Tenouk C & C++ | Winsock Prog. Example 1 | Main | Winsock Prog. Example 3 >|
Site Index | Download | Linux Socket | Winsock in .NET

WINSOCK 2
WINDOWS SOCKET: PROGRAM EXAMPLES PART 2

What do we have in this chapter?

1. socket()

2. Remarks

3. Some note

4. Notes for IrDA Sockets

5. Checking Winsock DLL Version Program Example

6. bind()

7. Return Values

8. Remarks

9. Using socket() and bind() Program Example

10. Another Program Example Using bind() and socket()

11. Notes for IrDA Sockets

12. closesocket()

13. Return Values

14. Remarks

15. Some Note

16. Notes for IrDA Sockets

17. Notes for ATM

Winsock Tutorial	My Training Period: hours
Winsock: Story 1 Winsock: Story 2 Winsock: Story 3 Winsock: Example 1 Winsock: Example 2 Winsock: Example 3 Winsock: Example 4 Winsock: Example 5 Winsock: Example 6 Winsock: Example 7 Winsock: Example 8 Winsock: Example 9	Machine's OS is standalone Windows Xp Pro with SP2 except whenever mentioned. Compiler used was Visual C++ 2003 .Net 1.1. Beware the codes that span more than one line. Program examples have been tested for Non Destructive Test. All information compiled for Windows 2000 (NT5.0) above and...

1. The story was discussed at Winsock introduction story.
2. Related functions, structures and macros used in the program examples have been dumped at Winsock structure & function 1 and Winsock structure & function 2.
3. Other related and required information (if any) not available in no. 2 can be found at MSDN & Visual C++ online reference.

Abilities:

- Able to understand the basic of networking such as TCP/IP.

- Able to understand Winsock implementation and operations through the APIs and program examples.

- Able to gather, understand and use the Winsock functions, structures and macros in your programs.

- Able to build programs that use Microsoft C/Standard C programming language and Winsock APIs.

socket()

Item	Description
Function	socket().
Use	To create a socket that is bound to a specific service provider.
Prototype	SOCKET socket(int af, int type, int protocol);
Parameters	See below.
Return value	See below.
Include file	<winsock2.h>
Library	ws2_32.lib.
Remark	See below

Table 1

af - [in] Address family specification.
type - [in] Type specification for the new socket.
The following are the only two type specifications supported for Windows Sockets 1.1 and for Winsock 2 there are support for RAW Socket, SOCK_RAW.

Type	Meaning
SOCK_STREAM	Provides sequenced, reliable, two-way, connection-based byte streams with an OOB data transmission mechanism. Uses TCP (Transport Control Protocol) for the Internet address family.
SOCK_DGRAM	Supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length. Uses UDP (User Datagram Protocol) for the Internet address family.

Table 2

In Windows Sockets 2, many new socket types will be introduced and no longer need to be specified, since an application can dynamically discover the attributes of each available transport protocol through the WSAEnumProtocols() function. Socket type definitions appear in winsock2.h, which will be periodically updated as new **socket types**, **address families**, and **protocols** are defined.

protocol - [in] Protocol to be used with the socket that is specific to the indicated address family.

Return Values

If no error occurs, socket() returns a descriptor referencing the new socket. Otherwise, a value of INVALID_SOCKET is returned, and a specific error code can be retrieved by calling WSAGetLastError().

Error code	Meaning
WSANOTINITIALISED	A successful WSAStartup() call must occur before using this function.

WSAENETDOWN	The network subsystem or the associated service provider has failed.
WSAEAFNOSUPPORT	The specified address family is not supported.
WSAEINPROGRESS	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEMFILE	No more socket descriptors are available.
WSAENOBUFS	No buffer space is available. The socket cannot be created.
WSAEPROTONOSUPPORT	The specified protocol is not supported.
WSAEPROTOTYPE	The specified protocol is the wrong type for this socket.
WSAESOCKTNOSUPPORT	The specified socket type is not supported in this address family.

Table 3

Remarks

The socket() function causes a socket descriptor and any related resources to be allocated and bound to a specific transport-service provider. Winsock will utilize the first available service provider that supports the requested combination of **address family**, **socket type** and **protocol** parameters. The socket that is created will have the overlapped attribute as a default. For Windows, the Microsoft-specific socket option, SO_OPENTYPE, defined in **mswsock.h** can affect this default. Sockets without the overlapped attribute can be created by using WSASocket(). All functions that allow overlapped operation (WSASend(), WSARecv(), WSASendTo(), WSARecvFrom(), and WSAIoctl()) also support non-overlapped usage on an overlapped socket if the values for parameters related to overlapped operation are null.

When selecting a protocol and its supporting service provider this procedure will only choose a base protocol or a protocol chain, not a protocol layer by itself. Unchained protocol layers are not considered to have partial matches on type or af either. That is, they do not lead to an error code of WSAEAFNOSUPPORT or WSAEPROTONOSUPPORT if no suitable protocol is found.

Some note

The manifest constant AF_UNSPEC continues to be defined in the header file but its use is strongly discouraged, as this can cause ambiguity in interpreting the value of the protocol parameter. **Connection-oriented sockets** such as SOCK_STREAM provide **full-duplex connections**, and must be in a connected state before any data can be sent or received on it. A connection to another socket is created with a connect() call. Once connected, data can be transferred using send() and recv() calls. When a session has been completed, a closesocket() must be performed.

The communications protocols used to implement a reliable, connection-oriented socket ensure that data is not lost or duplicated. If data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, the connection is considered broken and subsequent calls will fail with the error code set to WSAETIMEDOUT.

Connectionless, message-oriented sockets allow sending and receiving of **datagrams** to and from arbitrary peers using sendto() and recvfrom(). If such a socket is connected to a specific peer, datagrams can be sent to that peer using send() and can be received only from this peer using recv(). Support for sockets with type SOCK_RAW (RAW socket) is not required, but service providers are encouraged to support raw sockets as practicable. On Windows NT, raw socket support requires administrative privileges.

Notes for IrDA Sockets

For IrDA, keep the following in mind:

- The af_irda.h header file must be explicitly included.
- Only SOCK_STREAM is supported; the SOCK_DGRAM type is not supported by IrDA.
- The protocol parameter is always set to 0 for IrDA.

Checking Winsock DLL Version Program Example

```
// Microsoft Development Environment 2003 - Version 7.1.3088
// Copyright (r) 1987-2002 Microsoft Corporation. All Right Reserved
// Microsoft .NET Framework 1.1 - Version 1.1.4322
// Copyright (r) 1998-2002 Microsoft Corporation. All Right Reserved
//
// Run on Windows XP Pro machine, version 2002, SP 2
//
// <windows.h> already included
// WINVER = 0x0501 for Xp already defined in windows.h

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

int main()
{
WORD wVersionRequested;
WSADATA wsaData;
int wsaerr;

// Using MAKEWORD macro, Winsock version request 2.2
wVersionRequested = MAKEWORD(2, 2);
```

```c
        wsaerr = WSAStartup(wVersionRequested, &wsaData);
        if (wsaerr != 0)
        {
          /* Tell the user that we could not find a usable WinSock DLL.*/
            printf("The Winsock dll not found!\n");
            return 0;
        }

        else
        {
            printf("The Winsock dll found!\n");
            printf("The status: %s.\n", wsaData.szSystemStatus);
        }

        /* Confirm that the WinSock DLL supports 2.2.           */
        /* Note that if the DLL supports versions greater      */
        /* than 2.2 in addition to 2.2, it will still return   */
        /* 2.2 in wVersion since that is the version we         */
        /* requested.                                          */
        if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
        {
          /* Tell the user that we could not find a usable WinSock DLL.*/
            printf("The dll do not support the Winsock version %u.%u!\n",
        LOBYTE(wsaData.wVersion),HIBYTE(wsaData.wVersion));
            WSACleanup();
            return 0;
        }

        else
        {
            printf("The dll supports the Winsock version %u.%u!\n",
        LOBYTE(wsaData.wVersion),HIBYTE(wsaData.wVersion));
            printf("The highest version this dll can support: %u.%u\n",
        LOBYTE(wsaData.wHighVersion), HIBYTE(wsaData.wHighVersion));
        }

        //////////Create a socket////////////////////////
        //Create a SOCKET object called m_socket.
        SOCKET m_socket;
```

```c
// Call the socket function and return its value to the m_socket variable.
// For this application, use the Internet address family, streaming sockets, and
// the TCP/IP protocol.
// using AF_INET family, TCP socket type and protocol of the AF_INET - IPv4
m_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

// Check for errors to ensure that the socket is a valid socket.
if (m_socket == INVALID_SOCKET)
{
    printf("Error at socket(): %ld\n", WSAGetLastError());
    WSACleanup();
    return 0;
}
else
{
    printf("socket() is OK!\n");
}

return 0;
}
```
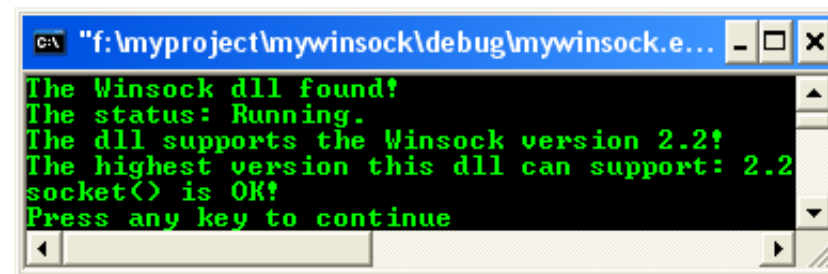
Sample output:

Figure 1

bind()

Item	Description
Function	bind().

Use	Associates a local address with a socket.
Prototype	`int bind(SOCKET s, const struct sockaddr* name, int namelen);`
Parameters	s - [in] Descriptor identifying an unbound socket. name - [in] Address to assign to the socket from the sockaddr structure. namelen - [in] Length of the value in the name parameter, in bytes.
Return value	See below.
Include file	<winsock2.h.>
Library	ws2_32.lib
Remark	See below.

Table 4

Return Values

If no error occurs, bind() returns zero. Otherwise, it returns SOCKET_ERROR, and a specific error code can be retrieved by calling WSAGetLastError()

Error code	Meaning
WSANOTINITIALISED	A successful WSAStartup() call must occur before using this function.
WSAENETDOWN	The network subsystem has failed.
WSAEACCES	Attempt to connect datagram socket to broadcast address failed because setsockopt() option SO_BROADCAST is not enabled.
WSAEADDRINUSE	A process on the computer is already bound to the same fully-qualified address and the socket has not been marked to allow address reuse with SO_REUSEADDR. For example, the IP address and port are bound in the AF_INET case). (See the SO_REUSEADDR socket option under setsockopt().)
WSAEADDRNOTAVAIL	The specified address is not a valid address for this computer.
WSAEFAULT	The name or namelen parameter is not a valid part of the user address space, the namelen parameter is too small, the name parameter contains an incorrect address format for the associated address family, or the first two bytes of the memory block specified by name does not match the address family associated with the socket descriptor s.
WSAEINPROGRESS	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEINVAL	The socket is already bound to an address.

WSAENOBUFS	Not enough buffers available, too many connections.
WSAENOTSOCK	The descriptor is not a socket.

Table 5

Remarks

The bind() function is used on an unconnected socket before subsequent calls to the connect() or listen() functions. It is used to bind to either connection-oriented (**stream** – TCP – Transmission Control Protocol) or connectionless (**datagram** – UDP – User Datagram Protocol) sockets. When a socket is created with a call to the socket() function, it exists in a namespace (address family), but it has no name assigned to it. Use the bind() function to establish the local association of the socket by assigning a local name to an unnamed socket. A name consists of three parts when using the Internet address family:

1. The address family.
2. A host address.
3. A port number that identifies the application.

In Windows Sockets 2, the name parameter is not strictly interpreted as a pointer to a sockaddr structure. It is cast this way for Windows Sockets 1.1 compatibility. Service providers are free to regard it as a pointer to a block of memory of size namelen. The first 2 bytes in this block (corresponding to the sa_family member of the sockaddr structure) must contain the address family that was used to create the socket. Otherwise, an error WSAEFAULT occurs. If an application does not care what local address is assigned, specify the manifest constant value **ADDR_ANY** for the sa_data member of the name parameter. This allows the underlying service provider to use any appropriate network address, potentially simplifying application programming in the presence of **multihomed hosts** that is, hosts that have more than one network interface and address.

For TCP/IP, if the **port is specified as zero**, the service provider assigns a unique port to the application with a value between **1024** and **5000**. The application can use getsockname() after calling bind() to learn the address and the port that has been assigned to it. If the Internet address is equal to INADDR_ANY, getsockname() cannot necessarily supply the address until the socket is connected, since several addresses can be valid if the host is multihomed. Binding to a specific port number other than port 0 is discouraged for client applications, since there is a danger of

conflicting with another socket already using that port number. When using bind() with the SO_EXCLUSIVEADDR or SO_REUSEADDR socket option, the socket option must be set prior to executing bind() to have any affect.

Using socket() and bind() Program Example

```
// Microsoft Development Environment 2003 - Version 7.1.3088
// Copyright (r) 1987-2002 Microsoft Corporation. All Right Reserved
// Microsoft .NET Framework 1.1 - Version 1.1.4322
// Copyright (r) 1998-2002 Microsoft Corporation. All Right Reserved
//
// Run on Windows XP Pro machine, version 2002, SP 2
//
// <windows.h> already included
// WINVER = 0x0501 for Xp already defined in windows.h

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

int main()
{
WORD wVersionRequested;
WSADATA wsaData;
int wsaerr;

// Using MAKEWORD macro, Winsock version request 2.2
wVersionRequested = MAKEWORD(2, 2);

wsaerr = WSAStartup(wVersionRequested, &wsaData);
if (wsaerr != 0)
{
    /* Tell the user that we could not find a usable */
    /* WinSock DLL.*/
    printf("The Winsock dll not found!\n");
    return 0;
}
else
{
```

```c
        printf("The Winsock dll found!\n");
        printf("The status: %s.\n", wsaData.szSystemStatus);
    }

    /* Confirm that the WinSock DLL supports 2.2.*/
    /* Note that if the DLL supports versions greater    */
    /* than 2.2 in addition to 2.2, it will still return */
    /* 2.2 in wVersion since that is the version we       */
    /* requested.                                         */
    if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
    {
        /* Tell the user that we could not find a usable */
        /* WinSock DLL.*/
        printf("The dll do not support the Winsock version %u.%u!\n", LOBYTE(wsaData.wVersion),
HIBYTE(wsaData.wVersion));
        WSACleanup();
        return 0;
    }
    else
    {
        printf("The dll supports the Winsock version %u.%u!\n", LOBYTE(wsaData.wVersion),
HIBYTE(wsaData.wVersion));
        printf("The highest version this dll can support: %u.%u\n",
LOBYTE(wsaData.wHighVersion), HIBYTE(wsaData.wHighVersion));
    }

    //////////Create a socket//////////////////////////
    //Create a SOCKET object called m_socket.
    SOCKET m_socket;

    // Call the socket function and return its value to the m_socket variable.
    // For this application, use the Internet address family, streaming sockets, and
    // the TCP/IP protocol.
    // using AF_INET family, TCP socket type and protocol of the AF_INET - IPv4
    m_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    // Check for errors to ensure that the socket is a valid socket.
    if (m_socket == INVALID_SOCKET)
    {
```

```
        printf("Error at socket(): %ld.\n", WSAGetLastError());
        WSACleanup();
        return 0;
    }
    else
    {
        printf("socket() is OK!\n");
    }

    /////////////////bind()/////////////////////////////
    // Create a sockaddr_in object and set its values.
    sockaddr_in service;

    // AF_INET is the Internet address family.
    service.sin_family = AF_INET;
    // "127.0.0.1" is the local IP address to which the socket will be bound.
    service.sin_addr.s_addr = inet_addr("127.0.0.1");
    // 55555 is the port number to which the socket will be bound.
    service.sin_port = htons(55555);

    // Call the bind function, passing the created socket and the sockaddr_in structure as
    parameters.
    // Check for general errors.

    if (bind(m_socket, (SOCKADDR*)&service, sizeof(service)) == SOCKET_ERROR)
    {
        printf("bind() failed: %ld.\n", WSAGetLastError());
        closesocket(m_socket);
        return 0;
    }
    else
    {
        printf("bind() is OK!\n");
    }

    return 0;
    }
```

Sample output:

Figure 2

Another Program Example Using bind() and socket()

Another example demonstrates the use of the bind() function.

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

int main()
{
  // Initialize Winsock
  WSADATA wsaData;
  int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
  if (iResult != NO_ERROR)
    printf("Error at WSAStartup().\n");
  else
    printf("Winsock dll is available.\n");

  // Create a SOCKET for listening for
  // incoming connection requests
  SOCKET ListenSocket;
  ListenSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (ListenSocket == INVALID_SOCKET)
  {
    printf("Error at socket(): %ld.\n", WSAGetLastError());
    WSACleanup();
    return 0;
  }
  else
  {
    printf("socket() is OK!\n");
  }

  // The sockaddr_in structure specifies the address family,
  // IP address, and port for the socket that is being bound.
  sockaddr_in service;
  // Address family - internet IPv4
  service.sin_family = AF_INET;
  // IP address
  service.sin_addr.s_addr = inet_addr("127.0.0.1");
  // Port number
  service.sin_port = htons(55555);

  // Bind the socket.
```

```c
    if (bind( ListenSocket, (SOCKADDR*) &service, sizeof(service)) == SOCKET_ERROR)
    {
      printf("bind() failed.\n");
      closesocket(ListenSocket);
      return 0;
    }
    else
    {
      printf("bind() is OK!\n");
    }

    WSACleanup();
    return 0;
}
```
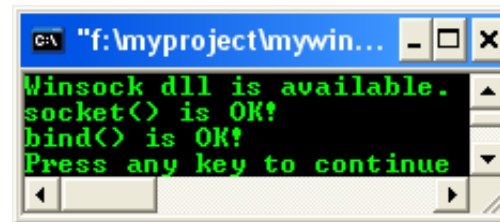
Sample output:

Figure 3

Notes for IrDA Sockets

Infrared Data Association, a term commonly referred to as IrDA, can be described as:

1. A protocol suite designed to provide wireless, walk-up, line-of-sight connectivity between devices.
2. The semi-transparent red window found on many laptops and some desktop computers.
3. An organization that creates, promotes, and standardizes IrDA technology (got to irda.org for more information on the IrDA organization).
4. The af_irda.h header file must be explicitly included.
5. Local names are not exposed in IrDA. IrDA client sockets therefore, must never call the bind function before the connect() function. If the IrDA socket was previously bound to a service name using bind, the connect function will fail with SOCKET_ERROR.

6. If the service name is of the form "LSAP-SELxxx," where xxx is a decimal integer in the range 1-127, the address indicates a specific LSAP-SEL xxx rather than a service name. Service names such as these allow server applications to accept incoming connections directed to a specific LSAP-SEL, without first performing an ISA service name query to get the associated LSAP-SEL. One example of this service name type is a non-Windows device that does not support IAS.

closesocket()

Item	Description
Function	closesocket().
Use	Closes an existing socket.
Prototype	int closesocket(SOCKET s);
Parameters	s - [in] Descriptor identifying the socket to close.
Return value	See below.
Include file	<winsock2.h>
Library	ws2_32.lib
Remark	See below.

Table 6

Return Values

If no error occurs, closesocket() returns zero. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code can be retrieved by calling WSAGetLastError().

Error code	Meaning
WSANOTINITIALISED	A successful WSAStartup() call must occur before using this function.
WSAENETDOWN	The network subsystem has failed.
WSAENOTSOCK	The descriptor is not a socket.
WSAEINPROGRESS	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEINTR	The (blocking) Windows Socket 1.1 call was canceled through WSACancelBlockingCall().
WSAEWOULDBLOCK	The socket is marked as non-blocking and SO_LINGER is set to a nonzero time-out value.

Table 7

Remarks

The closesocket() function closes a socket. Use it to release the socket descriptor s so that further references to s fail with the error WSAENOTSOCK. If this is the last reference to an underlying socket, the associated naming information and queued data are discarded. Any pending blocking, asynchronous calls issued by any thread in this process are canceled without posting any notification messages. Any pending overlapped send and receive operations (WSASend()/ WSASendTo()/ WSARecv()/ WSARecvFrom() with an overlapped socket) issued by any thread in this process are also canceled. Any event, completion routine, or completion port action specified for these overlapped operations is performed. The pending overlapped operations fail with the error status WSA_OPERATION_ABORTED.

An application should always have a matching call to closesocket() for each successful call to socket() to return any socket resources to the system. The semantics of closesocket are affected by the socket options SO_LINGER and SO_DONTLINGER as follows (SO_DONTLINGER is enabled by default; SO_LINGER is disabled).

Option	Interval	Type of close	Wait for close?
SO_DONTLINGER	Do not care	Graceful	No
SO_LINGER	Zero	Hard	No
SO_LINGER	Nonzero	Graceful	Yes

Table 8

If SO_LINGER is set with a zero time-out interval (that is, the linger structure members l_onoff is not zero and l_linger is zero), closesocket() is not blocked even if queued data has not yet been sent or acknowledged. This is called a hard or abortive close, because the socket's virtual circuit is reset immediately, and any unsent data is lost. Any recv() call on the remote side of the circuit will fail with WSAECONNRESET. If SO_LINGER is set with a nonzero time-out interval on a blocking socket, the closesocket() call blocks on a blocking socket until the remaining data has been sent or until the time-out expires. This is called a **graceful disconnect**. If the time-out expires before all data has been sent, the Windows Sockets implementation terminates the connection before closesocket() returns.

Enabling SO_LINGER with a nonzero time-out interval on a non-blocking socket is not recommended. In this case, the call to closesocket() will fail with an error of WSAEWOULDBLOCK if the close operation cannot be completed immediately. If closesocket() fails with WSAEWOULDBLOCK the socket handle is still valid, and a disconnect is not initiated. The application must call closesocket() again to close the socket. If SO_DONTLINGER is set on a stream socket by setting the l_onoff member of the LINGER structure to zero, the closesocket() call will return immediately and does not receive WSAEWOULDBLOCK whether the socket is blocking or non-blocking. However, any data queued for transmission will be sent, if possible, before the underlying

socket is closed. This is also called a **graceful disconnect**. In this case, the Windows Sockets provider cannot release the socket and other resources for an arbitrary period, thus affecting applications that expect to use all available sockets. This is the default behavior (SO_DONTLINGER is set by default).

Some Note

To assure that all data is sent and received on a connection, an application should call shutdown() before calling closesocket(). Also note, an FD_CLOSE network event is not posted after closesocket() is called. Here is a summary of closesocket() behavior:

- If SO_DONTLINGER is enabled (the default setting) it always returns immediately - connection is gracefully closed in the background.
- If SO_LINGER is enabled with a zero time-out: it always returns immediately - connection is reset/terminated.
- If SO_LINGER is enabled with a nonzero time-out:
  – With a blocking socket, it blocks until all data sent or time-out expires.
  – With a non-blocking socket, it returns immediately indicating failure.

Notes for IrDA Sockets

For IrDA, keep the following in mind:

- The af_irda.h header file must be explicitly included.
- The standard linger options are supported.
- Although IrDA does not provide a graceful close, IrDA will defer closing until receive queues are purged. Thus, an application can send data and immediately call the socket function, and be confident that the receiver will copy the data before receiving an FD_CLOSE message.

Notes for ATM

The following are important issues associated with connection teardown when using Asynchronous Transfer Mode (ATM) and Windows Sockets 2:

- Using the closesocket() or shutdown() functions with SD_SEND or SD_BOTH results in a RELEASE signal being sent out on the control channel. Due to ATM's use of separate signal and data channels, it is possible that a RELEASE signal could reach the remote end before the last of the data reaches its destination, resulting in a loss of that data. One possible solutions is programming a sufficient delay between the last data sent and the closesocket() or shutdown() function calls for an ATM socket.

- Half close is not supported by ATM.
- Both abortive and graceful disconnects result in a RELEASE signal being sent out with the same cause field. In either case, received data at the remote end of the socket is still delivered to the application.

Further reading and digging:

1. Linux Sockets: Story and program examples.
2. Linux and TCP/IP.
3. A complete info on Windows socket reference from MSDN which include managed and unmanaged API doc.
4. Structure, enum, union and typedef story can be found struct, enum, union & typedef tutorial.
5. For Multibytes, Unicode characters and Localization please refer to Multibyte, Unicode and wide characters (Story) and Win32 Windows & Users tutorial (Implementation).
6. Windows data type information is Win32 - Windows data types.
7. Check the best selling C / C++ and Windows books at Amazon.com.

Tenouk C & C++ | Winsock Prog. Example 1 | Main | Winsock Prog. Example 3 >|
Site Index | Download | Linux Socket | Winsock in .NET