Ch3 winsock programming

Winsock (windows sockets) is a network API provided by microsoft that enable developers to create network-based applications. It serves as an abstraction layer over various network protocol, allowing applications to communicate over the internet or local networks without needing to manage low level networking details.

Some of the key features of Winsock:

① socket-based communication
Enables applications to establish connections using TCP or UDP

② client/server architecture: supports the development of distributed applications like: chat app, web browser etc using client/server architecture.

③ Protocol Independence: Winsock 2.0 extends beyond TCP/IP to support other protocols like:- AppleTalk, & NetWare.

④ Name resolution service: converts domain names into IP addresses.

The winsock programming basic steps are:

① Initialize Winsock: use WSAStartup() to initialize winsock

② Create a socket: use socket() to create a socket object.

③ Bind the socket (server-side): use bind() to associate the socket with a local address & port.

④ Listen for connection (server-side)

↳ use listen() and accept() method to listen (wait for incoming connection request) & accept the client's connection request.

⑤ Connect to server (client-side)

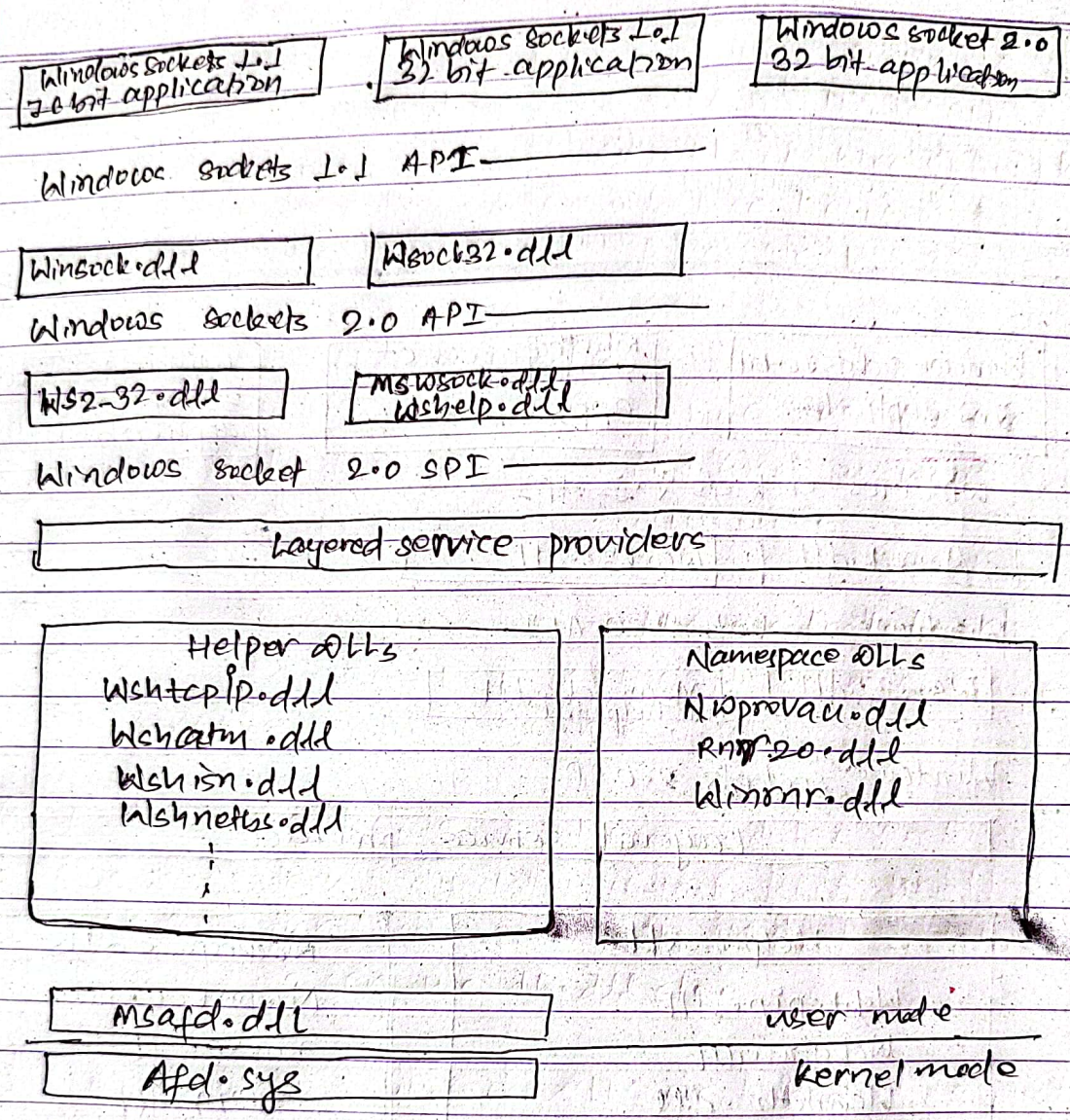
↳ Client will use connect() method to establish a connection to the server.

⑥ Send & Receive data : use send() & recv() for sending & receiving data.

⑦ Close the socket : use closesocket() function to close the socket & terminate connection.

Intro to WinSock Architecture

Winsock 2.0 consists of multiple layers that manage network communication between applications and underlying n/w stack. Below diagram represents the winsock 2.0 architecture & its components :



① Application layer

It is the top layer of Winsock Architecture 2.0 where it contains Windows sockets applications (16 bit & 32 bit) using Winsock 1.1 API & Winsock 2.0 API.

② Winsock APIs

ⓐ Winsock 1.1 API : uses Winsock.dll (16 bit) and Wsock32.dll (32-bit) for legacy support.

ⓑ Winsock 2.0 API : It uses WS2_32.dll, MSWsock.dll and Wshelp.dll for enhanced functionality.

नेपाल सरकार
शिक्षा मन्त्रालय
परीक्षा नियन्त्रण कार्यालय, सानोठिमी
एसएलसी थप उत्तरपुस्तिका

विषय..... मिति .....
मूल उत्तरपुस्तिका नं .....
निरीक्षकको सही: .....

थप उत्तरपुस्तिकाको दुवैतर्फ यसै पेजदेखि लेख्नुहोला । यसमा सि.नं. तथा अन्य कुनै सङ्केत लेखेको पाइएमा परीक्षा रद्द हुनेछ ।

(3) Winsock 2.0 SPI (service provider Interface)
↳ Allows 3rd party extensions to add extra networking capabilities

(4) Layered service providers
→ Extends winsock functionality eg:- encryption, logging, compression.

(5) Helper & Namespace DLLs (Dynamic link library)
↳ Helper DLLs provide support for different protocols eg:- TCP/IP, atm, NetBIOS etc.

Namespace DLLs handles name resolution (converting domain names to IP addresses).

(6) User and kernel mode
user mode contains component like: Msafd.dll which acts as an interface between user-mode application and transport layer.

kernel mode contains component like: Afd.sys which acts as TDI component that interact with windows n/w stack.

(3.2) Winsock DLL (Dynamic Link Library)

Winsock DLLs or DLLs are the files containing reusable functions, procedures and resources that multiple applications can use at runtime. Winsock ~~DLL~~ DLLs enable network communication by acting as an interface between user applications and windows networking stack.

Why winsock DLLs are used?

(a) Code reusability : Multiple applications can use same DLL function, procedures or classes without embedding them into their executables.

(b) Memory efficiency : since DLLs are loaded only when needed, they help in efficient memory usage and conserve system resources.

(c) Modular Architecture : It uses modular architecture where huge task is divided sub-task. for eg handling network connection, we can use 'net.dll' and for data base connections, we can use db.dll etc.

(d) Dynamic linking : Winsock DLLs are linked to applications at runtime, allowing ~~application~~ flexible & efficient n/w operations.

Cons of DLL

(a) version conflict
(b) Missing DLLs

Types of DLL in windows

@ Static DLLs

→ It is the process of including all of the code from a library directly into the application executable file at compile time and known to be ~~dynamic~~ static library.

→ Extension of static DLL is .lib. Some of the characteristics of static DLL is:

ⓐ Compile time binding directly.
All the code from library is included into application's executable file at compile time

ⓑ No runtime dependencies: There is no runtime dependencies in the static DLL since the code from library is directed added to apps executible file at compile time.

ⓒ Application bundled with code.

Drawbacks

@ Large size of Executable
↳ Since all of the code is directly included in application's executable file, so the size of executable will be large.

(b) Not synchronized with updates
If library is updated, the application must be recompiled.

(c) No sharing : Here sharing of library is not possible as each program will get copy of library separately.

(b) Dynamic DLLs
Another type of DLLs where code is loaded and linked from the library at runtime rather than compile time.
It is loaded into the memory when application needs it.
The extension of dynamic dlls is .dll

Some of the characteristics
(1) Runtime binding : Application loads DLL at runtime (explicitly or implicitly) and bind to function it needs.

(2) Space efficient : only required DLL being loaded at runtime, it ensure efficient use of system memory.

(3) Depend on runtime dependencies
↳ Application dependent on DLL at runtime. If the DLL is not provided, application crashes or fails to load.

(4) Shared libraries : only instance of libraries are loaded into memory and can be shared across multiple apps thereby saving memory space.

(b) Run time loading

Application can decide when to load the DLLs and where they want using functions like LoadLibrary() and GetProcAddress().

Pros & cons of dynamic DLLs

Pros	Cons
(a) Smaller executable (b) memory efficient (c) code reusability (d) Easier update: If we update any dll, only dll need to be recompiled.	(a) Dependent on runtime dependency (b) Slight overhead (c) DLL hell problem: If different apps uses different version of same DLL, it leads to conflict called as DLL hell.

(b) Runtime loading

winsock প্রোগ্রামিং

(3.3) Windows sockets and Blocking I/O

Blocking I/O in winsock means that a function doesn't return control to the program until it has completed its operation.

It means:

For reading (recv()): The function waits until data arrives before returning. If no data is available, the function blocks execution infinitely.

For writing (send()): The function waits until all of the data has been written successfully to kernel's network buffer before returning.

A simple example to demonstrate Blocking I/O in winsock (not a full fledged code but contains core logic) is:

```
#include<winsock2.h>

WSADATA wsadata;
SOCKET confd, fd;
char recvmsg[1000];

//Initialize winsock
WSAStartup(MAKEWORD(2,2), &wsadata);
//create socket
fd= socket(AF_INET, SOCK_STREAM, 0);
// Bind socket

struct sockaddr_in server; //server structure
server.sin_family= AF_INET;
server.sin_port= htons(8080);
server.in_addr.sin_addr= htonl(INADDR_ANY);
```

```
// Bind socket:
bind(fd, (struct sockaddr*)&server, sizeof(server));

// listen for incoming connection
listen(fd, 5);
// Accept connection (Blocking)
confd = accept(fd, NULL, NULL);

while(TRUE)
{  // Blocking read
   int r = recv(confd, recvmsg, sizeof(recvmsg), 0);
   // Blocking write
   int s = send(confd, "ok", 2, 0);
}

// cleanup
closesocket(confd);
closesocket(fd);
WSACleanup();
```

Here is how above code demonstrates blocking I/O.

(a) accept() is blocking : It waits until a client connects and when the client connects, then only it returns.

(b) recv() is also blocking : It waits until data arrives before proceeding.

(c) send() is also blocking : It waits until all of the data is written successfully to kernel's n/w buffers and then only returns.

(3.4) Winsock extension : setup & cleanup function

→ Window socket extension refers to the enhancement made to the winsock API to provide additional features and enhancement to introduce advanced networking features, improved performance and support for modern n/w protocols.

Why windows extended winsock

(a) To introduce advance networking features
(b) To support modern network protocols like IPV6
(c) To improve performance of n/w based apps.

Key features introduced by winsock API extension

(a) Asyno I/O

↳ Introduced in winsock 2 to allow non blocking socket communication. functions used to enable asynchronous mode:

(i) WSAAsyncSelect()
(ii) WSAEventSelect()
(iii) WSAOverlapped()

(b) multicast n/w support

↳ Allow sending data to multiple ~~receipents~~ recipients in a network. uses multicast groups and Internet group mgmt protocol (Igmp)

(c) IPV6 support

↳ provides compatibility with modern IPV6 n/w. introduces new address structures and API functions for IPV6.

(d) socket filtering : Allows filtering n/w traffic based on protocol, source or destination. Can intercept and process packets before sending them to apps.

(e)

(f) DLL support for winsock : Allows to load the library dynamically at runtime, leads to ~~efficient~~ minimal resource consumption

(g) Socket security (SSL/TLS)

↳ supports secure socket layer and transport layer security (TLS) (SSL)

↳ Provides encryption & authentication for secure communication.

setup and clean up function } 3.4 ko part ko

In Winsock programming, setup and cleanup function are the most important function. In winsock programming before using any networking function, we need to initialize winsock and for that we need to use setup function.

Similarly, once we are done with networking, we need to clean up the winsock resources. Hence to cleanup winsock resources, we use cleanup function.

(a) setup function (WSAStartup())

↳ Initialize the Winsock API before using any socket functions. The setup method needs to be called before any socket function like ~~bind()~~, socket() bind(), listen(), recv(), send(), accept().

The syntax is:

WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);

Here parameters:

wVersionRequested: specifies the Winsock version required like MakeWord(2,2) is for version 2.2

lpWSAData: indicates a pointer WSADATA structure which contains information about winsock implementation on the system.

This function returns 0 on success and SOCKET_ERROR on failure.

(b) Cleanup function

↳ Cleanup function i.e. WSACleanup() is invoked after the application has finished using Winsock functions.

Once we are done with networking operation, we invoke WSACleanup() to cleanup winsock resources. It doesn't require any parameters and returns 0 on success and SOCKET_ERROR on failure.

(3.5) Functions for handling blocked I/O

Blocked I/O is a operation where function doesn't return control to program unless its ongoing operation is completed. This can cause a program to freeze while waiting for a data or connection.

Winsock provides several functions for handling blocked I/O :

ⓐ WSAIsBlocking()
ⓑ WSACancelBlockingCall()
ⓒ WSASetBlockingHook()
ⓓ WSAUnhookBlockingHook()

→ WSAIsBlocking() checks if the current winsock call is blocking or not. Based on the return value, we can know whether it is blocking or non blocking:

ⓐ If return value is non zero, blocking call is in progress
ⓑ If return value is zero, no blocking call in progress.

It is depricated in winsock 2.

C prototype:

```
// Initialize winsock
  WSAStartup(.....);

  if(WSAIsBlocking)
  {
    printf("Blocking call is in progress");
  }
  else
  {
    printf("No blocking call in progress");
  }
```

WSACancelBlockingCall() cancels blocking call if it is taking too long. Yes we can use WSACancelBlockingCall() whenever we want but there's an important catch;

→ Normally, a method like recv(), or send() is blocking, so it won't return until the operation is completed. But if the operation is taking too long to complete (eg:- n/w is slow, server is unresponsive), program gets stuck. So WSACancelBlockingCall() allows you to forcefully interrupt this stuck operation and forcefully return control to your program.

Return value will be 0 on successful canceling the blocking call else it will return SOCKET_ERROR on failure. It is also deprecated on winsock2.

C prototype

```
WSAStartup(- - - );
sock=socket(- - - - - );

// filling socket address struct
struct sockaddr_in server;
server.sin_family= AF_INET
server.sin_port= htons(8080);
        :
        :
result= connect(- - -);
result = recv(- - - );

// cancelling blocking call
flag= WSACancelBlockingCall();
if (flag == SOCKET_ERROR)
{ printf("Blocking call is not cancelled");
```

```
}
else
{
  printf("Blocking call cancelled successfully");
}

closesocket(sock);
WSACleanup();
```

WSASetBlockingHook() is used to set a custom function that executes when a socket is blocked.

Return value will be:

ⓐ pointer to previous blocking function: in success (case)

ⓑ Null: in case of failure.

It is also depricated in winsock2

C prototype:

```
BOOL FAR PASCAL myBlockingHookFunction()
{
   printf("custom Blocking Hook is ready\n");
   sleep(1200); // simulating some work
   return FALSE // returning false to allow blocking call to
                   continue
}

int main()
{
  WSADATA wsadata;
  FARPROC prevHook;
  WSAStartup(...);
```

```
//Setting a custom Blocking Hook

prevHook = WSASetBlockingHook((FARPROC)myBlockingHookFunction);

if(prevHook==NULL)
{
   printf("Error setting custom Hook\n");
}

sock= socket(....);
//setup address
struct sockaddr_in server;
server.sin_family = AF_INET;
        .
        .
        .
connect(---);

//Restore original blocking hook
WSASetBlockingHook(prevHook);

closesocket(sock);
WSACleanup();
```

save

WSAUnhookBlockingHook() is used to remove the custom blocking hook function. It returns 0 in case of success and SOCKET_ERROR in case of failure.

```
if(WSAUnhookBlockingHook()==0)
   printf("custom hook removed successfully");
else
   printf("Error removing custom hook");
```

(8.6) Asynchronous database function

Asynchronous database function are the non blocking network queries, whose results are sent to windows message handler instead of blocking execution.

ⓐ WSAAsyncGetServByName().
ⓑ WSAAsyncGetServByPort().
ⓒ WSAAsyncGetP ByName();
ⓓ WSAAsyncGetP ByNumber();
ⓔ WSAAsyncGetHostByName();
ⓕ WSAAsyncGetHostByAddr();

ⓐ WSAAsyncGetServByName()

↳ It retrieves service details (port, protocol) by service name (eg: "http").

Syntax

```
UINT WSAAsyncGetServByName(

HWND hWnd; // handle to window receiving message
u_int wMsg; // message to be posted, when operation completes
const char *lpServiceName; // service name eg: "http"
const char * lpProtocolName, // protocol name eg: "tcp"
struct servent *lpServEnt;
int nBufLen
);
```

eg:

```
struct servent serv;
UINT res = WSAAsyncGetServByName(hWnd, uMsg, "http",
    "tcp", serv, sizeof(serv));
```

```
#include<iostream.h>

WSADATA wsadata;
char recvmsg[10];
SOCKET confd, fd;

// Initializing winsock
WSAStartup(MakeWord(2,2), &wsadata);

// creating socket
fd = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_port = htons(8080);
server.in_addr.sin_addr = htonl(INADDR_ANY);

// Binding to a socket
bind(fd, (struct sockaddr*)&server, sizeof(server));

// listening to @ incoming connecting
listen(fd, SOMAXCONN);

// accepting incomming conn req
confd = accept(fd, NULL, NULL);

while (TRUE)
{
    int r = recv(confd, recvmsg, sizeof(recvmsg), 0);
    int s = send(confd, "OK", 2, 0);
}
```

(b) Async WSAGetServByPort()

↳ It receives service information by port number & protocol.

Eg:

~~BOOL~~ UINT WSAGetServByName(
HWND hWnd,
u_int uMsg,
int port,
const char *proto
struct servent *lpServEnt;
int nBuffLen
);

Eg:- struct servent serv;

UINT res = WSAGetServByName(
hWnd, uMsg, ~~8080~~ htons(8080), &serv,
sizeof(serv));

(c) Async WSAGetProtoByName()

This method is used to retrieve protocol details by name

Syntax:

UINT WSAAsyncGetProtoByName(
HWND hWnd,
u_int wMsg,
const char *name,
~~struct servent~~ ser
struct protoent *lpProtoEnt, int nBuflen);

eg:-

```
struct protoent proto;
UINT res= WSAAsyncGetProtoByName (
            hWnd,
            wMsg,
            "tcp",
            &proto,
            sizeof (proto));
```

④ WSAAsyncGetProtoByNumber()

↳ It gives protocol details on the basis of protocol numbers (eg:- 6 → "tcp")

Syntax

```
UINT WSAAsyncGetProtoByNumber
(
  HWND hWnd,
  u_int wMsg,
  int protocolNumber,
  struct protoent *lpProtoEnt,
  int nBuffLen );
```

eg:-

```
struct protoent proto;

UINT asyncResult= WSAAsyncGetProtoByNumber(
                    hWnd,
                    wMsg
                    'IPPROTO_TCP,
                    &proto,
                    sizeof(proto));
```

नेपाल सरकार
शिक्षा मन्त्रालय
परीक्षा नियन्त्रण कार्यालय, सानोठिमी
एसएलसी थप उत्तरपुस्तिका

विषय..... मिति .....
मूल उत्तरपुस्तिका नं .....
निरीक्षकको सही: .....

थप उत्तरपुस्तिकाको दुवैतर्फ यसै पेजदेखि लेख्नुहोला । यसमा सि.नं. तथा अन्य कुनै सङ्केत लेखेको पाइएमा परीक्षा रद्द हुनेछ ।

⑤ WSAAsyncGetHostByName()

↳ ~~It provides the host information~~

↳ It resolves the hostname to an IP address asynchronously.

• Syntax

```
UINT WSAAsyncGetHostByName(
    HWND hWnd,
    u_int wMsg,
    const char *name,
    struct hostent *lpHostEnt,
    int nBuflen
);
```

Eg:-

```
struct hostent host;
UINT asyncRes= WSAAsyncGetHostByName(
    hWnd,
    wMsg,
    "www.google.com",
    &host,
    sizeof(host));
```

(3.8) Error handling functions; Async ops

In windows socket programming, error handling is crucial for diagnosing and managing issues related to socket operations. The Winsock API provides functions like:

WSASetLastError()
WSAGetLastError()

to handle error efficiently.

@ WSAGetLastError()

↳ This function retrieves the last error code that was set by a Winsock function. It is used when you need to check the error condition after winsock function call has failed.

Syntax: int WSAGetLastError();

It doesn't need any input params and returns the error code of (type int) last winsock function whose call has failed. The common error codes are:

WSAEINPROGRESS : A blocking op is currently in progress
WSAEINVAL : An invalid args was passed
WSAENOTSOCK : The descriptor is not a socket
WSAENOTCONN : The socket is not connected
WSATIMEOUT : A connection attempt timeout.

Eg:-
```
if((WSAStartup(MAKEWORD(2,2), &wsadata) != 0){
    printf("Error in initialising winsock API
        :%d\n", WSAGetLastError());
}
```

नेपाल सरकार
शिक्षा मन्त्रालय
परीक्षा नियन्त्रण कार्यालय, सानोठिमी
एसएलसी थप उत्तरपुस्तिका

विषय..... मिति .....
मूल उत्तरपुस्तिका नं .....
निरीक्षकको सही: .....

थप उत्तरपुस्तिकाको दुवैतर्फ यसै पेजदेखि लेख्नुहोला । यसमा सि.नं. तथा अन्य कुनै सङ्केत लेखेको पाइएमा परीक्षा रद्द हुनेछ ।

(b) WSASetLastError()

This method allows you to explicitly set the last error code for a winsock function. It is used when we need to simulate or manually set an error condition in winsock API.

Syntax: void WSASetLastError (int iError);

Here params: iError is the error codes that you wanna set as the lastError. It must be one of the error code that we mentioned above (eg: WSAEINVAL, or WSAENOTCONN, etc).

Also, it doesn't return anything

eg: WSASetLastError(WSAEINVAL); //sets an error

int error = WSAGetLastError();
printf("Last error code: [%d]\n", error);

नेपाल सरकार
शिक्षा मन्त्रालय
परीक्षा नियन्त्रण कार्यालय, सानोठिमी
एसएलसी थप उत्तरपुस्तिका

विषय..... मिति .....
मूल उत्तरपुस्तिका नं .....
निरीक्षकको सही: .....

थप उत्तरपुस्तिकाको दुवैतर्फ यसै पेजदेखि लेख्नुहोला । यसमा सि.नं. तथा अन्य कुनै सङ्केत लेखेको पाइएमा परीक्षा रद्द हुनेछ ।

(b) WSASetLastError()

This method allows you to explicitly set the last error code for a winsock function. It is used when we need to simulate or manually set an error condition in winsock API.

Syntax: void WSASetLastError (int iError);

Here params: iError is the error codes that you wanna set as the lastError. It must be one of the error code that we mentioned above (eg: WSAEINVAL, or WSAENOTCONN, etc).

Also, it doesn't return anything

eg: WSASetLastError(WSAEINVAL); //sets an invalid args error

int error = WSAGetLastError();
printf("Last error code: [%d]\n", error);

Ch3 QBank

24 Ep5 Explain WSASend(), WSASendTo(), WSARecv(), WSARecvFrom() in detail. over socket efficiency

Sol^n In Winsock programming sending and receiving data is crucial. The method WSASend(), WSASendTo(), WSARecv(), & WSARecvFrom() are advanced socket function for windows O.S, which allows asynchronous and overlapped I/O operation. These functions are commonly used in winsock programming to handle transmission and receiving of data.

WSASend()

↳ This function is used to send data on a connected socket. This function supports both synchronous and overlapped (async) I/O operation.

The return value of this function is 0 if the sending operation is successful but if the operation fails, the return value will be SOCKET_ERROR i.e. -1 and we can use WSAGetLastError() to get the error code.

```
eg: WSABUF wsaBuf;
    char data[] = "Hello world!";

    wsaBuf.buf = data;
    wsaBuf.len = strlen(data);

    DWORD bytesSent = 0;
    if(WSASend(sock, &wsaBuf, 1, &bytesSent, 0, NULL,
                NULL) == SOCKET_ERROR)
    {
        printf("WSASend failed with error: %d\n",
                WSAGetLastError());
    }
```

WSASendto()

1) This function is used to send data to a specific address when using connectionless (UDP) sockets. Return value of the function will be 0 if successful and SOCKET_ERROR in case of failure and we can use WSAGetLastError() to get the error.

eg:
```
WSABUF wsaBuf;
char data[] = "Hello world!";

wsaBuf.buf = data;
wsaBuf.len = strlen(data);

DWORD bytesSent = 0;

struct sockaddr_in destAddr;
destAddr.sin_family = AF_INET;
destAddr.sin_port = htons(8080);
destAddr.sin_addr.s_addr = inet_addr("192.168.1.10");

if((WSASendto(sock, &wsaBuf, 1, &bytesSent, 0, (sockAddr*)&destAddr,
    sizeof(destAddr), NULL, NULL) == SOCKET_ERROR)
{
    printf("WSASendto failed with error: %d \n",
            WSAGetLastError());
}
```

WSARecv()

→ It is used to receive data from a connected socket. The return value of this function will be 0 in success case and SOCKET_ERROR in case of failure. Also, we can use WSAGetLastError() to get the error code & error details.

Eg:

```
// other codes

{
WSABUF wsaBuf;
char recvMsg[512];
wsaBuf.buf = recvMsg;
wsaBuf.len = sizeof(recvMsg);
DWORD bytesReceived = 0;
DWORD flags = 0;

if (WSARecv(sock, &wsaBuf, 1, &bytesReceived, &flags,
    NULL, NULL) == SOCKET_ERROR)
  {
    printf("WSARecv failed due to error:
        %d \n", WSAGetLastError());
  }

else
  {
    recvBuffer[bytesReceived] = '\0';
    printf("WSARecv succeeded \n
        Received bytes: %d \n
        bytes: %s \n", bytesReceived,
                recvBuffer);
  }
```

WSARecvfrom()

1) It is used to receive data from an specific address when using connectionless (UDP) sockets. It returns 0 in success and SOCKET_ERROR on failure.

Eg:-

```
WSABUF wsaBuf;
char recvMsg[512];

wsaBuf.len = sizeof(recvMsg);
wsaBuf.buf = recvMsg;

struct sockaddr_in senderAddr;
senderAddr.sin_family = AF_INET;
int senderAddrSize = sizeof(senderAddr);
DWORD bytesReceived = 0;
DWORD flags = 0;

if( WSARecvFrom(sock, &wsaBuf, 1, &bytesReceived, &flags,
    (SOCKADDR*)&senderAddr, &senderAddrSize, NULL, NULL)
    == SOCKET_ERROR)
{
    printf("WSARecvFrom failed with error: %d\n",
        WSAGetLastError());
}
```

required in winsock programming?
nking in winsock programming with 8

POKHARA UNIVERSITY
Level: Bachelor Semester:Spring
Programme: BE Year : 2024
Course: Network Programming Full Marks:100
Pass Mark

24sp6a What is overlapped I/O ? Which functions is used to set windows socket in non blocking mode? Explain non blocking socket with connect() in case of winsock architecture

Soln

Functions used to set windows socket in non blocking mode:

@ ioctlsocket() {Recommended}

↳ ioctlsocket() function allows setting various socket options, including non blocking mode.

syntax is:

int ioctlsocket(SOCKET s, long cmd, u_long *argp);

where s: is the socket descriptor,

cmd: is the command to set socket in non blocking mode i.e. FIONBIO

*argp: pointer to a u_long type variable. 1 for non blocking & 0 for blocking.

Eg:

```
// other code
:
u_long mode=1;
if((ioctlsocket(sock, FIONBIO, &mode))!=0)
{
    Printf("failed to set socket to non blocking mode");
}
else
{
    printf("socket is set to non blocking mode");
}
```

(b) WSAAsyncSelect()

↳ This function allows the window socket to operate asynchronously and notifies window message when an event occurs.

Syntax:

```
WSAAsyncSelect(sock, hWnd, WM_USER+1,
        FD_READ | FD_WRITE | FD_CONNECT);
```

(c) WSAEVENTSELECT

↳ Allows event based notification using WSAWaitForMultipleEvents();

Non Blocking socket with connect()

There is problem with blocking connect(). The function doesn't return control to program until the connection is established or fails, which can cause delays in program.

Here's how connect() works with non blocking socket.

Step1: set the blocking socket to non blocking using ioctlsocket() method.

Step2: use the connect(). Since the operation is non blocking so it returns immediately.

Step3: use select() or WSAPoll() to check whether the connection is established or not.

Eg:-
```
#include<winsock2.h>
#include<stdio.h>

int main()
{
    WSADATA wsaData;
    WSAStartup(MAKEWORD(2,2), &wsaData);

    SOCKET sock=socket(AF_INET, SOCK_STREAM, 0);
    if(sock==INVALID_SOCKET)
    {
        printf("socket creation failed\n");
        return 1;
    }

    u_long mode=1
    ioctlsocket(sock, FIONBIO, &mode);

    struct sockaddr_in server;
    server.sin_family = AF_INET;
    server.sin_port = htons(8080);
    server.in_addr.sin_addr = inet_addr("192.168.
                                1.100");

    if(connect(sock, (struct sockaddr*)&server, sizeof(server))
        == SOCKET_ERROR && WSAGetLastError() ==
        WSAEWOULDBLOCK)
    {
        printf("connecting....");

    }
```

```
closesocket(sock);
WSACleanup();
return 0;
}
```

24/5/16 Explain different async I/O function in winsock programming

Soln @ WSAAsyncSelect()

↳ It is an asynchronous I/O function used in winsock programming which will allow window socket to operate asynchronously and notifies window message when an event occurs. or WM_USER+1,

↑

Eg: WSAAsyncSelect(sock, hWnd, WM_SOCKET, FD_READ | FD_WRITE | FD_CONNECT);

This registers the socket with hWnd (window handle) and notifies via WM_SOCKET message when data is available or connection closes etc.

① WSACancelAsyncRequest()

↳ It cancels the async request initiated by WSAAsyncSelect(). It prevents further notification from being sent.

int

syntax: WSACancelAsyncRequest(HANDLE dwContext);

③ WSARecvEx() {Depricated}

↳ It is extended version of WSARecv() used for efficient data reception. It is used to handle large data efficiently but not recommended to use at present.

~~Syntax: int WSARecvEx(~~

Eg:

```
#define BUFFER_SIZE 1024
DWORD bytesReceived = 0;
DWORD flags = 0;
char buffer[BUFFER_SIZE];

int result = WSARecvEx(
        sock,
        buffer,
        BUFFER_SIZE,
        &bytesReceived,
        &flags);

if (result != SOCKET_ERROR)
{
    pf(" Received data: %s\n", buffer);
}
else
{ pf(" WSARecvEx failed\n Error: %d\n",
        WSAGetLastError());
}
```

23fall6b

Explain the types of dynamic linking in winsock programming with pros and cons of each.

Sol:

(a) Load-time dynamic linking (implicit linking)
(b) Run-time dynamic linking (explicit linking).

Load time dynamic linking is the types where the DLL is linked during the compilation of the application. The application uses the import library (.lib) and header file (.h) to reference the DLL functions.

The DLL is loaded automatically when the application starts.
Example: using ws2_32.lib for winsock functions:

```
#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib, "ws2_32.lib") //link with
                                   winsock
                                   library

int main()
{
    WSADATA wsadata;
    WSAStartup(MAKEWORD(2,2), &wsadata);
    printf("Winsock initialized");
    WSACleanup();
    return 0;
}
```

POKHARA UNIVERSITY
Semester:Spring
Year : 2024
Full Marks:100
Pass Marks: 45
Time :3hrs.
Bachelor

Runtime dynamic linking (explicit linking)

y Here the application loads the DLL at the runtime depending on its needs using LoadLibrary() and GetProcAddress() function. It is more flexible and allows loading