

# WINSOCK 2

## WINDOWS SOCKET: PROGRAM EXAMPLES

### PART 3

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.

#### listen()

Item	Description
Function	listen().
Use	<a href="#">int listen(SOCKET s, int backlog);</a>
Prototype	Places a socket in a state in which it is listening for an incoming connection.
Parameters	<a href="#">s</a> - [in] Descriptor identifying a bound, unconnected socket.  <a href="#">backlog</a> - [in] Maximum length of the queue of pending connections. If set to SOMAXCONN, the underlying service provider responsible for socket <a href="#">s</a> will set the backlog to a maximum reasonable value. There is no standard provision to obtain the actual backlog value.
Return value	See below.
Include file	<winsock2.h>
Library	ws2_32.lib
Remark	See below.

Table 1

## Return Values

If no error occurs, listen() 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.
WSAEADDRINUSE	The socket's local address is already in use and the socket was not marked to allow address reuse with SO_REUSEADDR. This error usually occurs during execution of the bind() function, but could be delayed until this function if the bind was to a partially wildcard address (involving ADDR_ANY) and if a specific address needs to be committed at the time of this function.
WSAEINPROGRESS	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEINVAL	The socket has not been bound with bind().
WSAEISCONN	The socket is already connected.
WSAEMFILE	No more socket descriptors are available.
WSAENOBUFS	No buffer space is available.
WSAENOTSOCK	The descriptor is not a socket.
WSAEOPNOTSUPP	The referenced socket is not of a type that supports the listen operation.

Table 2

## Remarks

To accept connections, a socket is first created with the socket() function and bound to a local address with the bind() function, a backlog for incoming connections is specified with listen(), and then the connections are accepted with the accept() function. Sockets that are **connection oriented** those of type SOCK\_STREAM for example, are used with listen(). The socket s is put into passive mode where incoming connection requests are acknowledged and queued pending acceptance by the process. The listen() function is typically used by servers that can have more than one connection request at a time. If a connection request arrives and the queue is full, the client will receive an error with an indication of WSAECONNREFUSED.

If there are no available socket descriptors, listen() attempts to continue to function. If descriptors become available, a later call to listen() or accept() will

refill the queue to the current or most recent backlog, if possible, and resume listening for incoming connections.

An application can call `listen()` more than once on the same socket. This has the effect of updating the current backlog for the listening socket. Should there be more pending connections than the new backlog value, the excess pending connections will be reset and dropped. For IrDA sockets the `af_irda.h` header file must be explicitly included.

## Compatibility

The backlog parameter is limited (silently) to a reasonable value as determined by the underlying service provider. Illegal values are replaced by the nearest legal value. There is no standard provision to find out the actual backlog value.

## 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 = WSStartup(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;

// 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: %d.\n", WSAGetLastError());
    closesocket(m_socket);
    return 0;
}
else
{
    printf("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, 1) == SOCKET_ERROR)
    printf("listen(): Error listening on socket %d.\n", WSAGetLastError());
else
{
    printf("listen() is OK, I'm waiting for connections...\n");
}

return 0;
}

```

```

C:\> "f:\myproject\mywinsock\debug\mywinsock.exe"
The Winsock dll found!
The status: Running.
The dll supports the Winsock version 2.2!
The highest version this dll can support: 2.2
socket() is OK!
bind() is OK!
listen() is OK, I'm waiting for connections...
Press any key to continue

```

Figure 1

Another example that demonstrates the use of the listen() function.

### Program Example

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

int main()
{
    // Initialize Winsock
    WSADATA wsaData;
    int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
    if (iResult != NO_ERROR)
        printf("Server: Error at WSAStartup().\n");
    else
        printf("Server: WSAStartup() is OK!\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("Server: Error at socket(): %ld\n", WSAGetLastError());
        WSACleanup();
        return 0;
    }
    else
        printf("Server: 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;
    service.sin_family = AF_INET;
    service.sin_addr.s_addr = inet_addr("127.0.0.1");
    service.sin_port = htons(55555);

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

```

// Listen for incoming connection requests
// on the created socket
if (listen(ListenSocket, 1) == SOCKET_ERROR)
    printf("Server: listen(): Error listening on socket.\n");

printf("Server: I'm listening on socket, waiting for connection...\n");
WSACleanup();
return 0;
}

```

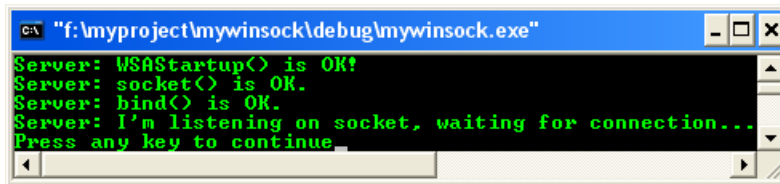


Figure 2

At this time our server is listening for connection on port 55555 but it still can't accept (establish) connection from clients or other servers.

### accept()

Item	Description
Function	accept().
Header	Permits an incoming connection attempt on a socket.
Prototype	<code>SOCKET accept(SOCKET s, struct sockaddr* addr, int* addrlen);</code>
Parameters	<p><code>s</code> - [in] Descriptor that identifies a socket that has been placed in a listening state with the listen() function. The connection is actually made with the socket that is returned by accept.</p> <p><code>addr</code> - [out] Optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the address family that was established when the socket from the sockaddr() structure was created.</p> <p><code>addrlen</code> - [in, out] Optional pointer to an integer that contains the length of addr.</p>
Return value	See below.
Include file	<winsock2.h>
Library	ws2_32.lib
Remark	See below.

Table 3

### Return Values

If no error occurs, accept() returns a value of type SOCKET that is a descriptor for the new socket. This returned value is a handle for the socket on which the actual connection is made. Otherwise, a value of INVALID\_SOCKET is returned,

and a specific error code can be retrieved by calling `WSAGetLastError()`. The integer referred to by `addrLen` initially contains the amount of space pointed to by `addr`. On return it will contain the actual length in bytes of the address returned.

Error code	Meaning
WSANOTINITIALISED	A successful <code>WSAStartup()</code> call must occur before using this function.
WSAECONNRESET	An incoming connection was indicated, but was subsequently terminated by the remote peer prior to accepting the call.
WSAEFAULT	The <code>addrLen</code> parameter is too small or <code>addr</code> is not a valid part of the user address space.
WSAEINTR	A blocking Windows Sockets 1.1 call was canceled through <code>WSACancelBlockingCall()</code> .
WSAEINVAL	The <code>listen()</code> function was not invoked prior to accept.
WSAEINPROGRESS	A blocking Windows Sockets 1.1 call is in progress, or the service provider is still processing a callback function.
WSAEMFILE	The queue is nonempty upon entry to accept and there are no descriptors available.
WSAENETDOWN	The network subsystem has failed.
WSAENOBUFS	No buffer space is available.
WSAENOTSOCK	The descriptor is not a socket.
WSAEOPNOTSUPP	The referenced socket is not a type that supports connection-oriented service.
WSAEWOULDBLOCK	The socket is marked as non-blocking and no connections are present to be accepted.

Table 4

## Remarks

The `accept()` function extracts the first connection on the queue of pending connections on socket `s`. It then creates and returns a handle to the new socket. The newly created socket is the socket that will handle the actual connection; it has the same properties as socket `s`, including the asynchronous events registered with the `WSAAsyncSelect()` or `WSAEventSelect()` functions. The `accept()` function can block the caller until a connection is present if no pending connections are present on the queue, and the socket is marked as blocking. If the socket is marked as non-blocking and no pending connections are present on the queue, `accept()` returns an error as described in the following. After the successful completion of `accept()` returns a new socket handle, the accepted socket cannot be used to accept more connections. The original socket remains open and listens for new connection requests.

The parameter `addr` is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the `addr` parameter is determined by the address family in which the communication is occurring. The `addrlen` is a value-result parameter; it should initially contain the amount of space pointed to by `addr`; on return it will contain the actual length (in bytes) of the address returned. The `accept()` function is used with connection-oriented socket types such as `SOCK_STREAM`. If `addr` and/or `addrlen` are equal to `NULL`, then no information about the remote address of the accepted socket is returned.

### 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 = WSStartup(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.
// using the htons for big-endian
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, 1) == 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;
}

return 0;
}
```

---

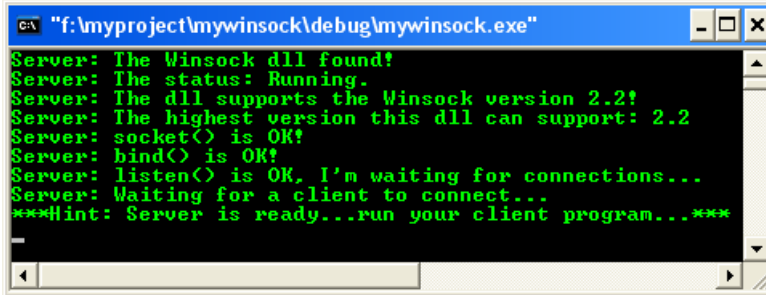


Figure 3

We can verify the connection listening and waiting using **netstat** command on Windows console as shown below. Make sure you do not terminate the previous program.

```

C:\>netstat -a 5

Active Connections

Proto Local Address          Foreign Address        State
TCP   mypersonal:http        0.0.0.0:0             LISTENING
TCP   mypersonal:epmap      0.0.0.0:0             LISTENING
TCP   mypersonal:https      0.0.0.0:0             LISTENING
TCP   mypersonal:microsoft-ds 0.0.0.0:0            LISTENING
TCP   mypersonal:1025       0.0.0.0:0             LISTENING
TCP   mypersonal:1026       0.0.0.0:0             LISTENING
TCP   mypersonal:1031       localhost:1032         ESTABLISHED
TCP   mypersonal:1032       localhost:1031         ESTABLISHED
TCP   mypersonal:55555      0.0.0.0:0             LISTENING
UDP   mypersonal:snmp       *:*
UDP   mypersonal:microsoft-ds *:*
UDP   mypersonal:isakmp     *:*
UDP   mypersonal:3456       *:*
UDP   mypersonal:4500       *:*
UDP   mypersonal:ntp        *:*
UDP   mypersonal:1900       *:*
^C
C:\>

```

Figure 4

The following is another example demonstrates the use of the accept() function. Press the **Ctrl + C** to terminate the server program.

### Program Example

```

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

int main()
{
    // Initialize Winsock.
    WSADATA wsaData;
    int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != NO_ERROR)
        printf("Server: Error at WSAStartup().\n");
    else
        printf("Server: WSAStartup() is OK.\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("Server: Error at socket(): %ld\n", WSAGetLastError());
        WSACleanup();
        return 0;
    }
    else
        printf("Server: 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;
service.sin_family = AF_INET;
service.sin_addr.s_addr = inet_addr("127.0.0.1");
service.sin_port = htons(55555);

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

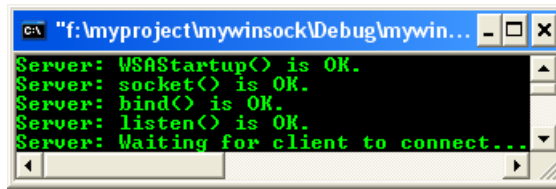
// Listen for incoming connection requests.
// on the created socket
if (listen(ListenSocket, 10) == SOCKET_ERROR)
    printf("Server: Error listening on socket.\n");
else
    printf("Server: listen() is OK.\n");

// Create a SOCKET for accepting incoming requests.
SOCKET AcceptSocket;
printf("Server: Waiting for client to connect...\n");

// Accept the connection if any...
while(1)
{
    AcceptSocket = SOCKET_ERROR;
    while(AcceptSocket == SOCKET_ERROR)
    {
        AcceptSocket = accept(ListenSocket, NULL, NULL);
    }
    printf("Server: accept() is OK.\n");
    printf("Server: Client connected...ready for communication.\n");
    ListenSocket = AcceptSocket;
    break;
}

WSACleanup();
return 0;
}

```

A screenshot of a Windows command prompt window. The title bar reads "C:\ "f:\myproject\mywinsock\Debug\mywin...". The window contains the following text:

```
Server: WSStartup() is OK.
Server: socket() is OK.
Server: bind() is OK.
Server: listen() is OK.
Server: Waiting for client to connect...
```

Figure 5

Now we can test our client-server communication by running both, the server and the client programs. In this example we test both programs on the same computer. You can try testing them on the network computer or Internet domain by changing the IP address (you can use **hostname** as well by adding some codes such as `gethostname()` etc.). Create a new project for the client program.

### 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
// A sample of client program
```

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

```
int main()
{
    // Initialize Winsock.
    WSADATA wsaData;
    int iResult = WSStartup(MAKEWORD(2,2), &wsaData);

    if (iResult != NO_ERROR)
        printf("Client: Error at WSStartup().\n");
    else
        printf("Client: WSStartup() is OK.\n");

    // Create a socket.
    SOCKET m_socket;
    m_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (m_socket == INVALID_SOCKET)
    {
        printf("Client: socket() - Error at socket(): %ld\n", WSAGetLastError());
    }
}
```

```

    WSACleanup();
    return 0;
}
else
    printf("Client: socket() is OK.\n");

// Connect to a server.
sockaddr_in clientService;

clientService.sin_family = AF_INET;
clientService.sin_addr.s_addr = inet_addr("127.0.0.1");
clientService.sin_port = htons(55555);

if (connect(m_socket, (SOCKADDR*)&clientService, sizeof(clientService)) ==
SOCKET_ERROR)
{
    printf("Client: connect() - Failed to connect.\n");
    WSACleanup();
    return 0;
}

// Send and receive data.
int bytesSent;
int bytesRecv = SOCKET_ERROR;
// Be careful with the array bound, provide some checking mechanism
char sendbuf[200] = "Client: Sending some test string to server...";
char recvbuf[200] = "";

bytesSent = send(m_socket, sendbuf, strlen(sendbuf), 0);
printf("Client: send() - Bytes Sent: %ld\n", bytesSent);

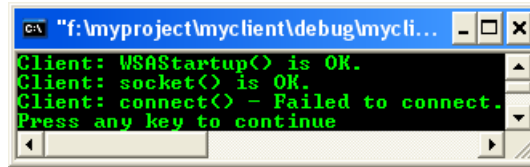
while(bytesRecv == SOCKET_ERROR)
{
    bytesRecv = recv(m_socket, recvbuf, 32, 0);
    if (bytesRecv == 0 || bytesRecv == WSAECONNRESET)
    {
        printf("Client: Connection Closed.\n");
        break;
    }
    else
        printf("Client: recv() is OK.\n");

    if (bytesRecv < 0)
        return 0;
    else
        printf("Client: Bytes received - %ld.\n", bytesRecv);
}

```

```
    return 0;
}
```

When we just run the client program, the following is the output.

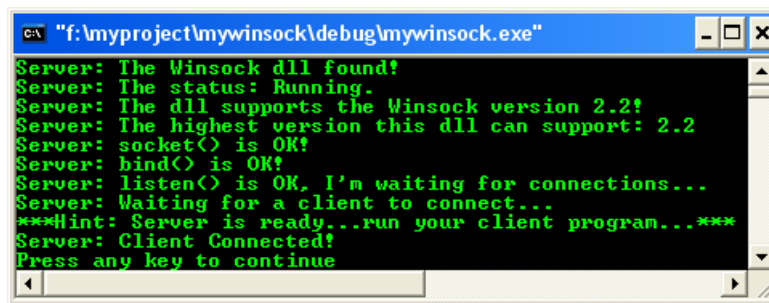


```
C:\> "f:\myproject\myclient\debug\mycli...
Client: WSStartup() is OK.
Client: socket() is OK.
Client: connect() - Failed to connect.
Press any key to continue
```

Figure 6

Next, run the previous server program and then run the client program. The Windows console outputs for the server and the client are shown below. In this example the server just accepts the connection from client and do not read the client's sent string (data).

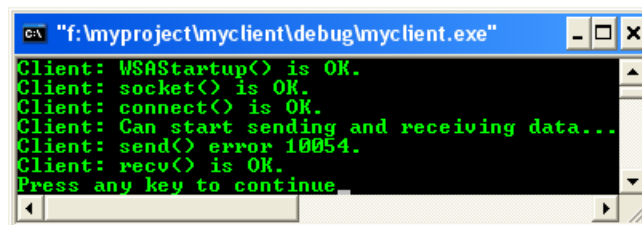
Server console output:



```
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...***
Server: Client Connected!
Press any key to continue
```

Figure 7

Client console output: The error code 10054 is WSAECONNRESET- "An existing connection was forcibly closed by the remote host". That is OK for the error at this stage.



```
C:\> "f:\myproject\myclient\debug\myclient.exe"
Client: WSStartup() is OK.
Client: socket() is OK.
Client: connect() is OK.
Client: Can start sending and receiving data...
Client: send() error 10054.
Client: recv() is OK.
Press any key to continue
```

Figure 8