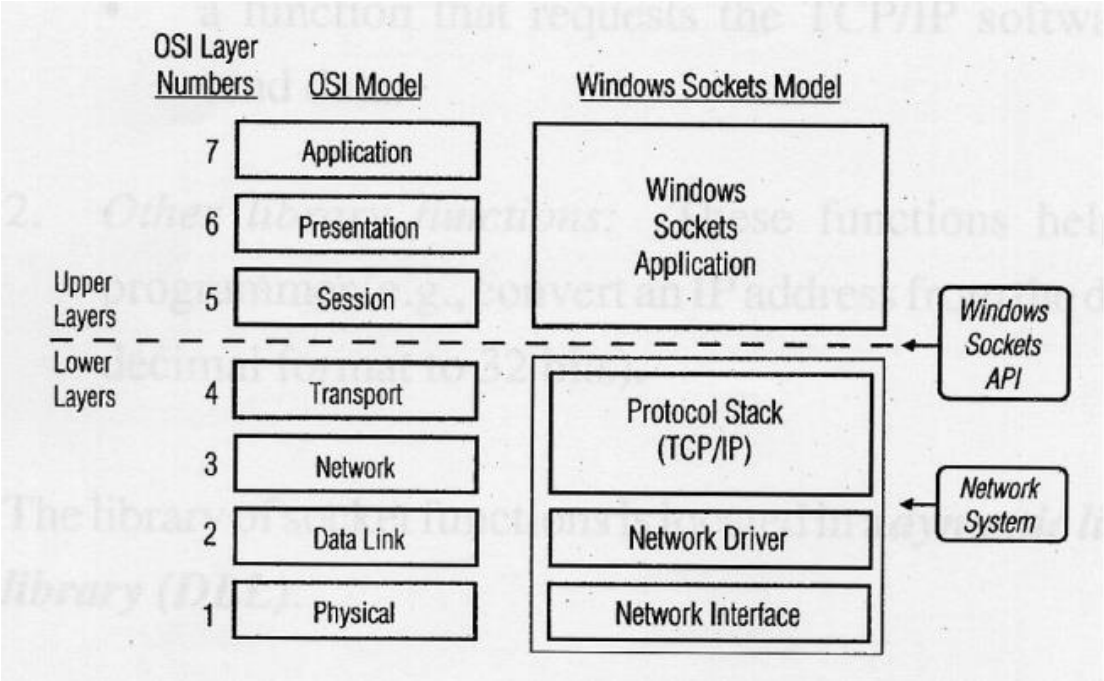
Windows Socket

– Different operating systems have different socket interfaces.

– In Mircosoft Windows, the socket interface is called Windows Socket, or WinSock, or Windows Socket API (application program interface).

Role of WinSock:



– WinSock provides a library of functions. These functions can be classified into two types:

• Primary socket functions: These functions perform specific operations to interact with the TCP/IP protocol software. Examples:

– a function that requests the TCP/IP software to establish a TCP connection to a remote server;

– a function that requests the TCP/IP software to send data.

Other library functions: These functions help the programmer (e.g., convert an IP address from the dotted decimal format to 32 bits).

– The library of socket functions is located in a dynamic linked library (DLL).

• A DLL is a library that is loaded into main memory only when the library is first used.

- WinSock is an interface but it is not a protocol.

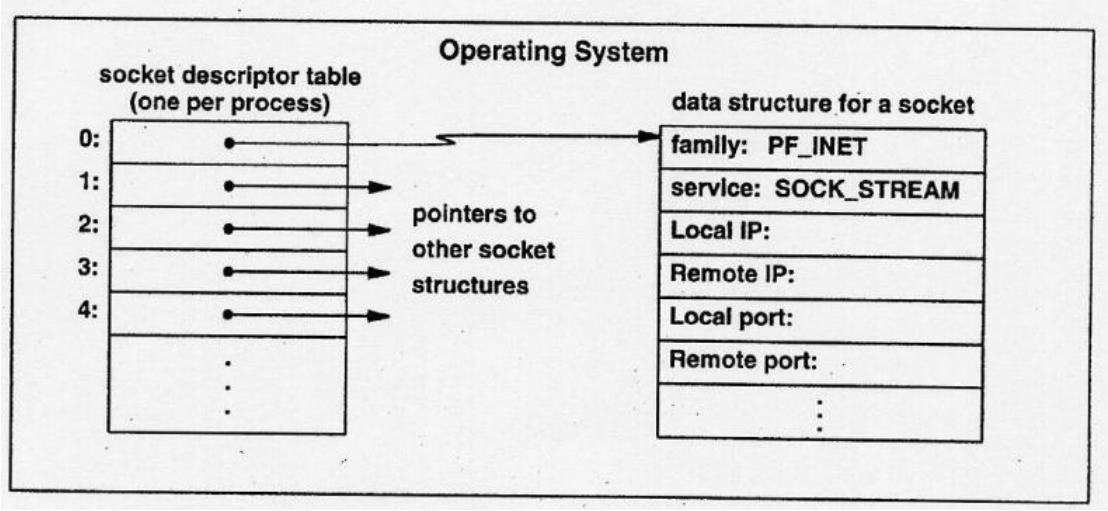
• If the client and the server use the same protocol suite (TCP/IP), then they can communicate even if they use different application program interfaces:

– There are cases where an interface to a protocol suite is adopted to another protocol suite.• e.g., WinSock API for the IPX/SPX protocol suite

– IPX (Internetwork Packet Exchange) is a networking protocol from Novell that interconnects networks that use Novell's Netware clients and servers. IPX is a datagram protocol. IPX works at the Network layer of communication protocols and is connectionless

Socket Descriptor

– Each socket is identified by an integer called socket descriptor, which is an unsigned integer.
– A process may open multiple sockets for multiple concurrent communication sessions (e.g., a web server is serving multiple browsers simultaneously).
• Windows keeps a table of socket descriptors for each process.
– Each socket descriptor is associated with a pointer, which points to a data structure that holds the information about the communication session of that socket.
– The data structure contains many fields, and they will be filled as the application calls additional WinSock functions.



Data types for TCP/IP endpoint address

```
struct  sockaddr_in {        /* struct to hold an address        */
    u_short sin_family;      /* type of address(always AF_INET) */
    u_short sin_port;        /* protocol port number             */
    struct  in_addr sin_addr; /* IP address (declared to be      */
                             /*  u_long on some systems)         */
    char    sin_zero[8];     /* unused (set to zero)             */
};
```

```
struct  in_addr {
    u_long    s_addr ;  /*  IP
   address  */
};
```

Assigning IP address
–Example 1
» The IP address of a server is 158.182.7.63. Its decimal value is 2662729535. We can specify the endpoint address for this server as follows:

```
struct sockaddr_in ServerAddr;
…
ServerAddr.sin_family = AF_INET;
ServerAddr.sin_port = 2000;
ServerAddr.sin_addr.s_addr = htonl (2662729535);
```

–Example 2

» We specify the endpoint address for a server as follows:

```
struct sockaddr_in ServerAddr;
…
ServerAddr.sin_family = AF_INET;
ServerAddr.sin_port = 2000;
ServerAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

» where the symbolic constant INADDR_ANY represents a wildcard address that matches any of the computer's IP address(es).

Sketch of a TCP Client and a TCP Server

– Using TCP, both the client and the server use

– Three major steps for communication:

- Initialize sockets.
- Communicate between sockets.
- Close the sockets.

Initialize Sockets

There are three initialization steps:

» Initialize WinSock DLL.

» Create a socket.

» Assign an endpoint address to a socket.

Initialize WinSock DLL

– Before using WinSock, an application calls WSAStartup().

» Then Windows binds to the WinSock DLL and allocates the necessary resources to this application.

– WSAStartup() requires two arguments:

» The 1st argument specifies the version of the requested WinSock.

» The 2nd argument returns information about the version of the WinSock that is actually used

```
#include <winsock.h>
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData) ;
```

Example

```
WSADATA     wsadata ;
…
WSAStartup (MAKEWORD(2, 0) , &wsadata );
```

Create a Socket

```
SOCKET socket ( int af,  int type,  int protocol, );
```

• Assign an Endpoint Address to a Socket

– After creating a socket, a server must assign its endpoint address to this socket.

» Then the client can identify the server's socket and send requests to this server.

–Steps

» The server specifies its endpoint address. In other words, it assigns the following three attribute values to a structure of type sockaddr_in:

protocol family (AF_INET for TCP/IP)
IP address,
port number.
» The server calls bind() to bind its endpoint address to the newly created socket.
int bind (SOCKET s, const struct sockaddr FAR * name, int namelen);
–Example
struct sockaddr_in ServerAddr;
…
/* Specify the server's endpoint address in ServerAddr here */
…
bind (s, (struct sockaddr *) &ServerAddr, sizeof(ServerAddr));

Communicate Between Sockets
Main Steps
– Setup a TCP connection:
» A client initiates to setup a TCP connection to a server.
» The server waits for and accepts connection requests from the clients.
– Send and receive data.
– Close the TCP connection.
In socket programming, a TCP connection and its associated socket are closed simultaneously.
Therefore, we will consider the 3rd step in the next subsection.
Client Initiates a TCP Connection
– After creating a socket, a client calls connect() to establish a TCP connection to a server.
– The function connect() has three arguments:
» the descriptor of the client's socket;
» endpoint address of the server;
» length of the 2nd argument.
int connect(SOCEKT s,Const struct sockaddr FAR* name, int namelen);
struct sockaddr_in ServerAddr;
…
/* Specify the server's endpoint address in ServerAddr here */
…
connect (s, (struct sockaddr*)&ServerAddr, sizeof(ServerAddr));

•Server Listens to Connection Requests
– After creating a socket, the server calls listen() to place this socket in passive mode.
» Then the socket listens and accepts incoming connection requests from the clients.
– Multiple clients may send requests to a server.
» The function listen() tells the OS to queue the connection requests for the server's socket.
– The function listen() has two arguments:
» the server's socket;
» the maximum size of the queue.
– The function listen() applies only to sockets used with TCP.
int listen(SOCKET s, int backlog);
–Example

listen(s, 1);
• Server Accepts a Connection Request
– The server calls accept() to
» extract the next incoming connection request from the queue
» Then the server creates a new socket for this connection request and returns the descriptor of this new socket.
– Remarks
» The server uses the new socket forthe new connection only. When this connection ends, the server closes this socket.
» The server still uses the original socket to accept additional connection requests.
» The function accept() applies only to stream (TCP) sockets.

SOCKET accept(SOCK s, struct sockaddr FAR* addr, int FAR* addrlen);
–Example
```
struct sockaddr_in ClientAddr;
int len;
SOCKET nsock;
…
len = sizeof ( ClientAddr );
nsock = accept( s, (struct sockaddr *)&ClientAddr, &len );
```

Send Data
– Both client and server calls send() to send data across a connection.
– The function send() copies the outgoing data into buffers in the OS kernel, and allows the application to continue execution while the data is being sent across the network.
– If the buffers become full, the call to send() may block temporarily until free buffer space is available for the new outgoing data.
int send(SOCKET s, const char FAR * buf, int len, int flags,);
–Example
```
char   *message="Hello world!";
…
send(s, message, strlen(message),0);
```
Receive Data
– Both the client and server call recv() to receive data through a TCP connection.
– Before calling recv(), the application must allocate a buffer for storing the data to be received.
– The function recv() extracts the data that has arrived at the specified socket, and copies them to the application's buffer.
int recv(SOCKET s, char FAR* buf, int len, int flags);
– Example: receive a small message with at most 5 characters
```
char  buf[5],  *bptr;
int buflen;
…
bptr = buf;
buflen = 5;
recv(s, bptr, buflen, 0);
```

Close the Sockets
Close a Socket

– Once a client or server finishes using a socket, it calls closesocket() to
» terminate the TCP connection associated with this socket, and
» deallocate this socket.

int closesocket(SOCKET s);

–Example

closesocket(s);

Clean Up

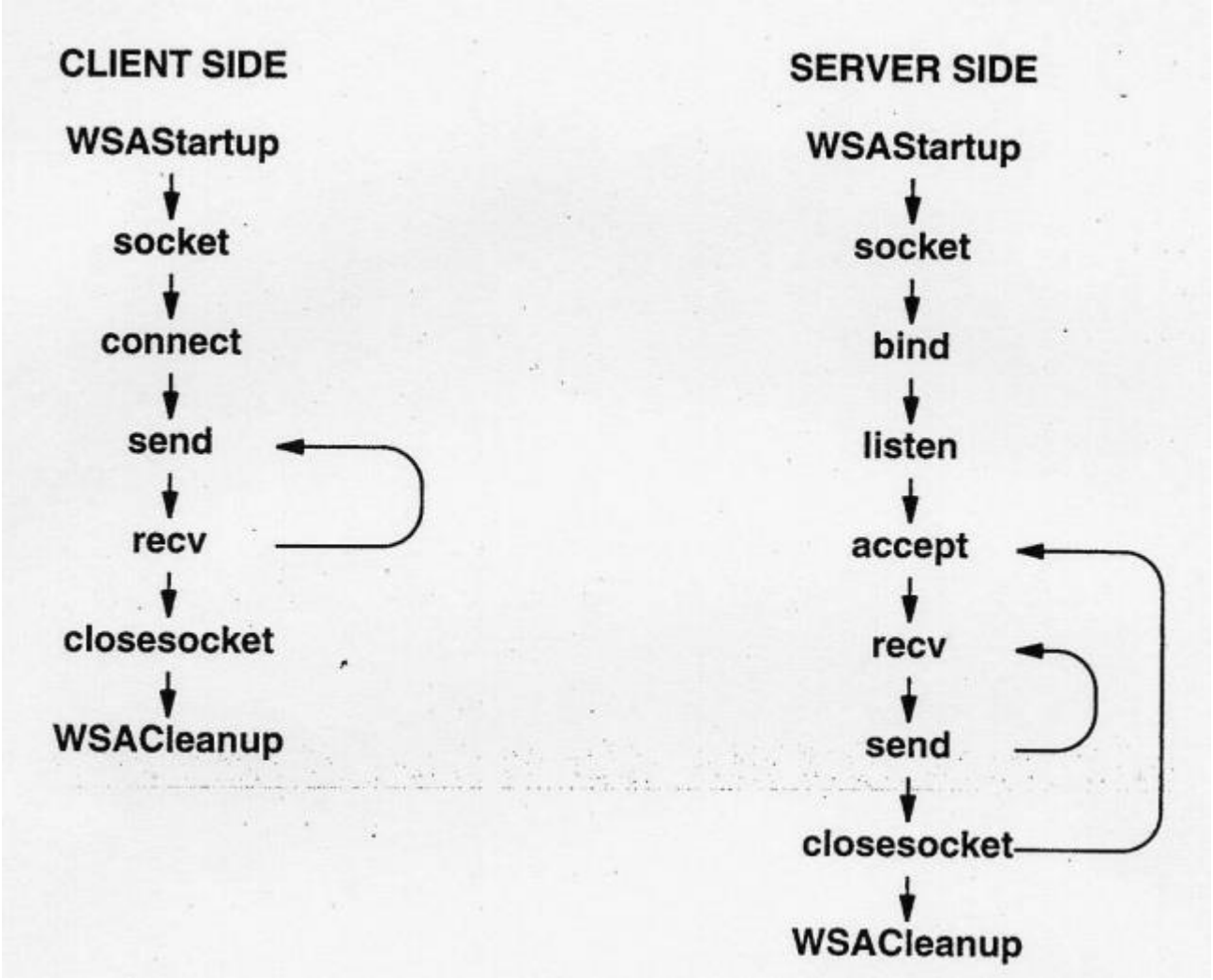
– When an application finishes using sockets, it must call WSACleanup() to deallocate all data structures and socket bindings.

int WSACleanup (void);

–Example

WSACleanup();

Sequence of Execution of WinSock Functions



/* tcpclient.c*/

```
#include  <stdio.h>
#include  <winsock2.h>
#define   WSVERS   MAKEWORD(2,0)
main()
{   WSADATA          wsadata;
SOCKET          s;
struct sockaddr_in ServerAddr;
```

```
char              *message="Hello",
buf[5],  *bptr;
int i,  buflen,  count;
/*  call WSAStartup() and socket() */
WSAStartup ( WSVERS , &wsadata ) ;
s = socket ( PF_INET , SOCK_STREAM , 0 );
/*  call connect() to connect to the server */
ServerAddr.sin_family = AF_INET ;
ServerAddr.sin_port = htons(2000) ;
ServerAddr.sin_addr.s_addr = htonl(2662729535);
connect(s, (struct sockaddr *)
&ServerAddr, sizeof(ServerAddr));
/* call send() to send a message to the
server  */
send(s, message, strlen(message), 0);
/* call recv() to receive a message from
the server  */
bptr = buf;   buflen = 5;
recv(s, bptr, buflen, 0);
/* Echo the received message from the
server */
for (i=0; i<5; ++i){
printf("%c", buf[i]);
}
printf("\n%s\n", "Bye bye!");
/* call closesocket() */
closesocket(s);
/* call WSACleanup() */
WSACleanup();
}
```

/*tcpserver.c*/

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

main()
{
WSADATA            wsadata;
SOCKET             s,  nsock;
struct sockaddr_in ServerAddr, ClientAddr;
char buf[5],  *bptr;
int i,  buflen,  count;
/* call WSAStartup() */
WSAStartup ( MAKEWORD(2,0) , &wsadata ) ;
/* call socket() */
s = socket ( PF_INET , SOCK_STREAM , 0 );
/* call bind() */
ServerAddr.sin_family = AF_INET;
ServerAddr.sin_port = htons(2000);
ServerAddr.sin_addr.s_addr = htonl(INADDR_ANY);
bind ( s, (struct sockaddr *) &ServerAddr, sizeof(ServerAddr) );
```

```
/* call listen() */
listen(s,1);
/* call accept() */
i = sizeof ( ClientAddr );
nsock = accept (s, (struct sockaddr *) &ClientAddr , &i );
/* call recv() to receive a message from
the client  */
bptr = buf;
buflen = 5;
recv ( nsock , bptr , buflen , 0 );
/* call send() to send the message
back to the client  */
send ( nsock, buf, strlen(buf), 0);
/* call closesocket() */
closesocket ( nsock );
closesocket ( s );
/* call WSACleanup() */
WSACleanup();
}
```