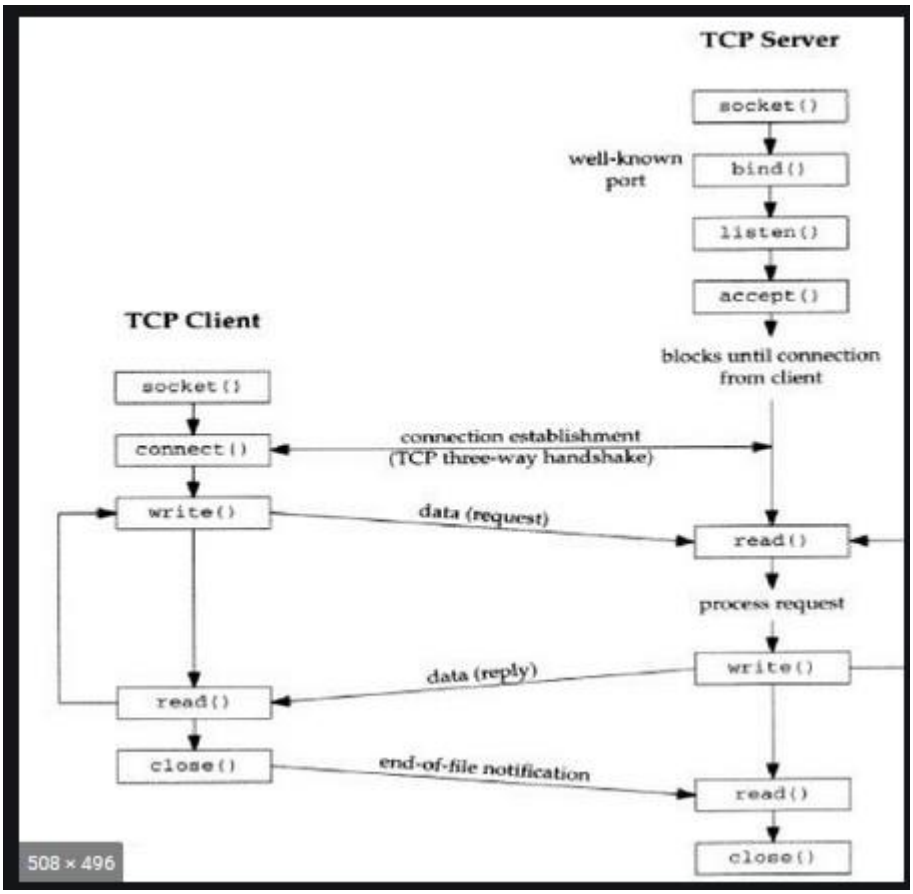
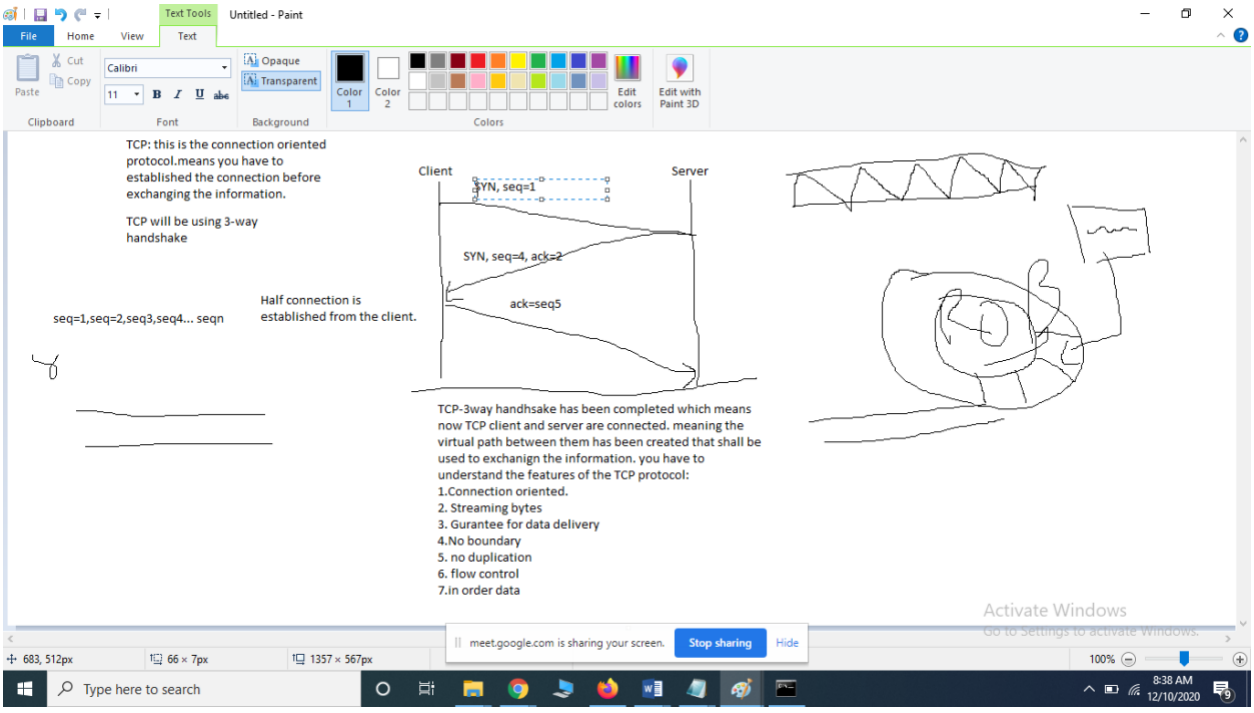
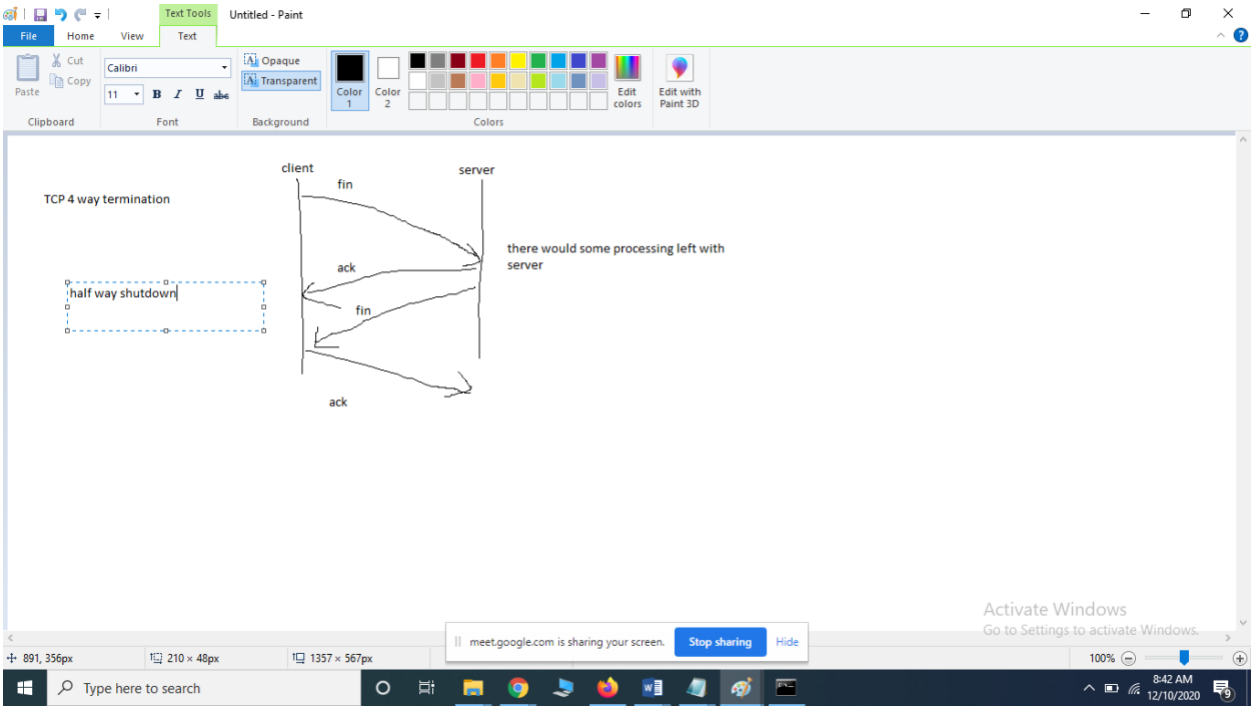


Network Protocol: IP, IPX…
IP: Network layer protocol responsible carrying the information between the hosts. It will be using IP address.(IPV4 & IPV6):192.168.218.1
Transport protocol: TCP, UDP, SCT . Responsible for carrying the information between processes. For this it will be using port number.: 1->65535 (65536)
This things is achieved by using the Client/Server Based paradigm.
Flow char for the client server based paradigm which contain the number of function involved in client and server.

Socket(): Endpoint create (Endpoint is the contacting point for any process to send and receive the data to and from the network). This function is used by client/server both.
Bind(): is used to map your endpoint with IP and port number and is only used by server.
Listen(): To make active socket into passive socket so that client request can be accepted (only for TCP).
Accept(): accepting the client connection from the server(only for TCP).
Read(): Read the data from the kernel buffer. In case of server read will shall be executed fires than followed by write function.
Write(): Write the information to kernel buffer.
Close(): close the socket means clearing all the resources allocated for that specific process. This function is used by both client and server.
Connect(): used by client to connect the server it is used in case of TCP only.





Header file:<sys/socket.h>
Prototype:
Int socket(int family,int type,int protocol);
Return: it will return brand new socket descriptor which is used to identify the end point (socket).
Param1(family): will basically talk about the types of family, and the types of family that we have are:

1. AF_INET: for internet domain socket to make communication using ip address.
2. AF_UNIX/ AF_LOCAL: for UNIX domain socket to make communication with in the system by utilizing the file.
3. AF_ROUTE: Routing socket.

Second parameter: type of the socket: define the features of the socket.

1. SOCK_STTREAM: this for streaming socket(TCP)
2. SCOK_DGRAM: this for datagram socket(UDP)
3. SOCK_SEQPACKET: sequenced packet socket
4. SOCK_RAW: raw socket.

Protocol: this will define what protocol you are supposed to use for the socket: actually you have to provide core protocol of Transport layer, which are (TCP, UDP, and SCTP).

If you supply zero on third parameter, system will automatically select the protocol for you based on second parameter supplied.

1. IPPROTO_TCP
2. IPPROTO_UDP
3. IPPROTO_SCTP

fd=socket(AF_INET,SOCK_STREAM,0);

Under successful execution of socket system call a positive integer value is return which greater than zero (Non negative value).

Header file: <sys/socket.h>

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

bind(fd,(struct sockaddr*)&serv_addr,sizeof(serv_addr)

second parameter from the bin function is type casting to the generic socket address structure of type sockaddr to the pointer. This is because we are dealing with sockaddr_in(internet domain specific address structure), however we are supposed to supply pointer to the generic socket address structure.

Return: success 0 else -1.

#include <sys/socket.h>

Int listen(int sockfd, int backlog);

Return : successful 0 else -1.

Listen function is responsible for creating the active socket to passive socket their by listening to the call from the client.

Sockfd: socket descriptor from the socket system call for identifying the socket.

Backlog: this is basically is maximum number of connection the kernel should queue for this (socfd) socket.

Two types of back log.

Incomplete connection queue: SYN is send by client but waiting for the ACK, for this socket will be in the SYN_RCVD state of tcp.

Complete connection queue: where TCP 3way-handshake has completed. The server will be on ESTABLISHED state of TCP.

listen(fd,5);// total connection that server can handle is 5.

Never set backlog to 0 else it will not able to accept the connection. Should be greater than 0.

#include<sys/socket.h>

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

It will accept the connection request form the client. Success new socket descriptor will be return , which is the actual connection for the client and server , more over it will be used to send and received the data to and from the client.

Under the successful execution of accept call the second parameter will contain the address information related with client;

Addrlen: this is value result argument:

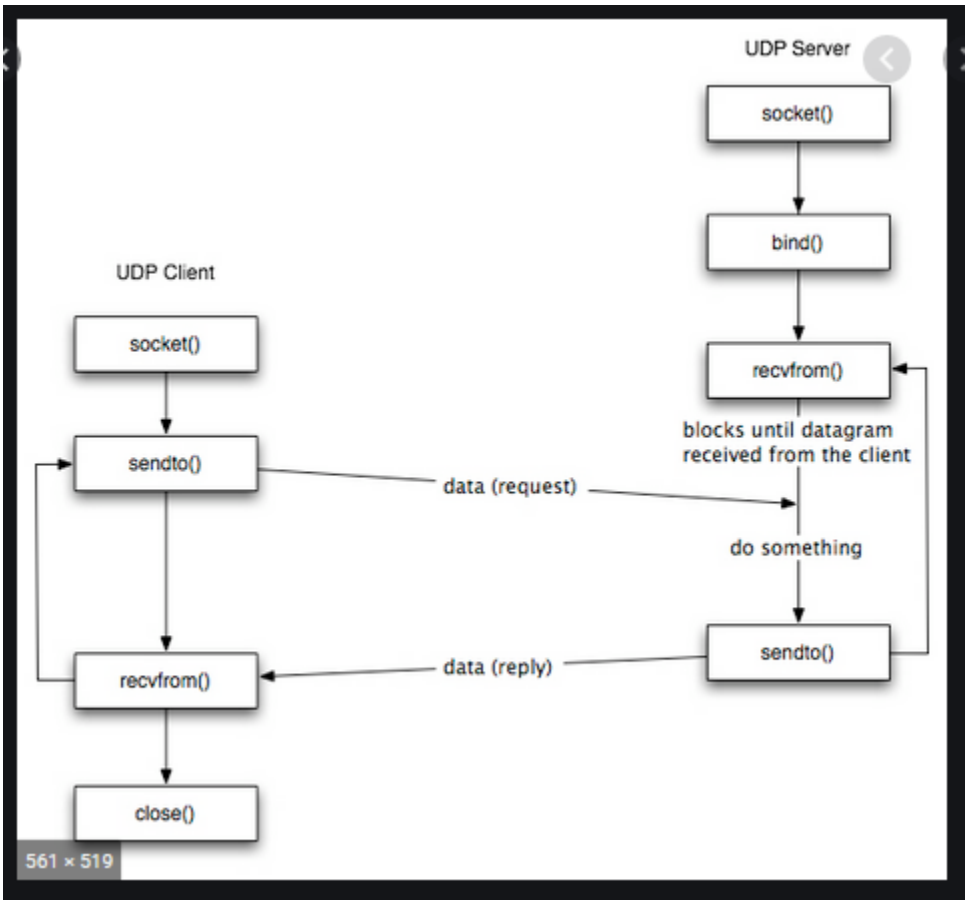
Sockfd=accept(fd,(struct sockaddr*)&cli_addr,&len);

Accept(fd,NULL,NULL);

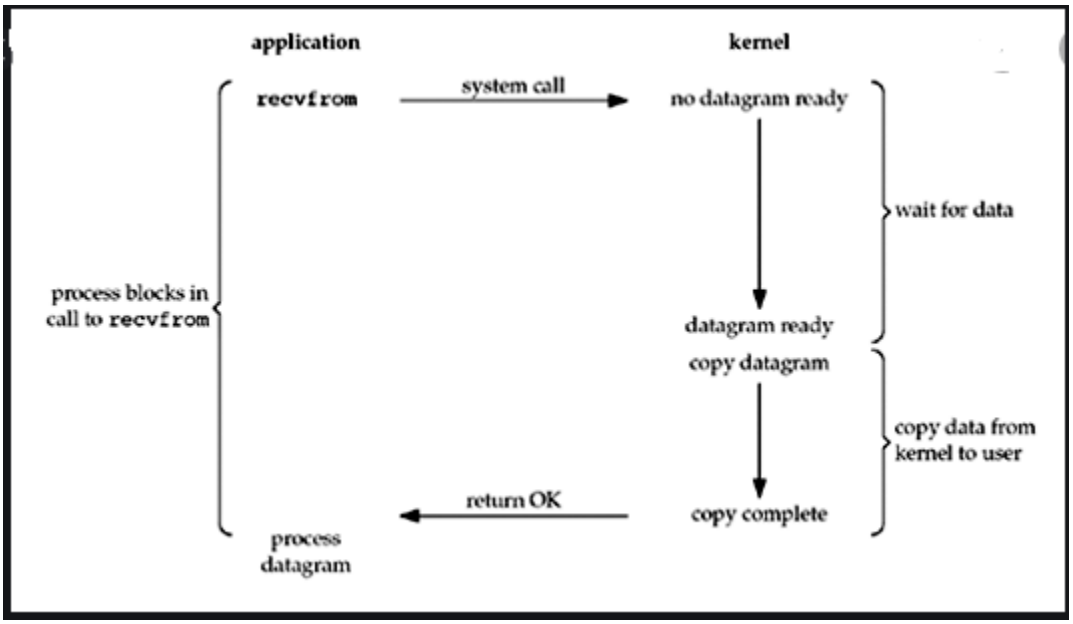
Int read(int sockfd,char * buff, int len);

int close(int fd);

successful execution return 0 else -1. Will simply mark socket to the closed and return to the process immediately. Now the socket descriptor is no longer usable.



1. Blocking I/O model.

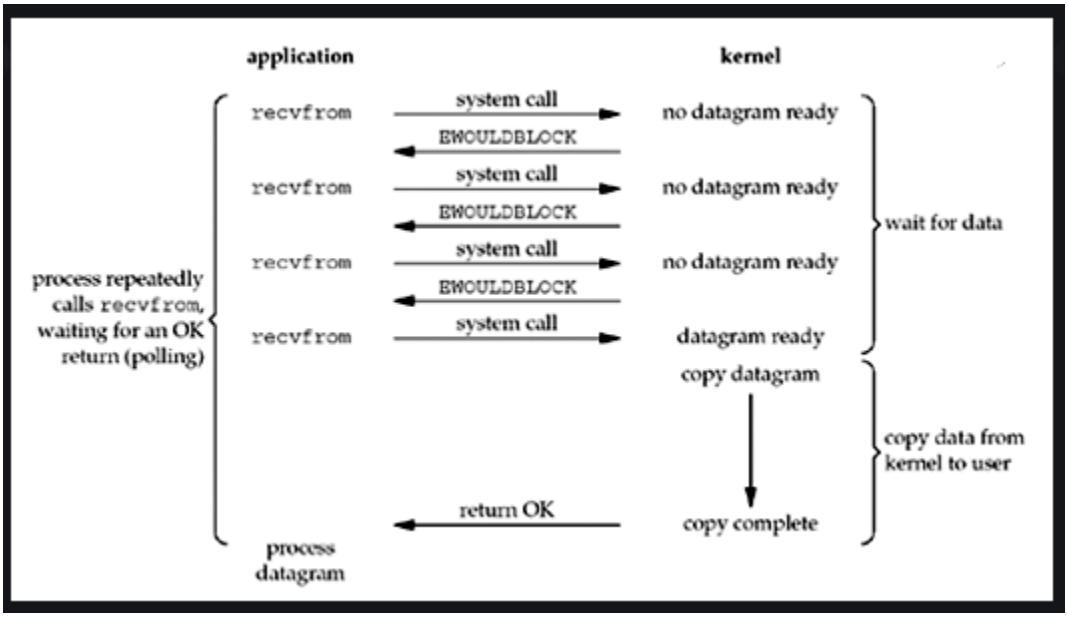


a. Process block in call to receive from unless datagram is ready or some error occur.
b. Once we execute recvfrom we are switching from application to kernel mode.

c. Process wait till the time datagram is ready, which means the kernel receive the data from the network and copy to application the buffer and return ok to application then application can process data.
d. In this kind of model our socket is blocked entire time reading the data.
e. Simplest/default model that we are using in our socket program.
f. Eg:

```
struct sockaddr_in serv;
int fd; char buff[100];
int size=100;
for(;;)
{
        Size=recvfrom(fd,buff,100,0,(struct
        socaddr*)&serv,sizeof(serv));//process will block here till data/Error
        Printf("Data receive from peer %s",buff);//Return Ok
}
```

2. Non-Blocking I/O

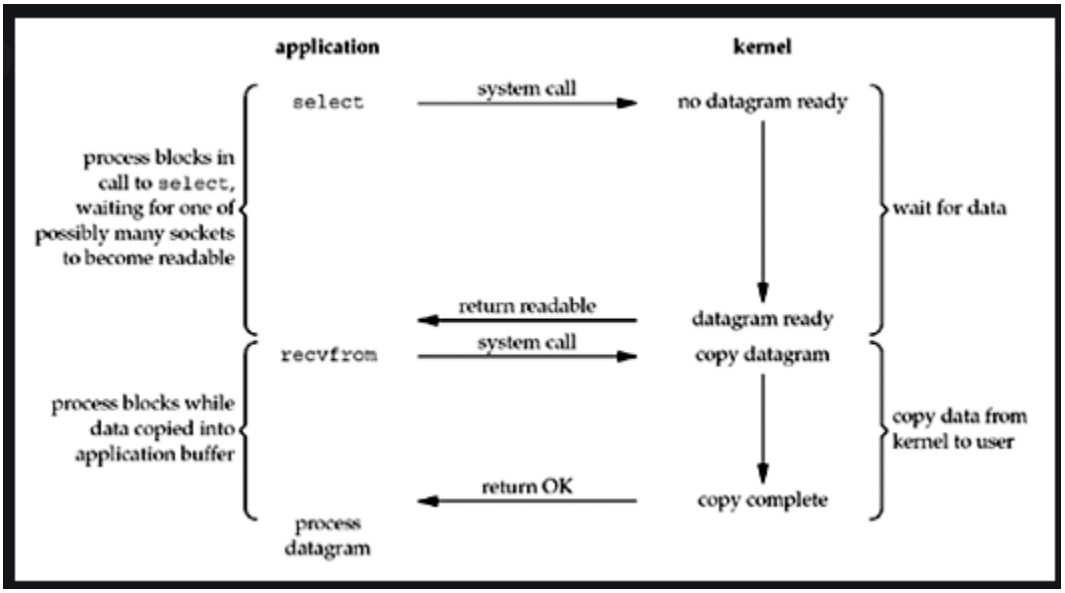


a. In this kind of model we are telling kernel not to put on sleep rather than to return error if datagram is not read. Kernel will return EWOULDBLOCK error rather than blocking the process, if data gram is not read.
b. We repeatedly call recvfrom unless data gram is ready.
c. We are repeatedly calling the recvfrom known as pooling. When datagram is ready process will be blocked till the time the kernel will copy the data from kernel to application and return ok
d. After the successful return of ok from kernel application can process the data. Means copy to application buffer.

e. This kind of model will waste the CPU time since frequently the process is asking for data availability to kernel.

```
struct sockaddr_in serv;
int fd; char buff[100];
int size=100;
int mode=1;//1 means setting socket to nonblocking mode
int res=ioctlsocket(fd,FIONBIO,&mode);//execute nonblocking function
for(;;)
{
        Size=recvfrom(fd,buff,100,0,(struct socaddr*)&serv,sizeof(serv));//
        process will not keep to sleep due to ioctlsocket() call.
        Printf("Data receive from peer %s",buff);//Return ok or error.
}
```

3. I/O multiplexing model



a. This I/O model use two call one select and another is recvfrom.
b. Select system call check for the data availability at kernel buffer and wait for the time out value if data available return immediately else wait for time out value.
c. Here process will wait for one of the socket descriptor to be ready eg: if reading data is ready, wiring data is ready, if any of the provided socket descriptor has occurred exception.
d. Syntax:

```
#include<sys/select.h>
#include<sys/time.h>
Int select(int maxfdp1, fd_set *readset,fd_set *writeset,fd_se *exceptset, const struct
timeval *timeout);
Struct timeval{
 Long tv_sec;
```

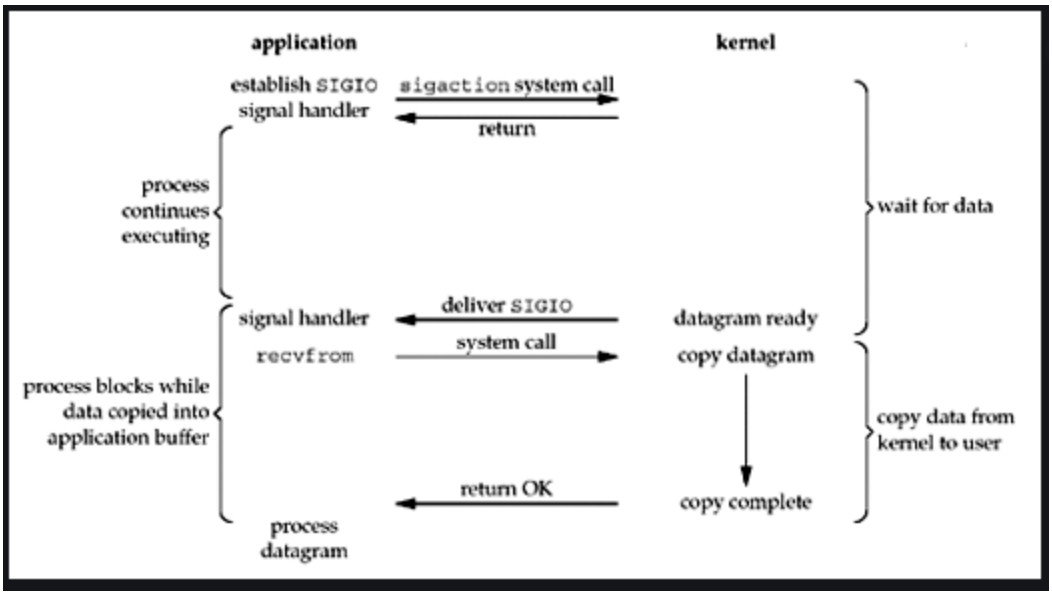
```
Long tv_usec;
};
Void FD_SET(int fd, fd_set *set);
Void FD_ZERO(fd_set *set);
```

e. Once select will return readable we call the next call recvfrom to read data from kernel where process will be block until data is copied to application buffer.
f. Application can process the data receive from kernel.
g. If timeval is set to zero the call will return immediately.

```
struct sockaddr_in serv;
int fd; char buff[100];
int size=100;
fd_set rfds;
struct timeval tval;
int retval;
FD_ZERO(&rfds);// removes all file descriptor from set.
FD_SET(fd,&rfds);// this macro add file descriptor to set.
Tval.tv_sec=5;//wait for 5 second
Tval.tva_usec=0;
for(;;)
        select(0,&rfd,0,0,&timeval);//return after 5 second if data will not
        available if yes then return immediately.

        Size=recvfrom(fd,buff,100,0,(struct
        socaddr*)&serv,sizeof(serv));//process will block here till data/Error
        Printf("Data receive from peer %s",buff);//Return Ok
}
```

4. Signal Driven I/O model



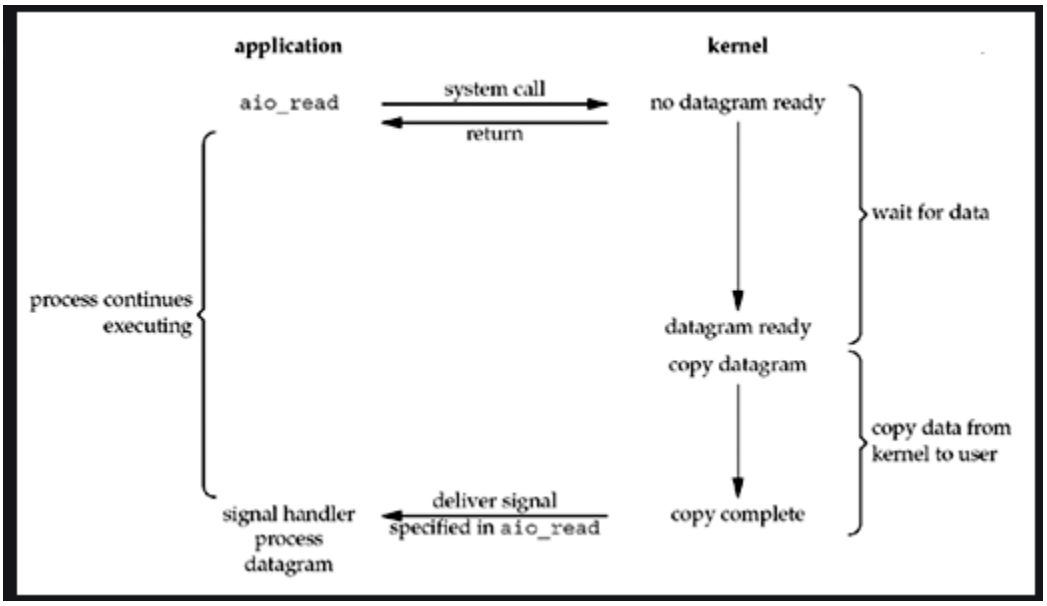
a. We use sigaction system call telling kernel to notify us with the SIGIO signal when the descriptor is ready.
b. For this we enable socket for signal driven I/O buy calling a signal handler using sigaction() system call. When this call is executed our process will not block we can continue with other processing and wait for signal from signal handler either data is ready to process or ready to read.
c. When the datagram is ready the sigaction() system call will return SIGIO to our process, here we can do two things.
    i. We can read the datagram from the signal handler by calling recvfrom and then notify the main loop that data is ready to processed or.
    ii. Notify the main loop and let it read the datagram.

#include<signal.h>
#include<sys.h>
Int sigaction(int sig,const struct sigaction *act,struct sigaction *aact);//0 success -1 failed

Sig: input parameter to define the list in control signal tag eg we will be supplying SIGIO(completion of input output). There are lots of control signal egSIGABRT(abnormal termination),SIGSTP(Stop signal for terminal) etc.
Act: (input) a pointer to the sigaction structure that describe action to be taken for the signal. Can be null.
Aact: (output) A pointer to a storage location where sigaction() can store sigaction structure . This structure contain the action currently associated with sig. can be null.

Struct sigaction act;
Act.sa_flags=SA_SIGINFO;//wants to send signal to calling function.
(Sigaction(SIGIO,&act,NULL);

Asynchronous I/O model

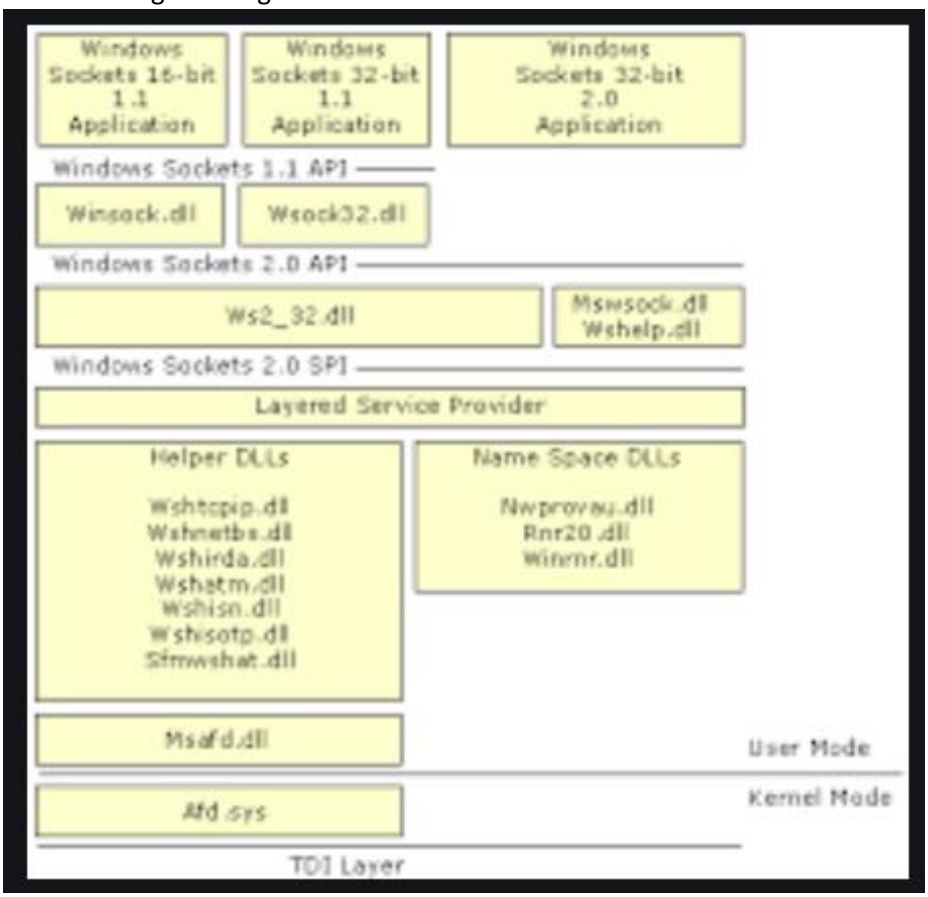


a. Aio_read() request asynchronous read function.
b. Here once the data is copied to application buffer then it is notify to application process.
c. Then application can process the data, the aio_read deliver signal to process on data ready.
d. Int aio_read(struct aiocb *aiocbp); //0 success -1 error.

Function used in UNIX:

1. Int socketpai(int family,int type,int protocol , int socfd[2]);//non zero ok -1 erro.
   #include<sys/socket.h>
   #include<sys/types.h>
   Int sockets[2];
   Socketpair(AF_UNIX,SOCK_STREAM,0,sockets);//two socket pair shall return on sockets. Only supported by unix.
2. Int getpeername(int s, struct sockaddr*addr,socklen_t * len);
   Int fd;struct sockaddr_in serv,cli;
   S=socket(AF_INEt,SOCK_STREM,0);
   Serv.sin_family=AF_INET;
   …
   Connect(fd,(struct sockaddr*)&serv,sizeof(serv));
   Len=sizeof(addr);
   Getpeername(fd,(struct sockaddr)&addr,&len);
   Printf(“Peer IP addres is %s”,inet_ntoa(addr.sin_addr));
   Printf(“Peer port is %d”,noths(addr.sin_port);
   Ioctl();
   Fcntl();
   Syslog();

Winsock Programming:



1. Window socket API 1.1(32 bits and 16 bits) and also this API support TCP/IP protocol suite.
    a. Winsock.dll (16 bits) : for 16 bits application.
    b. Wsock.dll(32 bits): for 32 bits application
2. Window socket API 2.0: can support AppleTalk, IXP/SPX many other as well as TCP/IP. Winsock 2.0 follows Window open system architecture (WOSA) model. It has two distinct part API for application and other is service provider interface for protocol stack and name space service provider. This two stack model to allow Winsock to provide multiple protocol support.
    a. Mswsock.dll: hidden library file in Window system, it is service provider for socket 2.0
    b. Wshelp.dll
3. Window socket 2.0 SPI: Concept of service provider is to allow different vendor to install their own brand of transport protocol such as TCP/IP, AppleTalk, IPX/SPX, in order to provide protocol independencies. Eg of name space service provider is DNS
4. Layer Service Provider
    a. Helpers DLL
    b. Name Space DLLs
    c. Msafd.dll
    d. Afd.sys
5. TDI layer

Dynamic-link Library (DLL)

Introduction: DLL in short is the Microsoft implementation of the shared library concept in Microsoft Windows systems. These library usually have file extension .DLL, OCX (library for active X control), DRV (for legacy system driver) etc.

Features of DLL:

1. DLL are same as EXE files but they cannot run directly for this it should be call from the EXE file
2. DLL and EXE are so much like same based on portable executable file format.
3. DLL can contain COM component and .NET libraries.
4. A DLL files consists of code and data that can be used by more than one program at the given instance of time.

Contain in DLL:

1. DLL files contain below listed items
    a. Functions
    b. Classes
    c. Variables
    d. UIs and Resources (Icon, images, files etc.)
    e. These files can be used by EXE or by other DLL

2. Use of DLLs
    a. Code modularization
    b. Code reuse
    c. Efficient memory management
    d. Reduce disk space
    e. Helps in Operating system and program load faster , run faster, and take less disk space
3. Types of DLLS
    a. Load-time dynamic linking (Static library with extension .lib):
        i. During the time of program compilation it will check for the DLL dependencies.
        ii. Here application has to explicitly call to the exported DLL function like local function.
        iii. To use the load time DLL one should provide the header file with (.h) extension and import a library file (.lib) when we are compiling and linking the application.
        iv. Steps iii will helps linker to provide the information to system required to load the DLL and resolve the exported DLLs function location at load time.
        v. Static library cannot be changed once compile and executable is created.
    b. Run-time dynamic linking(dynamic library .dll):
        i. This types of library are linked during the run time, when application gets executed and until block of code is called.
        ii. Here the file has to load explicitly by the application to access the code section or data.
        iii. DLL can be changed any time even after exe file is created.
        iv. Program loads a DLL at startup, using Win32 API load library.

v. In case if the DLL is dependent with other DLL, A program use the GetProcAddress or LoadResource to load the resource, in our case we will be using WSAStartup.

4. Window Socket in NT.
    a. Socket Calls:
        i. Creating Socket:
        SOCKET PASCAL FAR socket(int af, int type , int protocol)
        ii. Binding Socket:
        Int PASCAL FAR bind (SOCKET sock, const struct socaddr FAR * address, int addrlen)
        iii. Listening Socket:
        Int PASCAL FAR listen(SOCKET soc, int backlog)
        iv. Connecting Socket
        Int PASCAL FAR connect (SOCKET soc, const struct sockaddr FAR*name,int namelen)
        v. Accepting Socket
        SOCKET PASCAL FAR accept (SOCKET soc, const struct sockaddr FAR*addr,int FAR*addrlen);
        vi. Send/Receive
            1. In Stream socket (TCP)

```
Int PASCAL FAR send(SOCKET soc,
        Const char FAR* buffer,
        Int  len,
        Int flags,
        )
Int PASCAL FAR recv(SOCKET soc,
         char FAR* buffer,
        Int  len,
        Int flags,
        )
```

            2. In Datagram socket(UDP)

```
Int PASCAL FAR sendto( SOCKET soc,
const char FAR * buffer,
 int len,
int flags,
const struct sockaddr FAR* to,
 int tolen)

Int PASCAL FAR recvfrom( SOCKET soc,
const char FAR * buffer,
 int len,
int flags,
const struct sockaddr FAR* from,
 int fromlen)
```

vii. Terminating Socket

1. Int PASCAL FAR shutdown(SOCKET soc, int how)
    a. Description related to how flags
        i. Value=0, or SD_RECEIVE(shutdown receive operation)
        ii. Value=1, or SD_SEND(shutdown send operation)
        iii. Value=2, or SD_BOTH(shutdown both send/receive operation)
    b. Shutdown function is used to stop data transmission on the socket specified by the soc parameter, after shutdown the application can send and receive the data based on flag supplied. However to properly close the connection we have to call the closesocket() function.
    c. This kind of function is generally helpful for server where it is servicing more than one client and wants to stop the data receiving data from client.
2. Int PASCAL FAR closesocket(SOCKET soc)
    a. This function will will close the socket and release all the resources associated with the soc.

5. Windows Socket Extension
    a. Setup and CleanUp function:
        i. Setup function:

Int WSAStartup(

WORD wVersionRequired,
LPWSADAT lpWSAData
)

wVersionRequired: socket library version that we wish to use.
lpWSAData: A pointer to WSAData structure that is to receive the details of window socket implementations.
Return Value: on successful it will return zero else return one of the below error code:
WSASYSNOTREADY: underlying network subsystem is not ready for communication.
WSAVERNOTSUPPORTED: the version of windows socket version requested is not supported by this particular version of window socket implementation.
WSAEPROCLIM: A limit on the number of task supported by the windows socket implementation has been reached.
WSAEFAULT: the lpWSAData parameter is not valid.
Successful: under successful execution of the function it will load the DLL version of the requested socket API and return the information on

the lpWSAdata structure object. This function should be called in the beginning of the socket before calling the other socket system call. An application must call WSACleanup() function for each successful WSAStartup() function. WSACleanup() function

WSAData Structure:

```
Typedef struct WSAData{
        WORD  wVersion;
        WORD  wHighVersion;
        Char     szDescription[WASDESCRIPTION_LEN+1];
        Char     szSystemStatus[WSASYS_STATUS_LEN+1];
        Unsigned short  iMaxSockets;
        Unsigned short  iMaxUdpDg;
        Char FAR*       lpVendorInfo;
};
```

wVersion: the version of the Window socket implementation that the WS2_32.DLL expect call to use (Supported Version).
wHighVersion: the Highest version of Window socket specification that This DLL can support.
szDescription: Null terminated string in which WS2_32 copies the description about the Window socket implementation.
szSystemStatus: Null terminated string into which WS2_32.dll copies the relevant status and configuration information.
iMaxSockets: retained for backward compatibility.
iMaxUdpDg: Value should be ignored for Version 2 and onward
lpVendorInfo: should ignored for version 2 and onward. It is retained for compatibility with Window socket version 1.1

ii. Cleanup function

Int WSACleanup();
The function has no parameter
The return value is zero if operation is successful otherwise it will return SOCKET_ERROR, for detail information regarding the error one should call WSAGetLastError() function.
Cleanup function terminate socket operation for all the thread in multithread environment their by releasing all the resources allocated for the socket. Later this can be used by another application.
WSAStartup() and WSACleanup() is explicit function in windows to load and unload the DLL for initializing/de-initializing into the memory.

b. Error Handling Function:

i. Getting the Error

Int WSAGetLastError();
The function has no parameter
Return Value: value indicate the error code for this thread last windows operation failed.

Some of the error codes return by functions are

a. WSA_NOT_ENOUGH_MEMORY: insufficient memory available(8)
b. WSA_INVALID_PARAMETER: one or more parameter invalid(87)
c. WSAEACCES: Permission deny (10013) trying to access the socket in a way forbidden by the access permissions.
d. WSAEFAULT: Bad address error code(10014)
e. WSAENETRESET: Network dropped connection or reset(10052)

c. Error setting Function
    i. Set the error

    Void WSASetLastError(int iError)

    iError: integer that specify the error code to be return by a subsequent WSAGetLastError() call.

    Return Value: No value is return.

    A successful WSAStartup() call must execute before calling this function.

6. Function for Handling Blocked I/O (all function are not used in Socket 2 and above only supported by Windows socket implementation 1.1):
    a. WSAIsBlocking(): This function is not used by Window socket version 2 and is not directly exported by WS2_32.dll. Window socket application 1.1 that call this function are still supported through WINSOCK.dll, WSOCK32.dll. Blocking hook are generally used by single threaded application responsive during call to blocking functions. This function determine if a blocking call is in progress, basically this function allows a Winsock 1.1 application to determine if it is executing while waiting for a pervious block call to complete.

    Syntax:

    BOOL WSAAPI WSAIsBlocking();

    Parameter: no parameter

    Return value: the return value is TRUE if there is an outstanding blocking Awaiting completion in the current thread. Else it will return FALSE.

    b. WSAACancelBlockingCall(): Cancel a call to Windows socket that might be blocked. It basically cancels a blocking call that is in progress and any outstanding blocking operation for this thread. This function can be call in two scenario:
        i. First case,Suppose our application is processing a message which has been received while blocking call is in progress, in this case WSAIsBlocking() return TRUE.
        ii. Second case, A blocking call is in progress and Winsock has call back to the application blocking hook function as established by WSASetBlockingHook().

    Syntax:

    int WSAAPI WSAACancelBlockingCall();

    Parameter: Function has no parameter

    Return Value: on success return zero, else SOCKET_ERROR to get specific error Code call WSAGetLastError() functio

c. WSASetBlockingHook(): This function is used to associate an application-supplied function with Winsock that is called when a socket is blocked. This function is used by application when they want more complex message to process eg. Multi document interface (MDI) model. This function provide ability to execute its own function at "blocking" time in the place of default function.
   Syntax:
   FARPROC WSAAPI WSASetBlockingHook(FARPROC lpBlockFunc);
   lpBlockFunc: A pointer to the procedure instance address of the blocking function to be installed.
   Return value: the return value is a pointer to the procedure-instance of the previously installed blocking function. The application or the library that calls the WSASetBlockingHook function should save this return value so that it can be restored if necessary. In case if operations fails it will return NULL pointer, once can call WSAGetLastError() function to check the specific error code.
d. WSAUnhookBlockingHook(): this function disassociate the application-supplied function called when a socket is blocked. Basically this function removes any pervious blocking hook that has been installed and reinstall the default blocking mechanism. This function will always installed the default mechanism for blocking hook not the previous mechanism.
   In case if application wish to net the blocking hook i.e to establish the temporary blocking hook function and then revert to previous mechanism (whether the default one of previous mechanism established by earlier WSASetBlockingHook()) it must save the value and restore it return by WSASetBlockingHook() function.
   Argument: none
   Return Value: The return value is 0 if operation is successful else value SOCKET_ERROR will be return and to dig down we should call WSAGetLastError() function.

7. Asynchronous Database Function:
   a. WSAAsyncGetServByName() asynchronous form of getservbyname(): this function asynchronously retrieve service information that correspond to service name and port.
      HANDLE WSAAsyncGetServByName(
      HWND hwnd,
      U_int wmsg,
      Const char *name,
      Const char* porto,
      Char *buf,
      Int buflen
)
      Hwnd: Handle of the window that should receive a message when the Asynchronous request complete.
      Wmsg: Message to receive when asynchronous request completes.
      Name: Pointer to null terminated service name
      porto: this can be null, in which the function will search for correct proto for the Service attached with.
      Buf: pointer to the data area of type servent structure, the size must be greater

Than servent structure.
Buflen: size of the data area for the buf parameter, in bytes.
Return value: on success it will a nonzero values of type HANDLE that is the Asynchronous task handle for request. It can be used in two ways to cancel the Operation using WSACancelAsyncRequest, or it can be used to matchup the Operation and completion message, by examining the wparam parameter. in Case of error a zero value is return , we can call WSAGetLastError() to check the Specific error.

b. WSAAsyncGetServByPort()asynchronous form of getservbyproto(): this function asynchronously retrieves service information corresponds to port and porto.

```
HANDLE WSAAsyncGetServByName(
        HWND  hwnd,
        U_int    wmsg,
        Int       port,
        Const char* porto,
        Char *buf,
        Int buflen
)
```

Hwnd: same above
Wmsg: same above
Port: port assign to service in network byte order
Porto: same above
Buf,: same above
Buflen: same above
Return value: Same above.

c. WSAAsyncGetProtoByName()asynchronous form of getprotobyname(): this function Retrieves protocol that correspond to protocol name.

```
HANDLE WSAAsyncGetProtoByName (
        HWND  hwnd,
        U_int    wmsg,
        Const char* name,
        Char *buf,
        Int buflen
)
```

Hwnd: same above
Wmsg: same above
Name: Pointer to null-terminated protocol name to resolved.
Buf: same above
Buflen: same above
Return Value: Pointer to the data area to receive the protent data. In case of Failed it will return a zero value and one should call WSAGetLastError() to check Specific error code.

d. WSAAsyncGetProtoByNumber()asynchronous form of getprotobynumber(): this function asynchronously retrieves protocol information that correspond to protocol Number.

```
HANDLE WSAAsyncGetProtoByNumber (
        HWND  hwnd,
        U_int    wmsg,
        Int       number,
        Char *buf,
        Int buflen
)
```

Hwnd: same above
Wmsg: same above
Number: Protocol number to be resolved in host byte order.
Buf: same above
Buflen: same above
Return Value: Pointer to the data area to receive the protent data. In case of Failed it will return a zero value and one should call WSAGetLastError() to check Specific error code.

e. WSAAsyncGetHostByName()asynchronous form of gethostbyname(): this function Retrieves host information that correspond to the host name.

```
Void WSAAsyncGetServByName(

        HWND  hwnd,
        U_int    wmsg,
        Const char *      name,
        Char *buf,
        Int buflen
)
```

Hwnd: same above
Wmsg: same above
Name: pointer to null terminated name of host.
Buf: : Pointer to the data area to retrieve the hostent data.
Buflen: size of data area for the buf parameter, in bytes.
Return Value:none, for error the application indicate by the hWnd parameter Receives message in the wmsg parameter. the wparam contain in the Asynchronous task handle return by original function call. The high 16 bit of lParam can contain any error codes. Zero error codes means the asynchronous operation is successful.

.

f. WSAAsyncGetHostByAddr()asynchronous form of gethostbyaddr(): this function retrieves host information that correspond to an address.

```
Void WSAAsyncGetHostByAddr (
        HWND  hwnd,
        U_int    wmsg,
        Const char *      addr,
        Int               len,
        Int       type,
        Char *buf,
        Int buflen
)
```

Hwnd: same above
Wmsg: same above
addr: pointer to the network address for the host .
Len: length of address in bytes.
Type:type of the address.
Buf: contain pointer to the hostent structure data.
Buflen: same above
Return: None, however error code and buffer length should be extracted
From the lParam macros WSAGETASYNCERROR and WSAGETASYNCBUFLEN.

8. Synchronous Database Function:

a.
```
servent * getservbyname(
  const char *name,
  const char *proto
);
```

Name: a pointer to null terminated service name
Proto: A pointer to null terminated protocol name. if null proto will be selected
Based on service name.
Return: Servent structure else a null pointer and call WSAGetLastError().

b.
```
servent * getservbyport(
  int       port,
  const char *proto
);
```

Port: port of service in network byte order.
Proto: Optional pointer to a protocol name. if null returns the first service entry
For which the port matches the s_port of the servent structure.
Return: a Pointer to servent structure else return null pointer.

9. Structure used by database function:

a.
```
typedef struct protoent {
      char  *p_name;
```

```
char  **p_aliases;
short p_proto;
} PROTOENT, *PPROTOENT, *LPPROTOENT;
```

P_name: official name of protocol.
P_aliases: Null terminated array of alternate name
P_proto: protocol number in host byte order.

b. Hostent: this structure used to store information about a given host eg. Hostname, ipv4 address, etc.

```
typedef struct hostent {
 char  *h_name;
 char  **h_aliases;
 short h_addrtype;
 short h_length;
 char  **h_addr_list;
} HOSTENT, *PHOSTENT, *LPHOSTENT;
```

H_name: official name of host (PC).
H_aliases: A null terminated array of alternate name
H_addrtype: the type of address being returned
H_length: the length of byte of each address.
H_addr_list: A NULL terminated list of address for the host.

c. Servent: this structure is used to store or return the name and service for a given service name.

```
typedef struct servent {
 char  *s_name;
 char  **s_aliases;
#if ...
 char  *s_proto;
#if ...
 short s_port;
#else
 short s_port;
#endif
#else
 char  *s_proto;
#endif
} SERVENT, *PSERVENT, *LPSERVENT;
```

S_name: official name of service.
S_aliases: Null terminated array of alternative name
S_porto: name of protocol to be used when contacting a service.
S_port: the port number at which service can be contacted.

10. Asynchronous I/O functions:

a. WSACancelAsyncRequest(): this function cancel an incomplete asynchronous operation.

```
Int WSACancelAsyncRequest(
        HANDLE hAsyncTaskHandle
);
```

hAsyncTaskHandle: Handle that specify the asynchronous operation to be cancled.

Return: if successful zero else SOCKET_ERROR call WSAGetLastError() to get specific error.

b. WSAAsyncSelect(): this function request Window message-based notification of network events for a socket.

```
Int WSAAsyncSelect(
        SOCKET s,
        HWND hWnd,
        U_int wmsg,
        Long lEvent
        );
```

S: A descriptor that identify socket for which event notification is required
hWnd: A handle that identify Window that will receive message when a network Event occurs.
wmsg: A message to be received when network event occurs.
lEvent: A bitmask that specifies a combination of network events in which Application is interested.
Return: on success return zero else SOCKET_ERROR call WSAGetLastError() for Detail information.

c. WSARecvEx(): this function retrieves information from the connected socket or bound Connectionless socket. This is similar to recv() function except flag Parameter is used to return information.

```
Int WSARecvEx(
        SOCKET s,
        Char *buf,
        Int len,
        Int *flags,
        );
```

S: A descriptor that define a connected socket.
Buf: A pointer to the buffer to receive incoming data.
Len: the length, in bytes, of the buffer pointed to by buf parameter.
Flags: An indicator that specifying whether message is received full or Partially for datagram socket.
Retrun : on success it will return number of bytes received else SOCKET_ERROR is returned for detail information call WSAGetLastError().

11. Overlapped I/O socket:

Introduction: Overlapped I/O is name use for asynchronous socket in windows API. In order to use Overlapped I/O socket one has to pass overlapped structure to API function. The request function Is then return immediately and is completed by Operating system in background. The caller may optionally specify a win32 event handle to be raised when the operation was complete. Second a program may receive notification of an event via an I/O completion port (preferred method) . third last method to get I/O completion notification on overlapped socket Is to use ReadFileEx() and WriteFileEx().
Socket System call for Overlapped I/O socket:

a. Socket creation

```
SOCKET WSAAPI WSASocket(
        IN int    af,
        IN int    type,
        IN int    protocol,
        IN LPWSAPROTOCOL_INFO    lpProtocolInfo,
        IN GROUP                 g,
        IN DWORD                 dwFlags
);
```

Af: address family eg AF_INET
Type: socket type stream or datagram etc.
Protocol: protocol UDP , TCP used for that specific socket
lpProtocolInfo: A pointer to the type WSAPROTOCOL_INFO that define the characteristic Of socket to define.
G: reserved for future
dwFlags: the socket attribute specification.

b. WSAOVERLAPPED structure:

```
Typedef strcut WSAOVERLAPPED{
        DWORD       Internal;
        DWORD       InternalHigh;
        DWORD       Offset;
        DWORD       OffsetHigh;
        WSAEVENT hevent;
}WSAOVERLAPPED, FAR *LPWSAOVERLAPPED;
```

The Internal,InternalHigh,Offset,OffsetHigh fields all are used internally by the system and an application should not manipulate or directly use them.
Hevent: allow an application to associate an event object handle with this operation.

Winsock Programming demonstrating Multiple Functions:

```
// ServerSelect.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include<winsock2.h>
#include<Windows.h>
#include<stdlib.h>
#pragma comment(lib, "Ws2_32.lib")
#define PORT 8888
#define BUFFSIZE 200
/*Function for Handling Asynchronous connection and I/O operation*/
int handleConnection(SOCKET fd);
void sendData(SOCKET fd);
void recvData(SOCKET fd);
/*Windows Database Functions*/
void getHostName();
void getHostByName();
void getProtoByName();
void getServiceByName();
void getSockName(SOCKET);
void getPeerName(SOCKET confd);
int _tmain(int argc, _TCHAR* argv[])
{
	SOCKET fd, confd;//variable socket descriptro created to store
connected and non connected socket.
	struct sockaddr_in serv,cli;//two structure of type internet domain is
created
	WSADATA wsd; //Wsadata structure is created to store the information
return from kernel after successfuly execution of WSAStartup()
	int ret,b,l,result,len;
	DWORD ver = MAKEWORD(2, 2);//creating the variable to store
version,makeword will convert the int into DWROD datatypes need to use in
first parameter at WSAStartup()
	ULONG NONBlock = 0;//seting socket to non blocking mode

	if (ret = WSAStartup(ver, &wsd) != 0)//now we initalize the DLL and
check for the version supplied on it if success will return value on wsadata
strucute else return error.
	{
		printf("Error In Library initialization\n");
	}
	else{ printf("Successfully loaded library\n"); }
	//getHostName();
	// getHostByName();
	//getProtoByName();
	getServiceByName();

	serv.sin_family = AF_INET;
	serv.sin_port = htons(PORT);
```

```
	serv.sin_addr.S_un.S_addr = htonl(INADDR_ANY);

	fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
	//fd = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0,
WSA_FLAG_OVERLAPPED);

	//getSockName(fd);//this function must be call after socket() system
call

	if (fd > 0){ printf("End Point Created Successfully value is[%d]\n",
fd); }
	else{ printf("Error in Creating Socket\n"); }
	if( b = bind(fd, (struct sockaddr*)&serv, sizeof(serv))==0)
	{
		printf("Bind Successful");
	}
	else{ printf("Not able to Bind\n"); exit(-15); }
	if (l = listen(fd, 10) != -1)
	{
		printf("Program Listening at prot[%d] IP[%s]\n", PORT,
inet_ntoa(serv.sin_addr));
	}
	//confd = accept(fd, NULL, NULL); if you accpet the socket here the
socket is in blocking mode
	if (ioctlsocket(fd, FIONBIO, &NONBlock) == SOCKET_ERROR)//seting socket
to nonblocking mode.
	{
		printf("Soket not able to set as NonBlocking\n");
	}
	else{ printf("Ioctl operation on socket seting non block Ok\n"); }
	//confd = accept(fd, NULL, NULL);// now socket is in non blocking mode.

	/*actual code for handling multiple accept using select started*/
	len = sizeof(cli);
	while (TRUE)
	{
		result = handleConnection(fd);
		if (result>0)
		{
			printf("Select return for handling client request\n");
			confd = accept(fd, (struct sockaddr*)&cli, &len);
			//getPeerName(confd);//getpeername() system call will get
the socket information on connected socket.
			if (confd > 0){
				printf("New Client connection PORT[%d] and IP[%s]\n",
ntohs(cli.sin_port), inet_ntoa(cli.sin_addr));
				recvData(confd);
				sendData(confd);
				closesocket(confd);
			}
```

```
			}
			else if (result == 0){
				printf("Select time out\n");
			}
			else if (result == -1){
				WSACleanup();
				printf("Error in select\n",WSAGetLastError());
			}
		}
		return 0;
}
int handleConnection(SOCKET fd)
{
		struct timeval timeout;
		struct fd_set rfd;
		timeout.tv_sec = 30;
		timeout.tv_usec = 0;
		FD_ZERO(&rfd);//Initialize readdescriptor to zero
		FD_SET(fd, &rfd);//set descriptor for read notification.
		/*Note:value return by select one error -1 <0 if timeout 0 =0 data
ready greater than 0 >0*/
		//return select(0, &rfd, 0, 0, 0);// we set time val zero which will
not return unless the descriptor is ready
		return select(0, &rfd, 0, 0, &timeout);
}
void sendData(SOCKET fd)
{
		int s = 0;
		char msg[] = "Hello this is select server";
		handleConnection(fd);//Select in conjuction to sending data
		s = send(fd, msg,0, sizeof(msg));
		printf("No of Mesg send is[%d] msg=[%s]\n", s, msg);
}
void recvData(SOCKET fd)
{
		int r = 0;
		char msg[100];
		//handleConnection(fd);//select in conjuction to reading data
		r = recv(fd, msg, 0, 100);
		printf("No of bytes received is[%d] msgrecv=[%s]\n", msg);
}
void getHostName()
{
		/*char hostname[] = "DESKTOP-IH40FAV";
		struct hostent *ent = gethostbyname(hostname);
		if (ent == NULL){ printf("Error in returning IP\n"); exit(-15); }
		struct in_addr ipaddr = *(struct in_addr*)(ent->h_addr_list);
		printf("Host Ip address is[%s]\n", inet_ntoa(ipaddr));*/
		char *hostname = (char*)malloc(500 * sizeof(char));// allocated membory
to our pointer variable
```

```
	int r = gethostname(hostname, 100);
	if (r < 0){ printf("Error in retrieving Hostname[%d]\n",
WSAGetLastError()); }
	else{ printf("Hostname is[%s]\n", hostname); }
}
void getHostByName()
{
	const char*hostname = "DESKTOP-IH40FAV";
	struct sockaddr_in addr;
	struct hostent *ent;
	int i = 0;
	ent = gethostbyname(hostname);
	if (ent == NULL){ printf("Error in retrieving Information for hostent
struct gethostbynae()%s\n", WSAGetLastError()); }
	else
	{
		printf("Offical HostName is[%s]\n", ent->h_name);
		printf("Ipaddress Version[%d]\n", ent->h_addrtype);
		switch (ent->h_addrtype)
		{
		case AF_INET:
		{
			printf("Address Family is IPV4\n");
			while (ent->h_addr_list[i] != 0)
				addr.sin_addr.S_un.S_addr = (u_long)*ent-
>h_addr_list[i++];
			printf("IP address is[%s]\n", inet_ntoa(addr.sin_addr));
			break;
		}
		case AF_INET6:
			break;
		}
	}
}
void getProtoByName()
{
	char protocol[5][10] = { "ip", "tcp", "udp", "icmp", "rdp" };
	struct protoent *pr;
	int i = 0;
	for (i = 0; i < 5; i++)
	{
		pr = getprotobyname(protocol[i]);
		if (pr == NULL){ printf("Error in getting protoent structure
using getProtobyname()\n"); }
			{
				printf("getprotobyname() protocol Name is[%s]\n",
protocol[i]);
				printf("Protocal offical Name is[%s]\n", pr->p_name);
				printf("Protocol Number is[%d]\n", pr->p_proto);
			}
```

```
	}


}
void getServiceByName()
{
	struct servent *sr;
	char servicename[] = "http";
	int i = 0;
	char protocol[] = "tcp";
	sr = getservbyname(servicename,protocol);
	if (sr == NULL){ printf("Error in getservicebyname()[%s]\n",
WSAGetLastError()); }
	else
	{
		printf("getservbyname()\n");
		printf("Offical Name[%s]", sr->s_name);
		printf("Alias Name");
		while(1)
		{
			if (sr->s_aliases[i])
			{
				printf("[%s]", sr->s_aliases[i]);
				i++;
			}
			else{ break; }
		}
		printf("\nProtocol Name is[%s]\n", sr->s_proto);
		printf("protNumber listening at[%d]\n", ntohs(sr->s_port));
	}
}
void getSockName(SOCKET s)
{
	//Note:Should call after socket() system call otherwise you wont get
the socket descriptor.
	struct sockaddr name;
	struct sockaddr_in s_in;
	int len;
	len = sizeof(len);
	int i=0;
	//now binding dynamic port.
	name.sa_family = AF_INET;
	memset(name.sa_data, 0, sizeof(name.sa_data));
	int status = bind(s, &name, sizeof(name));
	if (status==NO_ERROR)
	{
		status = getsockname(s, (struct sockaddr*)&s_in,&len);
		if (status != NO_ERROR)
		{
			printf("Server Bound to port[%d]\n", ntohs(s_in.sin_port));
```

```
            }
      }
      else{ printf("Error in getting socket Name[%d]\n", WSAGetLastError());
}

}
void getPeerName(SOCKET confd)
{
      struct sockaddr_in addr;
      int len = sizeof(addr);
      int r = getpeername(confd,(struct sockaddr*)&addr, &len);

      if (r == 0)
      {
            printf("Address of Connected peer is[%s]\n",
inet_ntoa(addr.sin_addr));
      }
            }
```

UNIX Socket Programming Demonstrating Multiple function:

Server.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>//INADDR_ANY
#include<unistd.h>
#include<string.h>
#include<time.h>
#include<stdio.h>
#include<arpa/inet.h>
#include<errno.h>
#include<stdlib.h>
#define PORT 7777
int main(int argc,char * argv[])
{
        int fd=0;
        struct sockaddr_in serv_addr,cli_addr;
        int addrlen=0;
        char sendBuff[500]="Hello this is UDP Server..";
        char recvBuff[500];
        time_t ticks;
        addrlen=sizeof(cli_addr);
        fd=socket(AF_INET,SOCK_DGRAM,0);
        if(fd<0)
        {
                printf("Server:Unable to create endpoint\n");

                exit(-1);
        }else{  printf("Server:EndPoint created Successfully\n");}
        memset(&serv_addr,'0',sizeof(serv_addr));
        //memset(&sendBuff,'0',sizeof(sendBuff));

        serv_addr.sin_family=AF_INET;
    serv_addr.sin_port=htons(PORT);
        serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");

        if(bind(fd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))==0)
        {
                printf("UDP Server:Bind successfully..\n");
        }else
        {
                printf("UDP Server:Error in bind...\n");
                exit(-1);
        }
```

```
                size_t r,s;
                for(;;)
                {
                        r=recvfrom(fd,recvBuff,100,0,(struct
sockaddr*)&cli_addr,&addrlen);
                                printf("UDP Server:Message Received from client is and
Client Port[%d]  %s\n",ntohs(cli_addr.sin_port),recvBuff);
                        s=sendto(fd,sendBuff,100,0,(struct
sockaddr*)&serv_addr,sizeof(serv_addr));
                }

        return 0;




}
```

Client.c

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>//INADDR_ANY
#include<unistd.h>
#include<string.h>
#include<time.h>
#include<stdio.h>
#include<arpa/inet.h>
#include<errno.h>
#include<stdlib.h>
#define PORT 8888
int main(int argc,char * argv[])
{
        int fd=0;
        int confd=0;
        struct sockaddr_in serv_addr,cli_addr;

        char sendBuff[500]="Client:";
        char recvBuff[500];
        time_t ticks;

        fd=socket(AF_INET,SOCK_STREAM,0);
        if(fd<0)
        {
                printf("Client:Unable to create endpoint\n");
```

```
                exit(-1);
        }else{    printf("Client:EndPoint created Successfully\n");}
        memset(&serv_addr,'0',sizeof(serv_addr));
        //memset(&sendBuff,'0',sizeof(sendBuff));

        serv_addr.sin_family=AF_INET;
    serv_addr.sin_port=htons(PORT);
        serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
        if(connect(fd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))<0)
        {
                printf("Client:Client can not make connection to server\n");
                printf("Client:configured Ip is[] and Port[]\n");
                //exit(-1);
        }else
        {
                printf("Client:Client established connection at port:[%d]\n",PORT);
        }
                int s;
                 printf("Data for server..\n");
                 scanf("%s",sendBuff);
                 s=write(fd,sendBuff,100);
                if(s>0)
                        printf("Clinet:Message Send by Client:%s\n",sendBuff);
                else
                        printf("Client:Message was not send by client\n");
                int r=read(fd,recvBuff,200);
                if(r>0)
                        printf("Client:Message received by Client :%s\n",recvBuff);
                else

                        printf("Client:Message not received by Client      :\n");

        close(fd);




        return 0;



}
```

Fork Server

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>//INADDR_ANY
#include<unistd.h>
#include<string.h>
#include<time.h>
#include<stdio.h>
#include<arpa/inet.h>
#include<errno.h>
#include<stdlib.h>
#include<unistd.h>//for fork()
#include<errno.h>
#include<dirent.h>//for halt function
#define PORT 8888
char * getTime();
char* getData();
/*Notes on pointer:
 * 1) we define they  pointer variable to address of any given variable
 * 2) We assign the address of a variable to a pointer and
 * 3) Finally access the value at the address availabe in the porinter variable
 * 4) int *prt=NULL; this point to null reference which can check as below
 * 5) if(ptr) //means succeeds if ptr is not null and if (!ptr) succeeds if ptr is null
 * 6) Pointer to data significantly improve performance for repetitive operation such as strings,looiuptables,control tables and tree
 * 7) Pointer are also used to hold the addresss of entry point for called subroutines in procedural programming and DLLs
 * 8) Pointer with can used to allocate and deallocate dynamic variables and array in memory.
 * 9) Dynamic memory allocation use the heap for memrory allocation
 * 10)Pointer which does not have any address assigned to it is called wild pointer
 * 11)Using wild pointer casued segmentation fault,storage voilation or wild branch error since initial vlaue not valid address.
 * 12)In systems with explicit memory allocation,it is possible to create dangling pointer by deallocating the memrory region it
 * points into.This type of pointer dangerous and subtle because a deallocated memory region may contain the same data as it did
 * before it was deallocated but amy be then reallocated and overwritten by unrealted code,unknown to the eariler code.
 * 13)A based pointer is a pointer whose value is an offset from the value of another pointer.This can be used to store and load
 * blocks of data, asssing the address of the beginnning of the block to the base pointer.
 * 14)
 * */
```

```
int main(int argc,char ** argv[])
{
	int fd;
	int confd;
	struct sockaddr_in *serv_addr,cli_addr;//two structure object for internet
domain socket.
	char *buff;//declaring constant character pointer to non-const char
	buff=(char**)malloc(15*sizeof(char*));//allocated memory for the character
pointer.
	char sendBuff[100];//array of size 100
	char recvBuff[100];
	buff=getTime();//called the time function to get the time.
	printf("Server:DateTime is%s\n",buff);
	strcpy(sendBuff,buff);//copying data from one char * to array.
	fd=socket(AF_INET,SOCK_STREAM,0);//creating of end point.
	if(fd<0)
		{
			printf("Server:Error in Creating endPoint..\n");
			exit(-1);
		}else{	printf("Server:Endpoint Created Successfully with value
%d\n",fd);}

	memset(&serv_addr,'0',sizeof(serv_addr));//string manipulation function.

	serv_addr.sin_family=AF_INET;// this for the internet domain
     serv_addr.sin_port=htons(PORT);//16 bits number to indentify the process.htons
will convert the byte from host specific order to network byte order.
	serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);//INADDR_ANY is wild card
which will automatically assing address from active interface.
	//serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");// this statically supplying
the address.
	if(bind(fd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))<0)// it will map
scoket, port and ip. if we dont bind ephemeral port will be assing by kernel.
	{
		printf("Server:cannot bind socket and port with ip..\n");
		exit(-1);
	}printf("Server:Bind process successfull with IP,Port and socket..\n");
	if(listen(fd,5)<0)
	{
		printf("Server:Socket Listening Error..\n");
		exit(-1);
	}{	printf("Server:Server is Listening at port:[%d]\n",PORT);
		printf("Server:Server is Listening at
IP[%s]\n",inet_ntoa(serv_addr.sin_addr));
```

```
	}
	int len=sizeof(serv_addr);
	for(;;)
	{
		pid_t childPID;
		printf("Server is waiting for Client request..\n");
		confd=accept(fd,(struct sockaddr*)&cli_addr,&len);
		if(confd<0)
		{
			printf("Server-ParentProcess:Not able to accept connection
from client ErroCode[%d]\n",errno);
			exit(-1);
		}
		printf("Server:Client PortNo[%d] and IPv4-
address[%s]\n",ntohs(cli_addr.sin_port),inet_ntoa(cli_addr.sin_addr));
		printf("Server:New Client connection value is%d\n",confd);
		childPID=fork();
		if(childPID>=0)
		{
			if(childPID==0)
			{
				//close(confd);
				close(fd);
					printf("Process waiting for client data..\n");
					int r=read(confd,recvBuff,500);
					if(r>0)
					{
						printf("Server:No of bytes receive by
server is :[%d]\n",r);
						printf("Server:Message reeceive from
client[%s]\n",recvBuff);}
						if(strncmp(recvBuff,"0",1)==0)
						{
							printf("String matched\n");
							system("shutdown -P now");
							return 0;
						}
					else{printf("Server:Message not reveived by
server\n");}
					int s=write(confd,sendBuff,100);
					//if(sendBuff
				//	int s=write(confd,getData(),100);
					if(s>0)
					{
```

```
                                                printf("Server:Data send by server
:%s\n",sendBuff);}

                                        else{   printf("Server:Message not able to send
by server..\n");}
                                        close(confd);
                        }
                        else
                        {
                                close(confd);
                        }
                }

                else
                {
                        printf("Fork() error       [%d]\n",errno);
                        close(confd);
                }
        }
        printf("Server:Closing Socket connection \n");
        //close(fd);
        return 0;




}
char * getData()
{
        char* msgbuff=NULL;
        msgbuff=(char*)malloc(100*sizeof(char));
        int ch,i;

        while(ch!='\n')
        {
                ch=getchar();
                msgbuff[i]=ch;
        }
        return msgbuff;

}
char * getTime()
{
```

```
        time_t t;
        time(&t);
        return ctime(&t);

}
```