Tutorial Windows Sockets API - Partea I

Autor: Popescu Ionut aka Nytro
Contact: nytro_rst@yahoo.com
Website: http://rstcenter.com/

In acest tutorial va voi prezenta API-ul Winsock (Windows Sockets). Acest API reprezinta setul de functii de baza pentru accesul "brut" la Internet, la transferul datelor pe Internet.

O mare parte a tutorialului va fi prezentarea functiilor API. Nu voi prezenta toate detaliile inutile. Aceasta este prima parte, voi acoperi numai lucrurile de baza, mai multe detalii in articolele ce vor urma.

Exemplele din tutorial vor fi in Visual Basic 6 si C++. Programatorilor VB le recomand API Viewer 2004 (http://www.activevb.de/rubriken/apiviewer/index-apiviewereng.html), un program cu care puteti vedea declaratiile functiilor din DLL-uri, constante, tipuri etc. O veste buna pentru ei: folosind ce veti invata in acest tutorial veti putea crea aplicatii Internet fara a mai avea nevoie de fisiere auxiliare ca mswinsck.ocx sau msinet.ocx. Pentru orice informatie referitoare la o functie sau un tip, o constanta vizitati http://msdn.microsoft.com/en-us/default.aspx. Nu recomand acest tutorial incepatorilor, celor care nu stiu mai nimic. Il recomand celor obisnuiti cu folosirea functiilor API Windows, celor care stiu ce e o structura, care cunosc diferenta dintre transmiterea catre o functie a parametrilor prin valoare si transmiterea prin referinta, obisnuiti cu folosirea in mod frecvent a constantelor. Folosind un mediu ca VB6 sau C++, nu va fi nevoie de nimic pentru buna functionare a programului. Programul va functiona si pe XP si pe Vista, Windows 7 si e posibil sa ruleze foarte bine chiar si pe Windows 98. De exemplu prin folosirea unui mediu ca VB 2008 sau VC# 2008, programul va avea nevoie de platforma .NET Framework 2.0 pentru a rula.

Insa nu uitati ca functiile API pot fi folosite in orice limbaj de programare. Pentru programatorii C++ toate declaratiile se afla in Winsock2.h pentru versiunea 2.2 si winsock.h pentru versiunea 1.1, dar nu cred ca mai este cazul. Pentru Visual Basic 6 vor trebui declarate toate manual.

O API (*Application Programming Interface*) este o interfata de programare a aplicatiilor, un grup de functii care pot fi folosite de programatori pentru a scrie programe. Folosirea functiilor API in scrierea unui program este foarte grea dar foarte eficienta.

Prin anii 80, ARPA (*Advanced Research Projects Agency*) a oferit Universitatii Berkeley din California fonduri pentru a implementa protocoalele TCP/IP sub sistemul de operare UNIX.

Interfata Winsocket este o API pentru retelele TCP/IP. Windows Sockets e o API bazata pe Berkeley Sockets. Momentan, la versiunea 2.2, vechile functii se pastreaza din motive de compatibilitate, sunt in continuare folosite.

Un *socket* este un terminal pentru comunicatiile in retea, o "componenta" unei aplicatii prin care aplicatia trimite si receptioneaza date. Pentru o comunicatie in retea sunt necesare doua socket-uri care isi transfera date. De cele mai multe ori pentru comunicatii in retea se foloseste modelul client-server. Astfel, pentru un server: se creaza socketul, se "leaga" de o adresa apoi "asculta" pe un port cereri de conexiune de la clienti, si le accepta sau nu cand le primeste. Pentru un client: se creaza un socket, se conecteaza la server apoi incep sa se trimia/primeasca date. Dupa terminarea transferului de date se inchide conexiunea.

Comunicatia dintre doua socket-uri poate fi de doua feluri: comunictie orientata pe conexiune si comunicatie neorientata pe conexiune, pe baza de datagrame. La comunicatia orientata pe conexiune, trebuie sa existe o legatura logica intre socket-uri, o conexiune. La acest tip de conexiune se asigura ca

datele ajung in aceeasi ordine in care au fost trimise, se verifica impotriva erorilor, fara interventia programatorului. O comunicatie neorientata pe conexiune este nesigura: datele pot ajunge in alta ordine sau nu pot ajunge deloc, fara sa se cunoasca acest lucru. Nu se realizeaza o corectie a erorilor. Pentru comunicatia orientata pe conexiune se foloseste protocolul TCP iar pentru comunicatia neorientata pe conexiune se foloseste protocolul UDP. Voi reveni cu detalii.

Winsock API se afla in biblioteca *wsock32.dll* pentru Winsock 1.1 sau *ws2_32.dll* pentru Winsock 2.2.

Versiunile Winsock de pana acum sunt: 1.0, 1.1, 2.0, 2.1, 2.2. Versiunea curenta de Winsock este 2.2, cea care va fi prezentata in acest tutorial.

Un lucru important la Winsock este ca functiile pot fi cu blocare sau fara blocare. O functie cu blocare impiedica programul sa apeleze orice alta functie Winsock pana cand termina operatiile de retea, se asteapta executarea functiei pentru a se trece mai departe.

O parte din functiile specifice Windows sunt asincrone. Functiile asincrone sunt functii care nu produc o blocare. Ele returneaza de cele mai multe ori un handle de task, un identificator, si se efectueaza imediat, returnand handlerul, apoi, dupa ce efectueaza operatia trimit un mesaj handlerului unei ferestre pe care o primesc ca parametru impreuna cu mesajul pe care il trimit, astfel se stie cand functia a terminat operatia .

Numele functiilor specifice Windows incep cu WSA (*Windows Sockets API*).

<u>O scurta trecere in revista a functiilor, desigur, lista nu e completa:</u>
- **accept**() = Permite o cerere de conexiune
- **AcceptEx**() = Accepta o conexiune returnand adresa locala si adresa indepartata si primeste primul pachet trimis de client
- **bind**() = Asociaza un "nume" unui socket
- **closesocket**() = Inchide un socket
- **connect**() = Realizeaza conexiunea unui socket
- **ConnectEx**() = Conecteaza un socket si optional trimite date imediat dupa conexiune
- **DisconnectEx**() = Inchide o conexiune a unui socket dar permite ca acesta sa poata fi folosit in continuare
- **EnumProtocols**() = Obtine informatii despre un set de protocoale active
- **freeaddrinfo**() = Elibereaza memoria pe care functia getaddrinfo o aloca unei structuri addrinfo
- **FreeAddrInfoEx**() = Elibereaza memoria pe care functia GetAddrInfoEx o aloca unei structuri addrinfoex
- **FreeAddrInfoW**() = Elibereaza memoria pe care functia GetAddrInfoW o aloca unei structuri addrinfoW
- **gai_strerror**() = Este folosita pentru mesajele de eroare returnate de functia getaddrinfo
- **GetAcceptExSockaddrs**() = Parseaza datele obtinute prin apelul functiei AcceptEx
- **GetAddressByName**() = Interogheaza un namespace pentru a obtine informatii despre adresa pentru un serviciu specificat. Procesul e cunoscut ca Service Name Resolution. Un serviciu de retea se poate folosi de asemenea pentru a obtine adresa locala care poate fi folosita cu functia bind
- **getaddrinfo**() = Ofera o traducere a unui hostname ANSI intr-o adresa
- **GetAddrInfoW**() = Ofera o traducere a unui hostname Unicode la o adresa
- **gethostbyaddr**() = Obtine informatiile referitoare la host pentru o adresa
- **gethostbyname**() = Obtine informatiile referitoare la un host pentru un hostname dintr-o baza de date. Functia e veche, se recomanda folosirea functiei getaddrinfo in locul sau
- **gethostname**() = Obtine hostname-ul pentru calculatorul local
- **GetNameByType**() = Obtine numele unui serviciu de retea pentru un anumit tip de serviciu
- **getnameinfo**() = Ofera Name Resolution de la o adresa IPv4 sau IPv6 la un hostname ANSI si de la numarul unui port la numele unui serviciu ANSI
- **GetNameInfoW**() = Ofera Name Resolution de la o adresa IPv4 sau IPv6 la un hostname Unicode si de

la numarul unui port la numele unui serviciu Unicode
- **getpeername**() = Obtine adresa perechii la care este conectat un socket
- **getprotobyname**() = Obtine informatii despre un protocol dupa nume
- **getprotobynumber**() = Obtine informatii despre un protocol dupa un numar
- **getservbyname**() = Obtine informatii despre un protocol dupa un nume si un protocol
- **getservbyport**() = Obtine informatii despre un protocol dupa un port si un protocol
- **GetService**() = Obtine informatii despre serviciul din contextul unui namespace
- **getsockname**() = Obtine numele local pentru un socket
- **getsockopt**() = Obtine optiunile unui socket
- **GetTypeByName**() = Obtine GUID-ul (Globally Unique Identifier) tipului unui serviciu pentru un serviciu specificat prin nume
- **htonl**() = Converteste un numar pe 32 de biti din formatul hostului in formatul retelei (byte-order, big-endian)
- **htons**() = Converteste un numar pe 16 biti din formatul hostului in formatul retelei (byte-order, big-endian)
- **inet_addr**() = Converteste un sir reprezentand o adresa IP in formatul retelei, format necesar pentru structura in_addr
- **inet_ntoa**() = Converteste o adresa din formatul retelei in notatia cu punct
- **InetNtop**() = Converteste o adresa IPv4 sau IPv6 intr-un sir in formatul standard. Versiunea ANSI a functiei este inet_ntop
- **InetPton**() = Converteste o adresa IPv4 sau IPv6 din formatul standard text in formatul sau binar. Versiunea ANSI a functiei este inet_pton
- **ioctlsocket**() = Controleaza modul de intrare/iesire al unui socket
- **listen**() = Seteaza un socket in modul de ascultare
- **ntohl**() = Converteste un numar pe 32 de biti din formatul retelei in formatul hostului (byte-order, care este little-endian pe procesoarele Intel)
- **ntohs**() = Converteste un numar pe 16 biti din formatul retelei in formatul hostului (byte-order, care este little-endian pe procesoarele Intel)
- **recv**() = Primeste date de la un socket conectat
- **recvfrom**() = Primeste o datagrama si memoreaza adresa sursa
- **select**() = Determina starea unuia sau a mai multor socket-uri
- **send**() = Trimite date printr-un socket conectat
- **sendto**() = Trimite date catre o destinatie specifica
- **SetAddrInfoEx**() = Inregistreaza un nume, un serviciu si adresa specificata cu un namespace specificat
- **SetService**() = Inregistreaza sau sterge din Registry un serviciu sau un tip de serviciu dintr-unul sau mai multe namespace-uri
- **setsockopt**() = Seteaza optiuni pentru un socket
- **shutdown**() = Opreste trimiterea si receptionarea datelor pentru un socket
- **socket**() = Creaza un socket
- **TransmitFile**() = Trimite un fisier printr-un socket conectat
- **TransmitPackets**() = Trimite date printr-un socket conectat
- **WSAAccept**() = Accepta o conexiune si permite transferul datelor
- **WSAAddressToString**() = Converteste toate componentele unei structuri sockaddr intr-un sir usor de citit
- **WSAAsyncGetHostByAddr**() = Obtine asincron informatii despre un host corespunzator unei adrese
- **WSAAsyncGetHostByName**() = Obtine asincron informatii despre un host corespunzator unui nume
- **WSAAsyncGetProtoByName**() = Obtine asincron informatii despre un protocol dupa nume
- **WSAAsyncGetProtoByNumber**() = Obtine asincron informatii despre un protocol dupa un numar
- **WSAAsyncGetServByName**() = Obtine asincron informatii despre un serviciu corespunzator unui nume si unui protocol
- **WSAAsyncGetServByPort**() = Obtine asincron informatii despre un servici corespunzator unui port si unui protocol

- **WSAAsyncSelect**() = Cere Windows-ului notificare bazata pe mesaje pentru un socket
- **WSACancelAsyncRequest**() = Opreste o operatiune asincrona neterminata
- **WSACleanup**() = Elibereaza resursele (ws2_32.dll)
- **WSAConnect**() = Stabileste o conexiune cu un alt socket
- **WSAGetLastError**() = Intoarce codul ultimei erori
- **WSAHtonl**() = Converteste un unsigned long din formatul hostului in formatul retelei
- **WSAHtons**() = Converteste un unsigned short din formatul hostului in formatul retelei
- **WSAIoctl**() = Controleaza modul unui socket
- **WSANtohl**() = Converteste un unsigned long din formatul retelei in formatul hostului
- **WSANtohs**() = Converteste un unsigned short din formatul retelei in formatul hostului
- **WSARecv**() = Primeste date de la un socket conectat
- **WSARecvDisconnect**() = Primeste date apoi se deconcteaza daca socketul e unul orientat pe conexiune
- **WSARecvEx**() = Primeste date de la un socket conectat
- **WSARecvFrom**() = Primeste o datagrama si pastreaza adresa sursa
- **WSARecvMsg**() = Primeste date de la un socket conectat sau neconectat
- **WSASend**() = Trimite date printr-un socket conectat
- **WSASendDisconnect**() = Trimite date si se deconecteaza
- **WSASendMsg**() = Trimite date de la un socket conectat sau unul neconectat
- **WSASendTo**() = Trimite date catre o destinatie
- **WSASetLastError**() = Seteaza ultima eroare
- **WSASetSocketSecurity**() = Controleaza securitatea unui socket
- **WSASocket**() = Creaza un socket
- **WSAStartup**() = Initializeaza Winsock DLL (ws2_32.dll)
- **WSAStringToAddress**() = Converteste un sir numeric intr-o adresa sockaddr

Inainte de a apela majoritatea functiilor Winsock trebuie apelata functia **WSAStartUp**. Functia initializeaza Winsock DLL si ofera detalii despre versiunea Winsock existenta. Dupa terminarea operatiilor Winsock trebuie apelata functia **WSACleanUp**. **WSACleanUp** elibereaza resursele Winsock. O aplicatie poate apela de mai mute ori **WSAStartUp**, dar pentru fiecare apel al sau trebuie apelata **WSACleanUp**.

WSAStartUp necesita 2 parametri: primul, versiunea Winsock necesara pentru ca programul sa ruleze, iar al doilea, un pointer la o structura de tipul *WSADATA*:

Functia este definita astfel:

```
Private Declare Function WSAStartup Lib "ws2_32.dll" (ByVal wVersionRequired As Integer, ByRef
lpWSAData As WSADATA) As Long
int WSAStartup( WORD wVersionRequested, LPWSADATA lpWSAData);
```

Si structura:

```
Private Type WSADATA
        wVersion As Integer
        wHighVersion As Integer
        szDescription As String * WSADESCRIPTION_LEN
        szSystemStatus As String * WSASYS_STATUS_LEN
        iMaxSockets As Integer
        iMaxUdpDg As Integer
        lpVendorInfo As Long
End Type
```

```
typedef struct WSAData {
  WORD wVersion;
  WORD wHighVersion;
  char szDescription[WSADESCRIPTION_LEN+1];
  char szSystemStatus[WSASYS_STATUS_LEN+1];
  unsigned short iMaxSockets;
  unsigned short iMaxUdpDg;
  char FAR *lpVendorInfo;
} WSADATA,
 *LPWSADATA;
```

Membrii acestei structuri sunt:
- ***wVersion*** = Versiunea pe care se asteapta Winsock sa o primeasca , un numar pe 2 octeti
- ***wHighVersion*** = Versiunea cea mai mare pe care o poate oferi implementarea Winsock
- ***szDescription*** = Un string terminat in NULL in care Winsock copiaza o descriere a implementarii Winsock
- ***szSystemStatus*** = Un string terminat in NULL in care Winsock copiaza un status relevant sau informatie de configurare
- ***iMaxSockets*** = Numarul maxim de socketi care pot fi folositi. Acest parametru ar trebui sa fie ignorat versiunile Winsock 2.0 si mai mari. Este pastrat pentru compatibilitate cu Winsock 1.1
- ***iMaxUdpDg*** = Marimea maxima a unui mesaj datagrama. La fel, ar trebui sa fie ignorat in versiunile 2.0 sau mai mari
- ***lpVendorInfo*** = Un pointer catre informatia despre furnizor. La fel, ar trebui ignorat

Daca **WSAStartup** se incheie cu succes, returneaza 0. Daca nu, va returna una din valorile: *WSASYSNOTREADY* , *WSAVERNOTSUPPORTED* , *WSAENETDOWN* , *WSAEINPROGRESS* , *WSAEPROCLIM* , *WSAEFAULT*. Erorile sunt descrise la finalul articolului. Pentru Visual Basic 6, nu uitati sa le declarati explicit.

La sfarsit trebuie apelata functia **WSACleanup**. Daca **WSACleanup** se incheie cu succes returneaza 0, daca nu returneaza: *WSANOTINITIALISED* daca Winsock nu a fost initializat, adica daca nu a fost apelata functia **WSAStartup**, *WSAENETDOWN* daca e o problema de retea sau *WSAEINPROGRESS* daca o functie cu blocare nu s-a terminat. Pentru mesajele de eroare vedeti: *"Mesaje de eroare"*.

```
Private Declare Function WSACleanup Lib "ws2_32.dll" () As Long
int WSACleanup(void);
```

Un exemplu care ne arata informatiile din din structura *WSAData*. Pentru initializare vom cere versiunea 2.2 Winsock. Acest lucru il vom specifica in primul parametru al functiei **WSAStartup**. Acest parametru este un numar pe 2 bytes: primul byte reprezinta versiunea minora, al doilea reprezinta versiunea majora. Pentru acest lucru, in C++ ne puteam folosi de macro-urile: *MAKEWORD* care creaza un numar pe 2 bytes din 2 numere pe 1 byte, in cazul nostru 2 si 2, *HIWORD* care returneaza octetul cel mai semnificativ si *LOWORD* care returneaza octetul cel mai putin semnificativ. Pentru versiunea v2.2 putem cere si 0x202 (zecimal 514).

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

int main()
```

```
{
   WSADATA wsaData;
   int rezultat;

   // Apelam functia

   rezultat = WSAStartup(MAKEWORD(2, 2), &wsaData);

   // Daca nu s-a efectuat cu succes, verificam eroarea care a intervenit

   if(rezultat != 0)
   {
            if(rezultat == WSASYSNOTREADY) printf("A intervenit eroarea WSASYSNOTREADY \r\n");
            else if(rezultat == WSAVERNOTSUPPORTED) printf("A intervenit eroarea
WSAVERNOTSUPPORTED \r\n");
            else if(rezultat == WSAEINPROGRESS) printf("A intervenit eroarea WSAEINPROGRESS \r\n");
            else if(rezultat == WSAEPROCLIM) printf("A intervenit eroarea WSAEPROCLIM \r\n");
            else if(rezultat == WSAEFAULT) printf("A intervenit eroarea WSAEFAULT \r\n");

            // Eliberam resursele

             WSACleanup();
   }

   // Daca s-a efectuat cu succes, afisam informatile oferite

   else
   {
      printf("Versiune asteptata: %d.%d\n", HIBYTE(wsaData.wVersion), LOBYTE(wsaData.wVersion));
      printf("Versiune maxima: %d.%d\n", HIBYTE(wsaData.wHighVersion),
LOBYTE(wsaData.wHighVersion));
      printf("Descriere: %s\n", wsaData.szDescription);
      printf("Status: %s\n", wsaData.szSystemStatus);

      // Ignorate in Winsock2

      // printf("Nr. max. de socketi: %d\n", wsaData.iMaxSockets);
      // printf("Datagrama maxima: %d\n", wsaData.iMaxUdpDg);
      // printf("Furnizor: %s\n", wsaData.lpVendorInfo);

      // Eliberam resursele

      WSACleanup();
   }

   return 0;
```

}

Pentru Visual Basic va trebui sa definim niste functii care returneaza *HIBYTE* si *LOBYTE*, iar pentru versiune vom folosi o constanta cu valoarea 514 sau &H202 in hexazecimal.

```
Private Const VERSIUNE As Long = &H202
```

Vom defini functiile astfel:

```
Private Function LoByte(ByVal numar As Integer) As Byte
LoByte = numar And &HFF
End Function

Private Function HiByte(ByVal numar As Integer) As Byte
HiByte = (numar And &HFF00) / 256
End Function
```

Daca tot am ajuns aici macar sa explic ce fac functiile, cum se realizeaza aceasta separare a octetilor. Sa dam un exemplu: Avem numarul 258 care scris binar arata astfel:

00000001 00000010

Cel mai semnificativ octet este 00000001 iar cel mai putin semnificativ octet este 00000010.

Pentru *LoByte*, este ceva mai simplu, va trebui sa scapam de *HiByte*, de primul octet. Pentru acest lucru ne vom ajuta de operatiile la nivel de bit. Ne vom folosi de AND la nivel de bit. O operatie la nivel de bit gen nr1 AND nr2 returneaza un numar rezultat din operatii bit la bit logice intre fiecare bit al numerelor. Spre exemplu, in functia noastra *LoByte* numar And &HFF (acelasi lucru ca &H00FF, primul octet este 0), pentru numarul nostru care este 514 (00000010 00000010 sau &H0202) rezultatul va fi urmatorul:

```
514 :    00000010 00000010
&HFF : 00000000 11111111
------------------------
Rez :  00000000 00000010
```

Se realizeaza un AND logic intre fiecare bit, cu cel de sub el. O astfel de operatie este 1 daca ambii biti sunt 1 si 0 altfel. Deci pentru a avea 0 pe toate pozitiile unui byte, pentru acel byte folosim AND 0. Astfel toti bitii vor fi de 0 si rezultatul va fi 0. Pentru a pastra un byte asa cum e, vom folosi &HFF (0xFF - 255). Acest octet are toti bitii 1, si rezultatul va fi primul octet. Asadar operatia de mai sus seteaza primul octet la 0, care nu se mai ia in considerare si ramane decat ultimul octet.

Pentru *HiByte*, vom proceda cam la fel: vom scapa de cel mai putin semnificativ octet, il vom seta la 0:

```
514 :       00000010 00000010
&HFF00 : 11111111 00000000
--------------------------
Rez :    00000010 00000000
```

Pentru acest lucru, folosim numar AND &HFF00: primul octet este &HFF care binar este 11111111 iar al doilea octet este 0. Astfel setam ultimul octet la 0. Insa nu e de ajuns (si nici necesar). Va trebui sa scapam de el. Practic acest lucru se realizeaza printr-o operatie de mutare la nivel de bit, catre dreata cu 8 pozitii

(rez >> 8), adica sa mutam toti bitii cu 8 pozitii spre dreapta, asadar al doilea octet va deveni primul si rezultatul va fi primul octet (00000000 0000001). Insa cum nu putem face acest lucru in VB6, vom face ceea ce face practic aceasta mutare si anume vom imparti rezultatul la 256 (2 ^ 8). O mutare la dreapta cu x pozitii se realizeaza "manual" impartind numarul la 2 ^ x, si o mutare la stanga inmultind cu 2 ^ x. Acest lucru e usor de inteles, sa luam un exemplu. Avem numarul 6 care binar este 00000110, calculul se face astfel: 6 = [0 * (2 ^ 0)] + [1 * (2 ^ 1)] + [1 * (2 ^ 2)], cu alte cuvinte, fiecare bit "valoreaza" 2 ^ (x - 1), unde x e pozitia bitului incepand de la dreapta la stanga, adica ultimul bit are valoarea 1 (sau 0), penultimul 2 (sau 0), cel anterior 2 ^ 2 = 4 (sau 0) si asa mai departe. Si numarul final reprezinta suma acestori valori. Prin mutarea la dreapta cu o pozitie, bitii care sunt 1 ajung cu o pozitie mai in fata, asta inseamna ca valoarea lor se dubleaza. Asadar, si suma finala, numarul la care am mutat bitii se dubleaza.

Rezultatul final va fi: 00000010 00000000 / 256 adica 512 / 256 = 2. Ceea ce voiam. Operatia de eliminare a octetului mai putin semnificativ nu era neaparat necesara, dar asa impartirea se realizeaza corect. Daca nu il eliminam am fi avut 514 / 256.

Daca nu ati inteles nu-i nimic, nu este atat de important.

De asemenea vom mai folosi o functie pentru string-uri de lungime fixa. De multe ori este necesar sa folosim string-uri de lungime fixa cand folosim API-uri deoarece acestea nu inteleg practic string-urile normale din VB6. Un string de lungime fixa, are o lungime fixa, pe care nu o poate depasi. Cand este transmis prin referinta catre o functie API, aceasta scrie datele in string, iar restul caracterelor pana la sfarsit sunt caractere *NULL* (*vbNullChar*). Pentru a nu avea porbleme cu adaugarea sau afisarea datelor dupa string-ul nostru de lungime fixa va trebui sa scapam de acele null-uri, sa pastram decat textul nostru. Vom face acest lucru usor:

```
Private Function TrimNull(ByVal sir As String) As String
TrimNull = Left(sir, InStr(1, sir, vbNullChar) - 1)
End Function
```

Functia este simpla, la fel ca si ideea. Cautam pozitia primului *vbNullChar* (caracter null) si returnam textul pana la acel caracter. Gasim primul *NULL* folosind functia InStr care returneaza positia acestuia:

```
InStr(1, sir, vbNullChar)
```

Apoi copiem textul din stanga pana la acea pozitie - 1 ca sa nu copiem si *NULL*-ul. Astfel, nu vom avea probleme cand vom incerca de exemplu sa afisam un text dupa un string:

```
MsgBox "Test: " & string_lungime_fixa_cu_null & "zZzZz"
```

In acest caz, daca stringul nostru are *NULL*-uri la sfarsit, daca nu am scapat de ele, textul "zZzZz" nu va mai aparea in fereastra de mesaj, insa daca vom scapa de *NULL*-uri nu vom avea probleme.

Si in final, sa facem acelasi lucru ca si in C++, sa afisam cateva detalii continute de structura *wsaData* in VB6. Dupa cum observati in VB6 este ceva mai complicat: trebuie sa declaram API-urile, structura *wsaData* si constantele. De asemenea trebuie sa definim functiile de mai sus:

```
' Functiile API necesare

Private Declare Function WSAStartup Lib "ws2_32.dll" (ByVal wVersionRequired As Integer, ByRef
lpWSAData As wsaData) As Long
Private Declare Function WSACleanup Lib "ws2_32.dll" () As Long
```

```
' Constante necesare

Private Const WSADESCRIPTION_LEN As Long = 256
Private Const WSASYS_STATUS_LEN As Long = 128
Private Const WSASYSNOTREADY As Long = 10091&
Private Const WSAVERNOTSUPPORTED As Long = 10092&
Private Const WSAEINPROGRESS As Long = 10036&
Private Const WSAEPROCLIM As Long = 10067&
Private Const WSAEFAULT As Long = 10014&

' Structura wsaData

Private Type wsaData
   wVersion As Integer
   wHighVersion As Integer
   szDescription As String * WSADESCRIPTION_LEN
   szSystemStatus As String * WSASYS_STATUS_LEN
   iMaxSockets As Integer
   iMaxUdpDg As Integer
   lpVendorInfo As Long
End Type

' Versiunea 2.2

Private Const VERSIUNE As Long = &H202

' Functiile pentru operatile la nivel de bit

Private Function LoByte(ByVal numar As Integer) As Byte
LoByte = numar And &HFF
End Function

Private Function HiByte(ByVal numar As Integer) As Byte
HiByte = (numar And &HFF00) / 256
End Function

' Pentru a preveni anumite erori

Private Function TrimNull(ByVal sir As String) As String
TrimNull = Left(sir, InStr(1, sir, vbNullChar) - 1)
End Function

Private Sub Form_Load()

Dim wsa As wsaData
Dim rezultat As Integer
```

```
Dim mesaj As String

' Apelam functia

rezultat = WSAStartup(VERSIUNE, wsa)

' Daca nu s-a efectuat cu succes, verificam eroarea care a intervenit

If rezultat <> 0 Then
   If rezultat = WSASYSNOTREADY Then MsgBox ("A intervenit eroarea WSASYSNOTREADY")
   ElseIf rezultat = WSAVERNOTSUPPORTED Then MsgBox ("A intervenit eroarea
WSAVERNOTSUPPORTED")
   ElseIf rezultat = WSAEINPROGRESS Then MsgBox ("A intervenit eroarea WSAEINPROGRESS")
   ElseIf rezultat = WSAEPROCLIM Then MsgBox ("A intervenit eroarea WSAEPROCLIM")
   ElseIf rezultat = WSAEFAULT Then MsgBox ("A intervenit eroarea WSAEFAULT")

   ' Eliberam resursele

   WSACleanup
Else

   ' Daca s-a efectuat cu succes, afisam informatile oferite

   mesaj = "Versiune asteptata: " & HiByte(wsa.wVersion) & "." & LoByte(wsa.wVersion) & vbCrLf
   mesaj = mesaj & "Versiune maxima: " & HiByte(wsa.wHighVersion) & "." &
LoByte(wsa.wHighVersion) & vbCrLf
   mesaj = mesaj & "Descriere:" & TrimNull(wsa.szDescription) & vbCrLf
   mesaj = mesaj & "Status: " & TrimNull(wsa.szSystemStatus) & vbCrLf

   ' Ignorate in Winsock 2
   ' mesaj = mesaj & "Nr. max. de socketi: " & wsa.iMaxSockets & vbCrLF
   ' mesaj = mesaj & "Datagrama maxima: " & wsa.iMaxUdpDg & vbCrLF
   ' mesaj = mesaj & "Furnizor: " & wsa.lpVendorInfo & vbCrLF

   MsgBox mesaj

   WSACleanup

End If

End Sub
```

Crearea unui socket

Putem crea in 2 moduri un socket: folosind functia **socket** sau folosind functia specifica Winsock, **WSASocket**.

Functia **socket** are nevoie de 3 parametri: familia de adrese (af), tipul socketului (lType) si protocolul utilizat de socket (protocol).

Private Declare Function socket Lib "ws2_32.dll" (ByVal af As Long, ByVal lType As Long, ByVal protocol As Long) As Long
SOCKET WSAAPI socket(int af, int type, int protocol);

Familia de adrese poate lua urmatoarele valori:
- ***AF_UNSPEC*** = Familie de adrese nespecificata
- ***AF_INET*** = Familia de adrese IPv4
- ***AF_INET6*** = Familia de adrese IPv6
- ***AF_BTH/AF_IRDA*** = Familia de adrese Bluetooth/IrDa, e necesar adaptor si driver instalat (informativ)

De fapt poate lua mai multe valori, dar nu ne intereseaza. Pe noi ne intereseaza decat valorile *AF_INET* pentru IPv4 si *AF_INET6* pentru IPv6.

Tipul socketului poate fi:
- ***SOCK_STREAM*** = Socket bazat pe conexiune. Foloseste protocolul TCP si familia de adrese *AF_INET* sau *AF_INET6*
- ***SOCK_DGRAM*** = Socket fara conexiune, bazat pe datagrame. Foloseste UDP si familia de adrese *AF_INET* sau *AF_INET6*
- ***SOCK_RAW*** = Socket brut care permite aplicatiei accesul la protocoale inferioare: IP, ICMP, mai exact la headerele acestora

Protocolul poate fi:
- ***IPPROTO_TCP*** = Transmission Control Protocol, este permins cand familia de adrese este *AF_INET* sau *AF_INET 6* si tipul este *SOCK_STREAM*
- ***IPPROTO_UDP*** = User Datagram Protocol, este permins cand familia de adrese este *AF_INET* sau *AF_INET 6* si tipul este *SOCK_DGRAM*
- ***IPPROTO_IP*** = Pentru acces la protocolul IP
- ***IPPROTO_IPV6*** = Pentru acces la protocolul IPv6

Daca se foloseste 0 pentru aceasta valoare se va selecta automat un protocol. Daca familia de adrese este *AF_INET* sau *AF_INET6* si tipul socketului este *SOCK_RAW*, protocolul va aparea in headerul pachetelor IP sau IPv6.

Exemple de folosire a functiei:

```
// Pentru un socket orientat pe conexiune
socket_handle = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)

// Pentru un socket orientat pe datagrame
socket_handle = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)
```

Asa se creaza de cele mai multe ori un socket. Primul exemplu pentru protocolul TCP, iar al doilea pentru UDP. Diferenta dintre ele este ca pentru un socket orientat pe conexiune, pentru transferul datelor va trebui mai intai realizata conexiunea logica intre client si server. Dar astfel se va putea cunoaste aparitia unei erori, cum ar fi pierderea conexiunii, sau vom putea sti daca datele nu au ajuns la destinatie. Singurul dezavantaj, destul de neglijabil daca nu e vorba de cantitati mari de date trimise, ar fi faptul ca headerele TCP sunt mai mari decat cele UDP.

Daca functia se incheie cu succes returneaza un handle de socket, daca nu, returneaza *INVALID_SOCKET*, iar codul de eroare specific poate fi gasit cu **WSAGetLastError** care returneaza codul de eroare al ultimei erori aparute.

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

int main()
{
   WSADATA wsaData;
   SOCKET hSock;

   WSAStartup(MAKEWORD(2, 2), &wsaData);

   hSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

   printf("Handlerul socketului: %d\r\n", hSock);

   WSACleanup();

   return 0;
}
```

In C++ putem folosi tipul *SOCKET* care este de fapt *unsigned int*. Pentru Visual Basic 6 putem folosi *Long*. Practic acest handle de socket este un numar unic prin care socketul e recunoscut.

Si in Visual Basic 6 exemplul arata cam asa:

```
' Functiile pe care le vom apela

Private Declare Function WSAStartup Lib "ws2_32.dll" (ByVal wVersionRequired As Integer, ByRef
lpWSAData As wsaData) As Long
Private Declare Function WSACleanup Lib "ws2_32.dll" () As Long
Private Declare Function socket Lib "ws2_32.dll" (ByVal af As Long, ByVal lType As Long, ByVal protocol
As Long) As Long

' Constantele necesare

Private Const WSADESCRIPTION_LEN As Long = 256
Private Const WSASYS_STATUS_LEN As Long = 128

' Structura wsaData

Private Type wsaData
   wVersion As Integer
   wHighVersion As Integer
   szDescription As String * WSADESCRIPTION_LEN
```

```
    szSystemStatus As String * WSASYS_STATUS_LEN
    iMaxSockets As Integer
    iMaxUdpDg As Integer
    lpVendorInfo As Long
End Type

' Versiunea 2.2

Private Const VERSIUNE As Long = &H202

' Doar constantele pe care le folosim in acest exemplu
' Pentru o lista completa cu constante folositi API Viewer 2004
' Daca nu gasiti o constanta o cautati in winsock2.h sau ws2def.h pentru SDK mai nou

Private Const AF_INET As Long = 2
Private Const SOCK_STREAM As Long = 1
Private Const IPPROTO_TCP As Long = 6

Private Sub Form_Load()

Dim wsa As wsaData
Dim hSock As Long

WSAStartup VERSIUNE, wsa

' Cream socketul

hSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)

MsgBox "Handlerul socketului: " & hSock

WSACleanup

End Sub
```

Pentru a crea un socket putem folosi de asemenea functia specifica Winsock **WSASocket**.

Functia **WSASocket** are 6 parametri: familia de adrese (af), tipul (type), protocolul (protocol), un pointer la o structura *WSAPROTOCOL_INFO* care defineste caracteristicile socketului care va fi creat (*lpProtocolInfo*) (in VB6, ca sa nu mai declaram structura, vom folosi tipul *Long* si valoarea 0 pentru acest parametru in VB6 – functia necesita de fapt un pointer care e o de fapt o adresa de memorie pe 4 octeti, deci putem face asta), un parametru rezervat (g) si un parametru flag care specifica un atribut pentru socket (*dwFlags*). Primii 3 parametri sunt identici cu cei ai functiei socket. Daca este specificata o structura *WSAPROTOCOL_INFO* pentru al IV-lea parametru, se vor folosi datele din structura pentru a defini socketul. Pentru a crea un socket astfel, in VB6 vom folosi:

```
Private Declare Function WSASocket Lib "ws2_32.dll" Alias "WSASocketW" (ByVal af As Long,
ByVal lType As Long, ByVal protocol As Long, ByVal lpProtocolInfo As Long, ByVal g As Long,
```

ByVal dwFlags As Long) As Long

Atentie, functia practic nu e corecta. Parametrul *lpProtocolInfo* ar trebui sa fie de tipul *LPWSAPROTOCOL_INFO*, iar parametrul g de tipul *GROUP*. Si ambii ar trebuii transmisi prin referinta. Insa nu o sa ii folosim, si pentru a nu avea probleme ii vom transmite prin valoare, si ii vom declara de tipul Long (VB6), ceea ce practic nu este corect, dar asa nu mai e nevoie sa declaram acele structuri.

Declaratia corecta, in C++ este urmatoarea:

```
SOCKET WSASocket(int af, int type, int protocol, LPWSAPROTOCOL_INFO lpProtocolInfo, GROUP g, DWORD dwFlags);
```

Exemplu:

```
socket_handle = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, 0, 0, 0);
```

Daca functia se incheie cu succes va returna un handle de socket. Daca nu se incheie cu succes va returna *INVALID_SOCKET*.

Parametrul *dwFlags* poate lua mai multe valori, de exemplu poate lua valoarea *WSA_FLAG_OVERLAPPED*. Astfel, pentru socketul creat se vor putea folosi functiile **WSASend**, **WSASendTo**, **WSARecv**, **WSARecvFrom**, and **WSAIoctl** pentru mai multe operatii simultane. Mai poate lua si alte valori, dar doar pentru multicast, nu ne intereseaza.

De asemenea, pentru fiecare socket creat vor trebui eliberate la final resursele. Vom face acest lucru folosind functia **closesocket**:

```
int closesocket(SOCKET s);
Private Declare Function closesocket Lib "ws2_32.dll" (ByVal s As Long) As Long
```

Functia elibereaza resursele, si daca o face cu succes returneaza 0, in caz de eroare returneaza *SOCKET_ERROR*.

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

int main()
{
  WSADATA wsaData;
  SOCKET hSock;

  WSAStartup(MAKEWORD(2, 2), &wsaData);

  // Cream socketul

  hSock = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0,
WSA_FLAG_OVERLAPPED);

  printf("Handlerul socketului: %d\r\n", hSock);
```

```
   // Il distrugem

   closesocket(hSock);

   WSACleanup();

   return 0;
}
```

Pentru VB6 va trebui sa declaram in plus functiile **WSASocket** si **closesocket**. Restul e la fel.

```
Private Declare Function WSAStartup Lib "ws2_32.dll" (ByVal wVersionRequired As Integer, ByRef
lpWSAData As wsaData) As Long
Private Declare Function WSACleanup Lib "ws2_32.dll" () As Long
Private Declare Function WSASocket Lib "ws2_32.dll" Alias "WSASocketW" (ByVal af As Long, ByVal
lType As Long, ByVal protocol As Long, ByVal lpProtocolInfo As Long, ByVal g As Long, ByVal dwFlags
As Long) As Long
Private Declare Function closesocket Lib "ws2_32.dll" (ByVal s As Long) As Long

Private Const WSADESCRIPTION_LEN As Long = 256
Private Const WSASYS_STATUS_LEN As Long = 128

Private Type wsaData
   wVersion As Integer
   wHighVersion As Integer
   szDescription As String * WSADESCRIPTION_LEN
   szSystemStatus As String * WSASYS_STATUS_LEN
   iMaxSockets As Integer
   iMaxUdpDg As Integer
   lpVendorInfo As Long
End Type

Private Const VERSIUNE As Long = &H202
Private Const AF_INET As Long = 2
Private Const SOCK_STREAM As Long = 1
Private Const IPPROTO_TCP As Long = 6
Private Const WSA_FLAG_OVERLAPPED As Long = &H1

Private Sub Form_Load()

Dim wsa As wsaData
Dim hSock As Long

WSAStartup VERSIUNE, wsa

' Cream socketul
```

```
hSock = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, 0, 0, WSA_FLAG_OVERLAPPED)

MsgBox "Handlerul socketului: " & hSock

' Il distrugem

closesocket hSock

WSACleanup

End Sub
```

Este recomandata folosirea flagului *WSA_FLAG_OVERLAPPED*. Acest flag e setat automat daca socketul este creat cu functia **socket**, insa pentru functia **WSASocket** acesta trebuie specificat explicit. Daca nu se foloseste, socketul este unul cu blocare si daca nu este specificat nu putem seta un timeout pentru operatiile de trimitere/primire date. De exemplu, daca socketul este cu blocare, functia **recv** nu returneaza nici o valoare pana cand nu incheie primirea datelor. Acest lucru, dintr-o problema sau alta poate dura destul de mult, dar daca folosim acest flag putem seta un timeout. Alte functii care pot provoca astfel de probleme sunt: **recvfrom**, **send**, **sendto**.

Sa complicam putin lucrurile, sa cream un socket folosind o structura *WSAPROTOCOL_INFO*. Nu vom completa noi elementele structurii care reprezinta datele complete despre un protocol (*WSAPROTOCOL_INFO*), si vom cauta protocolul dorit folosind functia **WSAEnumProtocols**. Functia ofera informatii despre toate protocoalele de transport.

```
int WSAEnumProtocols(LPINT lpiProtocols, LPWSAPROTOCOL_INFO lpProtocolBuffer, LPDWORD lpdwBufferLength);

Private Declare Function WSAEnumProtocols Lib "ws2_32.dll" Alias "WSAEnumProtocolsA" (ByRef lpiProtocols As Long, ByRef lpProtocolBuffer As WSAPROTOCOL_INFOA, ByRef lpdwBufferLength As Long) As Long
```

Functia are 3 parametrii:
- **lpiProtocols** = Vector cu valori pentru intrarea *iProtocol* a structurii. Daca acest parametru e *NULL*, se vor returna toate protocoalele. Daca e specificat sunt returnate doar protocoalele specificate prin acest vector. Il folosim daca de exemplu vrem sa alegem doar protocoalele TCP
- **lpProtocolBuffer** = Un buffer pentru structuri *WSAPROTOCOL_INFO*. El va stoca informatiile despre protocoale
- **lpdwBufferLength** = Daca nu e specificata o valoare, functia va stoca in el marimea necesara bufferului. Bufferul necesita o marime destul de mare pentru stocarea datelor, functia stocheaza aici marimea necesara pe care o vom folosi intr-un alt apel al functiei pentru a putea scrie datele corect in buffer

Daca nu apare nici o eroare functia returneaza numarul de protocoale scrise in buffer. In caz de eroare returneaza *SOCKET_ERROR*, iar cu **WSAGetLastError** se pot obtine urmatoarele posibile erori: *WSANOTINITIALISED*, *WSAENETDOWN*, *WSAEINPROGRESS*, *WSAEINVAL*, *WSAENOBUFS*, *WSAEFAULT*. Pentru detalii despre continutul structurii *WSAPROTOCOL_INFO*: http://msdn.microsoft.com/en-us/library/ms741675(VS.85).aspx .

Pe noi nu ne intereseaza toate, sa vedem doar cateva elemente ale structurii:
- **dwServiceFlags1** - Un set de flag-uri (optiuni) pentru protocolul corespunzator. Aceasta intrare este un

numar pe 4 octeti. Fiecare flag care il poate avea un protocol reprezinta un bit al acestui numar. Daca un protocol are setat acel bit, inseamna ca prezinta caracteristicile acelui flag. Spre exemplu, daca un protocol are setat ultimul bit, inseamna ca are flag-ul *XP1_CONNECTIONLESS*, ceea ce inseamna ca acel protocol este unul fara conexiune bazat pe datagrame. Pentru a verifica un flag, vom folosi un "si" logic cu acel flag: *protocol.dwServiceFlags1 & XP1_CONNECTIONLESS*. Aceasta expresie este 1 numai daca acel protocol are acel flag setat
- **dwCatalogEntryId** - Un identificator unic oferit de ws2_32.dll pentru o astfel de structura
- **iVersion** - Versiunea protocolului
- **iAddressFamily** - Familia de adrese a protocolului, cea care se foloseste si la functia (**WSA**)**socket**
- **iSocketType** - Tipul socket-ului, cel care se foloseste si la functia (**WSA**)**socket**
- **iProtocol** - Protocolul, la fel, cel care se foloseste si la functia (**WSA**)**socke**t
- **iNetworkByteOrder** - Ordinea octetilor, bigendian sau littleendian
- **szProtocol** - Un nume asociat protocolului, sa poata fi citit (sir de caractere)

Sa vedem totusi cateva flag-uri pentru *dwServiceFlags1* si ce reprezinta acestea:
- **XP1_CONNECTIONLESS** - Protocolul este unul fara conexiune, orientat pe datagrame
- **XP1_GUARANTEED_DELIVERY** - Garanteaza ca pachetele ajung la destinatie
- **XP1_GUARANTEED_ORDER** - Garaneaza ca datele ajung la destinatie in ordinea in care au fost trimise
- **XP1_MESSAGE_ORIENTED** - Protocol orientat pe mesaje
- **XP1_EXPEDITED_DATA** - Suporta trimiterea datelor urgente
- **XP1_(DIS)CONNECT_DATA** - Suporta trimiterea datelor la (de)conectare
- **XP1_QOS_SUPPORTED** - Suporta Quality of Service

Folosirea acestei functii este putin mai complicata. Va trebui sa o apelam mai intai si sa nu specificam o valoare pentru ultimul parametru, deoarece nu stim ce marime trebuie sa aiba bufferul in care se vor memora datele de care avem nevoie. Vom trasnsmite pentru acest parametru o variabila prin referinta, iar functia va scrie in ea marimea necesara buffer-ului si va returna un mesaj de eroare (*WSAENOBUFS*). Dupa ce aflam aceasta marime, putem afla cate protocoale avem prin impartirea marimii totale la marimea unui structuri *WSAPROTOCOL_INFO* care reprezinta un protocol. Apoi vom crea un vector de *WSAPROTOCOL_INFO* si cand vom apela a doua oara functia, aceasta va avea marimea buffer-ului corecta si va scrie in vectorul nostru protocoalele disponibile. Apoi, pentru fiecare protocol in parte vom afisa mai multe informatii, mai mult sau mai putin utile.

Mai jos avem un exemplu de afisare a protocoalelor folosind aceasta functie.

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

int main()
{
   WSADATA wsaData;

   // Bufferul in care vom stoca datele despre protocoale

   WSAPROTOCOL_INFO *proto_info = NULL;
   int ret, i;

   // In aceasta variabila se va stoca marimea necesara bufferului

   DWORD marime_buffer;
```

```
  WSAStartup(MAKEWORD(2, 2), &wsaData);

  // Apelam functia si ne asteptam la eroare

  ret = WSAEnumProtocols(NULL, proto_info, &marime_buffer);

  // Verificam daca s-a produs o eroare, daca bufferul e de ajuns de mare, de obicei este prea mic

  if(WSAGetLastError() == WSAENOBUFS) printf("Bufferul este prea mic. Marime necesare: %d\n",
marime_buffer);

  // In marime_buffer e stocat acum spatiul necesar pentru buffer
  // Ne putem folosi de el pentru a calcula numarul de protocoale
  // Si cream un vector de WSAPROTOCOL_INFO[nr_de_protocoale]
  // Pentru asta impartim spatiul necesar la marimea unei structuri WSAPROTOCOL_INFO

  proto_info = new WSAPROTOCOL_INFO[marime_buffer / sizeof(WSAPROTOCOL_INFO)];

  // Apelam din nou functia care scrie datele in bufferul proto_info

  ret = WSAEnumProtocols(NULL, proto_info, &marime_buffer);

  // Functia returneaza nr. de protocoale, afisam acest numar

  printf("Nr. de protocoale: %d\n\n", ret);

  // Apoi afisam fiecare protocol

  for(i=0; i <= ret - 1; i++)
  {
    printf("\nProtocol: %s\n", proto_info[i].szProtocol);

    if(!proto_info[i].dwServiceFlags1) printf("- dwServiceFlags1: ( 0 ) Nici un flag \n");
    else printf("- dwServiceFlags1 ( %d ) : ", proto_info[i].dwServiceFlags1);

    // Verificam fiecare flag in parte

    if(proto_info[i].dwServiceFlags1 & XP1_CONNECTIONLESS) printf(" XP1_CONNECTIONLESS");
    if(proto_info[i].dwServiceFlags1 & XP1_GUARANTEED_DELIVERY) printf("
XP1_GUARANTEED_DELIVERY");
    if(proto_info[i].dwServiceFlags1 & XP1_GUARANTEED_ORDER) printf("
XP1_GUARANTEED_ORDER");
    if(proto_info[i].dwServiceFlags1 & XP1_MESSAGE_ORIENTED) printf("
XP1_MESSAGE_ORIENTED");
    if(proto_info[i].dwServiceFlags1 & XP1_PSEUDO_STREAM) printf(" XP1_PSEUDO_STREAM");
    if(proto_info[i].dwServiceFlags1 & XP1_GRACEFUL_CLOSE) printf("XP1_GRACEFUL_CLOSE");
```

```
      if(proto_info[i].dwServiceFlags1 & XP1_EXPEDITED_DATA) printf("XP1_EXPEDITED_DATA");
      if(proto_info[i].dwServiceFlags1 & XP1_CONNECT_DATA) printf("XP1_CONNECT_DATA");
      if(proto_info[i].dwServiceFlags1 & XP1_DISCONNECT_DATA) printf("
XP1_DISCONNECT_DATA");
      if(proto_info[i].dwServiceFlags1 & XP1_SUPPORT_BROADCAST) printf("
XP1_SUPPORT_BROADCAST");
      if(proto_info[i].dwServiceFlags1 & XP1_SUPPORT_MULTIPOINT) printf("
XP1_SUPPORT_MULTIPOINT");
      if(proto_info[i].dwServiceFlags1 & XP1_MULTIPOINT_CONTROL_PLANE) printf("
XP1_MULTIPOINT_CONTROL_PLANE");
      if(proto_info[i].dwServiceFlags1 & XP1_MULTIPOINT_DATA_PLANE) printf("
XP1_MULTIPOINT_DATA_PLANE");
      if(proto_info[i].dwServiceFlags1 & XP1_QOS_SUPPORTED) printf(" XP1_QOS_SUPPORTED");
      if(proto_info[i].dwServiceFlags1 & XP1_INTERRUPT) printf("XP1_INTERRUPT");
      if(proto_info[i].dwServiceFlags1 & XP1_UNI_SEND) printf("XP1_UNI_SEND");
      if(proto_info[i].dwServiceFlags1 & XP1_UNI_RECV) printf("XP1_UNI_RECV");
      if(proto_info[i].dwServiceFlags1 & XP1_IFS_HANDLES) printf("XP1_IFS_HANDLES");
      if(proto_info[i].dwServiceFlags1 & XP1_PARTIAL_MESSAGE) printf("
XP1_PARTIAL_MESSAGE");

      // Apoi afisam celelalte informatii din structura

      printf("\n- dwCatalogEntryId = %u\n", proto_info[i].dwCatalogEntryId);
      printf("- ProtocolChain.ChainLen = %d ", proto_info[i].ProtocolChain.ChainLen);

      // Verificam daca e un protocol de baza sau unul compus din mai multe "straturi"

      if (proto_info[i].ProtocolChain.ChainLen == 1) printf(" ( Base service provider )\n");
      else if (proto_info[i].ProtocolChain.ChainLen == 0) printf(" ( Layered service provider )\n");

      // Afisam si celelalte informatii utile din structura

      printf("- iVersion = %d\n", proto_info[i].iVersion);
      printf("- iAddressFamily = %d\n", proto_info[i].iAddressFamily);
      printf("- iMaxSockAddr = %d\n", proto_info[i].iMaxSockAddr);
      printf("- iMinSockAddr = %d\n", proto_info[i].iMinSockAddr);
      printf("- iProtocol = %d\n", proto_info[i].iProtocol);
      printf("- iNetworkByteOrder = %s\n", ((proto_info[i].iNetworkByteOrder == LITTLEENDIAN) ?
"LITTLEENDIAN" : "BIGENDIAN"));
      printf("- dwMessageSize = %u\n", proto_info[i].dwMessageSize);
    }

    delete []proto_info;
    WSACleanup();

    return 0;
```

```
}
```

In Visual Basic 6, pentru a putea folosi aceasta functie va trebui sa definim structura *WSAPROTOCOL_INFO* si structura *WSAPROTOCOLCHAIN*. De asemenea avem cateva constante noi. Exemplul de mai jos va afisa o eroare asteptata, numarul de protocoale si fiecare protocol in parte (atentie, vor aparea MULTE *MsgBox*-uri). Dupa cum observati este putin mai lung decat in C++:

```
Option Explicit

' Functiile de care avem nevoie

Private Declare Function WSAStartup Lib "ws2_32.dll" (ByVal wVersionRequired As Integer, ByRef
lpWSAData As wsaData) As Long
Private Declare Function WSACleanup Lib "ws2_32.dll" () As Long
Private Declare Function WSAEnumProtocols Lib "ws2_32.dll" Alias "WSAEnumProtocolsA" (ByVal
lpiProtocols As Long, ByRef lpProtocolBuffer As Any, ByRef lpdwBufferLength As Long) As Long
Private Declare Function WSAGetLastError Lib "ws2_32.dll" () As Long

' Constantele necesare

Private Const WSADESCRIPTION_LEN As Long = 256
Private Const WSASYS_STATUS_LEN As Long = 128
Private Const MAX_PROTOCOL_CHAIN As Long = 7
Private Const WSAPROTOCOL_LEN As Long = 255
Private Const VERSIUNE As Long = &H202
Private Const WSA_FLAG_OVERLAPPED As Long = &H1
Private Const WSAENOBUFS As Long = 10055&

' Flag-urile

Private Const XP1_CONNECT_DATA As Long = &H80
Private Const XP1_CONNECTIONLESS As Long = &H1
Private Const XP1_DISCONNECT_DATA As Long = &H100
Private Const XP1_EXPEDITED_DATA As Long = &H40
Private Const XP1_GRACEFUL_CLOSE As Long = &H20
Private Const XP1_GUARANTEED_DELIVERY As Long = &H2
Private Const XP1_GUARANTEED_ORDER As Long = &H4
Private Const XP1_IFS_HANDLES As Long = &H20000
Private Const XP1_INTERRUPT As Long = &H4000
Private Const XP1_MESSAGE_ORIENTED As Long = &H8
Private Const XP1_MULTIPOINT_CONTROL_PLANE As Long = &H800
Private Const XP1_MULTIPOINT_DATA_PLANE As Long = &H1000
Private Const XP1_PARTIAL_MESSAGE As Long = &H40000
Private Const XP1_PSEUDO_STREAM As Long = &H10
Private Const XP1_QOS_SUPPORTED As Long = &H2000
Private Const XP1_SUPPORT_BROADCAST As Long = &H200
Private Const XP1_SUPPORT_MULTIPOINT As Long = &H400
```

```
Private Const XP1_UNI_RECV As Long = &H10000
Private Const XP1_UNI_SEND As Long = &H8000

' Pentru intrarea iNetworkByteOrder a structurii

Private Const LITTLEENDIAN As Long = &H1
Private Const BIGENDIAN As Long = &H0

Private Type wsaData
  wVersion As Integer
  wHighVersion As Integer
  szDescription As String * WSADESCRIPTION_LEN
  szSystemStatus As String * WSASYS_STATUS_LEN
  iMaxSockets As Integer
  iMaxUdpDg As Integer
  lpVendorInfo As Long
End Type

' Pentru intrarea ProtocolChain, tipul protocolului, de baza sau compus din mai multe "straturi"

Private Type WSAPROTOCOLCHAIN
  ChainLen As Long
  ChainEntries(0 To MAX_PROTOCOL_CHAIN - 1) As Long
End Type

' Structura ce contine informatiile despre fiecare protocol in parte

Private Type WSAPROTOCOL_INFO
  dwServiceFlags1 As Long
  dwServiceFlags2 As Long
  dwServiceFlags3 As Long
  dwServiceFlags4 As Long
  dwProviderFlags As Long
  ProviderId(0 To 15)  As Byte
  dwCatalogEntryId As Long
  ProtocolChain As WSAPROTOCOLCHAIN
  iVersion As Integer
  iAddressFamily As Long
  iMaxSockAddr As Long
  iMinSockAddr As Long
  iSocketType As Long
  iProtocol As Long
  iProtocolMaxOffset As Long
  iNetworkByteOrder As Long
  iSecurityScheme As Long
  dwMessageSize As Long
```

```
  dwProviderReserved As Long
  szProtocol(0 To WSAPROTOCOL_LEN - 1) As Byte
End Type

' Pentru a evita aparitia erorilor provocate de sirurile de lungime fixa

Private Function TrimNull(ByVal sir As String) As String
TrimNull = Left(sir, InStr(1, sir, vbNullChar) - 1)
End Function

Private Sub Form_Load()

Dim wsa As wsaData
Dim proto_info() As WSAPROTOCOL_INFO    ' Vectorul de WSAPROTOCOL_INFO

' In marime_buffer vom memora marimea necesara buffer-ului

Dim ret, marime_buffer, protocoale, i As Long
Dim strProtocol As String    ' Vom stoca aici fiecare protocol in parte pentru a-l afisa in MsgBox

WSAStartup VERSIUNE, wsa

' Redimensionam vectorul pentru a avea un singur element, spatiu prea mic

ReDim proto_info(0) As WSAPROTOCOL_INFO

' Apelam functia si ne asteptam la eroare
' Dupa apelul functiei marime_buffer va stoca marimea necesara bufferului

ret = WSAEnumProtocols(0, proto_info(0), marime_buffer)

' Verificam daca a intervenit eroarea cu buffer prea mic si afisam un mesaj

If WSAGetLastError = WSAENOBUFS Then MsgBox "Buffer prea mic. Marime: " & marime_buffer

' Calculam nr. de protocoale folosindu-ne de marimea unui protocol si de marimea necesara tuturor

protocoale = (marime_buffer) / LenB(proto_info(0))

' Afisam nr. de protocoale ( Informativ )

MsgBox "Protocoale: " & protocoale

' Redimensionam vectorul la nr. de protocoale - 1 elemente ( vectorii sunt in baza 0 )

ReDim proto_info(0 To protocoale - 1) As WSAPROTOCOL_INFO
```

```
' Apelam din nou functia care scrie in proto_info() informatiile necesare

ret = WSAEnumProtocols(0, proto_info(0), marime_buffer)

For i = 0 To ret - 1

  ' Parcurgem vectorul si stocam intr-o variabila cateva informatii despre protocol
  ' szProtocol e un byte array, va trebui sa il convertim la string si sa scapam de NULL-urile de la sfarsit

  strProtocol = vbCrLf & "Protocol: " & TrimNull(StrConv(proto_info(i).szProtocol, vbUnicode)) & vbCrLf

  ' Verificam daca exista flaguri

  If proto_info(i).dwServiceFlags1 = 0 Then
    strProtocol = strProtocol & "- dwServiceFlags1: ( 0 ) Nici un flag " & vbCrLf
  Else
    strProtocol = strProtocol & "- dwServiceFlags1: ( " & CStr(proto_info(i).dwServiceFlags1) & " ) : " & vbCrLf
  End If

  ' Verificam existenta fiecarui flag in parte

  If proto_info(i).dwServiceFlags1 And XP1_CONNECTIONLESS Then strProtocol = strProtocol & " XP1_CONNECTIONLESS"
  If proto_info(i).dwServiceFlags1 And XP1_GUARANTEED_DELIVERY Then strProtocol = strProtocol & " XP1_GUARANTEED_DELIVERY"
  If proto_info(i).dwServiceFlags1 And XP1_GUARANTEED_ORDER Then strProtocol = strProtocol & " XP1_GUARANTEED_ORDER"
  If proto_info(i).dwServiceFlags1 And XP1_MESSAGE_ORIENTED Then strProtocol = strProtocol & " XP1_MESSAGE_ORIENTED"
  If proto_info(i).dwServiceFlags1 And XP1_PSEUDO_STREAM Then strProtocol = strProtocol & " XP1_PSEUDO_STREAM"
  If proto_info(i).dwServiceFlags1 And XP1_GRACEFUL_CLOSE Then strProtocol = strProtocol & " XP1_GRACEFUL_CLOSE"
  If proto_info(i).dwServiceFlags1 And XP1_EXPEDITED_DATA Then strProtocol = strProtocol & " XP1_EXPEDITED_DATA"
  If proto_info(i).dwServiceFlags1 And XP1_CONNECT_DATA Then strProtocol = strProtocol & " XP1_CONNECT_DATA"
  If proto_info(i).dwServiceFlags1 And XP1_DISCONNECT_DATA Then strProtocol = strProtocol & " XP1_DISCONNECT_DATA"
  If proto_info(i).dwServiceFlags1 And XP1_SUPPORT_BROADCAST Then strProtocol = strProtocol & " XP1_SUPPORT_BROADCAST"
  If proto_info(i).dwServiceFlags1 And XP1_SUPPORT_MULTIPOINT Then strProtocol = strProtocol & " XP1_SUPPORT_MULTIPOINT"
```

```
   If proto_info(i).dwServiceFlags1 And XP1_MULTIPOINT_CONTROL_PLANE Then strProtocol =
strProtocol & " XP1_MULTIPOINT_CONTROL_PLANE"
   If proto_info(i).dwServiceFlags1 And XP1_MULTIPOINT_DATA_PLANE Then strProtocol =
strProtocol & " XP1_MULTIPOINT_DATA_PLANE"
   If proto_info(i).dwServiceFlags1 And XP1_QOS_SUPPORTED Then strProtocol = strProtocol & "
XP1_QOS_SUPPORTED"
   If proto_info(i).dwServiceFlags1 And XP1_INTERRUPT Then strProtocol = strProtocol & "
XP1_INTERRUPT"
   If proto_info(i).dwServiceFlags1 And XP1_UNI_SEND Then strProtocol = strProtocol & "
XP1_UNI_SEND"
   If proto_info(i).dwServiceFlags1 And XP1_UNI_RECV Then strProtocol = strProtocol & "
XP1_UNI_RECV"
   If proto_info(i).dwServiceFlags1 And XP1_IFS_HANDLES Then strProtocol = strProtocol & "
XP1_IFS_HANDLES"
   If proto_info(i).dwServiceFlags1 And XP1_PARTIAL_MESSAGE Then strProtocol = strProtocol & "
XP1_PARTIAL_MESSAGE"

   ' Stocam si alte informatii din structura in variabila noastra string

   strProtocol = strProtocol & vbCrLf & "- dwCatalogEntryId = " & proto_info(i).dwCatalogEntryId &
vbCrLf
   strProtocol = strProtocol & "- ProtocolChain.ChainLen = " & proto_info(i).ProtocolChain.ChainLen

   ' Verificam si daca este un protocol de baza, sau unul compus din mai multe "straturi"

   If proto_info(i).ProtocolChain.ChainLen = 1 Then
      strProtocol = strProtocol & " ( Base service provider ) " & vbCrLf
   ElseIf proto_info(i).ProtocolChain.ChainLen = 0 Then
      strProtocol = strProtocol & " ( Layered service provider ) " & vbCrLf
   End If

   ' Celelalte informatii utile

   strProtocol = strProtocol & "- iVersion = " & proto_info(i).iVersion & vbCrLf
   strProtocol = strProtocol & "- iAddressFamily = " & proto_info(i).iAddressFamily & vbCrLf
   strProtocol = strProtocol & "- iMaxSockAddr = " & proto_info(i).iMaxSockAddr & vbCrLf
   strProtocol = strProtocol & "- iMinSockAddr = " & proto_info(i).iMinSockAddr & vbCrLf
   strProtocol = strProtocol & "- iProtocol = " & proto_info(i).iProtocol & vbCrLf
   strProtocol = strProtocol & "- iNetworkByteOrder = " & IIf(proto_info(i).iNetworkByteOrder =
LITTLEENDIAN, "LITTLEENDIAN", "BIGENDIAN") & vbCrLf
   strProtocol = strProtocol & "- dwMessageSize = " & proto_info(i).dwMessageSize & vbCrLf

   ' Afisam datele obtinute

   MsgBox strProtocol
Next
```

```
ReDim proto_info(0)
WSACleanup

Unload Me

End Sub
```

Dar cu ce ne ajuta aceste structuri la crearea unui socket? Simplu. Aceste structuri ne ofera toate informatiile necesare pentru selectarea unui protocol. Tot ce avem de facut e sa verificam ce protocol ne trebuie in functie de aceste structuri. Vom parcurge vectorul si vom selecta structura care ne ofera protocolul dorit. De exemplu, pentru TCP, vom verifica daca *iProtocol* e *IPPROTO_TCP* si familia de adrese este *AF_INET*. Deocamdata decat atat, nu ne intereseaza mai mult. In cazul meu am gasit 2 protocoale care ofera aceste facilitati: *MSAFD Tcpip [TCP/IP]* si *RSVP TCP Service Provider*, diferenta dintre ele fiind ca cel din urma ofera suport pentru QoS (Quality of Service) - care se foloseste pentru fuxuri multimedia gen VoIP, jocuri onlie... Nu avem nevoie.

Un alt lucru important pentru selectarea unui protocol folosind o structura *WSAPROTOCOL_INFO*, e faptul ca primii 3 parametrii ai functiei **WSASocket** (familia de adrese, tipul socketului si protocolul) vor trebui setati folosind constanta *FROM_PROTOCOL_INFO*. Un alt lucru important este primul parametru al functiei **WSAEnumProtocols**. Acesta reprezinta un vector de int in care specificam ce protocoale sa selecteze, de exemplu putem selecta doar protocoalele TCP. Asadar codul final va arata cam asa:

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

int main()
{
   WSADATA wsaData;
   WSAPROTOCOL_INFO *proto_info = NULL;
   int ret, i;
   DWORD marime_buffer;
   SOCKET hSock;

   WSAStartup(MAKEWORD(2, 2), &wsaData);

   // Apelam functia si asteptam eroare (buffer prea mic)

   ret = WSAEnumProtocols(NULL, proto_info, &marime_buffer);

   if(WSAGetLastError() == WSAENOBUFS) printf("Bufferul este prea mic. Marime necesare: %d\n",
marime_buffer);

   proto_info = new WSAPROTOCOL_INFO[marime_buffer / sizeof(WSAPROTOCOL_INFO)];

   ret = WSAEnumProtocols(NULL, proto_info, &marime_buffer);

   // Afisam cate protocoale avem acum
```

```
   printf("Nr. de protocoale: %d\n", ret);

   // Parcurgem vectorul cu protocoale si cautam ce dorim

   for(i=0; i <= ret - 1; i++)
   {
      // Verificam daca protocolul curent are ceea ce ne trebuie
      // Daca familia de adrese e AF_INET si protocolul este TCP (nu mai e nevoie sa verificam acest lucru,
informativ)

      if(proto_info[i].iAddressFamily == AF_INET && proto_info[i].iProtocol == IPPROTO_TCP)
      {
         printf("\nProtocol selectat: %s\n", proto_info[i].szProtocol);

         // Primii 3 parametrii sunt FROM_PROTOCOL_INFO

         hSock = WSASocket(FROM_PROTOCOL_INFO, FROM_PROTOCOL_INFO,
FROM_PROTOCOL_INFO, &proto_info[i], 0, WSA_FLAG_OVERLAPPED);
         printf("Handle socket creat: %d\n", hSock);

         closesocket(hSock);

         // Avem nevoie decat de un protocol, iesim din for

         break;
      }
   }

   // Eliberam resursele

   delete []proto_info;
   WSACleanup();

   return 0;
}
```

Pentru Visual Basic 6 vom avea nevoie de mai multe declaratii. Pot aparea multe probleme, mai ales cand folositi un API Viewer, cred ca mai bine scrieti manual declaratia pentru fiecare API folosind prototipul C++ de pe MSDN. In fine, cu modificarile de riguare la **WSASocket** (declaratie care nu era deloc bine facuta in API Viewer 2004), codul complet este:

```
Option Explicit

' Functiile de care avem nevoie

Private Declare Function WSAStartup Lib "ws2_32.dll" (ByVal wVersionRequired As Integer, ByRef
```

```
lpWSAData As wsaData) As Long
Private Declare Function WSACleanup Lib "ws2_32.dll" () As Long
Private Declare Function WSAEnumProtocols Lib "ws2_32.dll" Alias "WSAEnumProtocolsA" (ByVal
lpiProtocols As Long, ByRef lpProtocolBuffer As Any, ByRef lpdwBufferLength As Long) As Long
Private Declare Function WSAGetLastError Lib "ws2_32.dll" () As Long
Private Declare Function closesocket Lib "ws2_32.dll" (ByVal s As Long) As Long
Private Declare Function WSASocket Lib "ws2_32.dll" Alias "WSASocketW" (ByVal af As Long, ByVal
lType As Long, ByVal protocol As Long, ByRef lpProtocolInfo As WSAPROTOCOL_INFO, ByVal g As
Long, ByVal dwFlags As Long) As Long

' Constantele necesare

Private Const WSADESCRIPTION_LEN As Long = 256
Private Const WSASYS_STATUS_LEN As Long = 128
Private Const MAX_PROTOCOL_CHAIN As Long = 7
Private Const WSAPROTOCOL_LEN As Long = 255
Private Const VERSIUNE As Long = &H202
Private Const AF_INET As Long = 2
Private Const IPPROTO_TCP As Long = 6
Private Const WSA_FLAG_OVERLAPPED As Long = &H1
Private Const WSAENOBUFS As Long = 10055&
Private Const FROM_PROTOCOL_INFO As Long = -1

Private Type wsaData
  wVersion As Integer
  wHighVersion As Integer
  szDescription As String * WSADESCRIPTION_LEN
  szSystemStatus As String * WSASYS_STATUS_LEN
  iMaxSockets As Integer
  iMaxUdpDg As Integer
  lpVendorInfo As Long
End Type

' Pentru intrarea ProtocolChain, tipul protocolului, de baza sau compus din mai multe "straturi"

Private Type WSAPROTOCOLCHAIN
  ChainLen As Long
  ChainEntries(0 To MAX_PROTOCOL_CHAIN - 1) As Long
End Type

' Structura ce contine informatiile despre fiecare protocol in parte

Private Type WSAPROTOCOL_INFO
  dwServiceFlags1 As Long
  dwServiceFlags2 As Long
  dwServiceFlags3 As Long
```

```
  dwServiceFlags4 As Long
  dwProviderFlags As Long
  ProviderId(0 To 15)  As Byte
  dwCatalogEntryId As Long
  ProtocolChain As WSAPROTOCOLCHAIN
  iVersion As Integer
  iAddressFamily As Long
  iMaxSockAddr As Long
  iMinSockAddr As Long
  iSocketType As Long
  iProtocol As Long
  iProtocolMaxOffset As Long
  iNetworkByteOrder As Long
  iSecurityScheme As Long
  dwMessageSize As Long
  dwProviderReserved As Long
  szProtocol(0 To WSAPROTOCOL_LEN - 1) As Byte
End Type

' Pentru a evita aparitia erorilor provocate de sirurile de lungime fixa

Private Function TrimNull(ByVal sir As String) As String
TrimNull = Left(sir, InStr(1, sir, vbNullChar) - 1)
End Function

Private Sub Form_Load()

Dim wsa As wsaData
Dim proto_info() As WSAPROTOCOL_INFO
Dim ret, marime_buffer, protocoale, i, hSock As Long
Dim strProtocol As String

WSAStartup VERSIUNE, wsa

ReDim proto_info(0) As WSAPROTOCOL_INFO

' Apelam functia si ne asteptam la eroare
' Dupa apelul functiei marime_buffer va stoca marimea necesara bufferului

ret = WSAEnumProtocols(0, proto_info(0), marime_buffer)

' Verificam daca a intervenit eroarea cu buffer prea mic si afisam un mesaj

If WSAGetLastError = WSAENOBUFS Then MsgBox "Buffer prea mic. Marime: " & marime_buffer '
Informativ
```

```
' Calculam nr. de protocoale folosindu-ne de marimea unui protocol si de marimea necesara tuturor

protocoale = (marime_buffer) / LenB(proto_info(0))

' Afisam nr. de protocoale ( Informativ )

MsgBox "Protocoale: " & protocoale

' Redimensionam vectorul la nr. de protocoale - 1 elemente ( vectorii sunt in baza 0 )

ReDim proto_info(0 To protocoale - 1) As WSAPROTOCOL_INFO

' Apelam din nou functia care scrie in proto_info() informatiile necesare

ret = WSAEnumProtocols(0, proto_info(0), marime_buffer)

' Parcurgem vectorul si cautam protocolul care ne intereseaza

For i = 0 To ret - 1

   ' Verificam daca are familia de adrese AF_INET si protocolul este TCP

   If proto_info(i).iAddressFamily = AF_INET And proto_info(i).iProtocol = IPPROTO_TCP Then

      ' Memoram in variabila sir cateva informatii pe care le vom afisa

      strProtocol = vbCrLf & "Protocol: " & TrimNull(StrConv(proto_info(i).szProtocol, vbUnicode)) &
vbCrLf
      strProtocol = strProtocol & "- iAddressFamily = " & proto_info(i).iAddressFamily & vbCrLf
      strProtocol = strProtocol & "- iProtocol = " & proto_info(i).iProtocol & vbCrLf

      hSock = WSASocket(FROM_PROTOCOL_INFO, FROM_PROTOCOL_INFO,
FROM_PROTOCOL_INFO, proto_info(i), 0, WSA_FLAG_OVERLAPPED)

      strProtocol = strProtocol & vbCrLf & "Socket: " & hSock & vbCrLf

      ' Afisam datele obtinute

      MsgBox strProtocol

      Exit For

   End If
Next

' Eliberam resursele
```

```
ReDim proto_info(0)
closesocket hSock
WSACleanup

Unload Me

End Sub
```

Configurarea unui socket

Dupa cum am specificat, se foloseste modelul client-server. Serverul va trebui mai intai "legat" la o adresa IP si un port. Pentru asta vom folosi functia **bind**. Apoi, va asculta pe un anumit port. Vom folosi functia **listen**, iar cand va primi o cerere de conexiune, aceasta se va accepta folosind functia **accept**. Pentru client, dupa creare, doar ne conectam la un server folosind functia **connect** (pentru socket-uri orientate pe conexiune). Apoi putem incepe transferul de date intre client si server.

Functia **bind** are 3 parametri:

```
Private Declare Function bind Lib "ws2_32.dll" (ByVal s As Long, ByRef addr As sockaddr, ByVal
namelen As Long) As Long
int bind (SOCKET s, const struct sockaddr *name, int namelen);
```

Cei trei parametri ai functiei sunt: *s*, handlerul socketului pe care il vom lega la o adresa locala, cel creat folosind functia **socket** sau functia **WSASocket**, *name* va fi adresa pe care o vom asocia socketului, iar *namelen* marimea parametrului name, in bytes. Parametru *name* va trebui sa fie un pointer la o structura *sockaddr_in*. De fapt este un pointer la o structura *sockaddr*, dar *sockaddr* si *sockaddr_in* reprezinta de fapt acceasi structura, ambele sunt memorate pe 16 octeti si contin aceleasi date. Vom folosi *sockaddr_in* pentru ca este mai usor de folosit, este "sectionata" pe campuri si asta ne ajuta mult. Apoi va fi necesar sa convertim *sockaddr_in* la *sockaddr*, vom face asta in C++ folosind operatorul cast. Functia **bind** returneaza 0 daca se executa cu succes si *SOCKET_ERROR* daca intervine o eroare.

Putem spune ca aceasta functie ii atribuie un "nume" socketului nostru. Acest nume este compus din familia de adrese, adresa IP gazda si portul folosit.

Structura contine urmatoarele intrari:
- ***sin_family*** = familia de adrese
- ***sin_port*** = portul local folosit
- ***sin_addr*** = un pointer la o structura *in_addr* (va memora adresa IP locala la care legam socketul)
- ***sin_zero*** = nu il vom folosi

In VB6 declaram astfel structura, folosim numele sockaddr_in. Corespunde structurii din C.

```
Private Type sockaddr_in
  sin_family As Integer
  sin_port As Integer
  sin_addr As in_addr
  sin_zero As String * 8
End Type
```

```
struct sockaddr_in {
      short   sin_family;
      u_short sin_port;
      struct  in_addr sin_addr;
      char    sin_zero[8];
};
```

In VB6 vom defini *in_addr* astfel deoarece ne intereseaza decat un singur camp:

```
Private Type in_addr
  S_addr As Long
End Type
```

Structura este de fapt definita ca:

```
typedef struct in_addr {
 union {
  struct {
   u_char s_b1,s_b2,s_b3,s_b4;
  } S_un_b;
  struct {
   u_short s_w1,s_w2;
  } S_un_w;
  u_long S_addr;
 } S_un;
} IN_ADDR,
 *PIN_ADDR,
 FAR *LPIN_ADDR;
```

Structura *in_addr* se foloseste pentru a memora o adresa IP. Poate memora aceasta adresa IP (care se memoreaza pe 4 octeti, adresa IPv4) ca 4 numere pe cate un octet fiecare, 2 numere a cate 2 octeti fiecare sau un singur numar pe 4 octeti, pe care il vom folosi si noi. Din structura nu ne intereseaza decat *s_addr* (care este de fapt definit ca *S_un.S_addr*, sa nu uitam ca C si C++ fac diferenta intre majuscule si minuscule) care va fi adresa IP in formatul retelei, va fi un long, iar pentru conversie din string vom folosi functia **inet_addr**. Se poate specifica o anumita adresa sau se poate alege automat una folosind constanta *INADDR_ANY*. Intrarea *sin_port* va fi un (*unsigned*) *short* in formatul retelei. Pentru conversie vom folosi functia **htons** (host to network [unsigned] short) care primeste ca parametru numarul si il returneaza in formatul retelei, cel de care avem nevoie. Daca pentru aceasta intrare vom folosi valoarea 0, Windows Vista sau versiunile mai noi vor alege automat un port local intre 49152 si 65535 (se poate afla acest interval folosind comanda "*netsh int ipv4 show dynamicport tcp*"). Daca nu apare nici o eroare, functia va returna 0, in caz de eroare va returna *SOCKET_ERROR*, iar cu **WSAGetLastError** se poate obtine: *WSANOTINITIALISED* , *WSAENETDOWN* , *WSAEACCES* , *WSAEADDRINUSE* , *WSAEADDRNOTAVAIL* , *WSAEFAULT* , *WSAEINPROGRESS* , *WSAEINVAL* , *WSAENOBUFS* , *WSAENOTSOCK*.

```
u_short htons(u_short hostshort);
Private Declare Function htons Lib "ws2_32.dll" (ByVal hostshort As Integer) As Integer
```

Putem obtine adresa IP locala in mai multe feluri. De exemplu, putem folosi si functia **getaddrinfo**, dar o vom face in modul cel mai simplu, vom converti situl de caractere "*127.0.0.1*" care reprezinta adresa IP locala la un long, in formatul retelei, de care avem nevoie in structura *sockaddr_in*, folosind functia

inet_addr care face acest lucru pentru noi. Functia primeste ca parametru sirul de caractere ce reprezinta adresa IP in formatul cu puncte ("*127.0.0.1*") si returneaza acest IP in format Long, in formatul retelei. In cazul in care se specifica o adresa IP care nu este valida, se va returna *INADDR_NONE*.

unsigned long inet_addr(const char *cp);
Private Declare Function inet_addr Lib "ws2_32.dll" (ByVal cp As String) As Long

Exista o diferenta intre formatul retelei si formatul pe care il foloseste sistemul de operare. Sunt doua astfel de formate: *big-endian*, care salvaza cel mai important octet in cea mai "joasa" zona de memorie, si *little-endian* care face asta invers (exista totusi si sisteme ce folosesc *middle endian* - informativ). Cu alte cuvinte, ordinea in care sunt memorati octetii este opusa. Sa dau un mic exemplu. Sa spunem ca vrem sa trimitem cuiva, folosind Winsock numarul 256. Numarul 256, in binar este normal sa fie 00000001 0000000, acesta este formatul *big-endian*, care este si formatul retelei. Insa Windows-ul (procesoarele Intel) foloseste formatul *little endian* si memoreaza acest numar ca 00000000 00000001. Asadar, daca nu convertim numarul 256 in formatul retelei vom trimite de fapt numarul 1. Este o oarecare diferenta.

Asadar, pentru a pregati socketul nostru de actiune, vom putea proceda astfel:

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

int main()
{
   WSADATA wsaData;
   SOCKET hSock;
   sockaddr_in adresa;  // Structura noastra
   int rezultat;

   WSAStartup(MAKEWORD(2, 2), &wsaData);

   // Cream socketul

   hSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

         // Scriem informatiile necesare in structura

         adresa.sin_family = AF_INET;    // Setam familia de adrese in structura
         adresa.sin_addr.s_addr = inet_addr("127.0.0.1");    // Setam adresa locala la care vom lega
socketul in formatul retelei
         adresa.sin_port = htons(2222);    // Setam portul, in formatul retelei

         // Legam socketul la familia de adrese "Internet", adresa IP locala si portul 2222
         // Convertim structura sockaddr_in in sockaddr, asa e declarat parametrul functiei, acest tip
trebuie sa folosim

         rezultat = bind(hSock, (SOCKADDR*) &adresa, sizeof(adresa));

         // Daca a intervenit o eroare, afisam codul de eroare al acesteia
```

```
        if(rezultat) printf("A intervenit eroare cu codul: %d \r\n", WSAGetLastError());

   closesocket(hSock);

   WSACleanup();

   return 0;
}
```

Pentru Visual Basic 6, va trebui sa declaram functiile si structurile necesare. Putem folosi tipul *sockaddr_in* definit de noi ca tip de date pentru al doilea parametru al functiei **bind**. Structura *sockaddr_in* o putem defini ca mai sus, sau putem pune direct *s_addr As Long* in structura, pentru a nu mai folosi structura *in_addr*.

Asadar, codul complet pentru Visual Basic 6:

```
' Functiile pe care le vom apela

Private Declare Function WSAStartup Lib "ws2_32.dll" (ByVal wVersionRequired As Integer, ByRef
lpWSAData As wsaData) As Long
Private Declare Function WSACleanup Lib "ws2_32.dll" () As Long
Private Declare Function socket Lib "ws2_32.dll" (ByVal af As Long, ByVal lType As Long, ByVal protocol
As Long) As Long

' Al doilea parametru l-am setat ca sockaddr_in

Private Declare Function bind Lib "ws2_32.dll" (ByVal s As Long, ByRef addr As sockaddr_in, ByVal
namelen As Long) As Long
Private Declare Function inet_addr Lib "ws2_32.dll" (ByVal cp As String) As Long
Private Declare Function htons Lib "ws2_32.dll" (ByVal hostshort As Integer) As Integer
Private Declare Function WSAGetLastError Lib "ws2_32.dll" () As Long
Private Declare Function closesocket Lib "ws2_32.dll" (ByVal s As Long) As Long

' Constantele necesare

Private Const WSADESCRIPTION_LEN As Long = 256
Private Const WSASYS_STATUS_LEN As Long = 128
Private Const VERSIUNE As Long = &H202
Private Const AF_INET As Long = 2
Private Const SOCK_STREAM As Long = 1
Private Const IPPROTO_TCP As Long = 6

' Structura wsaData

Private Type wsaData
   wVersion As Integer
```

```
  wHighVersion As Integer
  szDescription As String * WSADESCRIPTION_LEN
  szSystemStatus As String * WSASYS_STATUS_LEN
  iMaxSockets As Integer
  iMaxUdpDg As Integer
  lpVendorInfo As Long
End Type

' Structura ce memoreaza adresa IP

Private Type in_addr
  S_addr As Long
End Type

' Structura pe care o vom folosi pentru "legare"

Private Type sockaddr_in
  sin_family As Integer
  sin_port As Integer
  sin_addr As in_addr
  sin_zero As String * 8
End Type

Private Sub Form_Load()

Dim wsa As wsaData
Dim hSock As Long
Dim rezultat As Integer
Dim adresa As sockaddr_in

WSAStartup VERSIUNE, wsa

' Cream socketul

hSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)

' Scriem informatiile necesare in structura

adresa.sin_family = AF_INET    ' Setam familia de adrese in structura
adresa.sin_addr.S_addr = inet_addr("127.0.0.1")    ' Setam adresa locala la care vom lega socketul in
formatul retelei
adresa.sin_port = htons(2222)     ' Setam portul, in formatul retelei

' Legam socketul la familia de adrese "Internet", adresa IP locala si portul 2222

rezultat = bind(hSock, adresa, Len(adresa))
```

```
' Daca a intervenit o eroare, afisam codul de eroare al acesteia

If (rezultat) Then MsgBox ("A intervenit eroarea cu codul: " & WSAGetLastError())

closesocket hSock

WSACleanup

End Sub
```

Mesaje de eroare (cele care pot sa apara folosind functiile de mai sus):
- ***WSANOTINITIALISED*** - Nu a fost initializat Winsock (functia *WSAStartUp* nu a fost apelata sau nu s-a terminat cu succes)
- ***WSAENETDOWN*** - Este o problema la retea (conexiune)
- ***WSAEACCES*** - Nu este acordata permisiunea. La *bind* apare cand o alta aplicatie foloseste acea adresa cu acces exclusiv
- ***WSAEADDRINUSE*** - Adresa e deja folosita. Se refera de fapt la protocol + adresa + port. Apare de exemplu cand o alta aplicatie foloseste acelasi port pe care functia *bind* incearca sa il foloseasca
- ***WSAEADDRNOTAVAIL*** - Adresa nu este valida. Apare la *bind* cand se specifica o adresa aiurea
- ***WSAEFAULT*** - Apare cand se foloseste un pointer invalid sau un buffer prea mic
- ***WSAEINPROGRESS*** - O operatie cu blocare este in curs de desfasurare
- ***WSAEINVAL*** - Apare cand se specifica un argument invalid
- ***WSAENOBUFS*** - Bufferul nu este destul de mare
- ***WSAENOTSOCK*** - Se incearca o operatie asupra ceva care nu este socket, probabil un socket invalid
- ***WSASYSNOTREADY*** - Apare la initializare cand sunt probleme: nu exista anumite componente sau se folosesc mai multe implementari odata
- ***WSAVERNOTSUPPORTED*** - Versiunea nu este suportata
- ***WSAEPROCLIM*** - Winsock are o limita referitoare la cate procese folosesc. Eroarea apare cand sunt prea multe astfel de procese

Referinte:

- *MSDN - http://msdn.microsoft.com/en-gb/default.aspx*
- *TCP/IP - Tim Parker, Mark Sportack*

Aceasta este prima parte a articolului. Continuarea, urmatoarea parte, va aparea in urmatorul numar al revistei.