

## Протокол №1

# Тема:

**Основи на интернет. Мрежови протоколи. HTTP. – част 2**

Дата:

Изготвил:

## Теоретична част:

### IP адрес

IP адресът е уникален адрес, използван от мрежови устройства (обикновено компютри). Използва се за разпознаване на устройствата, когато те си комуникират. Сегашният стандартен протокол в мрежите е IP версия 4. Неговите адреси са с големина 32 бита. Обикновено се записват като четири осембитови числа, разделени с точка, например: 194.145.63.12 или 212.50.1.217.

Разработен е нов протокол – IP версия 6, който все още не е широко разпространен. Адресите от този протокол са с големина 128 бита. Обикновено адресите се записват като осем шестнайсетични числа в интервала 0-FFFF, например: 2001:0db8:85a3:08d3:1319:8a2e:0370:7334.

### Domain Name Service (DNS)

Компютрите в Интернет се разпознават чрез IP адреси, но тези числови идентификатори не са лесни за запомняне от човек. Повечето хора предпочитат да работят с имена. Ако искате да прочетете новините от страницата на вестник "Капитал", ще ви е по-лесно да се сетите за [www.capital.bg](http://www.capital.bg), вместо за адреса 193.194.140.15. Затова е създадена системата DNS (Domain Name Service), която служи за управление на съответствията между IP адреси и имена (наричани домейни). Тя може да преобразува имена в адреси и обратно.

### Порт

На един и същ компютър обикновено работят повече от едно приложения. В общия случай компютърът има само една физическа връзка към мрежата. Тази връзка може да бъде използвана за комуникация с повече от едно приложение. Използвайки 16-битово число, наричано порт, разграничаваме комуникационните канали на различните приложения един от друг. Изпращачът, изпращайки данни за даден компютър, подава и номер на порт.

### Основни мрежови услуги

Услуга	Порт	Описание
HTTP	80	Достъп до уеб сайтове, ресурси и услуги
SMTP	25	Изпращане на e-mail
POP3	110	Извличане на e-mail
FTP	21	Достъп до отдалечени файлове
DNS	53	Извличане на IP по име на сървър и обратното
SSH	22	Сигурен достъп до отдалечен терминал

## Мрежов интерфейс

Мрежовият интерфейс е абстрактна структура, чрез която операционната система управлява изпращането и приемането на информация по мрежата. Възможно е една машина да бъде свързана към няколко мрежи едновременно. Тогава към всяка мрежа машината има различен мрежов интерфейс. Всеки интерфейс има различен IP адрес, съответно и машината има повече от един IP адрес.

## Протоколът TCP

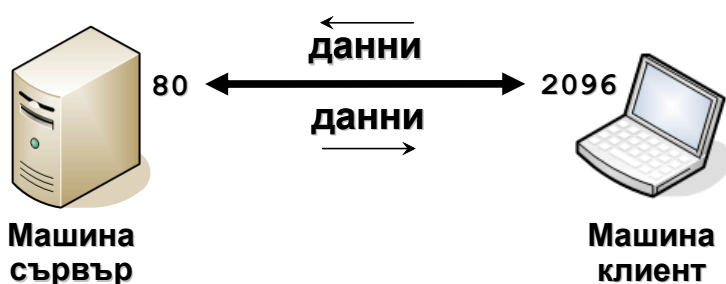
Протоколът TCP е един от най-широко разпространените протоколи за мрежова комуникация. Този протокол създава надежден двупосочен комуникационен канал за обмен на данни. Това гарантира, че изпратените данни ще пристигнат в същия ред, в който са изпратени. Ако данните не могат да се изпратят или получат, ще възникне грешка. Комуникационният канал съществува, докато някоя от двете страни не го прекрати. Комуникацията по протокола TCP се използва в приложения, в които редът на пристигане на данните и надеждността са важни.

## Протоколът UDP

Протоколът UDP позволява изпращане и приемане на малки независими един от друг пакети с данни, наречени **datagram пакети**. Не гарантира реда на пристигане на datagram пакетите, нито че те изобщо ще пристигнат. За сметка на това е по-бърз от протокола TCP. Използва се в приложения, в които скоростта на предаване на данните е по-важна от надеждността. Например, ако гледате видео материал по Интернет, няма да е от голямо значение, ако от време на време вместо една точка от екрана се появи звездичка. Но ще е от голямо значение, ако кадрите се забавят и се получава завличане на образа. В такива приложения е логично да се използва протоколът UDP.

## Как две отдалечени машини си "говорят"?

Ако искаме да осъществим връзка между два компютъра и да разменим определени данни, се нуждаем от приложение "клиент" и приложение "сървър":



Първо трябва да стартираме сървърното приложение, като го накараме да "слуша" на даден порт. Нека това е порт 80. Клиентското приложение се стартира на компютъра-клиент и се опитва да установи комуникационен канал, свързвайки се със сървърния компютър, като указва IP адреса и порта, към които иска да се свърже. За да е успешна комуникацията, е нужно клиентът да може да изпраща данни на сървъра, но и сървърът да може да изпраща на клиента. Когато сървърът изпраща данни на клиента, се нуждае не само от IP адрес, но и от порт. За целта или клиентът сам определя порта, или операционната система му задава такъв.

## Класове за мрежово програмиране в .NET

- Мрежовото програмиране на практика се състои в писане на код, който да управлява обмена на пакети данни по мрежата и да обработва получената информация. Класовете, които .NET Framework предлага за това, са разпределени в две основни именни пространства – **System.Net** и **System.Net.Sockets**.
- Чрез опростени класове като **TcpClient**, **TcpListener** и **UdpClient** лесно можем да реализираме комуникация съответно по TCP и UDP протокол.
- Освен тях, можем да използваме по-функционалния клас **Socket**, както и множеството помощни класове за програмиране на приложно ниво (application layer), чрез които да реализираме и да използваме съответните услуги (уеб-програмиране, DNS услуги, пощенски услуги и т.н.).

## Пространството System.Net.Sockets

Тук се намират споменатите по-горе основни класове за осъществяване на комуникация чрез сокети, както и няколко помощни класа, на които няма да се спираме подробно – класове за опции, за изключения и за мрежово програмиране при **мобилни устройства**.

- Класовете **TcpClient** и **TcpListener** служат за реализиране на връзка по TCP протокола. Първият клас се използва в клиентската част от приложението и чрез него се свързваме по TCP с отворен порт на отдалечена машина. Методите му позволяват връзка с определен сокет и приемане и изпращане на данни.
- **TcpListener** реализира сървърната част на връзката – чрез него "слушаме" на определен порт за идващи заявки връзки и установяваме връзка със съответния сокет.
- Класът **UdpClient** изпълнява задачата за осъществяване на комуникация по UDP протокола.
- Класът **Socket** реализира абстракцията на Berkeley Sockets API и е значително по-функционално обобщение на предните три класа. Чрез него можем да осъществим връзка по който и да е от протоколите на мрежово и по-ниски нива от OSI модела, например IP, IPv6, ICMP, IDP и други. Класът има методи както за слушане за връзки и установяване на връзка (connection), така и за изпращане и получаване на данни. Чрез класа **Socket** можем също да осъществяваме и асинхронно предаване на данни.
- Последният по-важен клас от това пространство е класът **NetworkStream** – специализация на обикновения клас за поток, който реализира специфичните за мрежов трансфер на данни особености.

## Пространството System.Net

Това пространство съдържа по-общ набор от класове, някои от които реализират услуги от приложно ниво, други са помощни класове, които използваме за удобство, трети служат за опции и т.н.

- Чрез класовете **HttpWebRequest** и **HttpWebResponse** можем да използваме HTTP услугите и да осъществяваме заявки с този протокол до различни уеб-ресурси. Чрез обработката на тези заявки и отговорите им можем лесно да построим прост вариант на обикновен уеб-браузър.

- Класът **Dns** и методите му ни дават достъп до DNS услугите за извличане на име на машина по IP адреса ѝ в мрежата и обратното.
- Класовете **IPAddress**, **IPHostEntry** и **IPEndPoint** служат за съхраняване на IP адреси. Първият представя един IP адрес, вторият е списък от съответни адреси и имена (по DNS), а третият е двойка от адрес и номер на порт.
- Класът **WebClient** е обобщен клас, чрез който можем да осъществяваме достъп до произволен ресурс чрез **URI (Uniform Resource Identifier)** във файловата система, интернет или локална мрежа.

### Въпроси и задачи:

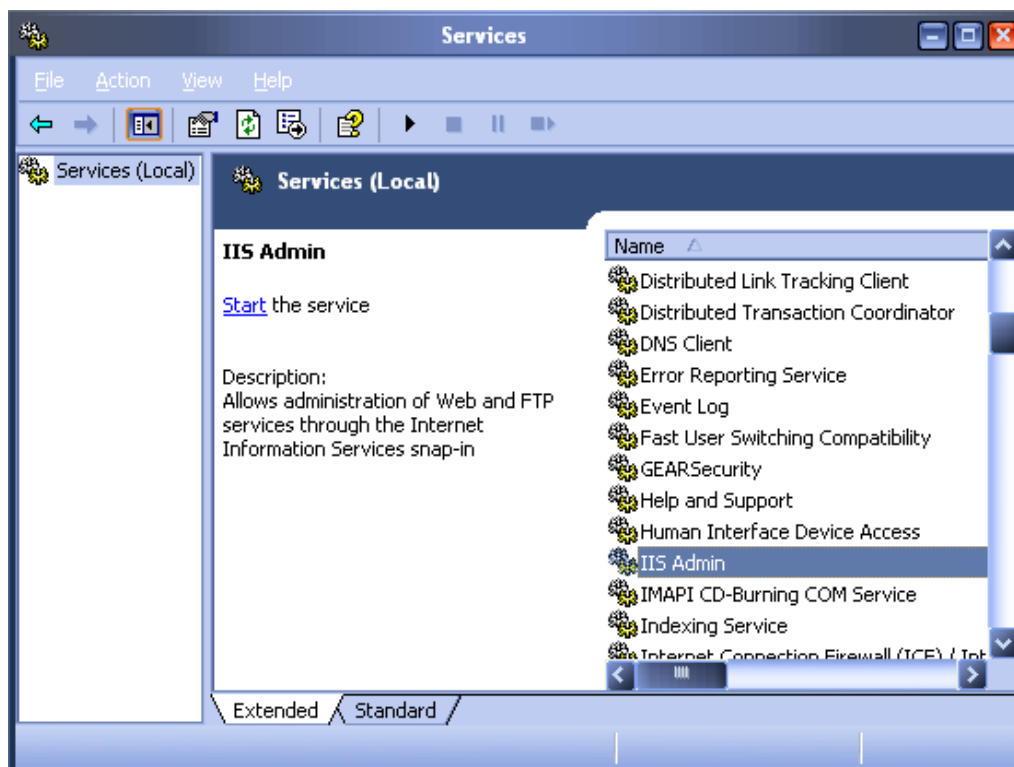
1. Обяснете понятията: IP адрес, DNS, порт, мрежов интерфейс, TCP, UDP и сокет връзка. Каква е разликата между протоколите TCP и UDP?
2. Опишете основните мрежови услуги в Интернет, какви протоколи използват и кои TCP портове.

### Задача 1:

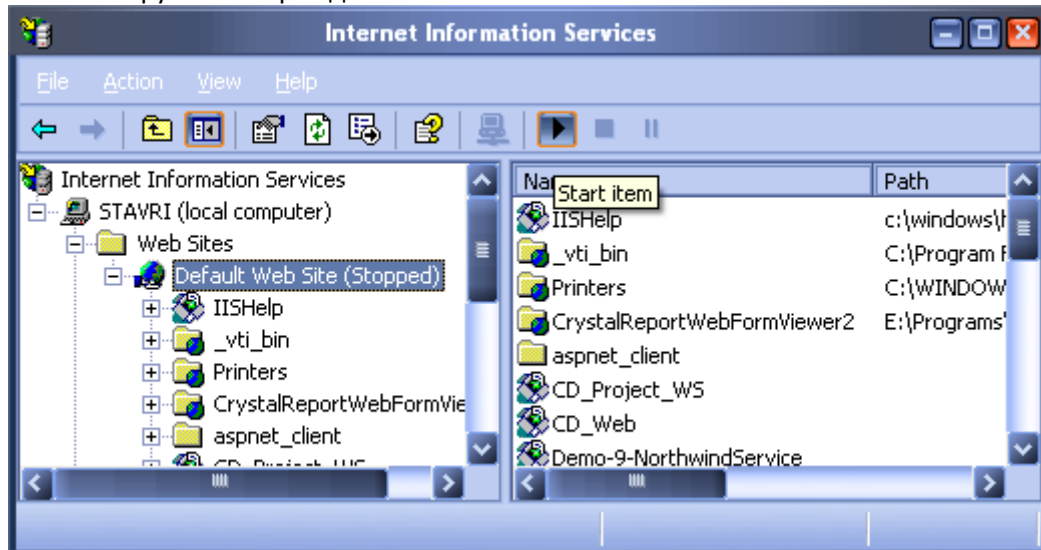
**Условие:** първо създаваме инстанция на класа **TcpClient**, а после се опитваме да се свържем с избрания хост на порт 80, където обикновено слуша HTTP сървър. След това извличаме потока, в който ще предаваме данни в отворения сокет. Чрез метода **Write(...)** ще изпратим една заявка **HTTP GET**, с която да поискаме съдържанието на заглавната страница на домейна, с който се сме свързали. После прочитаме отговора на сървъра като използваме метода **Read(...)** в цикъл докато резултатът от него стане 0 байта, което ще означава, че няма повече информация за четене. След това ще отпечатаме този отговор на екрана и ще завършим изпълнението на програмата.

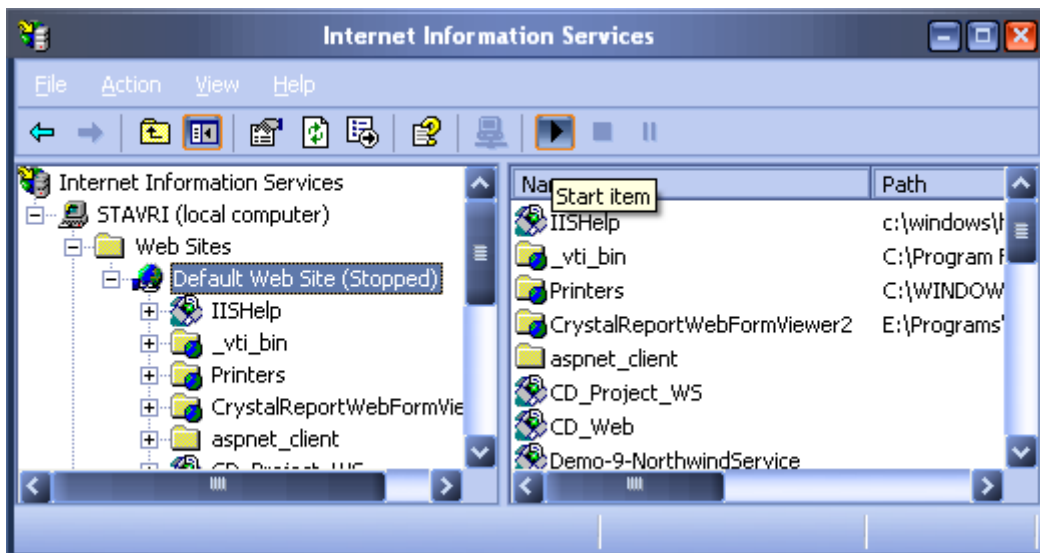
Необходимите стъпки за изграждането на примера са следните. Ако няма да използваме IIS, стъпките 1–3 може да се пропуснат.

1. Стартираме IIS, ако не е вече стартиран. Това може да се провери по следния начин – сървърът е стартиран, ако при поискването на адрес <http://localhost/> в браузъра се зарежда страницата на IIS. Ако това е така, можем да минем на стъпка 4. В противен случай изпълняваме стъпки 2 и 3 за стартиране на сървъра.
2. В Windows XP можем да стартираме услугата, която отговаря за IIS, като отидем в **Control Panel | Administrative Tools | Services**. Избираме услугата **IIS Admin** от списъка, след което избираме етикета **Extended** под списъка на услугите и щракваме върху препратката **Start the service**:



3. Сега отваряме **Control Panel | Administrative Tools | Internet Information Services** и чрез навигация в дървото отляво последователно отваряме **Local Computer | Web Sites | Default Web Site**. След това трябва да щракнем върху бутона **Start Item** върху лентата с инструменти горе вдясно:





1. При стартиран IIS, вече можем да създадем нов конзолен проект във Visual Studio .NET.
2. Въвеждаме кода на програмата. Обърнете внимание на употребата на класа **StreamWriter**, благодарение на който използваме метода **WriteLine(...)**:

```
using System;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Text;

class TcpClientDemo
{
    const int RECEIVE_BUF_SIZE = 4096;

    static void Main()
    {
        // Свържете се със сървъра
        TcpClient tcpClient = new TcpClient();

        try
        {
            tcpClient.Connect("localhost", 80);

            // Възможно е да се замени с "www.abv.bg"
```

```
    }

    catch (SocketException)
    {
        Console.WriteLine("Error: Unable to connect to the
                           server.");

        Environment.Exit(-1);
    }

    try
    {
        NetworkStream ns = tcpClient.GetStream();
        using (ns)
        {
            // Изпратете HTTP GET заявка до уеб сървъра
            try
            {
                StreamWriter writer = new StreamWriter(ns);
                writer.WriteLine("GET http://localhost/
HTTP/1.0");

                // Възможност "GET http://www.abv.bg/
HTTP/1.0"

                writer.WriteLine();
                writer.Flush();
            }
            catch (IOException)
            {
                Console.WriteLine("Error: Cannot send
request.");

                Environment.Exit(-1);
            }

            // Получаване на HTTP отговор от сървъра
```



```
        try
        {
            byte[] buf = new byte[RECEIVE_BUF_SIZE];
            while (true)
            {
                int bytesRead = ns.Read(buf, 0,
buf.Length);

                if (bytesRead == 0)
                {
                    // Сървърът прекъсна връзката
                    break;
                }

                string data =
Encoding.ASCII.GetString(buf, 0,
                bytesRead);

                Console.Write(data);
            }
        }
        catch (IOException)
        {
            Console.WriteLine("Error: Cannot read the
server                response.");

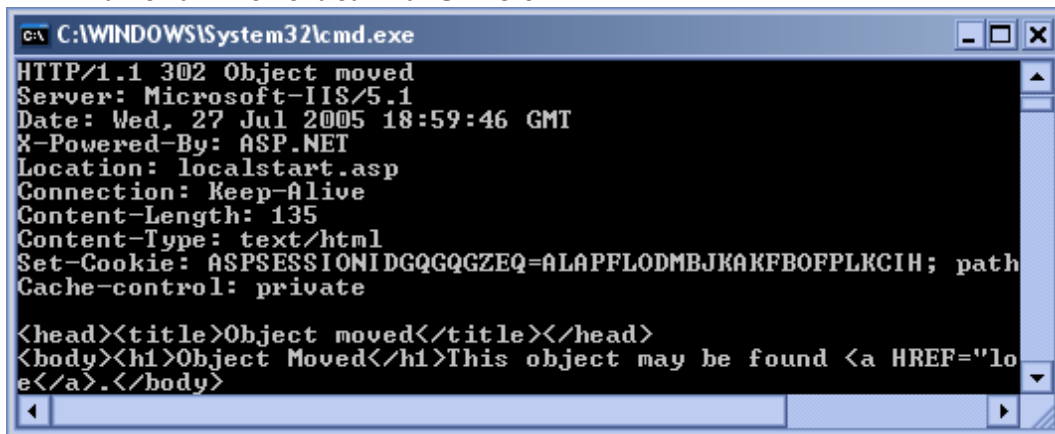
            Environment.Exit(-1);
        }
    }

    finally
    {
        // Затворете връзката със сървъра (ако все още е
отворен)

        tcpClient.Close();
    }
}
```

```
}  
  
}
```

1. Стартираме програмата. Очакваният изход е стандартният HTTP отговор, който трябва да съдържа тялото на поискания документ. Ако сме се свързали към IIS, отговорът ще ни казва, че обектът, който сме поискали, може да бъде намерен на адрес <http://localhost/localstart.asp>. Ако не нашето клиентско приложение, а някой истински браузър изпрати същата заявка, той ще бъде пренасочен именно към този адрес и ще изпрати автоматично нова заявка към него.



```
C:\WINDOWS\System32\cmd.exe  
HTTP/1.1 302 Object moved  
Server: Microsoft-IIS/5.1  
Date: Wed, 27 Jul 2005 18:59:46 GMT  
X-Powered-By: ASP.NET  
Location: localstart.asp  
Connection: Keep-Alive  
Content-Length: 135  
Content-Type: text/html  
Set-Cookie: ASPSESSIONIDGQGQZQE=ALAPFLODMBJKAKFBOFPLKCIH; path=  
Cache-control: private  
  
<head><title>Object moved</title></head>  
<body><h1>Object Moved</h1>This object may be found <a HREF="lo  
e</a>.</body>
```

1. Ако сме използвали IIS, можем да видим какво става, когато няма връзка към посочения сървър. Нека спрем IIS подобно на описания в т. 2 и т. 3 начин, но със **Stop...** вместо със **Start...**, и отново стартираме програмата, ще получим също очакван резултат: **"Error: Unable to connect to the server."**, защото порт 80 на локалната машина е затворен и методът **Connect (...)** предизвиква изключение.

## Задача 2:

**Условие** - Напишете приложение, което изпълнява DNS заявки (преобразува от име на машина към IP адрес).

Код:

```
var encodedUrl = Console.ReadLine();  
var decodeUrl = WebUtility.UrlDecode(encodedUrl);  
  
Console.WriteLine(decodeUrl);
```