IPK – Počítačové komunikace a sítě

Dokumentácia k projektu

Varianta EPSILON: Simple File Transfer Protocol v jazyku C#

Dátum: 24.4.2022

Autor: František Fúdor (xfudor00)

Obsah

Zoznam obrázkov

Simple File Transfer Protocol

RFC 913 - Simple File Transfer Protocol ďalej už len SFTP, je protokol, ktorý zabezpečuje kontrolu užívateľského prístupu, prenos súborov, výpis priečinkov, premenovanie súborov a odstraňovanie súborov. SFTP je veľmi podobný TFTP (Trivial File Transfer Protocol), ale o niečo užitočnejší a ľahšie implementovateľný. SFTP je možné implementovať s hocijakým spoľahlivým protokolom s 8-bitovým tokom dát. SFTP používa len jedno TCP pripojenie na rozdiel od TFTP, ktorý implementuje UDP a TCP, ktorý používa dva TCP pripojenia, jedno z toho je TELNET protokol.

Použitie SFTP

SFTP sa používa tak, že sa nadviaže TCP pripojenie na server s SFTP portom 115. Klient ďalej zadáva príkazy, ktoré sú v súlade s SFTP štandardom a čaká na odpoveď zo servera. SFTP príkaz sú vždy 4 ASCII písmená po ktorých nasleduje medzera, argument(y) a nakoniec NULL. Argument môže byť prázdny napríklad pri príkaze “DONE“. Tým pádom príkaz obsahuje len 4 ASCII písmená, za ktorými nasleduje NULL. Odpoveď zo serveru je vždy znak úspešnosti operácie, za ktorým nasleduje správa v ASCII, ktorá je ukončená NULL. Odpoveď môže byť aj napríklad znak odpovede ukončený s NULL.

Výpis možností príkazov z dokumentácie SFTP protokolu:

<command> : = <cmd> [<SPACE> <args>] <NULL>

<cmd> : = USER ! ACCT ! PASS ! TYPE ! LIST ! CDIR
KILL ! NAME ! DONE ! RETR ! STOR

<response> : = <response-code> [<message>] <NULL>

<response-code> : = + | - | | !

<message> can contain <CRLF>

Implementácia

Popis projektu

Projekt som vypracoval v jazyku C#, s ktorým už mám nejaké skúsenosti a hlavne kvôli tomu, že o spracovaní argumentov pomocou nugetového balíčká CommandLine som vedel už pred tým a jednoducho ma oslovil.

Projekt som vypracoval najskôr v systéme Windows s použitím Visual Studio 2022, s ktorým už mám taktiež skúsenosti a kvôli tomu, že sa kód príjemne debugguje. Neskôr som prešiel na systém Linux a to práve virtuálny stroj Ubuntu, v ktorom som hlavne riešil Makefile.

Projekt je štrukturovaný do dvoch C# projektov. Jeden z nich je server (ipk-simpleftp-server.csproj) a druhý je klient (ipk-simpleftp-client.csproj). Výsledkom sú dve konzolové aplikácie.

Projekt sa prekladá pomocou spomínaného Makefile, ktorý odstráni staré súbory, získa použité nugetové balíčky pre oba projekty, preloží server aj klienta a vygeneruje do root-u spustiteľné binárne súbory.

```
1  all : clean restore build publish
2
3  clean:
4          rm -rf ./client/bin
5          rm -rf ./client/obj
6          rm -rf ./server/obj
7          rm -rf ./server/bin
8          rm -rf ./ipk-simpleftp-client
9          rm -rf ./ipk-simpleftp-server
10
11 restore:
12         dotnet restore ./server/ipk-simpleftp-server.csproj
13         dotnet restore ./client/ipk-simpleftp-client.csproj
14
15 build:
16         dotnet build ./server/ipk-simpleftp-server.csproj
17         dotnet build ./client/ipk-simpleftp-client.csproj
18
19 publish:
20
21         dotnet publish ./server/ipk-simpleftp-server.csproj -r linux-x64 -o ./ -p:PublishSingleFile=true --self-contained true
22         dotnet publish ./client/ipk-simpleftp-client.csproj -r linux-x64 -o ./ -p:PublishSingleFile=true --self-contained true
23
```

Obrázok 1: Makefile

Implementačné detaily

Server (IPKServer.cs)

Vytvoril som projekt s názvom *ipk-simpleftp-server.csproj*, ktorý obsahuje priečinok (*namespace*) *server*. V ňom sú uložené jednotlivé triedy *IPKServer.cs*, v ktorom je implementovaná logika behu serveru:

1. Verejné properties

Deklarácia verejných *properties* serveru.

```
/// <summary>
/// Global properties such as paths, hostname, listener, dictionary with credentials and threads for async communication
/// </summary>
3 references
public static ManualResetEvent allDone = new ManualResetEvent(false);
1 reference
public static string CredentialsPath = "";
4 references
public static string Hostname = "";
2 references
public static string GlobalWorkingDirectory = "";
64 references
public static TcpListener _server;
3 references
public static Dictionary<string, string> credentials;
```

Obrázok 2: Verejné properties serveru

2. Spracovanie argumentov

Spracovanie argumentov pomocou nugetového balíčka *CommandLine*, pre ktoré sa vytvorí trieda *Options*, kde sa zvolia dané argumenty a ich vlastnosti.

```
/// <summary>
/// Nuget CommandLine options for parsing console arguments
/// Inspired by: https://github.com/commandlineparser/commandline
/// </summary>
2 references
public class Options
{
    [Option('i', "interface", Default = "any", Required = false, HelpText = "Interface on which server will be listening.")]
    2 references
    public string _interface { get; set; }

    [Option('p', "port", Default = 115, Required = false, HelpText = "Port on which server will be listening.")]
    1 reference
    public int _port { get; set; }

    [Option('u', "path_to_credentials", Required = true, HelpText = "Absolute path to database with credentials.")]
    3 references
    public string _path { get; set; }

    [Option('f', "path_to_working_directory", Required = true, HelpText = "Absolute path to working directory.")]
    3 references
    public string _dir { get; set; }


}
```

Obrázok 3: Spracovanie argumentov

3. Metóda Main()

Metóda *Main()*, ktorá volá *CommandLine Parser*.

```
/// <summary>
/// Main method calls ServerRun() method if arguments were parsed successfully otherwise CommandLine writes help on console output
/// </summary>
/// <param name="args">Console arguments</param>
0 references
static void Main(string[] args)
{
    CommandLine.Parser.Default.ParseArguments<Options>(args)
        .WithParsed(ServerRun);
}
```

Obrázok 4: Metóda Main() serveru

3.1. Úspech parsovania

Pri úspechu zavolá metódu *ServerRun()*, v ktorej je hlavná časť behu serveru.

3.2. Neúspech parsovania

Pri neúspechu (napríklad pri nepriložení povinného argumentu) sa vypíše pomoc.

```
ERROR(S):
  Required option 'u, path_to_credentials' is missing.
  Required option 'f, path_to_working_directory' is missing.

  -i, --interface                  (Default: any) Interface on which server will be listening.

  -p, --port                       (Default: 115) Port on which server will be listening.

  -u, --path_to_credentials        Required. Absolute path to database with credentials.

  -f, --path_to_working_directory  Required. Absolute path to working directory.

  --help                           Display this help screen.

  --version                        Display version information.
```

Obrázok 5: Neúspech argumentov serveru

4. Metóda ServerRun()

4.1. Ošetrenie vstupov od užívateľa

Metóda *ServerRun()*, ktorá na začiatku skontroluje, či zvolená cesta k pracovnému priečinku a prihlasovacím údajom existuje. Ďalej uloží cestu do pracovného súboru do verejnej *property* a vytvorí slovník, do ktorého sa uložia prihlasovacie údaje v tvare *username = password*.

```
//Gets credentials from console argument -u and store it into dictionary in format 'username = password'
//Inspired by: https://stackoverflow.com/a/39992125
List<string> credentialsList = File.ReadAllLines(opts._path).ToList();
credentials = new Dictionary<string, string>();
foreach (string line in credentialsList)
{
    string[] keyvalue = line.Split(':');

    if (keyvalue.Length == 2)
    {

        credentials.Add(keyvalue[0], keyvalue[1]);
    }
}
```

Obrázok 6: Slovník prihlasovacích údajov

4.2. Nastavenie serveru

Metóda získa *hostname* servera a uloží ho, potom vytvorí IP adresu, ktorá je na začiatku nastavená na *IPv6Any*, ktorá reprezentuje *::0*. Na základe vytvorenej IP adresy a portu, ktorý je na začiatku nastavený na 115 vytvorím *endpoint*, ktorý reprezentuje IP adresu a port. Na *socket* neskôr nabindujem daný *endpoint*, na ktorom sa začne počúvať.

Ak zvolené rozhranie užívateľom, prejdem všetky dostupné rozhrania, nájdem dané rozhranie a prepíšem IP adresu *endpointu* na IP adresu zvoleného rozhrania .

```
//Get all interfaces and try to find interface with name of console argument -f
//Inspired by: https://stackoverflow.com/a/10060249
foreach (NetworkInterface ni in NetworkInterface.GetAllNetworkInterfaces())
{
    //If interface is found get its IP address and rewrite variable address with it
    if (ni.Name.Equals(opts._interface))
    {
        foreach (UnicastIPAddressInformation ip in ni.GetIPProperties().UnicastAddresses)
        {
            address = ip.Address;
        }
    }
}
```

Obrázok 7: Hľadanie IP adresy rozhrania

Vytvorím serverový *socket*, nabindujem *endpoint* a nastavím mu aby nebral len IPv6, ale aj IPv4.

```
//Create TcpListener as server, get socket and set socket options to support both IPv6 and IPv4
//Inspired by: https://chiplegend.wordpress.com/2013/05/10/c-server-that-supports-ipv6-and-ipv4-on-the-same-port/
_server = new TcpListener(endPoint);
Socket serverSocket = _server.Server;
serverSocket.SetSocketOption(SocketOptionLevel.IPv6, SocketOptionName.IPv6Only, false);
```

Obrázok 8: Nastavenie serveru

4.3. Beh serveru

Vypíšem serverové informácie na konzolový výstup a spustím server vo *while()* cykle. Na začiatku cyklu zresetujem *property allDone*, ktorá funguje ako semafór pre asynchrónnu komunikáciu. Ďalej začnem počúvať asynchrónnu komunikáciu pomocou *callback* metódy, ktorá na začiatku nastaví *property allDone*, vytvorí *socket* pre klienta (potvrdí jeho pripojenie) a vďaka *callback* metóde začne znova počúvať pre pripojenie ďalšieho klienta. Ďalej vytvorím inštanciu triedy *IPKServerClient*, ktorá zabezpečuje SFTP komunikáciu medzi serverom a klientom a priamo rieši SFTP príkazy.

```
/// <summary>
/// TcpAcceptClient callback method - accept client (end of accepting) then start accepting again for another client (async)
/// </summary>
/// <param name="result">Callback result</param>
2 references
public static void TcpAcceptClient(IAsyncResult result)

{
    //Threads set
    allDone.Set();

    //Accept client
    TcpClient client = _server.EndAcceptTcpClient(result);

    //Start listening again
    _server.BeginAcceptTcpClient(TcpAcceptClient, _server);

    //Initialize new connected client
    IPKServerClient serverClient = new IPKServerClient(client, GlobalWorkingDirectory, Hostname, credentials);

    //Threads for queueing ClientRun() method in client class
    ThreadPool.QueueUserWorkItem(serverClient.IPKServerClientRun, client);
}
```

Obrázok 9: Callback metóda TcpAcceptClient

Server Client (IPKServerClient.cs)

1. Verejné properties

Deklarácia verejných *properties*

```
/// <summary>
/// Global properties
/// </summary>
7 references
private bool LoggedIn = false;
8 references
private string Account = "";
13 references
private string WorkingDirectoryPath = "";
3 references
private string Hostname = "";
3 references
private Dictionary<string, string> Credentials;
1 reference
private TcpClient _client;
3 references
private NetworkStream _stream;
101 references
private StreamReader _reader;
101 references
private StreamWriter _writer;
```

Obrázok 10: Verejné properties klienta

2. Metóda IPKServerClientRun()

Metóda IPKServerClientRun(), beží vo while() cykle, ktorý non-stop číta správy od klienta. Tieto správy premení na SFTP príkazy, ktoré pomocou switch() interpretuje (volá pomocné metódy SFTP príkazov).

3. Pomocné SFTP metódy

Každá pomocná metóda ošetrí argumenty príkazu a interpretuje príkaz. Po interpretácií posiela správu klientovi o jej úspešnosti.

3.1. Ukážky pomocných SFTP metód

Ukážka SFTP príkazu LIST s vloženou cestou k priečinku. Zistí, či je cesta, ktorú zadal existujúci priečinok. Získa informácie o priečinku pomocou *FileInfo*, vypíše a pošle klientovi hlavičku odpovede pre STFP príkaz LIST a pre každý súbor, ktorý našiel v danom priečinku pošle správu klientovi o jeho vlastnostiach. Nakoniec pošle správu klientovi „done“, ktorá mu hovorí, že má prestať čítať v cykle.

```
//Remove 'LIST V '
var path = data.Remove(0, 7);

//If path leads to existing directory
if (Directory.Exists(path))
{

    //Get name of directory
    FileInfo dirInfo = new FileInfo(path);

    //Write and send message "R: +{dir}: \n"
    string message = "R: +" + dirInfo.Name + ":";
    _writer.WriteLine(message);
    _writer.Flush();
    Console.WriteLine(message);

    //Get list of files in asked directory
    List<string> list = Directory.GetFiles(path).ToList();

    //Foreach file
    foreach (string s in list)
    {
        //Get file info
        FileInfo fileInfo = new FileInfo(s);

        //Concat message with informations of a file
        message = string.Concat(fileInfo.Name + "\t" + fileInfo.Length + "B\t" + fileInfo.CreationTimeUtc);

        //Write message to client
        _writer.WriteLine(message);
        _writer.Flush();
        Console.WriteLine(message);
    }

    //Write message to client - this message indicates stop reading in cycle
    _writer.WriteLine("done");
    _writer.Flush();

}
```

Obrázok 11: Ukážka SFTP LIST príkazu

Ukážka SFTP príkazu TOBE, ktorý premenuje zadaný súbor. Získa informácie o súbore, premiestni súbor do nového súboru s novým menom a starý súbor vymaže. Pošle správu klientovi o úspešnosti operácie.

```
//Try to rename file
try
{
    //Creates file with new name and moves content from old file into it then deletes old file
    FileInfo fi = new FileInfo(WorkingDirectoryPath + "/" + tmp);
    fi.MoveTo(WorkingDirectoryPath + "/" + _tmp);

    //Write message to client
    _writer.WriteLine("R: +<" + tmp + "> renamed to <" + _tmp + ">" + null);
    _writer.Flush();
    Console.WriteLine("R: +< " + tmp + " > renamed to < " + _tmp + " > " + null);
}
//If something went wrong
catch (Exception ex)
{
    //Write message to client
    _writer.WriteLine("R: -Rename aborted, file could not be renamed, don't send TOBE" + null);
    _writer.Flush();
    Console.WriteLine("R: -Rename aborted, file could not be renamed, don't send TOBE" + null);
}
```

Obrázok 12: Ukážka SFTP TOBE príkazu

Klient (IPKClient.cs)

1. Spracovanie argumentov

Spracovanie argumentov pomocou nugetového balíčka *CommandLine*, pre ktoré sa vytvorí trieda *Options*, kde sa zvolia dané argumenty a ich vlastnosti.

```
/// <summary>
/// Nuget CommandLine options for parsing console arguments
/// Inspired by: https://github.com/commandlineparser/commandline
/// </summary>
2 references
public class Options
{
    [Option('h', "IP", Required = true, HelpText = "IPv4 or IPv6 address of server.")]
    2 references
    public string _address { get; set; }

    [Option('p', "port", Default = 115, Required = false, HelpText = "Port on which server will be listening.")]
    2 references
    public int _port { get; set; }

    [Option('f', "cesta_k_adresari", Required = true, HelpText = "Absolute path to working directory.")]
    0 references
    public string _dir { get; set; }
}
```

Obrázok 13: Spracovanie argumentov klienta

2. Metóda Main()

Metóda *Main()*, ktorá volá *CommandLine Parser*.

```
/// <summary>
/// Main method calls ClientStart() method if arguments were parsed successfully otherwise CommandLine writes help on console output
/// </summary>
/// <param name="args">Console arguments</param>
0 references
static void Main(string[] args)
{
    CommandLine.Parser.Default.ParseArguments<Options>(args)
        .WithParsed(ClientStart);
}
```

Obrázok 14: Metóda Main() klienta

2.1. Úspech parsovania

Pri úspechu zavolá metódu *ClientStart()*, v ktorej je hlavná časť behu klienta.

2.2. Neúspech parsovania

Pri neúspechu (napríklad pri nepriložení povinného argumentu) sa vypíše pomoc.

```
ERROR(S):
  Required option 'h, IP' is missing.
  Required option 'f, cesta_k_adresari' is missing.

  -h, --IP                  Required. IPv4 or IPv6 address of server.

  -p, --port                (Default: 115) Port on which server will be listening.

  -f, --cesta_k_adresari    Required. Absolute path to working directory.

  --help                    Display this help screen.

  --version                 Display version information.
```

Obrázok 15: Neúspech spracovania argumentov klienta

3. Metóda ClientStart()

Nekoneční while() cyklus, ktorý sa non-stop snaží pripojiť na server (aj po vypnutí serveru). Je možné ho ukončiť skratkou CTRL+C.

3.1. Nastavenie klienta

Vytvorí sa socket pre klienta, klient sa pripojí na server so zvolenou IP adresou a portom, získa tok dát od serveru, pošle uvítaciu správu serveru, získa uvítaciu správu od servera a začne posielať príkazy na server.

```
//Connecting to the server
Console.Write("Connecting to " + opts._address + ":" + opts._port + "... ");
client.Connect(opts._address, opts._port);
Console.WriteLine("Connected!");

//Get network stream and it's writer and reader
NetworkStream stream = client.GetStream();
StreamWriter writer = new StreamWriter(stream, encoding: Encoding.ASCII);
StreamReader reader = new StreamReader(stream, encoding: Encoding.ASCII);
Console.WriteLine("------------------------------------------------");

//Write welcome message to server
writer.WriteLine("(opens connection to R)");
writer.Flush();
Console.WriteLine("S: (opens connection to R)");

//Get welcome message from server
string data = reader.ReadLine();
Console.WriteLine(data);
```

Obrázok 16: Nastavenie klienta

3.2. Beh klienta (posielanie príkazov na server)

Nekonečný while() cyklus, ktorý sa ukončí buď skratkou CTRL+C alebo zadaním SFTP príkazu DONE. V cykle sa non-stop číta z konzolového vstupu. Čo sa prečíta, to sa pošle na server cez dátový tok.

3.2.1. SFTP command DONE

Ak bol vložený SFTP príkaz DONE, ukonči dátový tok a vypni klienta.

```
//If message was DONE SFTP command
if (line.Equals("DONE"))
{
    //End client
    done = true;
    data = reader.ReadLine();
    Console.WriteLine(data);
    stream.Close();
    client.Close();
    break;
}
```

Obrázok 17: SFTP command DONE na klientovi

3.2.2. SFTP command LIST

Ak bol vložený SFTP príkaz LIST, začne čítať správy od klienta v cykle, až kým nedostane správu „done“, ktorá ukončí čítanie v cykle (pre veľký výpis obsahu priečinka)

```
//Try to parse LIST SFT command
try
{
    //If command was LIST {F|V}
    if (line.Substring(0, 6).Equals("LIST F") || line.Substring(0, 6).Equals("LIST V"))
    {
        //Read messages from server until message "done" was sent from server
        while ((data = reader.ReadLine()) != null)
        {
            //If message from server was "done"
            if (data.Equals("done"))
            {
                //Stop reading in cycle
                break;
            }
            //If message from server was access denied
            if (data.Equals("R: -Access denied, please logg in"))
            {
                //Write access denied on console
                Console.WriteLine(data);

                //Stop reading in cycle
                break;
            }

            //Write message from server on console
            Console.WriteLine(data);
        }
        //Continue writing commands
        continue;
    }
}
catch (Exception ex)
{
    //Empty
}
```

Obrázok 18: SFTP príkaz LIST na klientovi

Záver

Naučil som sa veľa nového v jazyku C#. Milo ma prekvapilo asynchrónna komunikácia v tomto jazyku a hlavne veľa nových poznatkov o dátovom toku medzi klientom a serverom.

Zdroje

Simple File Transfer Protocol:

- https://datatracker.ietf.org/doc/html/rfc913

Hľadanie IP adresy rozhrania:

- https://stackoverflow.com/a/10060249

TcpListener:

- https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.tcplistener?view=net-6.0#examples

TcpClient:

- https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.tcpclient?view=net-6.0

Nastavenie socketu pre IPv4 aj IPv6:

- https://chiplegend.wordpress.com/2013/05/10/c-server-that-supports-ipv6-and-ipv4-on-the-same-port/

Rozdelenie stringu do slovníka cez rozdeľovač „:“ :

- https://stackoverflow.com/questions/39991839/c-sharp-dictionary-how-to-read-key-value-from-file

Dotnet build:

- https://www.phillipsj.net/posts/building-dotnet-core-apps-old-school-with-make/

File Exist:

- https://www.c-sharpcorner.com/blogs/how-to-rename-a-file-in-c-sharp

Asynchrónny server:

- https://docs.microsoft.com/en-us/dotnet/framework/network-programming/asynchronous-server-socket-example

Asynchrónny klient:

- https://docs.microsoft.com/en-us/dotnet/framework/network-programming/asynchronous-client-socket-example