6. Server DNS folosind UDP – Implementare MFC

Un server UDP trebuie să asigure primirea datagramelor, prelucrarea acestora şi transmisia răspunsurilor. Faţă de serverul TCP, un server UDP nu trebuie să-şi facă probleme pentru realizarea conexiunilor şi administrarea lor, fiecare datagramă fiind tratată independent de celelalte.

În cadrul acestui capitol vom prezenta implementarea unui server DNS pe baza protocolului UDP ce implementează protocolul de nivel aplicație prezentat în capitolul anterior. Un asemenea server trebuie să fie echipat cu o bază de date ce conține o serie de înregistrări prin care se asociază fiecărui nume de domeniu o adresă IP în mod unic (cu toate că există cazuri când un nume de domeniu are ataşat mai multe IP-uri). Întrucât nu este vorba despre o disciplină pentru familiarizarea bazelor de date, vom considera înregistrările stocate într-un fişier. O asemenea abordare ne va permite să punem accent asupra considerentelor de comunicare între client şi server precum şi asupra organizării datelor în memoria serverului.

6.1 Cerințele aplicației server UDP-MFC

În continuare, vom trece în revistă cerințele unei aplicații server UDP ce trebuie să primească cereri şi să transmită înapoi răspunsuri specifice unui server DNS. Întrucât am apelat la utilizarea MFC, vom considera o interfață grafică ce asigură configurarea şi pornirea serverului. Controalele grafice trebuie să asigure introducerea căii şi a denumirii fişierului în care sunt stocate înregistrările DNS. Totodată, aceste controale trebuie să asigure alegerea portului pe care rulează serverul şi pornirea/oprirea acestuia.

Pentru implementarea interfeței de comunicare vom utiliza şi în cadrul acestui capitol clasa MFC `CAsyncSocket`. În concluzie, cerințele principale ce trebuie satisfăcute de aplicația server UDP sunt următoarele:

- Interfață grafică pentru configurarea, pornirea şi oprirea serverului;
- Primirea şi transmisia datagramelor UDP;
- Căutarea adreselor IP pe baza denumirilor primite într-un fişier ataşat.

6.2 Arhitectura aplicației

Pentru a satisface cerințele enumerate, vom utiliza o arhitectură server bazată pe o fereastră dialog, denumirea clasei asociate ferestrei fiind `CMyDNSServerDlg`. Utilizând Microsoft Visual Studio 2005, vom crea un nou proiect *MFC*, *MFC Application*, *Dialog Based*, cu opțiunea *Windows sockets* selectat (*Advanced Features*). Întrucât nu vom folosi caractere *Unicode*, se va deselecta *Use Unicode libraries* (*Application Type*). În continuare, vom considera că denumirea proiectului este *MyDNSServer*.

Arhitectura generală a aplicației ilustrată printr-o diagramă de clase este ilustrată în figura 6.1. Funcționalitatea soclului UDP este încapsulată în clasa `CAsyncSocket`, moştenită de clasa `CMyDNSServerSocket` ce va asigura apelul metodelor de transmisie şi

recepționare. Clasa `CMyDNSServerSocket` este instanțiată din clasa `CMyDNSClientDlg` atașată ferestrei dialog.

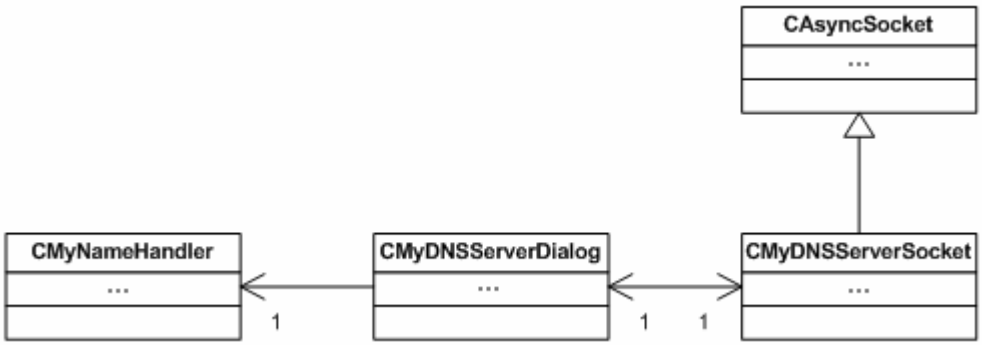

Figura 6.1 O parte din diagrama de clase a aplicației Server UDP-MFC

Pe lângă aceste clase, aplicația mai include şi o clasă de administrare a perechilor domeniu-IP, `CMyNameHandler`. Această clasă va asigura atât încărcarea perechilor din fişier dar şi căutarea acestora.

6.3 Construirea interfeței grafice

Interfața grafică utilizată, precum şi variabilele membru utilizate sunt ilustrate în figura 6.2. În controlul de editare *File* se introduce calea şi denumirea fişierului în care sunt stocate înregistrările, iar în controlul de editare *Port* se introduce portul utilizat de server.

Variabila membru ataşată controlului de editare *File* este `m_sFile`, de tipul `CString`, iar variabila membru ataşată controlului de editare *Port* este `m_nPort`, de tipul `UINT`. În urma apăsării buronului *CreateServer* se încarcă în memorie toate înregistrările din fişier şi se crează soclul server, cu schimbarea etichetei afişate pe buton în *Close*. La o nouă apăsare a aceluiaşi buton se distrug obiectele create la pasul anterior, incluzând închiderea soclului. Totodată, la a doua apăsare se schimbă înapoi eticheta butonului în *CreateSocket*.

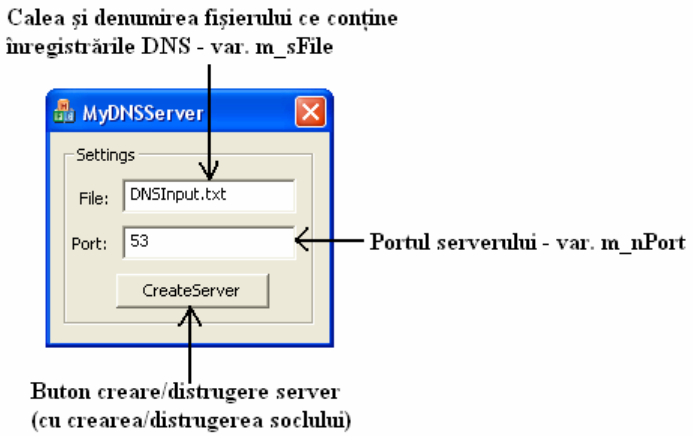

Figura 6.2 Interfața grafică şi variabilele membru ataşate aplicației Server UDP-MFC

6.4 Construirea clasei de administrare a denumirilor

Administrarea perechilor denumire-IP se realizează prin intermediul clasei `CMyNameHandler`. Prin această clasă se asigură încărcarea perechilor de denumiri dintr-un fişier în memorie şi căutarea acestora.

Fiecare *înregistrare* din fişier este alcătuită din două câmpuri: câmpul domeniu şi câmpul IP. Înregistrările sunr separate prin linie nouă, iar câmpurile prin una sau mai multe spații. Un exemplu de un asemenea fişier este următorul:

```
www.google.com      74.125.87.147
www.yahoo.com       87.248.113.14
www.upm.ro          193.226.19.204
www.netsoft.ro      193.226.116.155
www.ibs.ro          193.231.162.100
www.sysgenic.com    213.157.167.250
www.muresonline.ro  93.113.174.60
www.muresinfo.ro    89.47.237.70
```

Încărcarea acestor câmpuri în memorie se realizează prin operații uzuale de citire a fişierelor text. Înregistrările sunt stocate într-o structură de date de tipul `std::map` (din cadrul STL) ce asigură ordonarea datelor în funcție de anumiți parametrii. Pentru căutarea unui anumit domeniu se va utiliza o metodă publică. Declarația completă rezultată a acestei clase este dat în cele ce urmează:

```
#pragma once
#include <map>
using namespace std;

class CMyNameHandler
{
public:
      CMyNameHandler( void );
      virtual ~CMyNameHandler( void );
      bool loadNames( const CString& rFile );
      bool getIP( const CString& rDomain, CString& rIP );

private:
      map<CString,CString>    m_mapNames;
};
```

În cadrul acestei clase au fost incluse două metode:

```
      bool loadNames( const CString& rFile );
      bool getIP( const CString& rDomain, CString& rIP );
```

Prima metodă, `loadNames()` asigură încărcarea denumirilor din fişierul a cărui denumire se transmite ca parametru, iar a doua metodă, `getIP()` asigură căutarea unui IP pe baza unei denumiri date. Ambele metode sunt publice, fiind apelate din `CMyDNSClientDlg` de unde se instanțiază de altfel clasa `CMyNameHandler`.

Utilizarea clasei şablon `std::map` în locul unei liste sau vectori are avantajul accesului mult mai rapid la elementele stocate, timpul de căutare fiind unul logaritmic. Fiecare element este format dintr-o pereche <*cheie*, *valoare*>, unde *cheie* este o valoare unică, în cazul de față fiind denumirea domeniului, iar *valoare* reprezintă adresa IP asociată cheii. Ambele elemente din această pereche sunt de tipul `CString`.

Încărcarea structurii `m_mapNames` cu valorile citite din fişierul de intrare se realizează în metoda `loadNames()`. Pentru a asigura re-entranța acestei metode, înainte de încărcare se şterg toate elementele încărcate dintr-o execuție anterioară, această operație fiind urmată de citirea perechilor şi stocarea lor în `m_mapNames`:

```
bool CMyNameHandler::loadNames( const CString& rFile )
{
	FILE* pf = _tfopen( rFile, _T("rt") );
	if ( NULL == pf ) {
		return false;
	}

	m_mapNames.clear();

	_TCHAR pszDomain[ 4096 ];
	_TCHAR pszIP[ 1024 ];
	while( !feof( pf ) )
	{
		int nRet = _ftscanf( pf, _T("%4095s"), pszDomain );
		if ( EOF == nRet ) {
			break;
		}
		nRet = _ftscanf( pf, _T("%1023s"), pszIP );
		if ( EOF == nRet ) {
			break;
		}
		pszDomain[ 4093 ] = _T('\0');
		pszIP[ 1023 ]     = _T('\0');
		m_mapNames.insert( pair<CString,CString>(pszDomain,pszIP) );
	}

	fclose( pf );

	return true;
}
```

În cadrul metodei `getIP()` se utilizează metoda `find()` a clasei `std::map` pentru efectuarea operației de căutare. Definiția rezultată pentru această metode este următoarea:

```
bool CMyNameHandler::getIP( const CString& rDomain, CString& rIP )
{
	map<CString,CString>::iterator pos = m_mapNames.find( rDomain );
	if ( m_mapNames.end() == pos ) {
		return false;
	}

	rIP = (*pos).second;
	return true;
}
```

6.5 Construirea clasei de utilizare a soclului UDP

Pentru utilizarea soclului UDP vom moşteni clasa de încapsulare `CAsyncSocket` prin intermediul clasei `CMyDNSServerSocket`. Această clasă este instanțiată din `CMyDNSServerDlg` a cărei instanță este transmisă clasei `CMyDNSServerSocket` prin intermediul constructorului. Această legare a celor două clase asigură apelul metodelor clasei `CMyNameHandler` din cadrul `CMyDNSServerSocket`.

Metodele de utilizare a soclului UDP sunt cele prezentate în capitolul anterior. Diferențele constă în faptul că având un server UDP, numărul soclului va fi ales explicit de aplicație prin intermediul controalelor de editare, iar soclul nu va fi unul conectat întrucât serverul trebuie să deservească mai mulți clienți cu adrese IP diferite. Soclul poate fi legat însă de o anumită adresă IP prin intermediul ultimului parametru a metodei `Create()`, dacă se dorește ca aplicația să primească datagrame destinate unei anumite adrese IP. Acest mecanism se utilizează de regulă când avem mai multe plăci de rețea, fiecare placă având ataşată o altă adresă şi dorim ca serverul să deservească doar datagramele venite pe o anumită placă de rețea. Acest mecanism poate fi folosit şi pentru soclurile TCP.

Pentru crearea soclului vom utiliza o metodă publică:

```
bool createSocket( const unsigned short nPort );
```

Față de soclul client, soclul creat pentru server nu va fi unul conectat, iar numărul portului nu va fi unul auto-asignat întrucât un asemenea soclu ar avea un port necunoscut aplicațiilor client. Metoda de creare rezultată este următoarea:

```
bool CMyDNSServerSocket::createSocket( const unsigned short nPort )
{
      if ( m_bIsCreated ) {
            return false;
      }

      if ( !CAsyncSocket::Create( nPort, SOCK_DGRAM ) ) {
            return false;
      }

      return ( m_bIsCreated = true );
}
```

Distrugerea soclului se realizează în destructorul clasei, procedeu prezentat deja în capitolele anterioare:

```
CMyDNSServerSocket::~CMyDNSServerSocket()
{
      ShutDown( 2 );
      Close();
}
```

O dată ce soclul a fost creat, se pot recepționa cereri. Acestea sunt primite pentru procesare în metoda virtuală `OnReceive()`. În cadrul acestei metode vom apela două metode private, una pentru procesarea cererii `processRequest()` şi alta pentru transmiterea unei datagrame `sendDatagram()` utilizată pentru transmiterea răspunsului:

```
void CMyDNSServerSocket::OnReceive(int nErrorCode)
{
	if ( !nErrorCode )
	{
		_TCHAR pBuf[ 2049 ];
		int nRet = 0;
		CString sSrcAddr;
		UINT nSrcPort = 0;
		CString sResp;

		nRet = CAsyncSocket::ReceiveFrom( pBuf,
						  2048,
						  sSrcAddr,
						  nSrcPort );
		if ( SOCKET_ERROR == nRet ) {
			int nError = GetLastError();
			if ( WSAEMSGSIZE == nError ) {
				pBuf[ 2048 ] = _T('\0');

				// Se procesează cererea
				processRequest( pBuf, sResp );

				// Se transmite răspunsul
				sendDatagram( sSrcAddr, nSrcPort, sResp );
			}
			else {
				AfxMessageBox( _T("Receive ERROR"),
					       MB_OK | MB_ICONERROR );
			}
		}
		else {
			if ( nRet > 0 ) {
				pBuf[ nRet ] = _T('\0');

				// Se procesează cererea
				processRequest( pBuf, sResp );

				// Se transmite răspunsul
				sendDatagram( sSrcAddr, nSrcPort, sResp );
			}
		}
	}

	CAsyncSocket::OnReceive(nErrorCode);
}
```

Metoda privată de procesare a cererilor primite `processRequest()` asigură apelul metodei `resolveIP()` din cadrul `CMyDNSServerDlg` care la rândul ei va apela metoda `getIP()` din clasa `CMyNameHandler` pentru returnarea adresei IP corespunzătoare

denumirii domeniului dat. În cazul în care denumirea domeniului nu există, se construieşte un mesaj de eroare *NotFound*. Definiția rezultată este următoarea:

```
void CMyDNSServerSocket::processRequest( const _TCHAR* pszRequest,
                                         CString& rResp )
{
      CString sIP;
      rResp = pszRequest;
      if ( !m_pDlg->resolveIP( pszRequest, sIP ) )
      {
            rResp += _T(" ERROR ");
            rResp += _T("NotFound");
      }
      else {
            rResp += _T(" ");
            rResp += sIP;
      }
}
```

Ultima metodă definită pentru această clasă este `sendDatagram()`, o metodă privată prin care se asigură transmiterea unei datagrame la destinația specificată. Această metodă este utilizată pentru transmiterea răspunsurilor, indiferent de natura lor – de confirmare sau eroare. Definiția acesteia este dată în cele ce urmează:

```
void CMyDNSServerSocket::sendDatagram( const CString& rDestHost,
                                       const unsigned short nDestPort,
                                       const CString& rMsg )
{
      int nRet = CAsyncSocket::SendTo( rMsg,
                                       rMsg.GetLength(),
                                       nDestPort,
                                       rDestHost );
      if ( SOCKET_ERROR == nRet ) {
            AfxMessageBox( _T("Send ERROR"), MB_OK | MB_ICONERROR );
      }
}
```

6.6 Legarea claselor

Cele două clase prezentate anterior sunt instanțiate din clasa `CMyDNSServerDlg`, la o primă apăsare a butonului ilustrat în figura 6.2. Dacă inițializarea serverului este cu succes, la a doua apăsare a acestui buton instanțele create sunt distruse. În cazul în care nu se apasă butonul pentru distrugerea instanțelor, distrugerea trebuie să fie realizată în destructorul clasei `CMyDNSServerDlg`. Metoda ataşată tratării evenimentelor de apăsare a butonului este următoarea:

```
void CMyDNSServerDlg::OnBnClickedButton1()
{
      CWnd* pwnd = GetDlgItem( IDC_BUTTON1 );
      if ( NULL == pwnd ) {
            return;
      }
```

```
	CString sText;
	pwnd->GetWindowTextA( sText );
	if ( sText == _T("CreateServer") )
	{
		if ( NULL != m_pNames ) {
			delete m_pNames;
			m_pNames = NULL;
		}

		if ( NULL != m_pSocket ) {
			delete m_pSocket;
			m_pSocket = NULL;
		}

		UpdateData( TRUE );
		m_pNames = new CMyNameHandler();
		if ( !m_pNames->loadNames( m_sFile ) ) {
			AfxMessageBox( _T("Unable to open file\n"),
					MB_OK | MB_ICONERROR );
			return;
		}

		m_pSocket = new CMyDNSServerSocket( this );
		if ( !m_pSocket->createSocket( m_nPort ) ) {
			AfxMessageBox( _T("Unable to create socket\n"),
					MB_OK | MB_ICONERROR );
			return;
		}

		pwnd->SetWindowTextA( _T("Close") );
		AfxMessageBox( _T("Server successfully created"),
				MB_OK | MB_ICONINFORMATION );
	}
	else {
		if ( NULL != m_pNames ) {
			delete m_pNames;
			m_pNames = NULL;
		}

		if ( NULL != m_pSocket ) {
			delete m_pSocket;
			m_pSocket = NULL;
		}

		pwnd->SetWindowTextA( _T("CreateServer") );
	}
}
```

Metoda `resolveIP()`, apelată din clasa `CMyDNSServerSocket` este declarată în cadrul acestei clase ca o metodă publică. Definiția acesteia este următoarea:

```
bool CMyDNSServerDlg::resolveIP( const CString& rDomain, CString& rIP )
{
	if ( NULL == m_pNames ) {
		return false;
	}
```

```
    return ( m_pNames->getIP( rDomain, rIP ) );
}
```

6.7 Testarea aplicației server UDP-MFC

Pentru testarea aplicației construite se poate utiliza aplicația client prezentată în capitolul anterior. Dacă o asemenea aplicație nu este disponibilă se poate apela tot la utilitarul *Hercules* prezentat în capitolele anterioare.

Exercițiu.

Să se implementeze o aplicație server DNS bazat pe protocolul UDP prin utilizarea arhitecturii MFC pentru comunicarea pe socluri (i.e. clasa `CAsyncSocket`). Cerințele serverului DNS sunt cele specificate în cadrul acestui capitol.