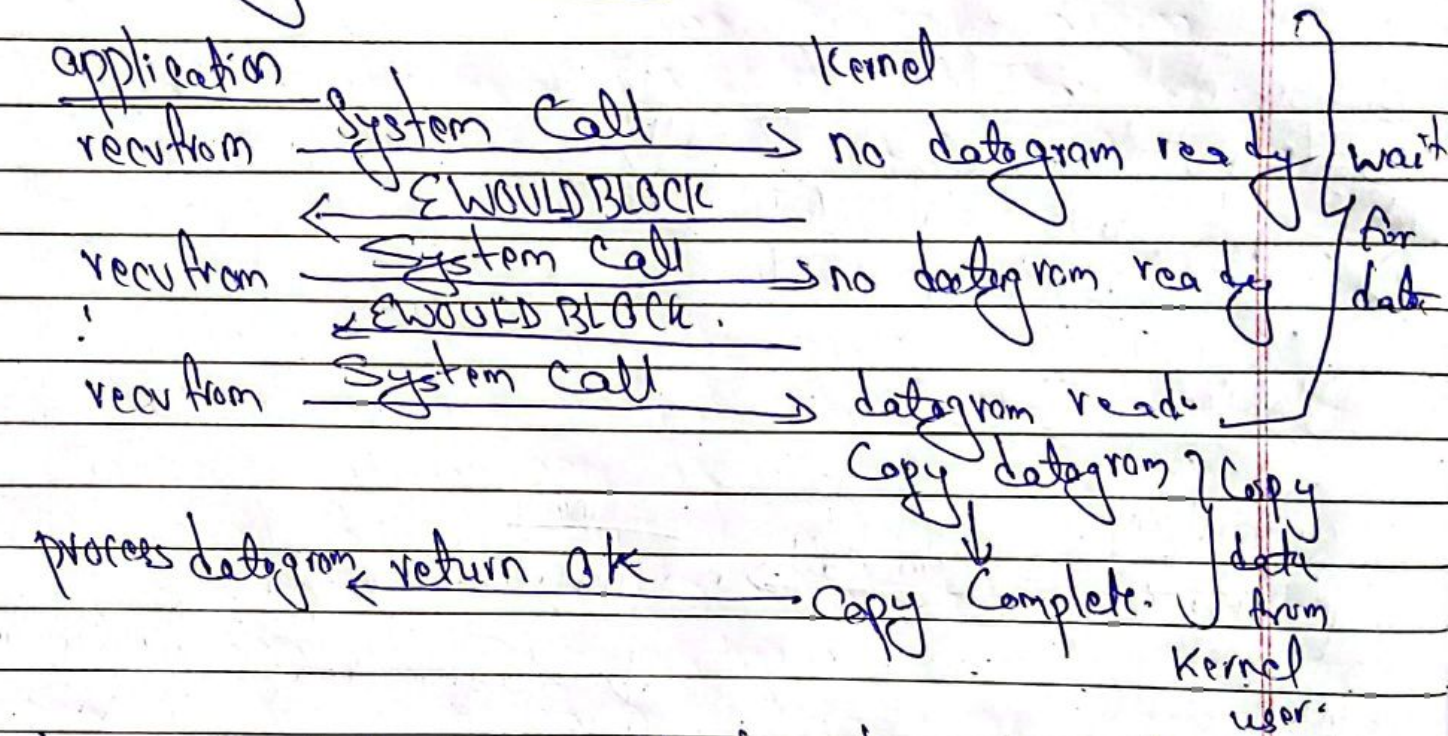
a) Blocking I/O
b) non-blocking I/O
c) I/O multiplexing (select & poll)
d) Signal driven I/O (SIGIO)
e) Asynchronous I/O

a) Blocking I/O



- By default, all socket are blocking.
- The process call recvfrom and the system call doesn't return until the datagram arrives and is copied into our application buffer or an error occurs.
- We say that our process is blocked the entire time from when it calls recvfrom until it returns.
- When recvfrom returns successfully our application process the datagram.

b) Non-Blocking I/O model



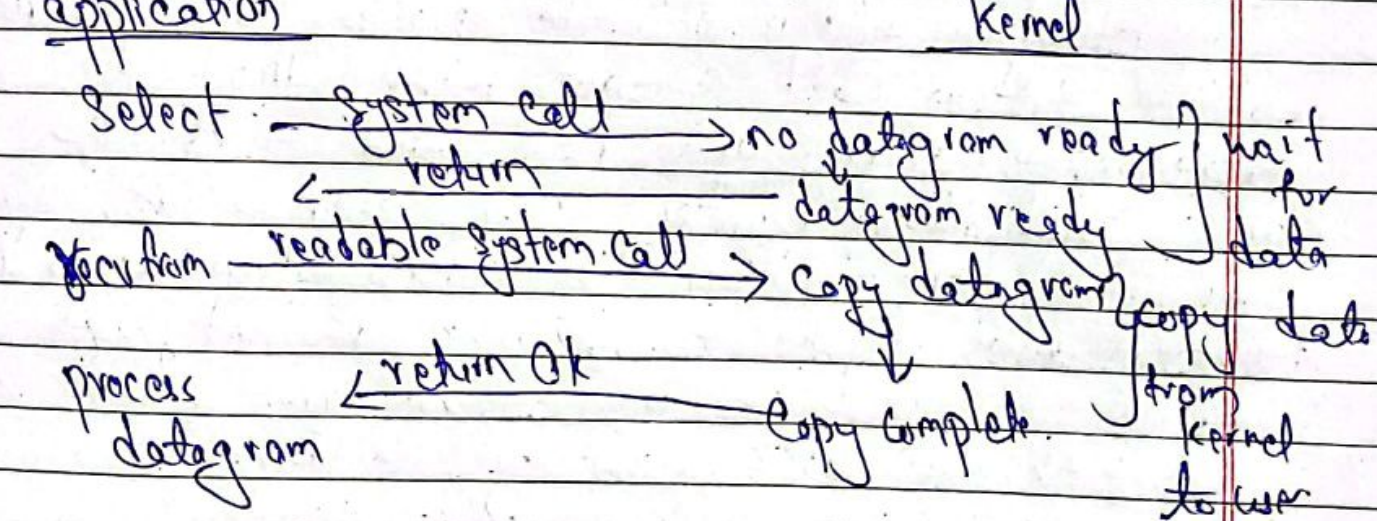
- When a socket is non-blocking it instruct the kernel as "when an I/O operation that the process requests cannot be completed without putting the process to sleep, do not put the process to sleep, but return an error instead.
- first three times that we call recvfrom there is no data to return, so the kernel immediately returns an error of EWOULDBLOCK instead.

→ fourth time we call 'recvfrom' a datagram is ready, it is copied into our application buffer and recvfrom returns successfully.

→ We then process data. When an application sits in a loop calling 'recvfrom' on a non-blocking descriptor like this, it is called polling.

→ the application is continually polling the kernel to see if some operation is ready. This is often a waste of CPU time.

③ I/O multiplexing model



- With I/O multiplexing we call 'select' & 'poll' and block in one of these two system calls, instead of blocking in the actual I/O system call
- We block in a call to select waiting for the datagram socket to be readable.
- When select returns that the socket is readable we then call recvfrom to copy the datagram into our application buffer.
- With select, we can wait for more than one descriptor to be ready.

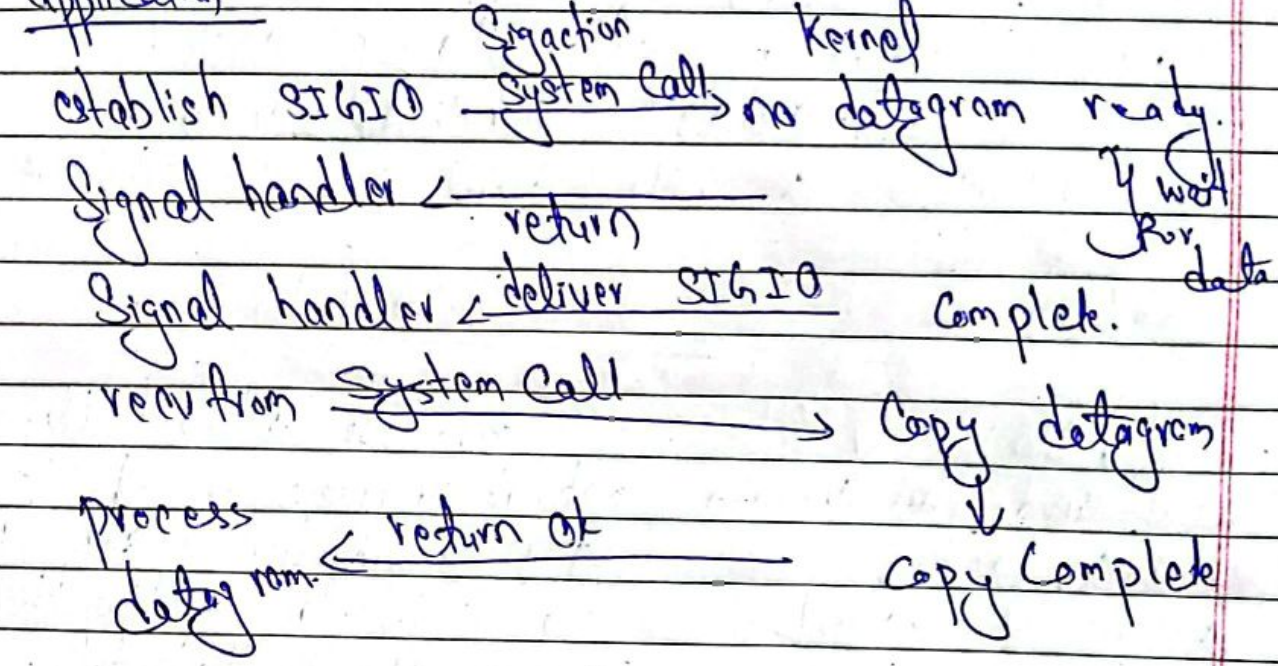
So, these are some of the I/O models in Unix System.

20) Differentiate blocking I/O model with non-blocking I/O model. Can we use signal driven I/O as asynchronous I/O? If yes, how? if no, why we cannot use Signal I/O as asynchronous I/O?

→ In blocking I/O model, the process call recvfrom and the system call doesnot return until the datagram arrives and is copied into our application buffer or an error occurs, but in non-blocking i/o model, when we call recvfrom and if the datagram is not ready it would return an error of EWOULDBLOCK i.e. it instruct the kernel as when an I/O operation that the process requests cannot be completed without putting the process to sleep, do not put the process to sleep, but return an error instead.

Blocking I/O	Non-blocking I/O
→ Initiate	→ Check,
|	Check
|	Check
| blocked.	Check
|	Check
|	↓ blocked
Complete	Complete.

→ Yes, we can use Signal driven I/O as asynchronous I/O



→ like Signal driven I/O, here also we enable the socket and install a signal handler using the Sigaction System. Call. The return from this system call is immediate and process continues; it is not blocked.

→ When datagram is Complete, the SIGIO signal is generated for our process.

→ After all the datagram is Complete, then only it will call the SIGIO signal which helps to minimize process block while Completing copy to the process datagram.

21) Differentiate fcntl() & ioctl() System call. Explain the meaning of following Socket option: SO_BROADCAST, SO_KEEPALIVE, SO_RCVTIMED, SO_REUSEADDR & SO_LINGER.

→

fcntl()	ioctl()
i) fcntl() stands for "file Control" and this function performs various descriptor Control operations.	i) Common use of ioctl() by network programs is to obtain information on all the hosts interface when the program starts the interface addresses. whether interface supports broadcasting the interface supports multicasting & Soon.
ii) int fcntl(int fd, int cmd, long arg);	ii) int ioctl(int fd, int request, ... /*void arg*/); -returns 0 if ok, -1 on error. -third argument is always a pointer, but the type of pointer depends on the request.
iii) to set socket for non-blocking I/O, we use F_SETFL, O_NDELAY BLOCK.	iii) to set socket for non-blocking I/O, we use FIONBIO.

fcntl()	ioctl()
(10) We cannot get # bytes in socket receive buffer using fcntl()	(19) We use FIOREAD to get # bytes in socket receive buffer.

a) SO_BROADCAST

- Enables or disables the ability of a process to send broadcast messages.
- it is supported only for datagram sockets.
- its default value is off.

b) SO_KEEPALIVE

- purpose of this option is to detect if the peer host crashes. The SO_KEEPALIVE option will detect half open connection and terminate them.

c) SO_RCVTIMEO and SO_SNDTIMEO

→ these options place a timeout on socket receives and sends.
→ timeout value is specified in a timeval structure.

```
struct timeval {
        long tv_sec;
        long tv_usec;
};
```

- to disable timeout the values in the timeval structure are set to 0.

d) SO-REUSEADDR

→ it allows a listening server to restart & bind its well known port even if previously established connections exist.

- it allows multiple instances of the same server to be started on the same port as long as each instance binds a different local ip address.

- it allows a single process to bind the same port to multiple sockets as long as each bind specifies a different local Ip address.

e) SO-LINGER

- Specifies how close operates for a connection-oriented protocol.
- the following structure is used.

```
Struct linger {
        int l_onoff;
        int l_linger;
};
// l_onoff → 0 = off  non-zero = on
// l_linger specifies seconds.
```

23) What are the meanings of SIOCATMARK, SIOCGPGRP and SIOCSPGRP? How do you get socket as non-blocking using fcntl() and ioctl(). illustrate with section of code.

→ SIOCATMARK

↳ return through the integer pointed to by the third argument a non-zero value. If the Sockets read pointer is currently at the out of band mark, or a zero value. If the read pointer is not at the out-of-band mark

→ SIOCGPGRP

→ return through the integer pointed to by the third argument either the process ID or the process group ID that is set to receive SIGIO or SIGURG signal for this ~~the~~ Socket. This request is identical to an fcntl of F_GETOWN. note that POSIX Standardizes the fcntl.

→ SIOCSPGRP

- Set either the process ID or process group ID to receive the SIGIO or SIGURG signal for this Socket ~~from~~ from the integer pointed to by the third argument. This request is identical to an fcntl of F_SETOWN. note that POSIX Standardizes the fcntl;

→ Code to enable non-blocking I/O using fcntl()

```
int flags;
        /* set a socket as non-blocking */
if((flags = fcntl(fd, F_GETFL, 0)) < 0)
        err_sys("F_GETFL error");
flags = O_NONBLOCK;

if(fcntl(fd, F_SETFL, flags) < 0)
        err_sys("F_SETFL error");
```

→ FIONBIO is used for non-blocking I/O in ioctl()
FIONBIO → the non-blocking flag for the socket is cleared or turned on, depending on whether the third argument to ioctl points to zero or non-zero value respectively. This request has the same effect as the O_NONBLOCK file ~~status flag, which can be cleared with the F_NONBLOCK file status flag~~ status flag which can be set and cleared with the F_SETFL command to the fcntl function.

→ int ioctl(int sd, unsigned long cmd, void *data)
- if *data is 0, ioctl clears non-blocking I/O. if data is not 0, sd is set for non-blocking I/O.

On = True
Status = ioctl(sfd, FIONBIO, &on);

24) What is daemon process and how does it started? How do you create the daemon process in Unix explain.

→ A daemon is a process that runs in the background and is not associated with a controlling terminal. Typically it is started when the system is bootstrapped.

→ We can create the Daemon process in Unix System in following ways :-

i) fork :- At first, we call fork and parent terminates and child continues. If the process was started as a shell command in the foreground when the parent terminates the shell. This automatically runs child process in the background.

ii) SetSid = it is a POSIX signal that creates a new session. The process becomes the session leader of new session becomes the process group leader of a new process group and has no controlling terminal.

iii) Ignore, SIGHUP and fork again.
→ We ignore SIGHUP and call fork again. When this function returns the parent is really the first child and it terminates leaving the second child running. The second

fork guarantees that daemon cannot automatically acquire a controlling terminal should it open a terminal device in the future.

iv) Change working directory

→ We change the working directory to root directory. Since file system can't be unmounted if the working directory is not changed.

v) Close any open descriptor

→ we close any open descriptors inherited from the processes that executes the daemon.

vi) Redirect stdin, stdout and stderr to /dev/null, so that a read from any of these descriptors returns 0 and kernel discards anything written to them.

vii) Use Syslogd for errors

→ the Syslogd daemon is used to log errors.

25) What is Syslogd()? What is the technique for logging messages from a daemon process? Explain with sample code.

⇒ Syslogd() is a daemons which runs in an infinite loop that call select waiting for any one of its three descriptors to be readable.

- a unix domain it reads the log messages & does what the configuration file says to do with that message.

→ the technique for logging messages from a daemon process is to call the syslog function.

```
#include <syslog.h>
void syslog (int priority, const char *message, ...);
```

→ Here, the priority argument is a combination of a level and facility. The message like a format string to printf with an addition of %m which is replaced with an error message corresponding to the value of errno.

→ the following call would be issued by a daemon when a call to rename function unexpectedly fails.

```
syslog (LOG_INFO | LOG_LOCAL2, "rename (%s, %s): %m", file1, file2);
```

Sample code

```
#include<stdio.h>
#include<unistd.h>
#include<syslog.h>
#include<fcntl.h>
#include<signal.h>
#define MAXFD 64
int main()
{
    int i;
    pid_t pid;
    if((pid=fork())<0)
        return -1;
    Signal(SIGHUP, SIG_IGN);
    if((pid=fork())<0)
        return -1;
    else if(pid)
        exit(0);
    chdir("/");
    for(i=0;i<MAXFD;i++)
        close(i);
    open("/dev/null", O_RDONLY);
    open("/dev/null", O_RDWR);
    open("/dev/null", O_RDWR);
    openlog("My Daemon:", LOG_PID, LOG_USER);
    syslog(LOG_ERR, "log message from process with id %d", getpid());
    closelog();
    return 0;
}
```

26) How close() is different from shutdown()? which functions set h_errno global variable? Differentiates errno and h_errno as well.

→

close()	shutdown()
i) Normal way to terminate connection is to call the close function. Closes the socket but the connection is still open for processes that shares this socket connection stays opened both for read & write.	i) shutdown() breaks the connection for all processes sharing the socket. A read will detect EOF & a write will receive SIGPIPE.
ii) close() decrements the descriptors reference count and closes the socket only if the count reaches 0.	ii) with shutdown we can initiate TCPs normal connection termination sequence regardless of the reference count.
iii) close terminates both directions of data transfer, reading & writing.	iii) It terminates one direction (half) of the connection.
iv) int close(int sockfd);	iv) int shutdown(int sockfd, int how to);

→ gethostbyname() & gethostbyaddr() set h_errno global variable.

gethostbyname() sets the global integer h_errno to one of the following constants defined by including <netdb.h>

- HOST_NOT_FOUND
- TRY_AGAIN
- NO_RECOVERY
- NO_DATA

Errno	h_errno.
i) if a process is blocked in a call to 'select' the return from the signal handler causes the function to return with 'errno' set to 'EINTR'.	i) It is a host error variable.
ii) Socket functions like select(), poll() set errno when an error occurs.	ii) gethostbyname() & gethostbyaddr() sets the global integer h_errno to one of the following constants defined by including <netdb.h> - HOST_NOT_FOUND, - TRY_AGAIN. - NO_RECOVERY. - NO_DATA.

27) How windows Socket is different from berkely Socket? Explain the role of setup(), Cleanup() and WSAStartup() function in winsock architecture.

→ Windows Socket is different from berkeys socket in following ways :-

i) Winsock supports TCP/IP domain for IPC on the same Computer as well as network Communication. In addition to TCP/IP domain, Socket in most UNIX implementation supports the UNIX domain for IPC on the same Computer.

ii) Return values of certain Berkeley's functions are different. for eg :- Socket() function returns -1 on failure in UNIX, in the winsock implementation returns INVALID_SOCKET.

iii) Certain Berkely functions have different names in winsock. for eg in UNIX, the close() system call is used to close a Connection in winsock, the function is called ~~close Socket()~~ closeSocket().

⇒ the role of setup() & Cleanup() are :-

(*) the winsock functions, the application needs are located in the dynamic library named WINSOCK.DLL or WSOCK32.DLL depending on whether the 16-bit or 32-bit version of windows is being targeted.

* the application is linked with either WINSOCK.LIB or WSock32.LIB as appropriate.

* the include file where the winsock function and structures are defined in named WINSOCK.H for both the 16-bit and 32-bit environments.

→ the following are the roles of WSAStartup functions:

* initialize the underlying windows Socket. Dynamic link library.

* Gives the TCP/IP stack under a chance to do any application. specific initialization.

* Also used to confirm that the version of winsock.DLL is compatible with requirements of the application.

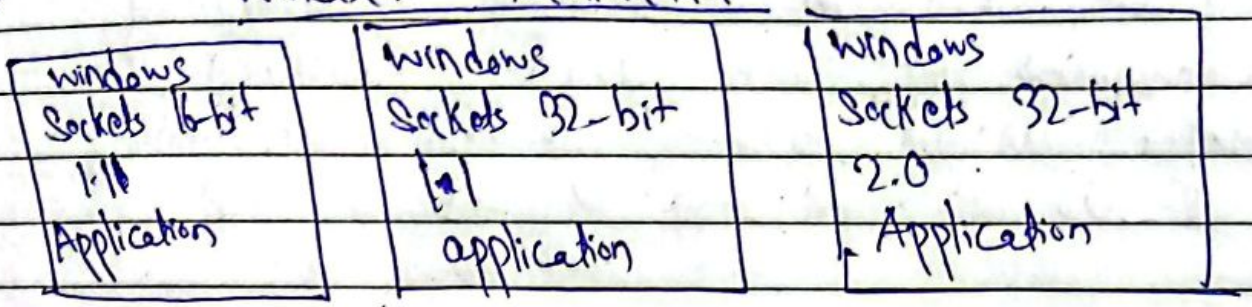
→ following are the roles of WSACLEANUP() function

* Used to terminate an applications use of winsock.

* Usually called after the applications's message loop has been terminated.

28) Explain winsock architecture. What are the types of dynamic linking in winsock programming?

→ Winsock Architecture



→ the top layer shows different network applications pertaining to different winsock API versions and windows architectures.
the two layers below it specify DLLs used for winsock API version 1.1 & 2.0 respectively.

→ Winsock provides a service provider Interface for creating winsock services, commonly referred to as the winsock SPI.

Afd.Sys is a windows driver. A driver is a small software program that allows your computer to communicate with hardware or connected devices.

→ there are two methods for calling a function in a DLL

i) In. Load-time dynamic linking: In this dynamic linking process, a module makes explicit calls to exported DLL functions as if they were local functions. This require to link the module with the import library for the DLL that contains the functions.

ii) Runtime Dynamic linking.
→ In this Dynamic linking, a module uses the load library or load library Ex function to load DLL at run time. After it is loaded, the module calls the GETProcAddress function to get the addresses of exported DLL functions.

29) differentiate blocked I/O with unblocked I/O. Explain all the functions in detail to handle blocked I/O in winsock architecture.

→

Blocked I/O	Unblocked I/O
i) it means when socket functions take indeterminate amount of time to executes it is said to block. calling function blocks further execution.	i) we use single thread to handle multiple concurrent connections.
ii) we either need to accept that we are going to wait for every I/O request or we need to fire off thread per request.	ii) we can send off multiple requests but we need to keep in mind that the data will not be available until some latter point

→ the functions that are needed to handle blocked I/O in winsock architecture are as follows:-

i) WSAAsyncSelect() = it works by sending a windows message to notify a window of a socket event.

int WSAAsyncSelect(SOCKET s, HWND hWnd, u_int wMsg, long lEvent);

Here,
→ s is a socket descriptor for which event notification is required.

→ hwnd is a windows handle that should receive a message when an event occure in socket.

→ wMsg is the message to be received by hwnd when a socket event occurs in Sockets.

→ lEvent is a bitmask that specifies the events in which the application is interested

→ WSAAsyncSelect() returns 0 on success and SOCKET_ERROR on failure. On failure WSAGetLastError Should be called.

30) What is overlapped I/O? which function is used to set windows socket in non-blocking mode? Explain non-blocking socket with connect() in the case of winsock architecture.

→ Overlapped I/O is also called asynchronous I/O in which form of input/output processing that permits other processing to continue before the transmission has finished.

→ ioctlsocket() function is used to set windows socket in non-blocking mode.

→ non-blocking socket with Connect()

- When connect is called on a blocking socket, the function blocks i.e. the function doesn't return until the TCP's 3-way handshake is completed (actually until SYN, Ack is received from remote end).

- the ioctlsocket function can be used to set Socket in non-blocking mode. with a non-blocking socket, the connect returns immediately without completion.

- In this case, Connect will return SOCKET_ERROR and WSAGETLastError will return WSAEWOULDBLOCK

- In this case, there are 3 possible scenarios:-

i) Use the select function to determine the completion of the connection request by checking to see if the socket is writeable. If connect fails, failure of the socket attempt is indicated in exceptfd (application must then call getsockopt SO_ERROR to determine the error value to describe why the failure occurred).

ii) if the application is using WSAAsyncSelect to indicate in Connect events, then the application will receive an FD_CONNECT notification indicating that the connect operation is complete (successfully or not).

iii) if the application is using WSAEventSelect to indicate in Connection events, then the associated event object will be signaled indicating that the Connect Operation is complete (successfully or not).

31) Why WSAGetLastError() is required in Winsock programming? Explain WSAGetOverlappedResult() function.

→ WSAGetLastError() returns that last winsock error that occurred. Because winsock isn't really part of the operating system but is instead a latter add-on errno (like in Unix and MS-Dos) couldn't be used.
As soon as a winsock API calls fail, the application should call WSAGetLastError() to receive specific details of the error.
prototype : int WSAGetLastError(void);

- if WSAGetOverlappedResult succeeds, the return value is TRUE. This means that the overlapped operation has completed successfully and that the value pointed to by lpcb transfer has been updated.
- if WSAGetOverlappedResult returns FALSE, this means that either the overlapped operation has not completed the overlapped operation completed but with errors or the overlapped operations completion status could not be determined due to errors in one or more parameters to WSAGetOverlappedResult.

32) In which approach of communication windows socket provides better functionalities than Unix Socket? What will happen if you call shutdown() and close() function? Do these functions are different from WSACleanup() function? Example how how not.

→ In Inter-process Communication, windows socket provides better functionalities than Unix Sockets.

- When close() is called, it decrements the descriptor's reference count and terminates both directions of data transfer, reading and writing.
- When shutdown() is called, it initiate TCP's normal connection termination sequence (the four segments beginning with a FIN). The action of function depends on the value of the 'how to' argument.

int shutdown(int sockfd, int howto);

→ Yes, Shutdown() & close() functions are different from WSACleanUp() function although all the function terminate the process

→ Shutdown() & close() are used for terminating Unix domain socket while WSAcleanup() are used to terminate an applications use of winsock.

→ for every call to WSAStartup() there has to be a matching call to WSACleanup(). but no such matching call is required in shutdown() and close() function.

33) What is the importance of backlog argument in listen() function? TCP sockets are full-duplex sockets. Is it possible to convert those sockets into half-duplex sockets? if possible - show the mechanism / if impossible - explain why.

→ the importance of backlog argument in listen() is to specify / instruct the socket to allow a maximum reasonable number of pending connections in the queue.

int listen(int sockfd, int backlog);

→ TCP sockets are full duplex sockets. But it is possible to convert those sockets into half-duplex sockets using shutdown() function. Since TCP connection is full duplex, there are times when want to tell the other end that we have finished sending even though that end might have more data to send us.

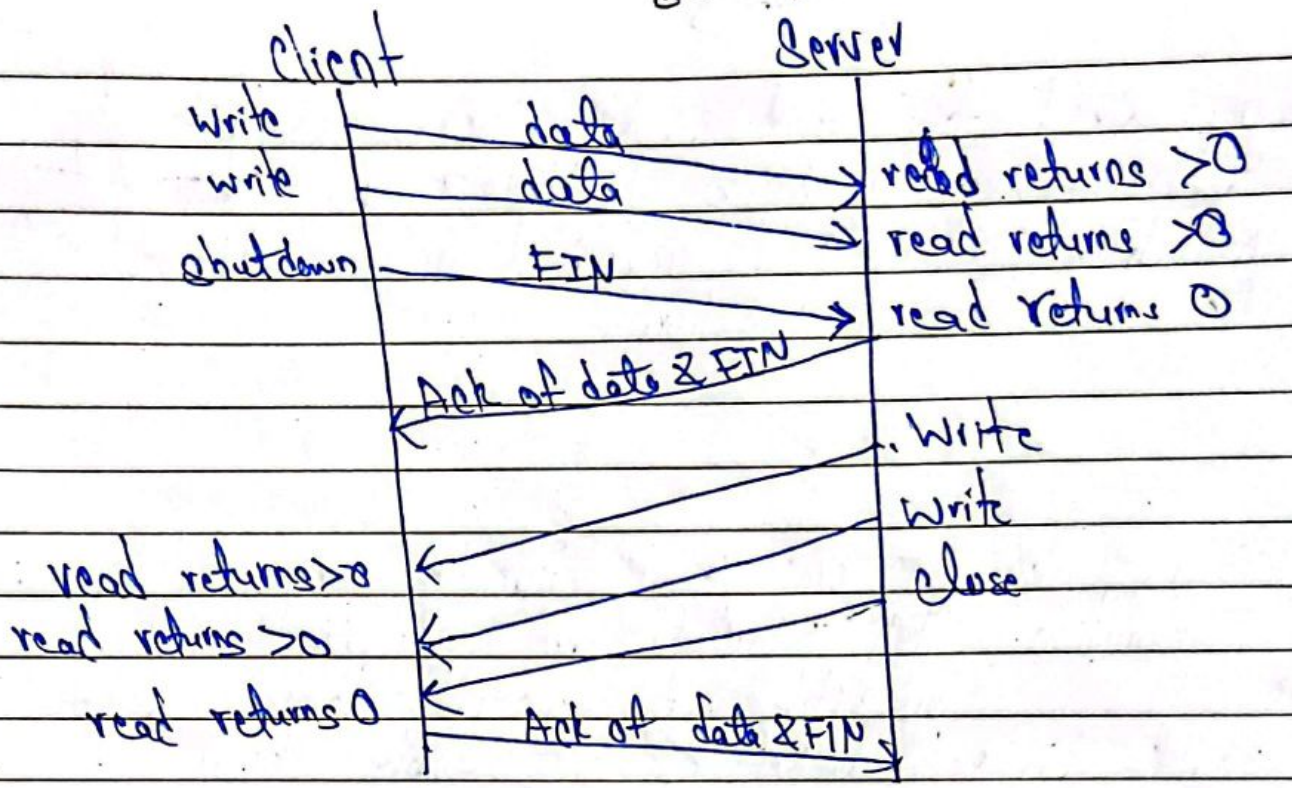


fig:- Calling Shutdown to close half of a TCP Connection

int shutdown(int sockfd, int howto);

→ the action of the function depends on the value of the how to argument.

Value of 'how to'	Description
i) SHUT_RD	- the read half of the connection is closed. No more data can be received on the socket and any data currently in the socket receive buffer is discarded.

(ii) SHUT_WR - the write half of the connection is closed. In the case of TCP, this is called half close. Any data currently in the socket send buffer will be sent, followed by TCP's normal connection termination sequence.

(iii) SHUT_RDWR - the read half and the write half of the connection are both closed. This is equivalent to calling shutdown twice: first with SHUT_RD and then with SHUT_WR.

34) What are the major differences between Unix and window socket? Is it possible to write a common network application that runs in both UNIX and windows? How would you do it. Show simple example program as well.

→ Yes, it is possible to write a common network application that runs in both LINUX and windows.

→ We do it using hybrid program or crossplatform program for UNIX and LINUX & windows program:

```
#if defined (_WIN32)
#if n def _WIN32_WINNT
 #define _WIN32_WINNT 0x0600
#endif

#include <winsock2.h>
#include <ws2tcpip.h>
#pragma Comment (lib, "ws2_32.lib")

#else
#include <sys/types.h>
#include <sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>
#endif
#include<stdio.h>
    int main ()
      #if defined (_WIN32)
      WSADATA d;
if(WSAStartup(MAKEWORD(2,2),&d)){
      printf (stderr, "failed to initialize.\n");
return 1;
}
#end if
      printf("Hello cross platform App\n");
      #if defined (_WIN32)
        WSACleanup();
      #endif
        return 0;
}
```

36) How do you implement Stream Communication in winsock? Describe each step with the help of relevant APIs.

→ We implement Stream Communication in winsock by

Client	Server
i) initialize winsock	i) Initialize winsock.
ii) Create a socket	ii) Create a socket.
iii) Connect to the server	iii) Bind the socket.
iv) Send & receive data	iv) listen on the socket for a client.
v) Dis Connect	v) Accept a Connection from a client.
	vi) Receive & Send data
	vii) Disconnect.

→ Steps for creating a socket for client :

i) to initialize winsock.
→ WSAStartup function is called to initiate use of WS2_32.dll.

ii) Create a socket
→ Declare an addrinfo object that contains a Sockaddr structure and initializes those values

iii) to connect to a Socket
→ Connect() is called, passing the created Socket and the sockaddr structures as parameters.

iv) Sending & receiving data on the client.
→ Send() & recv() are used to send and receive the data.

v) To disconnect & shutdown a socket.
→ closeSocket() & shutdown() are used.

→ Steps for Creating a Socket for the server.

i) to create a Socket for the server
- getaddrinfo() is used to determine the values in the sockaddr structure.
- Socket() is called and return its value to the listen socket variable.

ii) to bind a socket
→ bind() is called, passing the created socket and sockaddr structure returned from the getaddrinfo() as parameters.

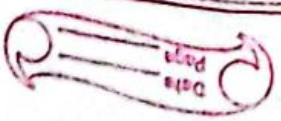
iii) To listen on a Socket

→ listen() is called, passing as parameters the created socket and a value for the backlog, maximum length of the queue of pending connection to accept.

iv) To accept a connection on a socket.
- accept() is called to accept a Connection on a socket.

v) Sending & receiving data on the server.
→ Send() & recv() functions are used for this operation.

vi) To disconnect & Shutdown a socket.
→ Shutdown() & ~~close socket~~() are used to disconnect & shutdown a Socket.



38) Write short notes on :- telnet, ftp, ping, ifconfig/ip config, netstat, rlogin.

→

① telnet

→ It is a popular remote login. Standard application that almost every TCP/IP implementation provides. It works both hosts that use different operating systems. Telnet uses option negotiation between the client & server to determine what features each end can provide.

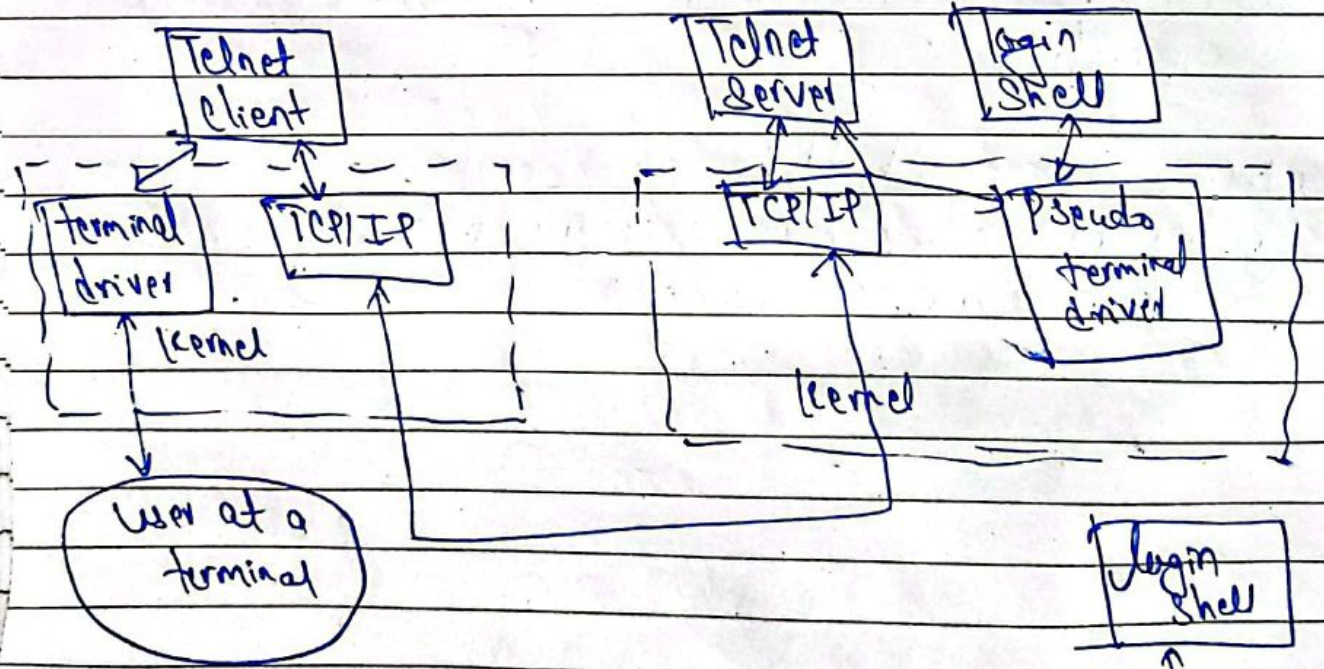


Fig:- Overview of Telnet Client-Server.

2) tftp

→ TFTP stands for Trivial File Transfer protocol. It is a simple lockstep protocol for transferring files (FTP) implemented on the top of the UDP/IP protocols. Using well-known port number 69.

→ TFTP was designed to be small & easy to implement & therefore it lacks most of the advance features offered by more robust file transfer protocols.

→ TFTP only reads/writes files from/to a remote server. It cannot list, delete or rename files & directories and it has no provisions for user authentication.

(3) Ping

→ Ping is a Computer network administration software utility used to test the reachability of host on an internet protocol network.

→ it measures the round-trip time for messages sent from the origination host to a destination Computer and echoed back to the source.

→ Ping Operates by sending internet Control message protocol (ICMP) echo request packets to the target host and waiting for an ICMP echo reply.

→ the results of the test usually include a statistical summary of the results, including the minimum, maximum, the mean, round-trip time & usually standard deviation of the mean

→ eg: Ping -c4 www.google.com // c = packet counts.

(4) Ifconfig / ipconfig

→ ipconfig - window
→ ifconfig - Unix and Unix-like systems
→ the ifconfig stands for "interface Configuration". It is used to assign an address to a network interface and/or Configure network interface parameters.

Examples

*list interfaces (only active)
→ ifconfig.

*list all interfaces
→ ifconfig -a

* Display the configuration of device eth0 only
  → ifconfig eth0.

* Enable and disable an interface
  → sudo ifconfig eth1 up
  → sudo ifconfig eth1 down.

* Configure an interface
  → sudo ifconfig eth0 net 192.168.0.10 netmask 255.255.255.0

(5) netstat

→ it means network statistics. The netstat is a command-line network utility tool to display the information about network connections. It can

i) display the routing table.
   netstat -r = shows routing table i.e. destination, gateway, genmask, flags, network interface, etc.

ii) display interface statistics
   netstat -i ÷ displays statistics for the network interface currently configured. If the -a option is also given, it prints all interfaces present in the kernel, not only those that have been configured currently.

iii) Display network Connection

→ netstat -ptcp = displays the information about TCP connections. The netstat provides statistics for the following : proto, local address, foreign address, TCP states.

(6) Rlogin

→ Rlogin is form Berkely Unix and was developed to work between Unix systems only but it has been ported to other operating systems also.

→ Rlogin appeared with 4.2BSD and was intended for remote login only between Unix hosts. This makes it a simpler protocol than Telnet, since option negotiation is not required when the operating system on the client & server are known in advance.

22) What are the socket options? Which functions are used to set & get a value of socket options? Explain them in detail with sample code.

→ Socket options are parameters for controlling various aspects like TCP retry attempts, timeouts, connection information, etc.

→ functions used to set & get a value of socket options are :-

i) getSockOpt & setSockOpt functions
ii) fcntl function
iii) ioctl function

1) getSockOpt() & setSockOpt()

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

int getSockopt (int sockfd, int level, int optname,
                void *optval, socklen_t *optlen);

int setSockopt(int sockfd, int level, int optname, const
                void *optval, socklen_t optlen);
```

→ Both functions returns 0 if Ok else -1 on error
→ 'Sockfd' must refer to an open socket descriptor.

→ the level indicates whether the socket option is general or protocol-specifies socket

→ the optval is a pointer to a variable from which the new value of the option is fetched by 'Setsockopt' or into which the current value of the option is stored by 'getsockopt'.

(11) fcntl() function

→ fcntl() stands for file Control and this function performs ~~various~~ various descriptor control operations.

→ the fcntl() provides the following features related to network programming.

(a) Non-blocking I/O = we can set the O_NONBLOCK file status flag using the F_SETFL command to set a socket as non-blocking.

(b) Signal driven I/O = we can set the O_ASYNC file status flag using the F_SETFL command, which causes the SIGIO signal to be generated when the status of a socket changes.

int fcntl (int fd, int cmd, long arg);

→ Each descriptor has a set of file flags that is fetched with the F_GETFL command and set with the F_SETFL command. The two flags that affect a socket are:-

O_NONBLOCK → nonblocking I/O
O_ASYNC → Signal-driven I/O

(iii) ioctl() function

→ the common use of ioctl by network programs (typically servers) is to obtain information on all the hosts interface when the program starts. The interface addresses whether interface supports broadcasting, the interface supports multicasting and so on.

```
#include <sys/ioctl.h>
int ioctl(int fd, int request, ... /* void * arg */);
```

We can divide the requests related to networking of six categories:-

(a) Socket operations.
(b) file operations
(c) Interface operations
(d) ARP cache operations
(e) Routing table operations
(f) STREAMS system.

37) Explain with the help of pseudo code the use of accept with select such that the accept function doesn't block.

→ the accept() system call is used with connection based socket types (SOCK_STREAM, SOCK_SEQPACKET). It extracts the first connection request on the queue of pending connection for the listening socket, sockfd, creates a new connected socket, & returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket sockfd is unaffected by this call.

→ the argument sockfd is a socket that has been created with socket, bound to a local address with bind, and is listening for connections after a listen. The argument addr is a pointer to a sockaddr structure. This structure is filled in with the address of the peer socket, as known to the communications layer.

SYNTAX

~~SOCKET WSAAPI accept(~~

SOCKET WSAADI accept(
Socket S,
SockAddr * addr
int * addrlen
);

parameter

S= A descriptor that identifies a socket that has been placed in a listening state with the listen function.

→ If no error occurs, accept returns a value of type SOCKET that is a descriptor for the new socket. This returned value is a handle for the socket on which the actual connection is made.

→ the accept function can block the caller until a connection is present if no pending connections are present on the queue and the socket is marked as blocking. If the socket is marked as non-blocking and no pending connections are present on the queue, accept returns an error as described in the following After the successful completion of accept returns a new socket handle the accepted socket cannot be used to accept more connections.

35) In which situations asynchronous IO is preferred than synchronous IO? Explain different asynchronous IO functions in winsock programming.

→ Asynchronous I/O is preferred than synchronous I/O when we want to continue doing something else, instead of waiting for I/O operation to finish.

→ Functions that are provided by winsock programming are WSASend, WSASendTo, WSARecv, WSARecvFrom, WSAIoctl

→ WSASend

```
int WSASend(
    _In_ SOCKET                              s,
    _In_ LPWSABUF                            lpBuffers,
    _In_ DWORD                               dwBufferCount,
    _Out_ LPDWORD                            lpNumberOfBytesSent,
    _In_ DWORD                               dwFlags,
    _In_ LPWSAOVERLAPPED                     lpOverlapped
    _In_ LPWSAOVERLAPPED_COMPLETION_ROUTINE  lpCompletionRoutine
);
```

→ WSASENDTO

```
int WSASendTo(
  -In- SOCKET                          s,
  -In- LPWSABUF                        lpBuffers,
  -In- DWORD                           dwBufferCount,
  -Out- LPDWORD                        lpNumberOfBytesSent,
  -In- DWORD                           dwFlags,
  -In- Const struct sockaddr           *lpTo,
  -In- int                             iToLen,
  -In- LPWSAOVERLAPPED                 lpOverlapped,
  -In- LPWSAOVERLAPPED_COMPLETION_ROUTINE  lpCompletionRoutine );
```

Here,

- s [in] = A descriptor identifying a socket.
- lpBuffers [in] = A pointer to an array of WSABUF structures
- dwBufferCount [in] = number of WSABUF structures in the lpBuffers array.
- g [in] = An existing socket group ID.
- dwFlags [in] = A set of flags used to specify additional Socket attributes.
- lpOverlapped [in] = A pointer to a WSAOVERLAPPED structure.
- lpCompletionRoutine (in) = A pointer to the completion routine called when the send operation has been completed.

→ WSARECV

```
int WSARecv (
    _In_ SOCKET                          s,
    _In_ LPWSABUF                        lpBuffers,
    _In_ DWORD                           dwBufferCount,
    _Out_ LPDWORD                        lpNumberOfBytesRecvd,
    _Inout_ LPDWORD                      lpFlags,
    _In_ LPWSAOVERLAPPED                 lpOverlapped
    _In_ LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

→ WSARECVFROM

```
int WSARecvFrom (
    _In_ SOCKET                          s,
    _Inout_ LPWSABUF                     lpBuffers,
    _In_ DWORD                           dwBufferCount,
    _Out_ LPDWORD                        lpNumberOfBytesRecvd,
    _Inout_ LPDWORD                      lpFlags,
    _Out_ struct sockaddr                *lpFrom
    _Inout_ LPINT                        lpFromlen
    _In_ LPWSAOVERLAPPED                 lpOverlapped
    _In_ LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

→ WSAIOCTL

int WSAIoctl(

_In_ SOCKET	s,
_In_ DWORD	dwIoControlCode,
_In_ LPVOID	lpvInBuffer,
_In_ DWORD	cbInBuffer,
Out LPVOID	lpvOutBuffer,
_In_ DWORD	cbOutBuffer,
Out LPDWORD	lpcbBytesReturned,
_In_ LPWSAOVERLAPPED	lpOverlapped,
_In_ LPWSAOVERLAPPED_COMPLETION_ROUTINE	lpCompletionRoutine);

These are the different asynchronous IO functions in Winsock programming.

A/B