ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Σύντομος Οδηγός για Socket Programming σε Windows

ΙΑΝΟΥΑΡΙΟΣ 2004

Σύντομος Οδηγός για Socket Programming σε Windows

Εισαγωγή

Επικεφαλίδα και Βιβλιοθήκες
Πρέπει να συμπεριλάβετε στην επικεφαλίδα τα αρχεία winsock.h winsock2.h και κάντε link το αρχείο wsock32.lib στο τελικό εκτελέσιμο.

```
#include <winsock2.h>
#include <winsock.h>
#include <windows.h>
```

WinSock: Αρχικοποίηση
Το πρώτο βήμα είναι η κλήση της WSAStartup για την εκκίνηση του interface στο WinSock.

```
WSADATA wsaData;
WORD version;
int error;

version = MAKEWORD( 2, 0 );

error = WSAStartup( version, &wsaData );

/* check for error */
if ( error != 0 )
{
   /* error occured */
   return FALSE;
}

/* check for correct version */
if ( LOBYTE( wsaData.wVersion ) != 2 ||
    HIBYTE( wsaData.wVersion ) != 0 )
{
   /* incorrect WinSock version */
   WSACleanup();
   return FALSE;
}

/* WinSock has been initialized */
```

Server: Δημιουργία ενός Socket

```
SOCKET server;

server = socket( AF_INET, SOCK_STREAM, 0 );
```

Server: Εκκίνηση Server

```
struct sockaddr_in sin;

memset( &sin, 0, sizeof (sin) );

sin.sin_family = AF_INET;
sin.sin_addr.s_addr = htonl(INADDR_ANY);
sin.sin_port = htons( 5000 );

if ( bind( server,( struct in_addr *)&sin, sizeof (sin )) == SOCKET_ERROR )
{
   /* could not start server */
   return FALSE;
}
```

Server: Ακούγοντας για Client

```
while ( listen( server, SOMAXCONN ) == SOCKET_ERROR );
```

Server: Αποδοχή μιας σύνδεσης

```
SOCKET client;
int length;

length = sizeof sin;
client = accept( server, ( struct in_addr *)&sin, &length );
```

Client: Δημιουργία ενός Socket

```
SOCKET client;

client = socket( AF_INET, SOCK_STREAM, 0 );
```

Client: Εντοπισμός Host

Από την πλευρά του client χρειάζεται να εντοπιστεί ο host σύμφωνα με το όνομα. Για παράδειγμα, όταν θέλουμε να συνδεθούμε με τον danaos.cslab.ntua.gr χρησιμοποιούμε την συνάρτηση gethostbyname.

```
struct hostent *host;
```

host = gethostbyname("danaos.cslab.ntua.gr ");Client: Connecting to Server

```
struct sockaddr_in sin;

memset( &sin, 0, sizeof sin );

sin.sin_family = AF_INET;
sin.sin_addr.s_addr = ((struct in_addr *)(host->h_addr))->s_addr;
sin.sin_port = htons( 5000 );

if ( connect( client ,( struct in_addr *)&sin, sizeof sin ) == SOCKET_ERROR )
{
   /* could not connect to server */
   return FALSE;
}
```

Αποστολή και Λήψη Δεδομένων
Με μια ενεργή socket σύνδεση μπορούμε να στείλουμε δεδομένα χρησιμοποιώντας την συνάρτηση send and receive data χρησιμοποιώντας την recv.

Κλείνοντας ένα Socket
Όταν θέλουμε να κλείσουμε ένα socket, μπορούμε να το κάνουμε καλώντας τη συνάρτηση closesocket.

```
closesocket( server );
```

WinSock: Shutdown
Once finsished with the WinSock system, we need to shut it down by calling the function WSACleanup.

```
WSACleanup();
```

Αρχικοποίηση του WinSock

Πριν από την χρήση των winsockets, το Winsock πρέπει να αρχικοποιηθεί. Πριν από αυτό, όπως έχει ήδη αναφερθεί πρέπει να συμπεριληφθούν όλες οι απαραίτητες επικεφαλίδες και βιβλιοθήκες. Έτσι, σε κάθε αρχείο κώδικα, που απαιτείται η χρήση των winsockets, πρέπει να κάνετε include την winsock.h επικεφαλίδα:

```
#include <winsock.h>
```

Επίσης, πρέπει να συμπεριλάβετε την wsock32.lib βιβλιοθήκη στην αρχή κάθε module. Για να γίνει αυτό, εφόσον δουλεύετε σε visual περιβάλλον, πρέπει να ανανεώσετε τα settings του project. Πατήστε Alt+F7, επιλέξτε το project σας στην λίστα επιλέξτε "All Configurations" από την λίστα αυτή. Πηγαίνετε στο "Link" tab,

και στον text-box για "Object/library modules" θα δείτε μια σειρά άλλων .lib αχρείων. Προσθέστε το wsock32.lib στη λίστα.

Αμέσως μετά είστε έτοιμοι να καλέσετε συναρτήσεις. Πριν αρχίσετε να ανοίγετε sockets, πρέπει να αρχικοποιήσετε το Winsock. Βάλτε αυτό τον κώδικα στη συνάρτηση εκκίνησης π.χ. στη WinMain()

```
WSAData wsa;
WSAStartup(MAKEWORD(1, 1), &wsa);
```

Αυτό λεει στο Winsock να ξεκινήσει και να είναι έτοιμο να δουλέψει. Εάν το ξεχάσετε θα λάβετε ένα μήνυμα λάθους (10093 (**WSANOTINITIALISED**) errors).

Δημιουργία ενός Server Socket

Για να ανοίξετε ένα socket για listening, πρέπει να:

1. Κατανείμετε ένα socket
2. Προσδέσετε το socket σε ένα port
3. Να πείτε σε ένα socket να αρχίσει να ακούει
4. Να το θέσετε στην ασύγχρονη κατάσταση (asynchronous mode) (non blocking)

Παρακάτω θα βρείτε μερικές βασικές δηλώσεις μεταβλητών: sin είναι η "socket address struct" και sock είναι το handle για το listening socket. Να σημειωθεί ότι το sock πρέπει να είναι προσβάσιμο globally και όχι μόνο τοπικά.

```
sockaddr_in sin;
SOCKET sock;
```

Πρώτα δεσμεύσετε ένα socket. Πριν κάνετε οτιδήποτε με ένα socket πρέπει να καλέσετε την socket() συνάρτηση για να πάρετε ένα socket handle. Αυτή η συνάρτηση επιστρέφει ένα SOCKET, ή SOCKET_ERROR στην περίπτωση γεγονότος λάθους.

```
if((sock = socket(PF_INET, SOCK_STREAM, 0)) == SOCKET_ERROR)
        return FALSE;
```

Στη συνέχεια πρέπει να προσδέσετε (bind) το νέο socket που δημιουργήθηκε σε ένα port, δηλ. να πείτε ποιο port μπορεί να χρησιμοποιήσει το socket. Εάν φτιάχνετε έναν client συνήθως πρέπει να προσδέσετε το socket στο port 0, έτσι ώστε το Winsock να επιλέξει ένα τυχαίο port. Σημειώστε ότι για οποιοδήποτε εκτός από το port 0, πρέπει να χρησιμοποιήσετε την htons() συνάρτηση. Αυτή μετατρέπει τον αριθμό σε ένα διαφορετικό network-order short.

```
/* Set the family for the socket to internet */
        sin.sin_family = AF_INET;

/* Set to use port 1234 */
        sin.sin_port = htons(1234);

        if(bind(sock, (sockaddr *)&sin, sizeof(sockaddr_in) ==
SOCKET_ERROR)
                return FALSE;
```

Στη συνέχεια, λετε στο socket να αρχίσει να άκουει. Αυτό πραγματοποιείται με τη συνάρτηση listen(). Αυτή έχει πολλές παραμέτρους αλλά η μόνη αξιοσημείωτη είναι η backlog, η οποία μας ενημερώνει πόσες *pending* connections θα κρατήσει, δηλαδή πόσες συνδέσεις, οι οποίες δεν έχουν απαντήσει μέχρι τώρα, θα κρατηθούν σε ένα αποθηκευτικό χώρο (not called accept on). Συνήθως, εάν το θέσετε ίσο με 100 δεν θα έχετε κανένα πρόβλημα.

```
        if(listen(sock, 200) == SOCKET_ERROR)
                return FALSE;
```

Τέλος, τις περισσότερες φορές θέλετε να θέσετε το socket σε κατάσταση (mode) non-blocking. Αυτό σημαίνει, ότι όταν συμβαίνει κάτι, λέτε στο winsock να σας στέλνει ένα μήνυμα αντί να κάνει blocking το κώδικά σας (δεν εξέρχεται από την συνάρτηση μέχρι την επιτυχή ολοκλήρωση). Αυτό επιτυγχάνεται με την WSAAsyncSelect(), χρησιμοποιώντας το FD_ACCEPT event. Αυτή η συνάρτηση είναι αρκετά πολύπλοκη.

```
/* Main.hWnd should contain the HWND of the window you're using for
socket stuff */
        WSAAsyncSelect(sock, Main.hWnd, WM_USER + 1, FD_ACCEPT);
```

Πρώτα από όλα, μπορείτε να χρησιμοποιήσετε την WSAAsyncSelect() μια φορά για ένα SOCKET. Επιπλέον κλήσεις της θα ακυρώσουν τις προηγούμενες. ΄Έτσι για να συνδυαστούν πολλαπλά γεγονότα μπορείτε να χρησιμοποιήσετε τον δυαδικό τελεστήl. π.χ.. FD_READ | FD_CLOSE

Επίσης, η WSAAsyncSelect θα στείλει το μήνυμα που καθορίσατε στο παράθυρο που επίσης έχετε καθορίσει. Μόλις το μήνυμα ληφθεί θα πάρετε επίσης το SOCKET στο οποίο το γεγονός (event) έλαβε χώρα στο *wParam*. Το *lParam* περιέχει το event που έλαβε χώρα στο low word, και τον error code στο high word. Το μήνυμα είναι αρκετά αυθαίρετο (arbitrary), αλλά πρέπει να είναι μοναδικό. Τώρα έχετε ένα socket σε κατάσταση listening.

Δημιουργία ενός Client Socket

Αφού έχει δημιουργηθεί ένα listening socket, πρέπει δημιουργηθεί μία κλήση προς αυτό. Το client socket πρέπει να εκτελέσει μια σειρά από λειτουργίες, σε σειρά, σε αντιστοιχία με το server socket. Αυτά παρατίθενται στη συνέχεια:

1. Δημιουργία του socket
2. Πρόσδεσή του σε κάποιο port (Αυτή τη φορά προσδένεται στο port 0, ώστε να επιλεγεί ένα τυχαίο port)
3. Απόπειρα σύνδεσης
4. Καθορισμός non-blocking

Εδώ παρατίθενται οι προκαταρτικές δηλώσεις. Και πάλι, το SOCKET sock πρέπει να είναι global.

```
        SOCKET sock;
        sockaddr_in LocalSin, RemoteSin;

/* This is the family for the socket */
        LocalSin.sin_family = AF_INET;
```

```
/* This is the port to connect from.  Setting 0 means use random port
*/
        LocalSin.sin_port = 0;

        RemoteSin.sin_family = AF_INET;
/* This is the port to connect to */
        RemoteSin.sin_port = htons(nPort);
```

Τώρα, πρέπει να προστεθεί η IP διεύθυνση προς την οποίο αποπειράται η σύνδεση στο remote address struct. Ωστόσο, δεν χρησιμοποιείται κάποιο string, αλλά network-byte order numbers... Η μετατροπή από string σε network-byte order γίνεται με τη συνάρτηση inet_addr():

```
        if((RemoteSin.sin_addr.S_un.S_addr = inet_addr(szIP)) ==
INADDR_NONE)
                return false;           // Error setting IP
```

Η δημιουργία του socket γίνεται με τον ίδιο τρόπο όπως και προηγουμένως...

```
        if((sock = socket(PF_INET, SOCK_STREAM, 0)) == SOCKET_ERROR)
                return false;           // Error creating socket
```

Τώρα το socket προσδένεται στο local port με χρήση της συνάρτησης bind(), όπως έγινε και κατά τη δημιουργία του server socket.

```
        if(bind(sock, (sockaddr *)&LocalSin, sizeof(sockaddr_in)) ==
SOCKET_ERROR)
                return false;           // Error binding socket
```

Στη συνέχεια, δίδεται εντολή στο socket να αποπειραθεί να συνδεθεί. Αυτό γίνεται με τη συνάρτηση connect().

```
        if(connect(sock, (sockaddr *)&RemoteSin, sizeof(sockaddr_in))
== SOCKET_ERROR)
                return false;
```

Και πάλι, θέτουμε το Winsock να στέλνει ένα μήνυμα όταν συμβαίνει κάποιο γεγονός (event). Εφ' όσον τα μόνα δύο ενδιαφέρονται events είναι η λήψη δεδομένων και το κλείσιμο του socket, χρησιμοποιούμε τα FD_READ | FD_CLOSE. Το WM_USER + 2 είναι αυθαίρετο, αλλά έχει σημασία να είναι μοναδικό.

```
/* Main.hWnd should contain the HWND of the window you're using for
socket stuff */
        WSAAsyncSelect(sock, Main.hWnd, WM_USER + 2, FD_READ |
FD_CLOSE);
```

Κλείσιμο του Socket

Το κλείσιμο ενός socket είναι πολύ εύκολο. Χρησιμοποιείται η συνάρτηση closesocket():

```
        closesocket(MySock);
```

Τερματισμός του Winsock

Στο τέλος κάθε προγράμματος πρέπει να τερματιστούν όλα τα αντικείμενα. Όταν χρησιμοποιούνται τα Winsock, αυτό γίνεται με τη συνάρτηση WSACleanup(). Αυτή η κλήση τοποθετείται στη συνάρτηση τερματισμού, ώστε να εκτελείται μόνο μία φορά.

```
        WSACleanup();
```

Αποστολή Δεδομένων

Σε τι χρησιμεύουν τα sockets αν δεν μπορείς να στείλεις δεδομένα;

Η αποστολή δεδομένων μέσα από ένα socket γίνεται με τη συνάρτηση send(). Να σημειωθεί ότι δέχεται ως παράμετρο το μήκος των δεδομένων προς αποστολή. Αν αποστέλλεται κείμενο, το μήκος προκύπτει με τη συνάρτηση strlen(), ενώ για οτιδήποτε άλλο, το μήκος προκύπτει με τον τελεστή sizeof.

```
        send(sock, "Test String", 12, 0);
-η-
        MyStruct a;
        send(sock, (char *)&a, sizeof(MyStruct), 0);
```

Γνωρίζοντας αυτό, ένας τρόπος για να αποσταλεί πλήθος strings φαίνεται στη συνέχεια. Σε περίπτωση που αποστέλλονται μόνο string, αυτή η μέθοδος είναι επαρκής.

```
        SendData(sock, "This is some data!");
        ...

int SendData(SOCKET sock, char * lpszData)
{
        return send(sock, lpszData, strlen(lpszData), 0);
}
```

Λήψη δεδομένων

Σε τι χρησιμεύει η αποστολή δεδομένων αν η άλλη πλευρά δεν τα λαμβάνει; Αυτό το κομμάτι κώδικα λαμβάνει δεδομένα από ένα socket, και ανοίγει ένα message box που περιέχει τα δεδομένα που ελήφθησαν.

Αυτό το κομμάτι κώδικα πρέπει να τοποθετηθεί στο WinProc code και για τον server και για τον client:

```
        switch(uMsg)
        {
        ...
        case WM_USER + 2:
                HandleData(wParam, lParam);
                break;
        ...
```

```
int HandleData(WPARAM wParam, LPARAM lParam)
{
        SOCKET sock = (SOCKET)wParam;
        WORD event = LOWORD(lParam);
        WORD error = HIWORD(lParam);

        if(event == FD_CLOSE)
        {
                closesocket(sock);
        }
        else if(event == FD_READ)
        {
                char szBuffer[1024];
                ZeroMemory(szBuffer, 1024);
                recv(sock, szBuffer, 1024, 0);

                MessageBox(Main.hWnd, szBuffer, "Received Data!",
MB_OK);

                closesocket(sock);
        }
        return TRUE;
}
```

Τώρα, όταν λαμβάνονται δεδομένα, αυτά τοποθετούνται σε ένα message box και το socket κλείνει. Σε περίπτωση που το port θέλει να κλείσει, το πρόγραμμα το αναγνωρίζει.

Διαχείριση Σφαλμάτων

Εν συντομία, η διαχείριση σφαλμάτων είναι απλή. Συνήθως, όταν συμβαίνει σφάλμα, η συνάρτηση επιστρέφει SOCKET_ERROR, και κάνει set ένα error code. Αυτό το error code λαμβάνεται με τη συνάρτηση WSAGetLastError().

```
        int Error = WSAGetLastError();

        if (Error == WSANOTINITIALISED)
                ...
```

Το MSDN online παρέχει μία λίστα των error codes, που για ευκολία παρατίθεται στο παράρτημα.

Αποδοχή Συνδέσεων

Εφόσον υπάρχει ένα socket που εκτελεί listen, απαιτείται να υπάρχει χειρισμός των αιτήσεων για συνδέσεις.

Στον κώδικα που δημιουργήθηκε το listening socket, καθορίστηκε να αποστέλλεται ένα συγκεκριμένο μήνυμα (πχ WM_USER + 1) σε ένα συγκεκριμένο παράθυρο (Main.hWnd). Τώρα θα δούμε πως αυτό το παράθυρο μπορεί να χειριστεί αυτό το μήνυμα. Στο WindowProc, συμπεριλαμβάνεται ο παρακάτω κώδικας:

```
        switch(uMsg)
        {
        ...
```

```
        case WM_USER + 1:
                HandleAccept(wParam, lParam);
                break;
        ...
}

int HandleAccept(WPARAM wParam, LPARAM lParam)
{
        SOCKET sock = (SOCKET)wParam;
        WORD event = LOWORD(lParam);
        WORD error = HIWORD(lParam);

        if(event == FD_ACCEPT)
        {
                sockaddr_in NewRemoteSin;
                SOCKET newsock = accept(sock,
(sockaddr*)&NewRemoteSin, NULL);

/* NOTE: Now NewRemoteSin contains the address of the new client.
        And newsock contains the socket that has the new connection
*/

                WSAAsyncSelect(newsock, Main.hWnd, WM_USER + 2,
FD_READ | FD_CLOSE);

                SendData(newsock, "Hello, welcome to my server.");
        }
        return TRUE;
}
```

Τώρα αυτό το παράθυρο αποδέχεται εισερχόμενες συνδέσεις, στέλνει στον νέο client το string "Hello, welcome to my server.", και περιμένει για απόκριση.

Εναλλακτικά –για την αποφυγή πολυπλοκότητας- μπορεί να γίνει χρήση της συνάρτησης select :

int select(int nfds, fd_set* readfds, fd_set* writefds, fd_set* exceptfds, const struct timeval* timeout)

Αναφορές

Το καλύτερο μέρος για να αναζητήσετε τις συναρτήσεις είναι η MSDN:
http://msdn.microsoft.com/library/default.asp

ΠΑΡΑΡΤΗΜΑ
Error Codes in the API

Τα ακόλουθα είναι μια λίστα από πιθανά error codes όπως καταγράφονται από **WSAGetLastError** call, καθώς και με τις περιγραφές τους.

WSAEACCES
(10013)
Permission denied.
An attempt was made to access a socket in a way forbidden by its access permissions. An example is using a broadcast address for **sendto** without broadcast permission being set using **setsockopt(SO_BROADCAST)**.

Another possible reason for the **WSAEACCES** error is that when the **bind** function is called (on Windows NT 4 SP4 or later), another application, service, or kernel mode driver is bound to the same address with exclusive access. Such exclusive access is a new feature of Windows NT 4 SP4 and later, and is implemented by using the **SO_EXCLUSIVEADDRUSE** option.

WSAEADDRINUSE
(10048)
Address already in use.
Only one usage of each socket address (protocol/IP address/port) is normally permitted. This error occurs if an application attempts to **bind** a socket to an IP address/port that has already been used for an existing socket, or a socket that wasn't closed properly, or one that is still in the process of closing. For server applications that need to **bind** multiple sockets to the same port number, consider using **setsockopt(SO_REUSEADDR)**. Client applications usually need not call **bind** at all - **connect** chooses an unused port automatically. When **bind** is called with a wildcard address (involving ADDR_ANY), a WSAEADDRINUSE error could be delayed until the specific address is committed. This could happen with a call to another function later, including **connect**, **listen**, **WSAConnect** or **WSAJoinLeaf**.

WSAEADDRNOTAVAIL
(10049)
Cannot assign requested address.
The requested address is not valid in its context. This normally results from an attempt to **bind** to an address that is not valid for the local machine. This can also result from **connect**, **sendto**, **WSAConnect**, **WSAJoinLeaf**, or **WSASendTo** when the remote address or port is not valid for a remote machine (for example, address or port 0).

WSAEAFNOSUPPORT
(10047)
Address family not supported by protocol family.
An address incompatible with the requested protocol was used. All sockets are created with an associated address family (that is, AF_INET for Internet Protocols) and a generic protocol type (that is, SOCK_STREAM). This error is returned if an incorrect protocol is explicitly requested in the **socket** call, or if an address of the wrong family is used for a socket, for example, in **sendto**.

WSAEALREADY
(10037)
Operation already in progress.
An operation was attempted on a nonblocking socket with an operation already in progress - that is, calling **connect** a second time on a nonblocking socket that is already connecting, or canceling an asynchronous request (WSAAsyncGetXbyY) that has already been canceled or completed.

WSAECONNABORTED
(10053)
Software caused connection abort.
An established connection was aborted by the software in your host machine, possibly due to a data transmission time-out or protocol error.

WSAECONNREFUSED
(10061)
Connection refused.

No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host—that is, one with no server application running.

WSAECONNRESET

(10054)

Connection reset by peer.

A existing connection was forcibly closed by the remote host. This normally results if the peer application on the remote host is suddenly stopped, the host is rebooted, or the remote host used a hard close (see **setsockopt** for more information on the **SO_LINGER** option on the remote socket.) This error may also result if a connection was broken due to keepalive activity detecting a failure while one or more operations are in progress. Operations that were in progress fail with **WSAENETRESET**. Subsequent operations fail with **WSAECONNRESET**.

WSAEDESTADDRREQ

(10039)

Destination address required.

A required address was omitted from an operation on a socket. For example, this error is returned if **sendto** is called with the remote address of ADDR_ANY.

WSAEFAULT

(10014)

Bad address.

The system detected an invalid pointer address in attempting to use a pointer argument of a call. This error occurs if an application passes an invalid pointer value, or if the length of the buffer is too small. For instance, if the length of an argument which is a **SOCKADDR** structure is smaller than the sizeof(**SOCKADDR**).

WSAEHOSTDOWN

(10064)

Host is down.

A socket operation failed because the destination host is down. A socket operation encountered a dead host. Networking activity on the local host has not been initiated. These conditions are more likely to be indicated by the error WSAETIMEDOUT.

WSAEHOSTUNREACH

(10065)

No route to host.

A socket operation was attempted to an unreachable host. See WSAENETUNREACH

WSAEINPROGRESS

(10036)

Operation now in progress.

A blocking operation is currently executing. Windows Sockets only allows a single blocking operation to be outstanding per task (or thread), and if any other function call is made (whether or not it references that or any other socket) the function fails with the WSAEINPROGRESS error.

WSAEINTR

(10004)

Interrupted function call.

A blocking operation was interrupted by a call to **WSACancelBlockingCall**.

WSAEINVAL

(10022)

Invalid argument.

Some invalid argument was supplied (for example, specifying an invalid level to the **setsockopt** function). In some instances, it also refers to the current state of the socket – for instance, calling **accept** on a socket that is not listening.

WSAEISCONN

(10056)

Socket is already connected.

A connect request was made on an already connected socket. Some implementations also return this error if **sendto** is called on a connected SOCK_DGRAM socket (For SOCK_STREAM sockets, the *to* parameter in **sendto** is ignored), although other implementations treat this as a legal occurrence.

WSAEMFILE

(10024)

Too many open files.

Too many open sockets. Each implementation may have a maximum number of socket handles available, either globally, per process, or per thread.

WSAEMSGSIZE

(10040)

Message too long.

A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the buffer used to receive a datagram was smaller than the datagram itself.

WSAENETDOWN

(10050)

Network is down.

A socket operation encountered a dead network. This could indicate a serious failure of the network system (that is, the protocol stack that the Windows Sockets .dll runs over), the network interface, or the local network itself.

WSAENETRESET

(10052)

Network dropped connection on reset.

The connection has been broken due to keep-alive activity detecting a failure while the operation was in progress. It can also be returned by **setsockopt** if an attempt is made to set SO_KEEPALIVE on a connection that has already failed.

WSAENETUNREACH

(10051)

Network is unreachable.

A socket operation was attempted to an unreachable network. This usually means the local software knows no route to reach the remote host.

WSAENOBUFS

(10055)

No buffer space available.

An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

WSAENOPROTOOPT

(10042)

Bad protocol option.

An unknown, invalid or unsupported option or level was specified in a **getsockopt** or **setsockopt** call.

WSAENOTCONN

(10057)

Socket is not connected.

A request to send or receive data was disallowed because the socket is not connected and (when sending on a datagram socket using **sendto**) no address was supplied. Any other type of operation might also return this error – for example, **setsockopt** setting SO_KEEPALIVE if the connection has been reset.

WSAENOTSOCK

(10038)

Socket operation on non-socket.

An operation was attempted on something that is not a socket. Either the socket handle parameter did not reference a valid socket, or for **select**, a member of an **fd_set** was not valid.

WSAEOPNOTSUPP

(10045)

Operation not supported.

The attempted operation is not supported for the type of object referenced. Usually this occurs when a socket descriptor to a socket that cannot support this operation, for example, trying to accept a connection on a datagram socket.

WSAEPFNOSUPPORT

(10046)

Protocol family not supported.

The protocol family has not been configured into the system or no implementation for it exists. Has a slightly different meaning to **WSAEAFNOSUPPORT**, but is interchangeable in

most cases, and all Windows Sockets functions that return one of these specify **WSAEAFNOSUPPORT**.

WSAEPROCLIM

(10067)

Too many processes.

A Windows Sockets implementation may have a limit on the number of applications that may use it simultaneously. **WSAStartup** may fail with this error if the limit has been reached.

WSAEPROTONOSUPPORT

(10043)

Protocol not supported.

The requested protocol has not been configured into the system, or no implementation for it exists. For example, a **socket** call requests a SOCK_DGRAM socket, but specifies a stream protocol.

WSAEPROTOTYPE

(10041)

Protocol wrong type for socket.

A protocol was specified in the **socket** function call that does not support the semantics of the socket type requested. For example, the ARPA Internet UDP protocol cannot be specified with a socket type of SOCK_STREAM.

WSAESHUTDOWN

(10058)

Cannot send after socket shutdown.

A request to send or receive data was disallowed because the socket had already been shut down in that direction with a previous **shutdown** call. By calling **shutdown** a partial close of a socket is requested, which is a signal that sending or receiving or both have been discontinued.

WSAESOCKTNOSUPPORT

(10044)

Socket type not supported.

The support for the specified socket type does not exist in this address family. For example, the optional type SOCK_RAW might be selected in a **socket** call, and the implementation does not support SOCK_RAW sockets at all.

WSAETIMEDOUT

(10060)

Connection timed out.

A connection attempt failed because the connected party did not properly respond after a period of time, or the established connection failed because the connected host has failed to respond.

WSATYPE_NOT_FOUND

(10109)

Class type not found.

The specified class was not found.

WSAEWOULDBLOCK

(10035)

Resource temporarily unavailable.

This error is returned from operations on nonblocking sockets that cannot be completed immediately, for example **recv** when no data is queued to be read from the socket. It is a non-fatal error, and the operation should be retried later. It is normal for WSAEWOULDBLOCK to be reported as the result from calling **connect** on a nonblocking SOCK_STREAM socket, since some time must elapse for the connection to be established.

WSAHOST_NOT_FOUND

(11001)

Host not found.

No such host is known. The name is not an official host name or alias, or it cannot be found in the database(s) being queried. This error may also be returned for protocol and service queries, and means the specified name could not be found in the relevant database.

WSA_INVALID_HANDLE

(OS dependent)

Specified event object handle is invalid.

An application attempts to use an event object, but the specified handle is not valid.

WSA_INVALID_PARAMETER

(OS dependent)

One or more parameters are invalid.

An application used a Windows Sockets function which directly maps to a Win32 function. The Win32 function is indicating a problem with one or more parameters.

WSAINVALIDPROCTABLE

(OS dependent)

Invalid procedure table from service provider.

A service provider returned a bogus procedure table to WS2_32.dll. (Usually caused by one or more of the function pointers being NULL.)

WSAINVALIDPROVIDER

(OS dependent)

Invalid service provider version number.

A service provider returned a version number other than 2.0.

WSA_IO_INCOMPLETE

(OS dependent)

Overlapped I/O event object not in signaled state.

The application has tried to determine the status of an overlapped operation which is not yet completed. Applications that use **WSAGetOverlappedResult** (with the *fWait* flag set to FALSE) in a polling mode to determine when an overlapped operation has completed get this error code until the operation is complete.

WSA_IO_PENDING

(OS dependent)

Overlapped operations will complete later.

The application has initiated an overlapped operation which cannot be completed immediately. A completion indication will be given at a later time when the operation has been completed.

WSA_NOT_ENOUGH_MEMORY

(OS dependent)

Insufficient memory available.

An application used a Windows Sockets function which directly maps to a Win32 function. The Win32 function is indicating a lack of required memory resources.

WSANOTINITIALISED

(10093)

Successful WSAStartup not yet performed.

Either the application hasn't called **WSAStartup** or **WSAStartup** failed. The application may be accessing a socket which the current active task does not own (that is, trying to share a socket between tasks), or **WSACleanup** has been called too many times.

WSANO_DATA

(11004)

Valid name, no data record of requested type.

The requested name is valid and was found in the database, but it does not have the correct associated data being resolved for. The usual example for this is a host name -> address translation attempt (using **gethostbyname** or **WSAAsyncGetHostByName**) which uses the DNS (Domain Name Server), and an MX record is returned but no A record – indicating the host itself exists, but is not directly reachable.

WSANO_RECOVERY

(11003)

This is a non-recoverable error.

This indicates some sort of non-recoverable error occurred during a database lookup. This may be because the database files (for example, BSD-compatible HOSTS, SERVICES, or PROTOCOLS files) could not be found, or a DNS request was returned by the server with a severe error.

WSAPROVIDERFAILEDINIT

(OS dependent)

Unable to initialize a service provider.

Either a service provider's DLL could not be loaded (**LoadLibrary** failed) or the provider's **WSPStartup/NSPStartup** function failed.

WSASYSCALLFAILURE

(OS dependent)

System call failure.
Returned when a system call that should never fail does. For example, if a call to **WaitForMultipleObjects** fails or one of the registry functions fails trying to manipulate the protocol/name space catalogs.

WSASYSNOTREADY

(10091)
Network subsystem is unavailable.
This error is returned by **WSAStartup** if the Windows Sockets implementation cannot function at this time because the underlying system it uses to provide network services is currently unavailable. Users should check:

- That the appropriate Windows Sockets DLL file is in the current path.
- That they are not trying to use more than one Windows Sockets implementation simultaneously. If there is more than one WINSOCK DLL on your system, be sure the first one in the path is appropriate for the network subsystem currently loaded.
- The Windows Sockets implementation documentation to be sure all necessary components are currently installed and configured correctly.

WSATRY_AGAIN

(11002)
Non-authoritative host not found.
This is usually a temporary error during host name resolution and means that the local server did not receive a response from an authoritative server. A retry at some time later may be successful.

WSAVERNOTSUPPORTED

(10092)
WINSOCK.DLL version out of range.
The current Windows Sockets implementation does not support the Windows Sockets specification version requested by the application. Check that no old Windows Sockets .dll files are being accessed.

WSAEDISCON

(10094)
Graceful shutdown in progress.
Returned by **WSARecv** and **WSARecvFrom** to indicate that the remote party has initiated a graceful shutdown sequence.

WSA_OPERATION_ABORTED

(OS dependent)
Overlapped operation aborted.
An overlapped operation was canceled due to the closure of the socket, or the execution of the **SIO_FLUSH** command in **WSAIoctl**.

Built on Monday, August 16, 1999