MINISTERUL EDUCATIEI REPUBLICII MOLDOVA

UNIVERSITATEA TEHNICA A MOLDOVEI

Facultatea „Calculatoare, Informatică și Microelectronică”

Departamentul Automatica și Tehnologii Informaționale

RAPORT

Programarea în rețea

Lucrare de laborator Nr. 6

Tema: Proiectarea și programarea aplicației client server

A elaborat: st. gr. TI-142 Țurcan Tudor

A verificat: asist. univ. Donos Eugenia

Chișinău 2017

Cuprins

Scopul lucrării

Proiectarea și programarea aplicației de tip client-server.

https://github.com/TurcanTudor/Programarea-in-Retea.git

Sarcina lucrării

De creat o aplicație client-server utilizând Sockets API. De implementat totodată și un protocol propriu, cu o serie de comenzi :

e) serviciu director distant (autentificare, navigare, copiere fișiere, etc.)

f) mesagerie dintre utilizatori cu diverse tipuri de mesaje (public, privat,grup);

Considerații teoretice

Sistemul de operare foloseşte TDF (Tabela Descriptorilor de Fişier) pentru a păstra pointeri la structuri interne pentru fişierele cu care lucrează procesul. Aplicaţia (procesul) foloseşte aceşti descriptori pentru accesarea unui fişier. Socket-urile, servesc la comunicarea în reţea, şi asemeni descriptorilor de fişier sunt păstraţi tot în TDF. Creearea unui socket se face prin apelul funcţiei sistem socket(…). La acest apel sistemul de operare alocă o noua structură de date ce păstrează informaţiile necesare la comunicare şi actualizează pointerul în TDF la această structură. Înainte ca socket-ul să poată fi utilizat în aplicaţii câmpurile structurii trebuie actualizate conform cerinţelor aplicaţiei prin alte apeluri sistem.

Un socket poate fi pasiv dacă este folosit de un proces server în vederea aşteptării unor cereri de conectare, sau activ dacă este folosit de un proces client pentru a iniţia o conexiune.

Cele mai importante informaţii din structura sunt adresa IP şi numărul de port pentru protocol. Protocoalele TCP/IP definesc punctul de comunicaţie ca perechea adresa IP şi numărul de port pentru protocol. Alte familii de protocoale definesc adresa punctului de comunicaţie în alte moduri. Deoarece socketul este utilizabil cu multe familii de protocoale el nu specifică modul de definire a adresei punctului de comunicaţie şi nici formatul unei adrese pentru un protocol particular.

Pentru ca familiile de protocoale să aibă libertatea de a alege o reprezentare pentru adrese, socket-ul defineşte o familie de adrese pentru fiecare tip de adresa. Protocoalele TCP/IP folosesc o reprezentare unică pentru adrese, familia ardeselor fiind specificată prin constanta simbolica AF_INET (nu familia de protocoale PF_INET, deşi ambele valori sunt identice).

În practică, programele de aplicaţie folosesc o structură predefinită cu doua câmpuri: identificatorul familiei de adrese (2 octeţi) şi adresa (14 octeţi). Din raţiuni de portabilitate, în cazul protocoalelor TCP/IP se recomandă utilizarea ei numai pe post de structură de păstrare adresa (singura referire este la câmpul sa_family).

Desfășurarea lucrării

Programarea bazata pe Socket-uri este nucleul de bază a programării în rețea pe Windows și Linux. .NET oferă o platformă puternică pentru implementarea și folosirea socket-urilor utilizând limbajul C#.

Utilizând în programarea pe rețea socket-urile, nu are loc accesul direct a dispozitivului de rețea pentru a trimite și primi pachete. În schimb este creat un conector intermediar care gestionează interfața pentru programarea la rețea. Astfel socket-ul este un conector care conectează aplicația noastră la o interfață de rețea a computerului. Pentru a transmite și a primi date către și de la rețea trebuie să apelăm metodele socket-ului nostru (1).

Astfel utilizând socket-urile am creat o aplicație client-server, unde clientul trimite comenzi către server, iar acesta la rândul lui prelucrează datele primite de la client și returnează un anumit răspuns.

Mai întâi de toate vom analiza partea client aplicației noastre. Pentru acesta creăm mai întâi o variabilă care va fi un punct de conectare pentru aplicația noastră către un anumit serviciu de pe un server anumit.

Cu ajutorul clasei *Sockets* din .NET creăm o variabilă care ne va permite să utilizăm metode și proprietăți pentru comunicarea in rețea. În așa mod variabila *clientSocket* o inițializăm cu anumiți parametri. Primul specifică schema de adresare pe care socket-ul nostru o va utiliza. Al doilea parametru specifică tipul de socket, iar al treilea parametru specifică protocolul prin care ca avea loc transmiterea de date dintre client și server (figura 1).

```
private  Socket _clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)
public frm_Main()
```

Fig. 1 Inițializarea unui Socket

După inițializarea acestei variabile are loc scrierea funcționalului părții client. Astfel introducem o metoda ce este operata asyncron și returneaza tipul metodei ce inițiaza o operație asyncrona si le transmite metodelor invocate de AsyncCallBack care sunt delegate atunci cind operatiile asyncrone sunt indeplinite. Structura acestui bloc se poate de vizualizat în figura de mai jos (figura 2).

```
private void ReceiveData(IAsyncResult ar)
{
    Socket socket = (Socket)ar.AsyncState;
    int received = socket.EndReceive(ar);
    byte[] dataBuf = new byte[received];
    Array.Copy(receivedBuf, dataBuf, received);
    lb_stt.Text = (Encoding.ASCII.GetString(dataBuf));
    //rb_chat.AppendText("\nServer: " + lb_stt.Text);
    _clientSocket.BeginReceive(receivedBuf, 0, receivedBuf.Length, SocketFlags.None, new AsyncCallback(ReceiveData), _clientSocket);
}
```

Fig. 2 Codul sursă a funcției ReceiveData

La introducerea unei comenzi are loc codificarea acesteia într-o secvență de biți. Mai apoi se apelează metoda *Send()* caracteristică socket-ului nostru, având ca parametri comanda noastră în formatul menționat mai sus, lungimea comenzii și comportamentul socket-ului. Acesta se realizează la apasarea butonului Connect al aplicației noastre.

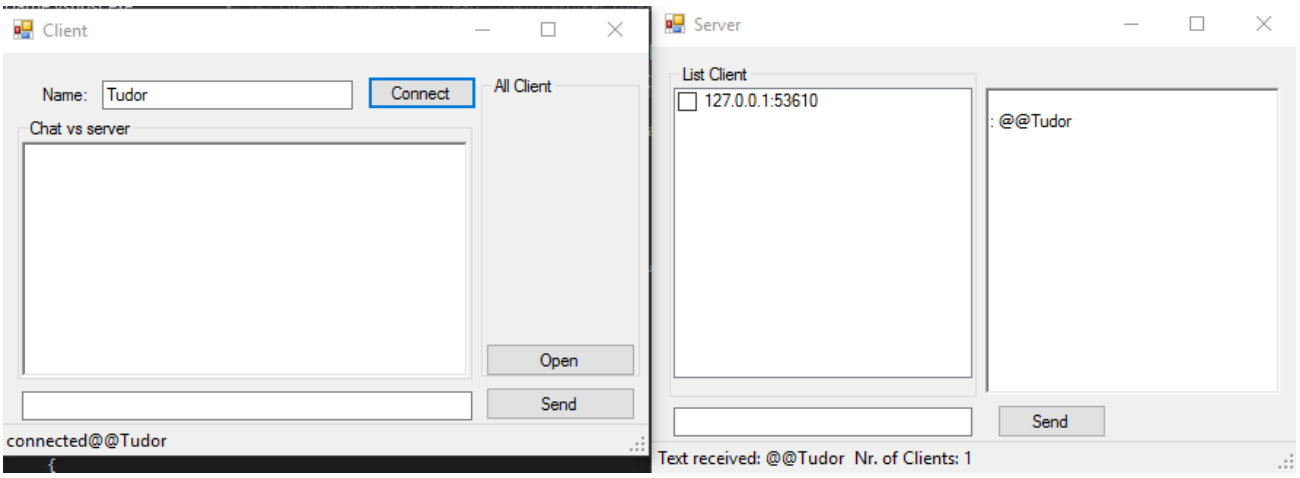

Fig. 3 – Conectarea la server

Pentru primirea datelor de la server de la un anumit serviciu apelat, creăm o variabilă pentru stocarea datelor trimise de acesta. Aplicând funția *Sendata()* ce are ca scop crearea variabilelor necesare

transformate in bytes și prelucrate cu ajutorul operațiilor asyncrone intru thread separat ,iar metoda callback preia un parametru AsyncResult ce este obtinut segvential din resultatul operatiei asyncrone.

```
void Sendata(Socket socket,string str)
        {
            byte[] data = Encoding.ASCII.GetBytes(str);
            socket.BeginSend(data, 0, data.Length, SocketFlags.None, new
AsyncCallback(SendCallback), socket);
            _serverSocket.BeginAccept(new AsyncCallback(AppceptCallback), null);
               }
```

După trimiterea datelor si preluarea lor putem transmite mesaje la clienții ce sau conectat la server iar alegînd clientul dorit îi transmitem anume unia din ei.Pentru aceasta bifam checkboxul dorit si apasam butonul send de la form-ul serverului.

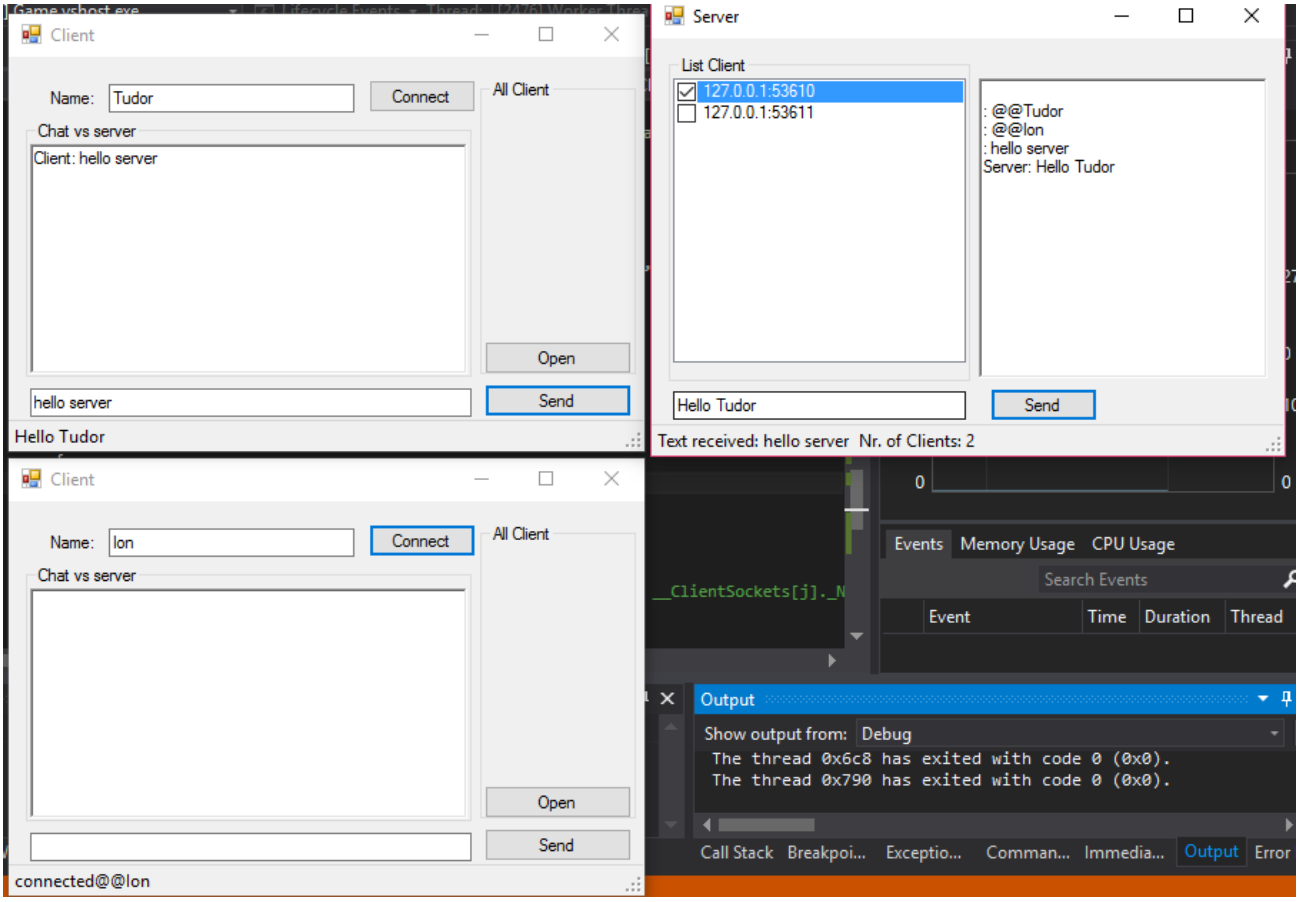

Fig. 4 Partea Client a aplicației

Realizarea aceste actiuni are loc cu ajutorul urmatorei segvențe de cod:

```
private void btnSend_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < list_Client.SelectedItems.Count; i++)
            {
                string t = list_Client.SelectedItems[i].ToString();
                    for (int j = 0; j < __ClientSockets.Count; j++)
```

```
                    {
                        if (list_Client.GetItemChecked(j))
                    {
                            Sendata(__ClientSockets[j]._Socket, txt_Text.Text);

                    }
                }
            }
            rich_Text.AppendText("\nServer: " + txt_Text.Text);
        }
    }
```

Aicea are loc cautarea prin lista de clinți și verificarea daca unul sau mai mulți clienți au fost aleși pentru transmiterea mesajului.

Partea Server a aplicației este responsabilă de efectuarea unor anumite calcule sau oferirea căror va informații. La fel ca și la partea client a aplicației are loc crearea unui punct de acces la rețea care va trebuie să primească de la client activități provenite de la orice interfețe de rețea, dar portul de comunicarea va rămâne același. Pe lingă aceasta mai sunt create si care liste de tip soket pentru threadul nostru și pentru căpătarea informației de la clienți.

```
private  byte[] _buffer = new byte[1024];
public List<SocketT2h> __ClientSockets { get; set; }
List<string> _names = new List<string>();
private  Socket _serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
public frm_Server()
{
    InitializeComponent();
    CheckForIllegalCrossThreadCalls = false;
    __ClientSockets = new List<SocketT2h>();
}
```

Fig. 5 Inițializarea unui Server Socket

Deja pentru server se va folosi metoda ReceiveCallBack pentru a asocia socket-ul creat de noi cu punctul de acces la rețea de la partea client. Aicea are loc concentrarea conexiunilor client cu serverul .

```
private  void ReceiveCallback(IAsyncResult ar)
        {
            Socket socket = (Socket)ar.AsyncState;
            if (socket.Connected)
            {
                int received;
                try
                {
                    received = socket.EndReceive(ar);
                }
                catch (Exception)
                {
                   for (int i = 0; i < __ClientSockets.Count; i++)
                    {
```

```
if(__ClientSockets[i]._Socket.RemoteEndPoint.ToString().Equals(socket.RemoteEndPoint.ToStrin
g()))
                {
                            __ClientSockets.RemoveAt(i);
                            lb_soluong.Text = "clients connected to server :
"+__ClientSockets.Count.ToString();
                        }
                    }

                    return;
                }
                if (received!=0)
                {
                    byte[] dataBuf = new byte[received];
                    Array.Copy(_buffer, dataBuf, received);
                    string text = Encoding.ASCII.GetString(dataBuf);
                    lb_stt.Text = "Text received: " + text;

                    string reponse = string.Empty;

                    for (int i = 0; i < __ClientSockets.Count; i++)
                    {
                        if
(socket.RemoteEndPoint.ToString().Equals(__ClientSockets[i]._Socket.RemoteEndPoint.ToString(
)))
                        {
                            rich_Text.AppendText("\n" + __ClientSockets[i]._Name + ": " +
text);
                        }
                    }

                    if (text == "bye")
                    {
                        return;
                    }
                    reponse = "connected" + text;
                    Sendata(socket, reponse);
                }
```

Pentru identificarea comenzilor de către server pentru accesarea sunt afișate intrun label poziționat in form-ul clientului .

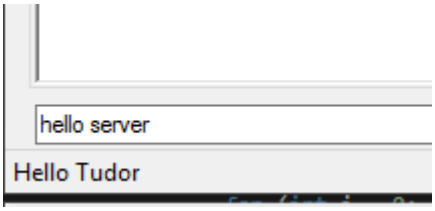

Fig. 6- Recevice data de la server

O alta functionalitate pe care o poseda aplicația noastră este de copiere a fișierelor intru-un anumit directoriu , pe care noi il alegem. Deci pentru a accesa aceasta funcționalitate tastam butonul Open ,unde ne va aparea încă 2 interfețe cu care vom interacționa.

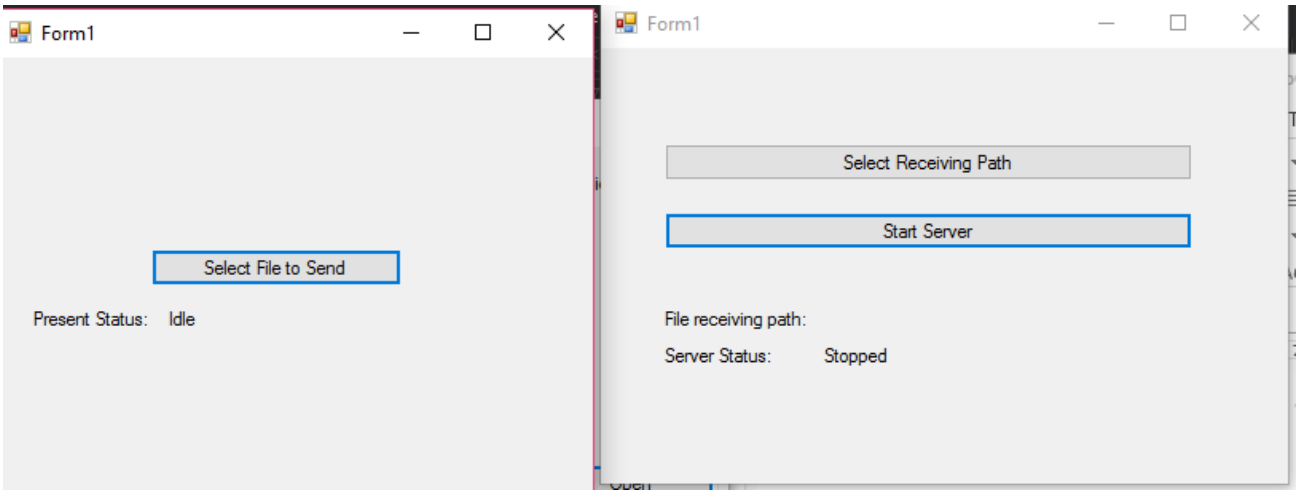

Fig. 7 Transmiterea fșierului

După cum observam mai sus pentru a incepe copierea unui fișier dintăi pornim serverul dupa care alegem path-ul dorint unde se va copia fișierul nostru , și in ultimul rind selectam fișierul dorit care va fi copiat.În urma efectuarii acestor acțiuni ne va aparea informatia data dupa care putem selecta fișierul:

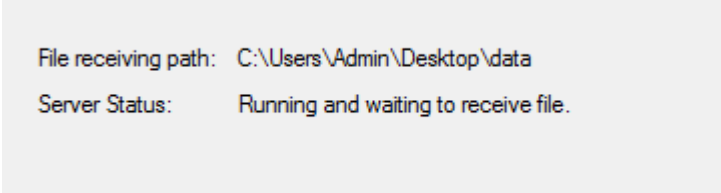

Fig. 8 Închiderea conexiunilor

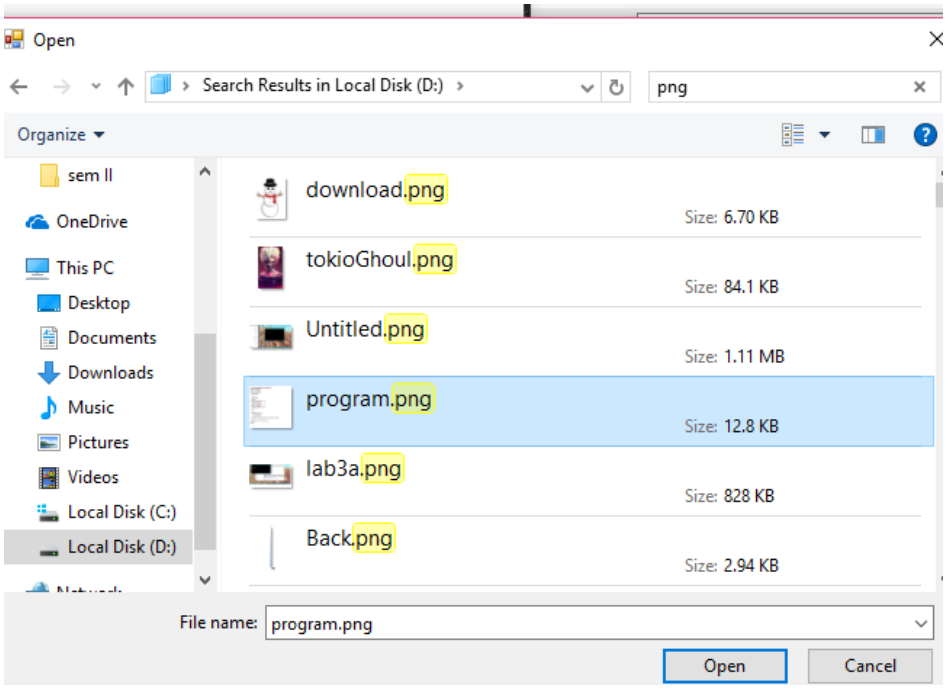

Fig. 8 Alegerea fișierului dorit

Fig. 8 Rezultatul așteptat

Concluzie

În cadrul acestei lucrări de laborator am realizat o aplicație bazată pe arhitectura client-server. Astfel clientul este partea activă ce inițiază conexiunea cu serverul și solicită servicii de la acesta, iar serverul este partea pasivă ce oferă servicii și realizează careva operațiuni.

Pentru aceasta s-au utilizat socket-uri pentru că acestea oferă accesul la interfețe de nivel jos. Principalele atribute ce sunt utilizate de socket-uri sunt IP și porturile. Adresele IP sunt unice ceea ce permite identificarea unui computer într-o rețea. Totodată am folosit porturile pentru ca aplicația noastră client server să poată comunica între ele. Aceasta include transmiterea și primirea datelor.

Bibliografie

1. **ALLEXY.** TCP/IP Chat Application Using C#. *http://www.codeproject.com/.* [Online] [Cited: May 22, 2017.] http://www.codeproject.com/.

2. **2016 Microsoft.** IPEndPoint Class. *msdn.microsoft.com.* [Online] [Cited: May 24, 2017.] https://msdn.microsoft.com/en-us/library/system.net.ipendpoint%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396.

3. —. Socket.Listen Method (Int32). *msdn.microsoft.com.* [Online] [Cited: Mai 24, 2017.] https://msdn.microsoft.com/en-us/library/system.net.sockets.socket.listen%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396.

4. AsyncCallback Delegate [Online] [Cited:Mai 20, 2017.] https://msdn.microsoft.com/en-us/library/system.asynccallback(v=vs.110).aspx