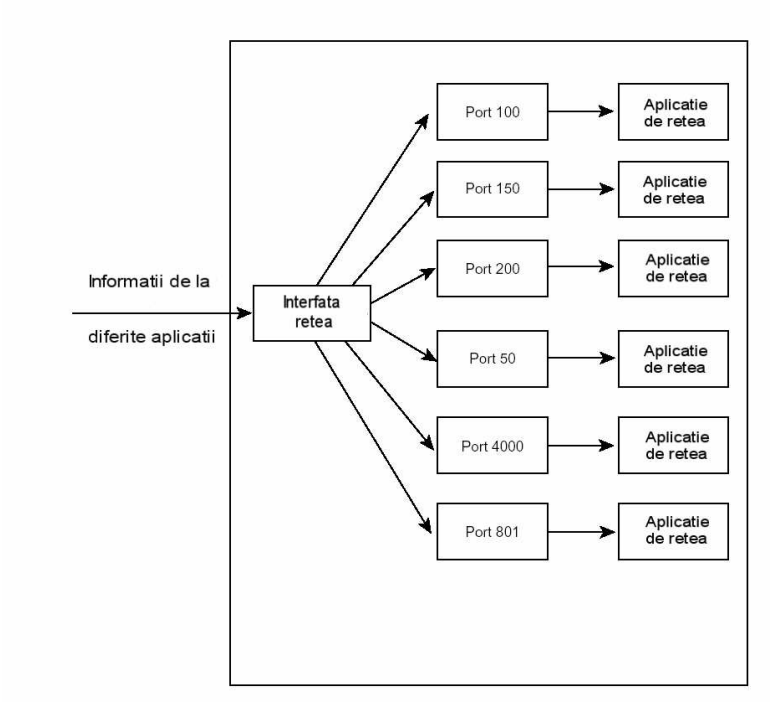
# Aplicatie folosind accesarea intreruperilor si a spatiului de memorie al chip-ului folosind interfata stndardizata WinSock

Datorita exploziei internetului in lume, tot mai multe aplicatii doresc sa comunice in retea. De aceea, si producatorii de sisteme de operare s-au gandit sa includa suport pentru aplicatiile de retea.

Majoritatea aplicatiilor care ruleaza astazi sunt de tip client/server. Aceasta implica o aplicatie care „asculta" reteaua si o alta care incearca sa se conecteze. A doua aplicatie trebuie sa stie adresa primeia pentru a se putea conecta.




Odata realizata legatura, datele pot fi schimbate intre cele doua aplicatii.

In perioada lui Windows 3.11, cand suportul pentru retea nu era inclus in OS, se puteau cumpara de la diversi distribuitori protocoale de retea, si de aici apareau probleme de comunicatie si portabilitate. Iata de ce Microsoft a pus bazele WinSock, ca un API standard pentru toate comunicatiile de retea, indiferent de soft-ul folosit.

Desi aplicatiile variaza de la simple la foarte complexe, toate au functii construite in jurul interfetei WinSock.

Pentru conectarea folosind Winsock este nevoie sa se cunoasca adresa ip a calculatorului la care vrem sa ne conectam precum si portul pe care acesta l-a deschis comunicarii.

Modul in care un socket opereaza este similar cu modul de lucru pentru fisiere: socketul reprezinta un obiect folosit pentru citirea si scrierea mesajelor care sunt trasferate intre aplicatii.

O conexiune folosind socketuri necesita insa informatii diferite fata de lucrul cu fisiere. Pentru a deschide un fisier este nevoie de numele acestuia si de locul in care se afla; pentru deschiderea unui socket, trebuie cunoscut calculatorul pe care ruleaza aplicatia cu care dorim sa comunicam si portul care urmeaza sa fie folosit.

O analogie a acestui proces este cea a liniilor telefonice. Astfel, daca se doreste contactarea unei persoane intr-o cladire cu multe birouri, se formeaza mai intai numarul de telefon, apoi se selecteaza interiorul dorit. Similar porturile sunt folosite pentru a directiona diferitele conexiuni pe care le poate avea un calculator la retea. Daca se specifica un calculator diferit sau un port diferit se pot realiza conexiuni eronate, asa cum si un telefon poate fi format gresit. Daca aplicatia nu functioneaza pe calculatorul destinatie este posibil sa nu se primeasca nici un raspuns la incercarea de conectare.

Doar o singura aplicatie poate asculta un port pe un anumit calculator. Daca exista mai multe aplicatii, fiecare trebuie sa foloseasca portul propriu.

**Crearea unui socket**

Atunci cand se construiesc aplicatii in Visual C++ care folosesc serviciile de retea, se foloseste clasa de baza CAsyncSocket care ofera comunicatie folosind socketuri.

Pentru a crea un socket pentru a fi folosit intr-o aplicatie, trebuie declarata variabila CAsyncSocket ca membru al uneia din clasele aplicatiei:

```
class CMyDlg : public CDialog
        {...
```

```
private:
CAsyncSocket m_sMySocket;
};
```

Inainte insa de a folosi obiectul socket, trebuie apelata metoda Create. Aceasta creaza un socket si il pregateste pentru a fi folosit. Modul de apelare a metodei Create depinde de cum dorim sa folosim socketul. Daca acesta va fi folosit pentru a ne conecta la o alta aplicatie (de exemplu va fi folosit pentru aplicatia client) atunci nu este nevoie sa trimitem parametri pentru metoda Create:

```
if (m_sMySocket.Create())
        { // Continue on
        }
else
        // Perform error handling here
```

Daca un socket va fi folosit pentru a asculta o alta aplicatie (de exemplu va fi folosit pentru aplicatia server), atunci trebuie sa ii trimitem cel putin numarul portului pe care socketul trebuie sa il asculte:

```
if (m_sMySocket.Create(4000))
        {// Continue on
        }
else
        // Perform error handling here
```

Se pot trimite si alti parametri in apelarea metodei Create, de exemplu tipul socketului care se doreste a fi creat, evenimentele la care socketul trebuie sa raspunda, sau adresa la care trebuie sa asculte socketul in cazul in care calculatorul are mai mult de o placa de retea.

**Crearea unei conexiuni**

Odata creat socketul, se poate deschide o conexiune cu el. Trei etape insotesc deschiderea unei conexiuni: doi dintre acestea au loc pe server (aplicatia care asculta conexiunea) iar a treia are loc la client (aplicatia care realizeaza cererea de conexiune).

Pentru client, deschiderea unei conexiuni consta in apelarea metodei Connect. Clinetul trebuie sa asigure doi parametri pentru metoda Connect: numele calculatorului sau adresa de retea, si portul aplicatiei la care se conecteaza. Iata doua metode de folosire a metodei Connect:

```
if (m_sMySocket.Connect("thatcomputer.com", 4000))
{// Continue on
}
else
// Perform error handling here
```

A doua forma este:

```
if (m_sMySocket.Connect("178.1.25.82", 4000))
{// Continue on
}
else
// Perform error handling here
```

Odata stabilita conexiunea se declanseaza un eveniment care semnalizeaza conectarea sau problemele care au aparut in cazul in care conectarea nu s-a putut face.

Pentru server, adica sistemul care asculta, aplicatia trebuie sa comunice socketului sa asculte la conexiunile care sosesc, prin apelarea metodei Listen. Parametrul acestei metode specifica numarul de conexiuni care pot astepta intr-o stiva, in asteptarea realizarii conexiunii. Implicit valoarea este de 5. Apelarea se face astfel:

```
if (m_sMySocket.Listen())
{// Continue on}
else
// Perform error handling here
```

Atunci cand o alta aplicatie incearca sa se conecteze, este generat un eveniment care semnalizeaza aplicatiei prezenta unei cereri de conexiune. Acceptarea conexiunii se face prin folosirea

metodei Accept, care foloseste o a doua variabila CAsyncSocket, care este conectata la cealalta aplicatie.
In cazul receptiei unei cereri de conexiune socketul care asculta creaza un alt socket,care este conectat la cealalta aplicatie, care insa nu trebuie sa aiba metoda Create deoarece metoda Accept este cea care creaza socketul. Apelarea metodei Accept se face astfel:

```
if (m_sMySocket.Accept(m_sMySecondSocket))
{
// Continue on
}
else
// Perform error handling here
```

Acum aplicatia care doreste conectarea este legata la al doile socket al aplicatiei server.

**Primirea/Trimiterea mesajelor**

Functiile de primire/trimitere a mesajelor printr-un socket folosesc un buffer generic si, deoarece nu conteaza tipul datelor care sunt transmise, se poate transmite chiar text.
Pentru trimiterea datelor printr-o conexiune se foloseste metoda Send. Aceasta are doi parametri: primul este un pointer la bufferul care contine data care trebuie transmisa(daca mesjul este o variabila CString se foloseste operatorul LPCTSTR pentru a transmite variabila CString catre buffer); al doilea este lungimea bufferului. In cazul aparitei unei erori Send returneaza SOCKET_ERROR. Folosirea metodei Send se face astfel:

```
CString strMyMessage;
int iLen;
int iAmtSent;
...
iLen = strMyMessage.GetLength();
iAmtSent = m_sMySocket.Send(LPCTSTR(strMyMessage), iLen);
if (iAmtSent == SOCKET_ERROR)
{// Do some error handling here
 }
else
{// Everything's fine
 }
```

Cand aplicatia de la celalalt capat al coneiunii primeste datele, este generat un eveniment, si se foloseste metoda Receive. Aceasta are aceeasi parametri ca metoda Send, cu o deferenta: primul parametru este un pointer catre bufferul in care va fi copiat mesajul primit; al doilea parametru contine dimensiunea bufferului pentru a stoca datele primite. In cazul aparitei unei erori Receive returneaza SOCKET_ERROR. Daca mesajul primit este text, datele pot vi direct copiate intr-o variabila CString. Folosirea metodei Receive se face astfel:

```
char *pBuf = new char[1025];
int iBufSize = 1024;
int iRcvd;
CString strRecvd;
iRcvd = m_sMySocket.Receive(pBuf, iBufSize);
if (iRcvd == SOCKET_ERROR)
{// Do some error handling here
}
else
{pBuf[iRcvd] = NULL;
strRecvd = pBuf;
// Continue processing the message
```

}

Este important sa se plaseze in buffer NULL dupa ultimul character primit pentru a preveni preluarea unor date existente anterior.

**Inchiderea conexiunii**

Aceasta se face prin apelarea metodei Close, astfel:

**m_sMySocket.Close();**

Aceasta este printer putinele metode care nu returneaza nici un cod de stare.

**Evenimentele pentru un Socket**

Clasa CAsyncSocket are functii associate fiecarui eveniment care poate sa apara intr-o comunicatie: primire mesaje, transmitere mesaje, incheiere conexiune, etc. Aceste functii au un singur parametru de tip integer, care trebuie verificat pentru a afla ce eroare a avut loc. Functiile sunt:
OnAccept, OnClose, OnConnect, OnReceive, OnSend.

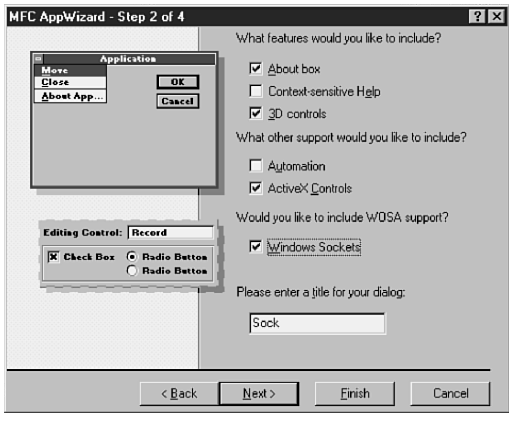
**Detectia erorilor**

Atunci cand functiile membre returneaza o eroare, fie ca este FALSE sau SOCEK_ERROR, se poate apela GetLastError pentru a intercepta codul de eroare. Acest lucru se face in felul urmator:

```
int iErrCode;
iErrCode = m_sMySocket.GetLastError();
switch (iErrCode)
{
case WASNOTINITIALISED:
...
}
```

Codurile pentru Winsock sunt standard si poate fi regasita semnificatia acestora.

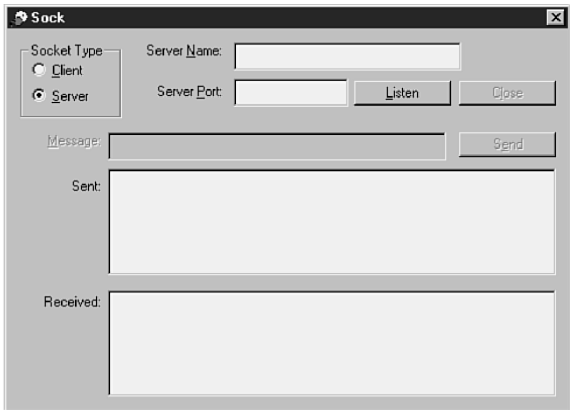
**Construirea aplicatiei**




Aplicatia a fost dezvoltata in Visual C++ 6 si foloseste clasele MFC Winsock pentru a oferi foarte usor acces la retea. Clasa de baza CAsyncSocket asigura comunicatii complet controlate de evenimentele inregistrate la socket-uri.

Pentru inceput se creaza un nou proiect MFC AppWizard, si i se da un nume corespunzator: Sock. Se va folosi o aplicatie bazata pe dialoguri, si care include suport pentru Windows Sockets.

Dialogul principal va contine un set de butoane prin care se precizeaza daca aplicatia este de tip client sau server, un alt set de cutii de text editabile in care sa se poata introduce numele calculatorului si numarul portului de pe server la care la care ne conectam, un buton care sa selecteze inceperea conexiunii(ascultarea mediului de transmisie pentru severe respectiv inceperea incercarii de conectare pentr client), un buton pentru deconectare, o cutie de text editabila pentru scrierea mesajelor insotita de un buton pentru trimiterea mesajului, precum si doua cutii pentru a pastra mesajele trimise respectiv primite:




| Object | Property | Setting |
|---|---|---|
| Group Box | ID | IDC_STATICTYPE |
| | Caption | Socket Type |
| Radio Button | ID | IDC_RCLIENT |
| | Caption | &Client |
| | Group | Checked |
| Radio Button | ID | IDC_RSERVER |
| | Caption | &Server |
| Static Text | ID | IDC_STATICNAME |
| | Caption | Server &Name: |
| Edit Box | ID | IDC_ESERVNAME |
| Static Text | ID | IDC_STATICPORT |
| | Caption | Server &Port: |
| Edit Box | ID | IDC_ESERVPORT |
| Command Button | ID | IDC_BCONNECT |
| | Caption | C&onnect |
| Command Button | ID | IDC_BCLOSE |
| | Caption | C&lose |
| | Disabled | Checked |
| Static Text | ID | IDC_STATICMSG |
| | Caption | &Message: |
| | Disabled | Checked |
| Edit Box | ID | IDC_EMSG |
| | Disabled | Checked |
| Command Button | ID | IDC_BSEND |
| | Caption | S&end |
| | Disabled | Checked |
| Static Text | ID | IDC_STATIC |
| | Caption | Sent: |
| List Box | ID | IDC_LSENT |
| | Tab Stop | Unchecked |
| | Sort | Unchecked |
| | Selection | None |
| Static Text | ID | IDC_STATIC |
| | Caption | Received: |
| List Box | ID | IDC_LRECVD |
| | Tab Stop | Unchecked |
| | Sort | Unchecked |
| | Selection | None |

Odata construit dialogul, se foloseste ClassWizard pentru atasarea variabilelor la controalele dialogului ca mai jos:

| Object | Name | Category | Type |
|---|---|---|---|
| IDC_BCONNECT | m_ctlConnect | Control | CButton |
| IDC_EMSG | m_strMessage | Value | CString |
| IDC_ESERVNAME | m_strName | Value | CString |
| IDC_ESERVPORT | m_iPort | Value | int |
| IDC_LRECVD | m_ctlRecvd | Control | CListBox |
| IDC_LSENT | m_ctlSent | Control | CListBox |
| IDC_RCLIENT | m_iType | Value | int |

Pentru a putea refolosi butonul de conectare pentru a plasa aplicatia server in modul ascultare, trebuie adaugata o functie pentru ambele butoane radio, care va schimba textul afisat pe butonul de comanda in functie de modul care este selectat. Pentru a adauga aceasta functie aplicatiei, trebuie adaugata o functie pentru evenimentul BN_CLICKED pentru controlul ID-ul IDC_RCLIENT, functie cu numele OnRType. Trebuie sa se adauge aceeasi functie si evenimentului BN_CLICKED pentru control ID-ul IDC_RSERVER. Modificarile se fac astfel:
Functia CSockDlg OnRType:

```
void CSockDlg::OnRType()
{
// TODO: Add your control notification handler code here
// Sync the controls with the variables
UpdateData(TRUE);
// Which mode are we in?
if (m_iType == 0) // Set the appropriate text on the button
m_ctlConnect.SetWindowText("C&onnect");
else
m_ctlConnect.SetWindowText("&Listen");
}
```

Dupa compilarea si rularea aplicatie, textul ar trebui sa se modifice la apasarea butonului radio de selectie a tipului aplicatiei: server respectiv client.

Pentru a putea captura si raspunde la aparitia evenimentelor de la socket, trebuie creata o noua clasa cu rol de preluare a evenimentelor si trimitere a lor catre clasele care asigura dialogul. Trebuie avuta in vedere si tratarea erorilor care pot sa apara.

Pentru crearea acestei clase, se selecteaza: Insert → New Class. Tipul acesteia trebuie lasat cu valoarea implicita MFC Class. I se va da numele CMySocket, si va face parte din CasyncSocket (clasa de baza).

Odata creata clasa, trebuie sa ii adaugam o variabila membru pentru a servi drept pointer la fereastra parinte de dialog. Tipul variabilei va fi CDialog*, numele variabilei m_pWnd, iar accesul la ea drept privat. Trebuie adaugata si o metoda clasei pentru stabilirea pointerului; tipul este void, accesul public si este declarata ca SetParent(CDialog* pWnd):

```
void CMySocket::SetParent(CDialog *pWnd)
{
// Set the member pointer
m_pWnd = pWnd;
}
```

Singurul lucru care trebuie facut este adaugarea functiilor de tratare a evenimentelor. Pentru evenimentul OnAccept trebuie adaugata o functie dupa cum urmeaza:

```
void CMySocket::OnAccept(int nErrorCode)
{
// Were there any errors?
if (nErrorCode == 0)
// No, call the dialog's OnAccept function
((CSockDlg*)m_pWnd)->OnAccept();
}
```

Functii similare trebuiesc adaugate pentru OnConnect, OnClose, OnReceive si OnSend. Dupa ce aceste functii au fost adaugate, trebuie inclus fisierul header pentru dialogul aplicatiei astfel:

```
// MySocket.cpp: implementation file
//
#include "stdafx.h"
#include "Sock.h"
#include "MySocket.h"
#include "SockDlg.h"
```

Dupa ce au fost adaugate toate functiile la clasa socket, trebuie adaugate doua variabile la clasa de dialog: una care sa asculte la cererile de conectare si alta care sa se conecteze la cealalta aplicatie. Deoarece sunt necesare doua obiecte, se adauga doua variabile memble la clasa (CSockDlg). Ambele variabile au tipul CMySocket, iar accesul este privat. Una dintre ele se va numi m_sListenSocket – folosita pentru cererile de conectare, iar a doua m_sConnectSocket – folosita pentru trimiterea mesajelor.

Odata adaugate variabilele, trebuie adaugat codul de initializare a acestora: implicit tipul aplicatiei este client, iar numele serverului este loopback, portul 4000. impreuna cu aceste variabile trebuie stabiliti parametrii celor doua obiecte socket, astfel incat sa indice catre clasa de dialog (prin adaugarea codului de mai jos in functia OnInitDialog din CSockDlg):

```
BOOL CSockDlg::OnInitDialog()
{
CDialog::OnInitDialog();
// Add "About..." menu item to system menu.
…….
SetIcon(m_hIcon, FALSE); // Set small icon
// TODO: Add extra initialization here
// Initialize the control variables
m_iType = 0;
m_strName = "loopback";
m_iPort = 4000;
// Update the controls
UpdateData(FALSE);
// Set the socket dialog pointers
m_sConnectSocket.SetParent(this);
m_sListenSocket.SetParent(this);
return TRUE; // return TRUE unless you set the focus to a control
}
```

**Conectarea la aplicatie**

Atunci cand utilizatorii se conecteaza la aplicatie folosind butonul „connect" prima etapa care trebuie parcursa este dezactivarea tuturor controalelor dialogului (pentru a nu lasa utilizatorul sa schimbe parametrii conexiunii). In acest sens, se apeleaza functia Create pentru variabila socket corespunzatoare in functie de rolul aplicatiei (clinet/server). In final trebuie apelata functia Connect sau Listen pentru initializarea conexiunii pentru fiecare din cele doua aplicatii. Pentru a adauga aceasta functionalitate aplicatiei, se deschide Class Wizard si se adauga evenimentului BN_CLICKED pentru butonul Connect functia:

```
void CSockDlg::OnBconnect()
{
// TODO: Add your control notification handler code here
// Sync the variables with the controls
UpdateData(TRUE);
// Disable the connection and type controls
GetDlgItem(IDC_BCONNECT)->EnableWindow(FALSE);
GetDlgItem(IDC_ESERVNAME)->EnableWindow(FALSE);
GetDlgItem(IDC_ESERVPORT)->EnableWindow(FALSE);
GetDlgItem(IDC_STATICNAME)->EnableWindow(FALSE);
GetDlgItem(IDC_STATICPORT)->EnableWindow(FALSE);
GetDlgItem(IDC_RCLIENT)->EnableWindow(FALSE);
GetDlgItem(IDC_RSERVER)->EnableWindow(FALSE);
GetDlgItem(IDC_STATICTYPE)->EnableWindow(FALSE);
// Are we running as client or server?
```

```
if (m_iType == 0)
{
// Client, create a default socket
m_sConnectSocket.Create();
// Open the connection to the server
m_sConnectSocket.Connect(m_strName, m_iPort);
}
else
{
// Server, create a socket bound to the port specified
m_sListenSocket.Create(m_iPort);
// Listen for connection requests
m_sListenSocket.Listen();
}
}
```

Pentru completarea conexiunii trebuie adaugate functiile la clasa de dialog pentru evenimentele OnAccept si OnConnect. Aceste functii aplelate de socket nu au nevoie de parametri, si nu returneaza nimic. Pentru functia OnAccept (chemata de aplicatia care asculta), se apeleaza functia obiectului socket Accept, trimitandu-i variabila Odata realizata conexiunea, se poate activa cutia de dialog pentru trimiterea de mesaje catre cealalta aplictie.

Pentru adaugarea acestei functii la aplicatie, trebuie adaugata o functie membra la clasa de dialog CSockDlg. Tipul este void, iar accesul este public.

```
void CSockDlg::OnAccept()
{
// Accept the connection request
m_sListenSocket.Accept(m_sConnectSocket);
// Enable the text and message controls
GetDlgItem(IDC_EMSG)->EnableWindow(TRUE);
GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
}
```

Pentru aplicatia clint, dupa realizarea conexiunii, singurele lucruri care trebuie realizate sunt activarea controalelor de introducere si trimitere a mesajelor. Trebuie activat si butonul Close pentru a permite inchiderea conexiunii de la client (nu si de la server). Pentru a adauga aceasta functie aplicatiei, trebuie adaugata o noua functie membru la clasa de dialog CSockDlg. Tipul este void, iar accesul este public. Codul este urmatorul:

```
void CSockDlg::OnConnect()
{
// Enable the text and message controls
GetDlgItem(IDC_EMSG)->EnableWindow(TRUE);
GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
GetDlgItem(IDC_BCLOSE)->EnableWindow(TRUE);
}
```

In acest moment mai trebuie adaugate la clasa de dialog cateva functii, necesare clasei socket. Acestea sunt: OnSend, OnReceive si OnClose. Tipul acestora este void si accesul public.

Daca aplicatia se compileaza corect, se pot porni pe acelasi calculator doua instante ale aceleiasi aplicatii, din care una este plasata in modul ascultare, iar alta in modul client. Se porneste aplicatia server, se plaseaza in modul ascultare, apoi se porneste aplicatia client si se incearca realizarea conexiunii.

**Receptia si transmisia datelor**

Deoarece aplicatiile sunt capabile sa realizeze conexiunea, trebuie doar sa se adauge functionalitate butoanelor de transmisie si receptie. In timpul conexiunii, utilizatorul poate trimite mesaje, care vor aparea in fereastra de mesaje trimise a apasarea butonului Send, si poate primi mesaje care apar in fereastra de mesaje primite automat.

La apasarea butonului Send trebuie verificat daca exista un mesaj de transmis, care este lungimea acestuia, si se realizeaza transmiterea acestuia precum si adaugarea sa la lista de mesaje. Pentru adaugarea acestei functii la aplicatie, foosind Class Wizard se adauga o functie la evenimentul apasatii butonului Send: IDC_BSEND.

```
void CSockDlg::OnBsend()
{
// TODO: Add your control notification handler code here
int iLen;
int iSent;
// Sync the controls with the variables
UpdateData(TRUE);
// Is there a message to be sent?
if (m_strMessage != “”)
{// Get the length of the message
  iLen = m_strMessage.GetLength();
  // Send the message
  iSent = m_sConnectSocket.Send(LPCTSTR(m_strMessage), iLen);
  // Were we able to send it?
  if (iSent == SOCKET_ERROR)
  {
  }
else
{// Add the message to the list box.
  m_ctlSent.AddString(m_strMessage);
  // Sync the variables with the controls
  UpdateData(FALSE);
  }
 }
}
```

Atunci cand se primeste un mesaj se activeaza evenimentul OnReceive, iar mesajul trebuie preluat folosind functia Receive. Odata preluat mesajul, trebuie convertit in CString si adaugat la lista de mesaje primite. Aceasta functionalitate se adauga prin modificarea functiei OnReceive.

```
void CSockDlg::OnReceive()
{
char *pBuf = new char[1025];
int iBufSize = 1024;
int iRcvd;
CString strRecvd;
// Receive the message
iRcvd = m_sConnectSocket.Receive(pBuf, iBufSize);
// Did we receive anything?
if (iRcvd == SOCKET_ERROR)
{
}
else
{
```

```
// Truncate the end of the message
pBuf[iRcvd] = NULL;
// Copy the message to a CString
strRecvd = pBuf;
// Add the message to the received list box
m_ctlRecvd.AddString(strRecvd);
// Sync the variables with the controls
UpdateData(FALSE);
}
}
```

Acum aplicatiile se pot conecta si trimite mesaje intre ele.

**Terminarea conexiunii**

La apasarea butonului Close, aplicatia server primeste evenimentul OnClose si termina conexiunea. Aplicatia server ramane in modul de ascultare pentru a stabili noi conexiuni cu alte aplicatii. In acest sens se va edita functia OnClose, adaugandu-se codul urmator:

```
void CSockDlg::OnClose()
{
// Close the connected socket
m_sConnectSocket.Close();
// Disable the message sending controls
GetDlgItem(IDC_EMSG)->EnableWindow(FALSE);
GetDlgItem(IDC_BSEND)->EnableWindow(FALSE);
GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
GetDlgItem(IDC_BCLOSE)->EnableWindow(FALSE);
// Are we running in Client mode?
if (m_iType == 0)
{
// Yes, so enable the connection configuration controls
GetDlgItem(IDC_BCONNECT)->EnableWindow(TRUE);
GetDlgItem(IDC_ESERVNAME)->EnableWindow(TRUE);
GetDlgItem(IDC_ESERVPORT)->EnableWindow(TRUE);
GetDlgItem(IDC_STATICNAME)->EnableWindow(TRUE);
GetDlgItem(IDC_STATICPORT)->EnableWindow(TRUE);
GetDlgItem(IDC_RCLIENT)->EnableWindow(TRUE);
GetDlgItem(IDC_RSERVER)->EnableWindow(TRUE);
GetDlgItem(IDC_STATICTYPE)->EnableWindow(TRUE);
}
}
```

Pentru butonul Close, trebuie apelata functia OnClose. Pentru a ii adauga functionalitate butonului, trebuie folosit Class Wizard pentru adaugarea functiei OnClose:

```
void CSockDlg::OnBclose()
{
// TODO: Add your control notification handler code here
// Call the OnClose function
OnClose();
}
```

Acum aplicatiile pot sa se conecteze, sa trimit informatii si sa se deconecteze. Aplicatia este acum terminata.

**Mod de lucru al aplicatiei:**

1. Se selecteaza modul server pe un calculator
2. se selecteaza modul client pe al doilea calculator
3. se pune serverul in modul „asteapta client” si se conecteaza clientul la ip-ul si portul la care se afla serverul
4. cand legatura e stabilita, se pot trimite mesaje.

   OBS: O adresa folositoare este adresa de test pentru TCP/IP: adresa de loopback (127.0.0.1) care este de fapt adresa propriului calculator. Aceasta permite rularea unei conexiuni client/server pe un singur calculator!
5. pentru terminare, doar clientul are posibilitatea deconectarii (serverul nu il poate deconecta decat daca e inchis programul)