Práce se sítí – System.Net

738. Zasíláme textovou zprávu prostřednictvím protokolu UDP

Protokol UDP je nespojový protokol, jehož zprávy jsou posílány datagramy. Díky své nespojovosti a poměrné jednoduchosti oproti ostatním protokolům je určen hlavně k rychlé, ale nespolehlivé komunikaci. Pro používání tohoto protokolu obsahuje prostředí .NET třídu UdpClient.

Chceme-li poslat textový řetězec vzdálenému cíli pomocí protokolu UDP, nejprve se k němu připojíme metodou Connect třídy System.Net.Sockets.UdpClient na portu, jehož číslo je argumentem této metody. Poté zakódujeme textový řetězec, který byl také parametrem metody, na pole bajtů a pošleme je metodou Send, kde jej přijme naslouchající aplikace.

```
public void PosliZpravu(string zprava, int port)
{
   UdpClient  client = new UdpClient();
   client.Connect(IPAddress.Loopback, port);
   byte[] data = Encoding.ASCII.GetBytes(zprava);
   client.Send(data, data.Length);
}
```

739. Jak naslouchat příchozím textovým zprávám pomocí protokolu UDP

Pokud chceme přijmout textovou zprávu pomocí protokolu UDP, použijeme následující metodu s jediným argumentem, který říká, na jakém portu budeme příchozím zprávám naslouchat.

```
public string Naslouchej(int port)
{
  UdpClient client = new UdpClient(port);                          //1.
  IPEndPoint host = new IPEndPoint(IPAddress.Any, 0);              //2.
byte[] prijateBajty = client.Receive(ref host);                   //3.
string prijatyString = Encoding.ASCII.GetString(prijateBajty); //4.
  return prijatyString;                                            //5.
}
```

Na prvním řádku metody jsme v konstruktoru třídy UdpClient nastavili port, na kterém budeme naslouchat. Instance třídy IPEndPoint nám říká, od kterého příjemce bude paket přijat (v tomto případě jej můžeme přijmout od kohokoliv). Na třetím řádku přijímáme pole bajtů metodou Receive, které poté převedeme na textový řetězec třídou Encoding. Ten pak vrátíme příkazem return.

740. Přijímáme zprávy pouze z konkrétní IP adresy

Může se stát, že budeme potřebovat přijímat pakety pouze od konkrétního odesilatele. Tato situace nastává, když očekáváme více zpráv od různých odesilatelů. K tomu, abychom mohli přijímat pakety pouze od jednoho určitého odesilatele, změníme v předchozí metodě argument v konstruktoru třídy IPEndPoint takto:

```
IPEndPoint host = new IPEndPoint(IPAdress.Parse("nějaká IP adresa"));
```

Jako argument metody Parse třídy IPAdress použijeme IP adresu odesilatele, jehož pakety chceme přijímat.

741. Přijímáme zprávy z několika konkrétních IP adres

Pokud si přejeme přijímat zprávy prostřednictvím protokolu UDP od několika určitých odesilatelů, nejprve uložíme výčet jejich IP adres do pole a poté zavoláme tuto metodu.

```
public string Naslouchej(int port)
{
   string[] adresy = new string[3] { "55.66.77.88", "88.77.66.133",
                                     "88.102.234.250" };
   bool nalezeno = false;
   UdpClient client = new UdpClient(port);
   IPEndPoint host  = new IPEndPoint(IPAddress.Any, 0);
   byte[] prijateBajty = client.Receive(ref host);
   foreach (string adresa in adresy)
   {
     if (host.ToString().Contains(adresa))
     {
        nalezeno = true; break;
     }
   }
   if (!nalezeno)
   {
     throw new Exception
     ("Přijatá zpráva není poslána z požadované IP adresy");
   }
   return Encoding.ASCII.GetString(prijateBajty);
}
```

Zprávu nejprve přijmeme metodou Receive, a pokud se IP adresa odesilatele shoduje s jednou z námi požadovaných IP adres uložených v poli, vrátíme příkazem return přijatou zprávu, v opačném případě vyvoláme výjimku.

742. Ověření validity IP adresy

Jestliže potřebujeme ověřit, zda je IP adresa napsána správně, použijeme tuto metodu:

```
IPAddress adresa = null;
public bool IsIPValid(string ip)
{
    return  IPAddress.TryParse(ip, out adresa);
}
```

Pokud jako argument metody použijeme třeba "88.4a", metoda TryParse třídy IPAddress vrátí false, protože se nejedná o korektní IP adresu. Použijeme-li však jako argument třeba „107.0.0.1" metoda vrátí true, protože tato IP adresa je použitelná.

743. Identifikace chyby pomocí chybových zpráv WinSock

Tato metoda vypisuje chybové hlášky socketových spojení. Použijeme ji tam, kde nám náš kód pracující se sockety generuje výjimky a my neznáme jejich příčinu.

```
public void ZjistiDruhChyby()
{
 try
```

```
 {
   //Kód pracující s UDP či jiným socketovým protokolem
 }
 catch(SocketException r)
 {
    int kód = r.ErrorCode;
    switch (kód)
    {
      case 1004:
       Console.WriteLine("Interrupted function call");
       break;
      case 10013:
       Console.WriteLine("Permission denied");
       break;
      case 10014:
       Console.WriteLine("Bad address");
       break;
      case 10048:
       Console.WriteLine("Address already in use");
       break;
    }
    .
    .
    .   // Případně další druhy chybových hlášení.
  }
}
```

Další druhy chybových hlášení můžete najít na http://msdn2.microsoft.com/en-us/library/ms740668.aspx.

744. Poslání souboru pomocí protokolu UDP

Jestliže chceme poslat soubor pomocí protokolu UDP, poslouží nám následující jednoduchá metoda.

Jako první argument zadáme port, na kterém bude přenos probíhat, druhým argumentem bude cesta k souboru, který bude poslán, a poslední argument nám poslouží jako IP adresa cíle, jemuž posíláme data.

```
public void PosliSoubor(int port, string cesta, string cilovaIP)
{
  UdpClient client = new UdpClient();
  client.Connect(IPAddress.Parse(cilovaIP), port);
  byte[] data = System.IO.File.ReadAllBytes(cesta);
  client.Send(data, data.Length);
}
```

Instance třídy UdpClient se připojí ke vzdálenému cíli na určitém portu. Poté pomocí metody ReadAllBytes třídy System.IO.File přečteme všechny bajty souboru, jehož cestu jsme deklarovali jako argument metody, a toto pole bajtů pošleme metodou Send.

745. Jak přijmout soubor protokolem UDP

Pokud chceme přijmout soubor protokolem UDP, použijeme následující metodu. Jako první argument metody zadáme port, na kterém bude metoda naslouchat. Druhým argumentem bude cesta k souboru, kam jej po přijetí uložíme.

```
public void PrijmiSoubor(int port, string cestaSouboru)
{
  UdpClient client = new UdpClient(port);
  IPEndPoint host = new IPEndPoint(IPAddress.Any, 0);
  byte[] přijatéBajty = client.Receive(ref host);
  File.WriteAllBytes(cestaSouboru, přijatéBajty);
}
```

Naslouchame pomocí metody Receive, která vrací přijaté byty. Ty poté zapíšeme metodou WriteAllBytes třídy System.IO.File do souboru, jehož cestu jsme deklarovali jako parametr metody.

746. Zjištění velikosti přijatých nebo odeslaných dat

Následující metoda zjištuje velikost souboru (který dostane jako pole bajtů) a vrací jeho velikost ve správných jednodkách (b, Kb nebo Mb).

```
static string[] přípona = {"B", "KB", "MB", "GB", "TB", "PB", "EB"};
public string ZjistiVelikost(byte[] bajty)
{
  float velikost = bajty.Length;
  int kilo = 1024;
  for(int i = 0; i < 7; i++)
  {
    if(velikost < kilo)
      return velikost + " " +přípona[i];
    velikost /= kilo;
  }
  throw new ArgumentException("moc veliké");  // To nemůže nastat
}
```

747. Jak zaslat textovou zprávu prostřednictvím TCP-spojení

Protokol TCP je spojovanou službou, tj. službou, která mezi dvěma aplikacemi naváže spojení, tj. vytvoří na dobu spojení virtuální okruh. Oproti UDP je sice pomalejší, ale zato mnohem bezpečnější. V prostředí .NET máme pro práci s tímto protokolem k dispozici třídy TcpListener a TcpClient.

Chceme-li poslat textovou zprávu protokolem TCP, vytvoříme instanci třídy TcpClient, jejímuž konstruktoru zadáme IP adresu, které chceme data poslat, a port, na kterém bude přenos probíhat. Poté získáme datový proud metodou GetStream již zmiňované třídy TcpClient a ten použijeme v konstruktoru třídy System.IO.StreamWriter. Třída StreamWriter do datového proudu zapíše naši zprávu pomocí metody Write a nakonec spojení uzavře metodou Close.

```
public void PosliZpravu(string zprava,int port, string ip)
{
  TcpClient client = new TcpClient(ip, port);
  // Vytvoříme datový proud ke vzdálenému cíli,
  System.IO.Stream stream = client.GetStream();
  System.IO.StreamWriter wr = new System.IO.StreamWriter(stream);
  // do kterého zapíšeme třídou StreamWriter naši zprávu.
  wr.Write(zprava);
  // Nakonec datový proud uzavřeme.
  wr.Close();
}
```

748. Jak přijmout textové zprávy pomocí protokolu TCP

K tomu, abychom mohli přijmout textovou zprávu prostřednictvím protokolu TCP, použijeme třídu TcpListener. Tato metoda očekává v konstruktoru dva parametry: prvním z nich je IP adresa představovaná instancí třídy IPAddress, druhým pak port, na kterém bude TcpListener naslouchat. Ke spuštění naslouchání slouží metoda Start, která naslouchá do té doby, dokud se z druhé strany nepřipojí odesilatel reprezentovaný instancí třídy TcpClient.

Toho přijmeme metodou AcceptTcpClient a převezmeme jeho datový proud, který nese zprávu, metodou GetStream. Zprávu ukrytou v datovém proudu přečteme metodou ReadToEnd třídy System.IO.StreamReader a vrátíme ji příkazem return.

```
public string Naslouchej(int port)
{
   TcpListener listener = new TcpListener(IPAddress.Loopback, port);
   TcpClient client = null;
   listener.Start();
   client = listener.AcceptTcpClient();
   System.IO.Stream st = client.GetStream();
   System.IO.StreamReader reader = new System.IO.StreamReader(st);
   string přijatáZpráva = reader.ReadToEnd();
   st.Close();
   return přijatáZpráva;
}
```

749. Posílání souboru pomocí protokolu TCP

V této metodě nejprve vytvoříme instanci třídy IPAddress. Ta bude obsahovat IP adresu cílového počítače, kterou použijeme v konstruktoru třídy IPEndPoint. Tato třída představuje tzv. „koncový bod“, tzn. že nám říká, kam data námi posílaného souboru směřují. Poté vytvoříme instanci třídy Socket, připojíme ji metodou Connect k cílovému počítači a pošleme mu vybraný soubor metodou SendFile třídy TcpClient. Je-li soubor poslán, oba sockety (jak klientský, tak serverový) vypneme metodou Shutdown a nakonec Socket uzavřeme metodou Close.

```
public void PošliSoubor(string ip, int port, string cesta)
{
  IPAddress ipAddr = IPAddress.Parse(ip);
  IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, port);
  Socket client = new Socket(AddressFamily.InterNetwork,
  SocketType.Stream, ProtocolType.Tcp);
  client.Connect(ipEndPoint);
  client.SendFile(cesta);
  client.Shutdown(SocketShutdown.Both);
  client.Close();
}
```

750. Tři způsoby k uzavírání socketových spojení

Metoda Shutdown třídy Socket bezpečně vypne posílající a přijímající socketová spojení. Mějme na paměti, že bychom měli vždy před uzavřením socketu metodou Close zavolat metodu Shutdown, která zabezpečí, že všechna data byla již přijata. Této metodě můžeme zadat tři různé parametry podle toho, jak chceme socket uzavřít.

Následující tabulka ukazuje hodnoty výčtu SocketShutdown pro tuto metodu.

Send	Vypne posílající socket.
Receive	Vypne přijímající socket.
Both	Vypne oba sockety zároveň.

Tabulka 37: Popis položek výčtového typu SocketShutdown určující způsob uzavření socketových spojení

751. Jak poslat e-mailovou zprávu pomocí třídy SmtpClient

Protokol SMTP (Simple Mail Transfer Protocol) je protokol založený na protokolu TCP. Jeho úkolem je přenášet poštu mezi jednotlivými počítači. Prostředí .NET obsahuje pro práci s tímto protokolem třídu System.Net.Mail.SmtpClient.

Chceme-li poslat e-mailovou zprávu, použijeme třídu System.Net.Mail.SmtpClient, jejíž metodou Send odešleme zprávu. Předtím je však důležité nastavit v konstruktoru název hostitelského serveru. Je samozřejmé, že každý server má tento název jiný, u seznam.cz je jím mx1.seznam.cz, u jahoo.com je to nomail.yahoo.com atd. Jako argumenty metody Send použijeme adresu odesilatele, adresu příjemce, záhlaví a tělo zprávy.

```
string odKoho = "odesilatel@domena.cz";
string komu = "prijemce@domena.cz";
string zahlavi = "Záhlaví zprávy";
string zprava = "Tělo zprávy";
SmtpClient cli = new SmtpClient(hostitel);
cli.Send(odKoho, komu, zahlavi,zprava);
```

752. Poslání několika e-mailových zpráv

Tato metoda pošle několik e-mailových zpráv různým příjemcům, jejichž adresy a názvy hostitelských serverů jsou uloženy ve dvourozměrném poli. V prvcích pole s druhým indexem rovným 0 jsou uloženy adresy a v prvcích s druhým indexem rovným 1 jsou jména hostitelských serverů. O poslání e-mailové zprávy se stará metoda Send třídy System.Net.SmtpClient. Jako argumenty této metody použijeme adresu odesilatele, adresu příjemce, záhlaví a tělo zprávy.

```
 void PosliMaily(string[,] a, string odKoho, string zahlavi, string zprava)
{
    for (int x = 0; x < a.GetUpperBound(0); x++)
    {
        SmtpClient cli = new SmtpClient(a[x,1]);
        cli.Send(odKoho, a[x,0], zahlavi, zprava);
    }
}
```

753. Jak poslat e-mailovou zprávu s přílohou

Příloha e-mailové zprávy je v prostředí .NET vyjádřena instancí třídy Attachment, která jako parametr ve svém konstruktoru očekává cestu k souboru, který chceme poslat. Ke zprávě ji přiložíme metodou Attachments.Add. Zprávu poté pošleme funkcí Send třídy SmtpClient.

```
void PošliSoubor(string od,string komu,string zprava,string server, string
soubor, string predmet)
{
    SmtpClient cli = new SmtpClient(server);
    // Vytvoříme novou instanci třídy Attachment, která představuje
    // přílohu zprávy.
    Attachment příloha = new Attachment(soubor);
```

```
    // Samotnou e-mailovou zprávu představuje třída MailMessage, do
    // jejíhož konstruktoru jsme zadali údaje - od koho je e-mail
    // poslán,komu je určen, předmět zprávy a obsah.
    MailMessage mess = new MailMessage(od,př,předmět,zpráva);
    // Ke zprávě připojíme přílohu metodou Add.
    mess.Attachments.Add(příloha);
    // Nakonec zprávu pošleme.
    cli.Send(mess);
}
```

754. Jak poslat několik souborů v jedné e-mailové zprávě

Pokud chceme v jedné e-mailové zprávě poslat několik souborů, použijeme metodu Attachments. Add třídy System.Net.Mail.MailMessage, která do vnitřní kolekce této třídy přidá objekt typu System.Net.Mail.Attachment. Třída System.Net.Mail.Attachment představuje přílohu zprávy. Soubory můžeme přidávat postupným přidáváním, kdy na každém řádku přidáme jeden soubor,

```
MailMessage mail = new MailMessage();
mail.Attachments.Add(new Attachment("C:\A.txt"));
mail.Attachments.Add(new Attachment("C:\B.jpg"));
mail.Attachments.Add(new Attachment("C:\C.zip"));
.
.
```

Cesty k souborům můžeme také mít uložené v poli:

```
foreach(string filePath in pole)
{
    mail.Attachments.Add(new Attachment(filePath));
}
```

755. Odstranění jedné nebo několika příloh z e-mailové zprávy

Jak jsem již uvedl v tipu 445, e-mailovou zprávu představuje třída System.Net.Mail.MailMessage. K této e-mailové zprávě můžeme připojit přílohu metodou Attachments.Add. Co však udělat, když budeme chtít ze zprávy odstranit některou z příloh?

K odstranění přílohy můžeme použít metodu Attachments.Remove a jako argument použijeme právě tu přílohu, kterou chceme ze zprávy odstranit. K odstranění přílohy z kolekce příloh v instanci třídy MailMessage na určité pozici použijeme metodu RemoveAt a jako argument nám poslouží integer, který udává v kolekci příloh index přílohy, kterou chceme odstranit. K odstranění všech příloh ze zprávy použijeme metodu Attachments.Clear.

756. Nastavení priority e-mailových zpráv

Jestliže chceme nastavit prioritu odesílané e-mailové zprávy, použijeme vlastnost Priority třídy System.Net.Mail.MailMessage, která očekává parametr výčtového typu třídy System.Net.MailPriority. Tato třída nabízí tři možné hodnoty priorit: Low (nízká priorita), Medium (střední, implicitně nastavená priorita) a High (vysoká).

```
MailMessage zpráva = new MailMessage(odKoho,příjemce,předmět,zpráva);
// Nastavili jsme vysokou prioritu zprávy.
zpráva.Priority = MailPriority.High;
```

757. Jak poslat e-mailovou zprávu asynchronně

Metoda SendAsync třídy System.Net.Mail.SmtpClient posílá e-mailovou zprávu asynchronním způsobem, což znamená, že metoda, ve které je umístěn kód, nečeká na odeslání zprávy, ale ihned pokračuje dále. Chceme-li zrušit odesílání zprávy, použijeme metodu SendAsyncCancel.

```
SmtpClient client;

void SendAsync(string odKoho, string komu, string záhlaví, string zpráva, string
hostitel)
{
   MailMessage zprava = new MailMessage(odKoho, komu, záhlaví, zpráva);
   client = new SmtpClient(hostitel);
   client.SendAsync(zprava, null);
}
void Cancel()
{
   // Jestliže posílání zprávy ještě nezačalo, vyvoláme výjimku.
   if (client == null)
   {
      throw new Exception("Zpráva ještě nezačala být odesílána");
   }
   // Metoda SendAsyncCancel třídy SmtpClient zruší asynchronní
   // odesílání.
   client.SendAsyncCancel();
}
```

758. Zjištění hostitelského jména pomocí IP adresy

Přetížená metoda GetHostByAddress třídy Dns očekává jeden argument, který představuje IP adresu vzdáleného cíle, a vrací jeho hostitelské jméno uložené ve vlastnosti HostName.

```
string ip = System.Net.Dns.GetHostByAddress(IP).HostName;
```

759. Zjištění hostitelské IP adresy

```
string hostName = Dns.GetHostByName("http://google.com ").HostName;
string nazev = Dns.Resolve(hostName).AddressList[0].ToString();
```

Zadáme-li jako argument metody třeba „http://google.com“ – metoda vrátí IP adresu této webové stránky.

760. Zjištění názvu lokálního počítače

Následující metoda je určena pro zjištění názvu lokálního počítače. K tomu nám poslouží jednoduchá funkce GetHostName třídy System.Net.Dns, která vrací název lokálního počítače jako řetězec znaků.

```
string uzivatelovoJmeno = Dns.GetHostName();
```

761. Zjištění IP adresy lokálního počítače

Následující kód vrací řetězec obsahující IP adresu lokálního počítače.

```
string ip = Dns.Resolve(Dns.GetHostName()).AddressList[0].ToString();
```

762. Stažení souboru pomocí třídy WebClient

Chceme-li stáhnout soubor z nějakého vzdáleného zdroje, použijeme třídu System.Net.WebClient a její metodu DownloadFile, která přijímá dva argumenty: adresu souboru, který chceme stáhnout, a cestu, kam jej chceme uložit.

```
WebClient w = new WebClient();
w.DownloadFile(adresa, cestaSouboru);
```

763. Stáhování souboru jako pole bajtů

Někdy potřebujeme stáhnout soubor rovnou jako pole bajtů, abychom s ním mohli dále manipulovat, aniž bychom jej museli někam ukládat. K tomuto účelu nám poslouží metoda DownloadData třídy WebClient, která vrátí stažený soubor jako pole bajtů.

```
WebClient w = new WebClient();
byte[] bajtySouboru = w.DownloadData(adresa);
```

764. Zjištění velikost staženého souboru

Jestliže potřebujeme stáhnout soubor z nějaké url a zároveň vrátit jeho velikost v bajtech, musíme nejprve vytvořit třídu System.Net.WebClient a poté stáhnout soubor metodou DownloadFile, kde první parametr je cesta ke stahovanému souboru a druhým argumentem je cesta, kam tento soubor chceme uložit. Zjištění velikosti v bajtech provedeme pomocí vlastnosti Lenght třídy System.IO.FileInfo.

```
int ZiskejVelikost(string adresa, string kamUložit)
{
   WebClient w = new WebClient();
   w.DownloadFile(adresa, kamUložit);
   System.IO.FileInfo fi = new System.IO.FileInfo(kamUložit);
   return fi.Lenght;
}
```

765. Jak stáhnout soubor a vrátit jeho obsah jako pole bajtů

Ke stažení souboru použijeme metodu DownloadFile třídy System.Net.WebClient. Tato metoda očekává dva argumenty: url souboru, který chceme stáhnout, a cestu, kam jej chceme uložit. Poté metodou ReadAllBytes třídy System.IO.File získáme pole bajtů daného souboru a vrátíme jej příkazem return.

```
byte[] Stahni(string adresa, string cestaSouboru)
{
   // Vytvoříme novou instanci třídy WebClient.
   WebClient w = new WebClient();
   // Metoda DownloadFile stáhne soubour z adresy dané prvním
   // argumentem této metody do složky na disku, jejíž
   // umístění představuje druhý argument.
   w.DowloadFile(adresa, cestaSouboru);
   // Metoda ReadAllBytes přečte všechny bajty souboru, které uloží do
   // souboru a vrátí je příkazem return.
   return System.IO.File.ReadAllBytes(cestaSouboru);
}
```

766. Jak stahovat data asynchronně třídou WebClient

Pokud metody DownloadFile a DownloadData používáme ve formulářích Windows, můžeme si všimnout, že během stahování formulář „zamrzne", což znamená, že s ním nejde nijak manipulovat do té doby, dokud se daný soubor nestáhne. Abychom mohli stahovat soubor a přitom volně pracovat s formulářem, můžeme použít metody pro asynchronní stahování DownloadFileAsync a DownloadDataAsync. Metoda DownloadFileAsync očekává dva argumenty. Prvním argumentem je třída System.Net.Uri, která představuje adresu souboru, jejž chceme stáhnout, druhým argumentem je cesta, kam chceme soubor uložit. Metoda DownloadDataAsync očekává jako argument pouze adresu souboru, který chceme vrátit jako pole bajtů.

767. Událost reagující na dokončení stahování

Když třída WebClient dokončí stahování souboru, vyvolá událost DownloadFileCompleted, která nám oznamuje dokončení stahování.

```
void Stahuj(string adresaSouboru, string cestaSouboru)
{
   WebClient w = new WebClient();
   // Objektu WebClient přiřadíme událost DownloadFileCompleted
   w.DownloadFileCompleted += new EventHandler(Stahnuto);
   w.DownloadFile(adresaSouboru, cestaSouboru);
}

void Stahnuto(object sender, AsyncCompletedEventArgs e)
{
   // Kód oznamující dokončení stahování.
}
```

768. Jak průběžně zjišťovat velikost stáhnutých dat

Jestliže potřebujeme průběžně zjišťovat velikost stáhnutých dat během stahování a zobrazit je v nějakém ovládacím prvku, např. v komponentě ProgressBar, použijeme následující metodu. Nejprve vytvoříme metodu, která začne stahovat určitý soubor, a souběžně se stahováním spustíme časovač (timer) s nastaveným intervalem, který bude průběžně zjišťovat velikost již stažených dat a zobrazovat ji. Po dokončení stahování se automaticky zavolá událost DownloadFileCompleted třídy WebClient, která oznamuje ukončení stahování.

```
Timer timer = new Timer();
string filePath;
WebClient c;
void ZacniStahovat(string adresa, string cestaSouboru, int interval)
{
     // Vytvoříme novou instanci třídy WebClient.
     c = new WebClient();
     // Objektu timer přiřadíme událost Tick.
     timer.Tick += new EventHandler(timer_Tick);
     // U objektu c nastavíme událost pro dokončení stahování.
     c.DownloadFileCompleted +=
     new  AsyncCompletedEventHandler(c_DownloadFileCompleted);
     // Nastavíme interval časovače.
     timer.Interval = interval;
     filePath = cestaSouboru;
     progressBar1.Maximum = (int)ZískejVelikostSouboru(adresa);
     Uri u = new Uri(adresa);
```

```
     // Spustíme časovač...
     timer.Start();
     // ...a začneme asynchronně stahovat.
     c.DownloadFileAsync(u, cestaSouboru);
}
void timer_Tick(object sender, EventArgs e)
{
     // Při každém "tiknutí" časovače přečteme velikost souboru.
     System.IO.FileInfo i = new System.IO.FileInfo(filePath);
     // Kterou následně zobrazíme v progressbaru.
     progressBar1.Value = (int)i.Length;
}
void c_DownloadFileCompleted(object sender, AsyncCompletedEventArgs e)
{
     // Po dokončení stahování časovač zastavíme a zobrazíme zprávu
     // oznamující jeho dokončení.
     timer.Stop();
     MessageBox.Show("Stahování dokončeno");
}
long ZískejVelikostSouboru(string adresa)
{
     Uri u = new Uri(adresa);
     WebRequest req = (WebRequest)WebRequest.Create(u);
     WebResponse webres = req.GetResponse();
     return webres.ContentLength;;
}
```

769. Průběžně zjišťujeme rychlost stahování

Jestliže chceme zjistit rychlost, se kterou data pomocí třídy System.Net.WebClient stahujeme, poslouží nám tento vzorec:

velikost stažené části souboru / počet uplynulých sekund = přenosová rychlost

V praxi to může vypadat tak, že budeme mít jednu metodu, která asynchronně stahuje soubor, a jeden časovač, který se každou sekundu dotáže na prozatímní velikost stahovaného souboru, poté vydělí počet kilobytů, které jsme zatím stáhli, počtem uplynulých sekund a vypíše výsledek.

```
int sec = 0;
string filePath = "nějaká url";
void timer_Tick(object sender, EventArgs e)
{
 sec++;
 System.IO.FileInfo i = new System.IO.FileInfo(filePath);
 int přenosováRychlost = (i.Length / sec) /1024;
 label1.Text = "Přenosová rychlost je " + přenosováRychlost.ToString() +" kb/s";
}
```

770. Jak přerušit probíhající asynchronní stahování

V případě, že stahujeme nějaký soubor pomocí třídy WebClient a z nějakého důvodu chceme toto stahování přerušit, můžeme použít následující dvě metody. První metoda stahuje soubor metodou DownloadFileAsync a druhá toto stahování přeruší.

```
using System.Threading;

Thread th  = new Thread(new ThreadStart(Stahuj));
```

```
void static Stahuj(string adresa, string filePath)
{
  WebClient client = new WebClient();
  client.DownloadFileAsync(new Uri(adresa), filePath);
}

void ZrusStahovani()
{
  // Metoda Abort třídy Thread přeruší podproces, který má na
  // starosti stahování.
   th.Abort();
}
```

Stahování započneme spuštěním podprocesu, tj. metodou Start třídy Thread:

```
void ZapocniStahovani()
{
   th.Start();
}
```

771. Zjištění velikosti souboru, který je umístěn na webu

Chceme-li zjistit velikost určitého souboru na webovém serveru, můžeme použít tuto metodu, která vrací velikost souboru v bajtech. Nejprve vytvoříme požadavek na vzdálený cíl pomocí třídy WebRequest a vyčkáme na odpověď, kterou zajišťuje třída WebResponse. K zjištění velikosti souboru v bajtech nám poslouží právě její vlastnost ContentLenght.

```
long ZískejVelikostSouboru(string adresa)
{
  Uri u = new Uri(adresa);
  WebRequest req = (WebRequest)WebRequest.Create(u);
  WebResponse webres = req.GetResponse();
  return webres.ContentLength;
}
```

Pokud soubor na vzdáleném cíli neexistuje, metoda vrátí -1.

772. Zjištění typu souboru umístěného na webu

Ke zjištění typu souboru (zda se jedná o obrázek, text apod.) použijeme vlastnost ContentType třídy System.Net.WebResponse. Jako argument metody použijeme adresu souboru, jehož typ chceme zjistit.

```
string ZjistiTypSouboru(string adresaSouboru)
{
  // Vytvoříme požadavek na určitou webovou stránku, jejíž URL je
  // argumentem metody.
  WebRequest req = (WebRequest)WebRequest.Create(adresaSouboru);
  // Získáme odpověď vzdáleného cíle.
  WebResponse webres = req.GetResponse();
  // Vlastnost ContentType nám říká, o jaký typ souboru, ke kterému jsme
  // měli požadavek, se jedná.
  return webres.ContentType;
}
```

Jestliže jako parametr této metody použijeme cestu k nějakému obrázku ve formátu gif (který je umístěn třeba na nějaké webové stránce), metoda vrátí „image/gif“, u HTML-stránky „text/html“ atd.

773. Stahujeme zdrojový kód stránky pomocí datového proudu

Ke stáhnutí zdrojového kódu stránky použijeme webový dotaz, který je představován třídou System.Net.WebRequest. Po zaslání dotazu obdržíme od serveru odpověď, která je představována třídou System.Net.WebResponse. Tato třída nám předá datový proud metodou GetResponseStream; jeho obsah přečteme metodou ReadToEnd třídy System.IO.StreamReader.

```
 string StáhniZdroj(string adresaStránky)
{
  // Vytvoříme požadavek na určitou webovou stránku.
  WebRequest req = WebRequest.Create(adresaStránky);
  // Zašleme požadavek a vyčkáme na odpověď vzdáleného zdroje.
  WebResponse resp = req.GetResponse();
  // Třída Stream nám poslouží jako datový proud ke vzdálenému cíli.
  System.IO.Stream s = resp.GetResponseStream();
  // Jako argument konstruktoru třídy StreamReader použijeme již
  // zmiňovaný datový proud
  System.IO.StreamReader sr = new System.IO.StreamReader(s);
  // Třídou StreamReader přečteme obsah stránky a vrátíme jej příkazem
  // return.
   return sr.ReadToEnd();
}
```

774. Jak stáhnout soubor pomocí datového proudu

K tomu, abychom mohli stáhnout nějaký soubor, který je uložen na vzdáleném serveru, musíme nejprve zjistit jeho velikost, abychom mohli alokovat dostatečné velké bytové pole, do kterého se byty souboru později uloží. Poté metodou GetResponseStream získáme datový proud ke vzdálenému cíli, třídou BinaryReader přečteme všechny bajty souboru a uložíme je do pole. Poté toto pole předáme metodě WriteAllBytes třídy System.IO.File. Tato metoda vytvoří nový soubor a pole zapíše do něj.

```
void StahniSoubor(string url, string filePath)
{
  Uri u = new Uri(url);
  WebRequest req = (WebRequest)WebRequest.Create(u);
  WebResponse webres = req.GetResponse();
  long velikostSouboru = webres.ContentLength;
  System.IO.Stream s = webres.GetResponseStream();
  System.IO.BinaryReader b = new System.IO.BinaryReader(s);
  byte[] p = b.ReadBytes((int)velikost);
  System.IO.File.WriteAllBytes(filePath, p);
}
```

775. Zjišťujeme, zda odpověď serveru byla z cache

Vlastnost IsFromCache třídy System.Net.WebResponse zjišťuje, zda soubor, na který se dotazujeme, je na serveru uložen v cachi. Pokud ano, metoda vrací true, jinak vrátí false.

```
bool JeZCache(string url)
{
  Uri u = new Uri(url);
  WebRequest req = (WebRequest)WebRequest.Create(u);
  WebResponse webres = req.GetResponse();
  return webres.IsFromCache;
}
```

776. Zjišťujeme datum poslední úpravy webové stránky

Chceme-li zjistit datum poslední úpravy určité webové stránky, použijeme vlastnost LastModified třídy HttpWebResponse. Tato třída představuje odpověď, která nám bude po zadání dotazu metodou GetResponse třídy HttpWebRequest vrácena serverem.

```
string ZjistiPosledníUpravu (string url)
{
  HttpWebRequest req = (HttpWebRequest)WebRequest.Create(url);
  HttpWebResponse res = (HttpWebResponse)req.GetResponse();
  return res.LastModified.ToString();
}
```

777. Získání názvu serveru

Následující metodou posíláme serveru pomocí třídy HttpWebRequest dotaz na jeho název. Odpověď serveru získáme pomocí vlastnosti Server třídy HttpWebResponse.

```
string ZjistiNázevServeru (string url)
{
  HttpWebRequest req = (HttpWebRequest)WebRequest.Create(url);
  HttpWebResponse res =  (HttpWebResponse)req.GetResponse();
  // Příkaz return vrátí název serveru, třeba
  // "Apache/2.0.54 (Debian GNU/Linux)".
  return res.Server;
}
```

778. Přistupujeme k internetovým zdrojům přes proxy-server

Chceme-li vyřizovat naše požadavky na určitý server a jsme-li připojeni přes proxy-server, můžeme použít následující metodu. První argument metody je název proxyserveru, který použijeme v konstruktoru třídy WebProxy, druhým argumentem je webová adresa vzdáleného zdroje, jemuž chceme požadavek zaslat. Poté tuto nově vytvořenou třídu přiřadíme třídě HttpWebRequest pomocí vlastnosti Proxy, čímž jsme nastavili, že veškeré požadavky půjdou právě přes tento proxy-server.

```
void NastavProxyPřipojení(string názevProxy, string url)
{
   // Nejprve vytvoříme webový požadavek na nějakou url.
   HttpWebRequest req = (HttpWebRequest)WebRequest.Create(url);
   // Následuje vytvoření instance nové třídy WebProxy, do jejího
   // konstruktoru zadáme název proxy, přes kterou
   // chceme požadavky na server vyřizovat.
   WebProxy proxy = new WebProxy(názevProxy);
   // Instanci třídy WebProxy přiřadíme instanci třídy HttpWebRequest
   // pomocí její vlastnosti Proxy
   req.Proxy = proxy;
   // Nyní můžeme vyřizovat naše požadavky přes proxy-server.
   .
   .
   .
   .
}
```

Seznam proxy-serverů můžete najít na tzv. proxy-listech. Jeden poměrně obsáhlý proxy-list najdete na této webové adrese: http://samair.ru/proxy/