

WINSOCK 2

WINDOWS SOCKET: PROGRAM EXAMPLES

PART 5

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 protocol, client, server and [TCP/IP](#) suite.
- 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.

recv()

Item	Description
Function	recv().
Use	Receives data from a connected or bound socket.
Prototype	int recv(SOCKET s, char* buf, int len, int flags);
Parameters	s - [in] Descriptor identifying a connected socket. buf - [out] Buffer for the incoming data. len - [in] Length of buf, in bytes flags - [in] Flag specifying the way in which the call is made.
Return value	See below.
Include file	<winsock2.h>
Library	ws2_32.lib
Remark	See below.

Table 1

Return Values

If no error occurs, `recv()` returns the number of bytes received. If the connection has been gracefully closed, the return value is 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 <code>WSAStartup()</code> call must occur before using this function.
WSAENETDOWN	The network subsystem has failed.
WSAEFAULT	The <code>buf</code> parameter is not completely contained in a valid part of the user address space.
WSAENOTCONN	The socket is not connected.
WSAEINTR	The (blocking) call was canceled through <code>WSACancelBlockingCall()</code> .
WSAEINPROGRESS	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAENETRESET	The connection has been broken due to the keep-alive activity detecting a failure while the operation was in progress.
WSAENOTSOCK	The descriptor is not a socket.
WSAEOPNOTSUPP	<code>MSG_OOB</code> was specified, but the socket is not stream-style such as type <code>SOCK_STREAM</code> , OOB data is not supported in the communication domain associated with this socket, or the socket is unidirectional and supports only send operations.
WSAESHUTDOWN	The socket has been shut down; it is not possible to receive on a socket after <code>shutdown()</code> has been invoked with <code>how</code> set to <code>SD_RECEIVE</code> or <code>SD_BOTH</code> .
WSAEWOULDBLOCK	The socket is marked as non-blocking and the receive operation would block.
WSAEMSGSIZE	The message was too large to fit into the specified buffer and was truncated.
WSAEINVAL	The socket has not been bound with <code>bind()</code> , or an unknown flag was specified, or <code>MSG_OOB</code> was specified for a socket with <code>SO_OOBINLINE</code> enabled or (for byte stream

	sockets only) len was zero or negative.
WSAECONNABORTED	The virtual circuit was terminated due to a time-out or other failure. The application should close the socket as it is no longer usable.
WSAETIMEDOUT	The connection has been dropped because of a network failure or because the peer system failed to respond.
WSAECONNRESET	The virtual circuit was reset by the remote side executing a hard or abortive close. The application should close the socket as it is no longer usable. On a UDP-datagram socket this error would indicate that a previous send operation resulted in an ICMP "Port Unreachable" message.

Table 2

Remarks

The `recv()` function is used to read incoming data on connection-oriented sockets, or connectionless sockets. When using a **connection-oriented protocol**, the sockets must be connected before calling `recv()`. When using a **connectionless protocol**, the sockets must be bound before calling `recv()`. The local address of the socket must be known. For server applications, use an explicit `bind()` function or an implicit `accept()` or `WSAAccept()` function. Explicit binding is discouraged for client applications. For client applications, the socket can become bound implicitly to a local address using `connect()`, `WSAConnect()`, `sendto()`, `WSASendTo()`, or `WSAJoinLeaf()`. For connected or connectionless sockets, the `recv()` function restricts the addresses from which received messages are accepted. The function only returns messages from the remote address specified in the connection. Messages from other addresses are (silently) discarded.

For **connection-oriented sockets** (type `SOCK_STREAM` for example), calling `recv()` will return as much information as is currently available—up to the size of the buffer specified. If the socket has been configured for in-line reception of OOB data (socket option `SO_OOBINLINE`) and OOB data is yet unread, only OOB data will be returned. The application can use the `ioctlsocket()` or `WSAIoctl()` `SIOCATMARK` command to determine whether any more OOB data remains to be read.

For connectionless sockets (type `SOCK_DGRAM` or other message-oriented sockets), data is extracted from the first enqueued datagram (message) from the destination address specified by the `connect()` function. If the datagram or message is larger than the buffer specified, the buffer is filled with the first part of the datagram, and `recv()` generates the error `WSAEMSGSIZE`. For unreliable protocols (for example, UDP) the excess data is lost; for reliable protocols, the

data is retained by the service provider until it is successfully read by calling `recv()` with a large enough buffer.

If no incoming data is available at the socket, the `recv()` call blocks and waits for data to arrive according to the blocking rules defined for `WSARecv()` with the `MSG_PARTIAL` flag not set unless the socket is non-blocking. In this case, a value of `SOCKET_ERROR` is returned with the error code set to `WSAEWOULDBLOCK`. The `select()`, `WSAAsyncSelect()`, or `WSAEventSelect()` functions can be used to determine when more data arrives.

If the socket is connection oriented and the remote side has shut down the connection gracefully, and all data has been received, a `recv()` will complete immediately with zero bytes received. If the connection has been reset, a `recv()` will fail with the error `WSAECONNRESET`. The flags parameter can be used to influence the behavior of the function invocation beyond the options specified for the associated socket. The semantics of this function are determined by the socket options and the flags parameter. The latter is constructed by using the bitwise OR operator with any of the following values.

Value	Meaning
MSG_PEEK	Peeks at the incoming data. The data is copied into the buffer but is not removed from the input queue. The function subsequently returns the amount of data that can be read in a single call to the <code>recv()</code> (or <code>recvfrom()</code>) function, which may not be the same as the total amount of data queued on the socket. The amount of data that can actually be read in a single call to the <code>recv()</code> (or <code>recvfrom()</code>) function is limited to the data size written in the <code>send()</code> or <code>sendto()</code> function call.
MSG_OOB	Processes Out Of Band (OOB) data.

Table 3

Program Example

The following example demonstrates the use of the `recv()` function on server and client.

```
// 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
// Server program, using TCP
```

```

#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("Server: The Winsock dll not found!\n");
return 0;
}
else
{
printf("Server: The Winsock dll found!\n");
printf("Server: 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("Server: The dll do not support the Winsock version %u.%u!\n",
LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion));
WSACleanup();
return 0;
}
else
{
printf("Server: The dll supports the Winsock version %u.%u!\n",
LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion));
printf("Server: 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("Server: Error at socket(): %ld\n", WSAGetLastError());
    WSACleanup();
    return 0;
}
else
{
    printf("Server: 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("Server: bind() failed: %ld.\n", WSAGetLastError());
    closesocket(m_socket);
    return 0;
}
else
{
    printf("Server: bind() is OK!\n");
}

```

```

}

// Call the listen function, passing the created socket and the maximum number
of allowed
// connections to accept as parameters. Check for general errors.
if (listen(m_socket, 10) == SOCKET_ERROR)
    printf("Server: listen(): Error listening on socket %ld.\n", WSAGetLastError());
else
{
    printf("Server: listen() is OK, I'm waiting for connections...\n");
}

// Create a temporary SOCKET object called AcceptSocket for accepting
connections.
SOCKET AcceptSocket;

// Create a continuous loop that checks for connections requests. If a connection
// request occurs, call the accept function to handle the request.
printf("Server: Waiting for a client to connect...\n" );
printf("***Hint: Server is ready...run your client program...***\n");
// Do some verification...
while (1)
{
    AcceptSocket = SOCKET_ERROR;
    while (AcceptSocket == SOCKET_ERROR)
    {
        AcceptSocket = accept(m_socket, NULL, NULL);
    }
    // else, accept the connection...
    // When the client connection has been accepted, transfer control from the
    // temporary socket to the original socket and stop checking for new
connections.
    printf("Server: Client Connected!\n");
    m_socket = AcceptSocket;
    break;
}

int bytesSent;
int bytesRecv = SOCKET_ERROR;
char sendbuf[200] = "This string is a test data from server";
// initialize to empty data...
char recvbuf[200] = "";

// Send some test string to client...
printf("Server: Sending some test data to client...\n");
bytesSent = send(m_socket, sendbuf, strlen(sendbuf), 0);

```

```

if (bytesSent == SOCKET_ERROR)
    printf("Server: send() error %ld.\n", WSAGetLastError());
else
{
    printf("Server: send() is OK.\n");
    printf("Server: Bytes Sent: %ld.\n", bytesSent);
}

// Receives some test string from client...and client
// must send something lol...
bytesRecv = recv(m_socket, recvbuf, 200, 0);

if (bytesRecv == SOCKET_ERROR)
    printf("Server: recv() error %ld.\n", WSAGetLastError());
else
{
    printf("Server: recv() is OK.\n");
    printf("Server: Received data is: \"%s\"\n", recvbuf);
    printf("Server: Bytes received: %ld.\n", bytesRecv);
}

WSACleanup();
return 0;
}

```

```

C:\> "f:\myproject\mywinsock\Debug\mywinsock.exe"
Server: The Winsock dll found!
Server: The status: Running.
Server: The dll supports the Winsock version 2.2!
Server: The highest version this dll can support: 2.2
Server: socket() is OK!
Server: bind() is OK!
Server: listen() is OK, I'm waiting for connections...
Server: Waiting for a client to connect...
***Hint: Server is ready...run your client program...***

```

Figure 1

The following is the client program. As usual, you have to run the server program first.

```

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

int main()
{
    // Initialize Winsock
    WSADATA wsaData;
    int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
    if (iResult != NO_ERROR)

```

```

printf("Client: Error at WSASStartup().\n");
else
printf("Client: WSASStartup() is OK.\n");

// Create a SOCKET for connecting to server
SOCKET ConnectSocket;
ConnectSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (ConnectSocket == INVALID_SOCKET)
{
printf("Client: Error at socket(): %ld.\n", WSAGetLastError());
WSACleanup();
return 0;
}
else
printf("Client: socket() is OK.\n");

// The sockaddr_in structure specifies the address family,
// IP address, and port of the server to be connected to.
sockaddr_in clientService;
clientService.sin_family = AF_INET;
clientService.sin_addr.s_addr = inet_addr("127.0.0.1");
clientService.sin_port = htons(55555);

// Connect to server.
if (connect(ConnectSocket, (SOCKADDR*)&clientService, sizeof(clientService))
== SOCKET_ERROR)
{
printf("Client: Failed to connect.\n");
WSACleanup();
return 0;
}
else
printf("Client: connect() is OK.\n");

// Declare and initialize variables.
int bytesSent;
int bytesRecv = SOCKET_ERROR;
char sendbuf[100] = "Client: Sending some data.";
char recvbuf[100] = "";

while(bytesRecv == SOCKET_ERROR )
{
bytesRecv = recv(ConnectSocket, recvbuf, 100, 0);
if (bytesRecv == 0 || bytesRecv == WSAECONNRESET)
{
printf("Client: Connection Closed.\n");
break;
}
}

```

```

}
else
{
    printf("Client: recv() is OK.\n");
    printf("Client: Bytes received: %ld\n", bytesRecv);
}
}

// Send and receive data.
bytesSent = send(ConnectSocket, sendbuf, strlen(sendbuf), 0);
printf("Client: Bytes sent: %ld\n", bytesSent);
WSACleanup();
return 0;
}

```

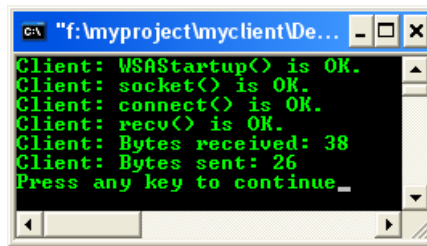


Figure 2

And the previous server's console output.

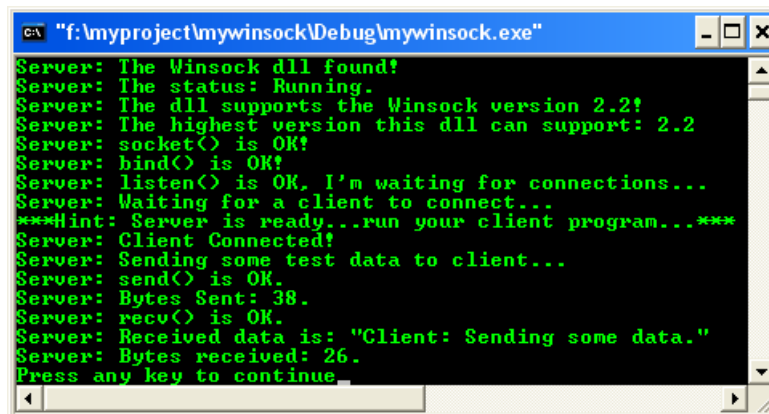


Figure 3

send()

Item	Description
Function	send().
Use	Sends data on a connected socket.
Prototype	<code>int send(SOCKET s, const char* buf, int len, int flags);</code>

Parameters	<p>s - [in] Descriptor identifying a connected socket.</p> <p>buf - [in] Buffer containing the data to be transmitted.</p> <p>len - [in] Length of the data in buf, in bytes.</p> <p>flags - [in] Indicator specifying the way in which the call is made.</p>
Return value	See below.
Include file	<winsock2.h>
Library	ws2_32.lib
Remark	See below.

Table 4

Return Values

If no error occurs, send returns the total number of bytes sent, which can be less than the number indicated by len. 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.
WSAEACCES	The requested address is a broadcast address, but the appropriate flag was not set. Call setsockopt() with the SO_BROADCAST socket option to enable use of the broadcast address.
WSAEINTR	A blocking Windows Sockets 1.1 call was canceled through WSACancelBlockingCall().
WSAEINPROGRESS	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEFAULT	The buf parameter is not completely contained in a valid part of the user address space.
WSAENETRESET	The connection has been broken due to the keep-alive activity detecting a failure while the operation was in progress.
WSAENOBUFS	No buffer space is available.
WSAENOTCONN	The socket is not connected.
WSAENOTSOCK	The descriptor is not a socket.
WSAEOPNOTSUPP	MSG_OOB was specified, but the socket

	is not stream-style such as type SOCK_STREAM, OOB data is not supported in the communication domain associated with this socket, or the socket is unidirectional and supports only receive operations.
WSAESHUTDOWN	The socket has been shut down; it is not possible to send on a socket after shutdown() has been invoked with how set to SD_SEND or SD_BOTH.
WSAEWOULDBLOCK	The socket is marked as non-blocking and the requested operation would block.
WSAEMSGSIZE	The socket is message oriented, and the message is larger than the maximum supported by the underlying transport.
WSAEHOSTUNREACH	The remote host cannot be reached from this host at this time.
WSAEINVAL	The socket has not been bound with bind(), or an unknown flag was specified, or MSG_OOB was specified for a socket with SO_OOBINLINE enabled.
WSAECONNABORTED	The virtual circuit was terminated due to a time-out or other failure. The application should close the socket as it is no longer usable.
WSAECONNRESET	The virtual circuit was reset by the remote side executing a hard or abortive close. For UDP sockets, the remote host was unable to deliver a previously sent UDP datagram and responded with a "Port Unreachable" ICMP packet. The application should close the socket as it is no longer usable.
WSAETIMEDOUT	The connection has been dropped, because of a network failure or because the system on the other end went down without notice.

Table 5

Remarks

The send() function is used to write outgoing data on a connected socket. For message-oriented sockets, care must be taken not to exceed the maximum packet size of the underlying provider, which can be obtained by using getsockopt() to retrieve the value of socket option SO_MAX_MSG_SIZE. If the

data is too long to pass atomically through the underlying protocol, the error WSAEMSGSIZE is returned, and no data is transmitted.

The successful completion of a send does not indicate that the data was successfully delivered.

If no buffer space is available within the transport system to hold the data to be transmitted, send will block unless the socket has been placed in non-blocking mode. On non-blocking stream oriented sockets, the number of bytes written can be between 1 and the requested length, depending on buffer availability on both client and server computers. The select(), WSAAsyncSelect() or WSAEventSelect() functions can be used to determine when it is possible to send more data.

Calling send with a zero len parameter is permissible and will be treated by implementations as successful. In such cases, send() will return zero as a valid value. For message-oriented sockets, a zero-length transport datagram is sent. The flags parameter can be used to influence the behavior of the function beyond the options specified for the associated socket. The semantics of this function are determined by the socket options and the flags parameter. The latter is constructed by using the bitwise OR operator with any of the following values.

Value	Meaning
MSG_DONTROUTE	Specifies that the data should not be subject to routing. A Windows Sockets service provider can choose to ignore this flag.
MSG_OOB	Sends OOB data (stream-style socket such as SOCK_STREAM only).

Table 6

For IrDA Sockets the af_irda.h header file must be explicitly included.