МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

***Факультет информационных технологий и робототехники***

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**Отчет**

**по лабораторной работе № 8**

по дисциплине: ”Компьютерные системы и сети ”

на тему: ***”***ПРОТОКОЛ HTTP***”***

Выполнил**:** студент группы *10701118*

Воробей И.А

Приняла**:** Белова С.В.

Минск 2020

**ВАРИАНТ 9**

**Цель работы**: Изучить протокол HTTP. Изучить назначение и структуру URI-имен. Разработать HTTP-клиент и HTTP-сервер.

**Задание 1**

Написать консольное приложение, реализующее основные функции HTTP-сервера. Должен поддерживаться запрос GET. Сервер должен быть сконфигурирован на определенный каталог, где расположены html-файлы. На экране отображать весь протокол общения HTTP клиента с HTTP сервером: все приходящие от клиента запросы и все отправляемые ему ответы. Тестирование и демонстрацию программы-сервера производить при помощи Web браузера и своего Web-клиента.

Я написал сервер который отвечает на запросы клиентов по адресу <http://localhost:8000/Varabei_10701118/>

Браузер разбирает uri на три части: протокол, хост, адрес ресурса на веб сервере. Далее браузер должен превратить хост в IP адрес. Для этого используется DNS. Если ранее этот адрес был запрошен, то IP берется из кеша, иначе браузер обращается к DNS серверу.

Имея IP адрес, браузер пытается установить TCP соединение(с подтверждением получения информации).

Далее браузер формирует HTTP 1)заголовок(метод,resource location, version) ; 2)заголовки запроса; 3) тело Рис.2.

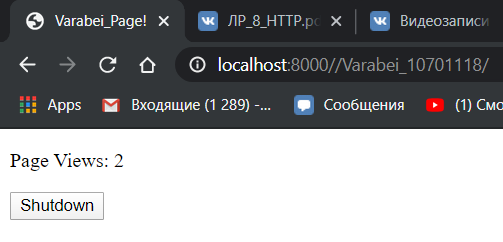
 В случае успеха клиент получает HTTP заголовок, содержащий 1)протокол; 2)код ответа; 3) сообщение. Потом заголовки HTTP ответа сервера и наконец HTML код который может быть отображен в браузере Рис.1.

Рис 1 – адресная строка и страница

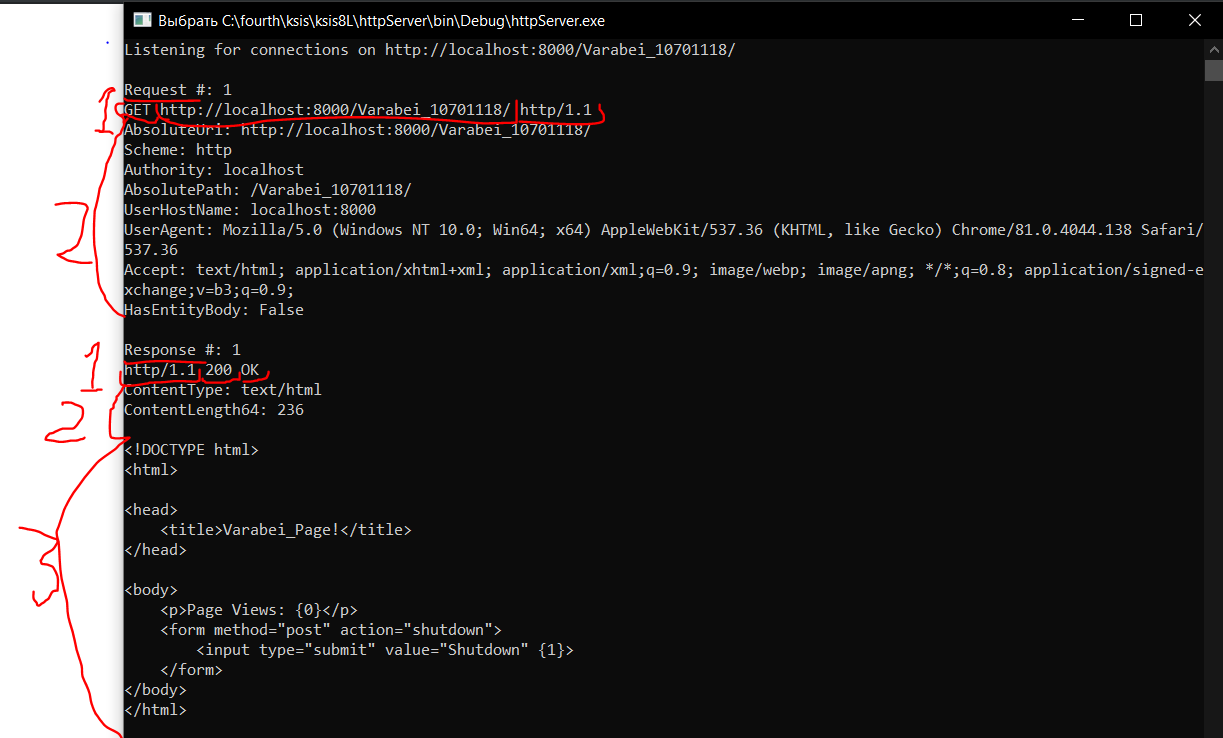


Рис. 2 – диалог сервера с браузером

using System;

using System.IO;

using System.Net;

using System.Text;

using System.Threading.Tasks;

namespace httpServer

{

class HttpServer

{

public static HttpListener listener;

public static string url = "http://localhost:8000/Varabei\_10701118/";

public static int pageViews = 0;

public static int requestCount = 0;

public static string pageData = ReadFile(@"C:\fourth\ksis\ksis8L\httpServer\page.html");

public static async Task HandleIncomingConnections()

{

bool runServer = true;

while (runServer)

{

HttpListenerContext ctx = await listener.GetContextAsync();

HttpListenerRequest req = ctx.Request;

HttpListenerResponse resp = ctx.Response;

OutRequestInfo(req);

string disableSubmit = !runServer ? "disabled" : "";

byte[] data = Encoding.UTF8.GetBytes(String.Format(pageData, ++pageViews, disableSubmit));

resp.ContentType = "text/html";

resp.ContentEncoding = Encoding.UTF8;

resp.ContentLength64 = data.LongLength;

OutResponseInfo(resp);

if (req.HttpMethod == "POST" && req.Url.AbsolutePath == "/Varabei\_10701118/shutdown")

{

Console.WriteLine("Shutdown requested");

runServer = false;

}

await resp.OutputStream.WriteAsync(data, 0, data.Length);

resp.Close();

}

}

static void OutRequestInfo(HttpListenerRequest req)

{

Console.WriteLine("Request #: {0}", ++requestCount);

Console.WriteLine(req.HttpMethod + " " + req.Url.AbsoluteUri

+ " " + req.Url.Scheme + "/" + req.ProtocolVersion);

Console.WriteLine("AbsoluteUri: " + req.Url.AbsoluteUri);

Console.WriteLine("Scheme: " + req.Url.Scheme);

Console.WriteLine("Authority: " + req.Url.Host);

Console.WriteLine("AbsolutePath: " + req.Url.AbsolutePath);

Console.WriteLine("UserHostName: " + req.UserHostName);

Console.WriteLine("UserAgent: " + req.UserAgent);

Console.WriteLine("Accept: " + ArrayToStr(req.AcceptTypes));

Console.WriteLine("HasEntityBody: " + req.HasEntityBody);

Console.WriteLine();

}

static void OutResponseInfo(HttpListenerResponse resp)

{

Console.WriteLine("Response #: {0}", requestCount);

Console.WriteLine("http/" + resp.ProtocolVersion + " " + resp.StatusCode + " " + resp.StatusDescription);

Console.WriteLine("ContentType: " + resp.ContentType);

Console.WriteLine("ContentLength64: " + resp.ContentLength64);

Console.WriteLine();

Console.WriteLine(pageData);

Console.WriteLine();

}

static string ArrayToStr(string[] ms)

{

string res = "";

for (int i = 0; i < ms.Length; i++)

{

res += ms[i] + "; ";

}

return res;

}

public static string ReadFile(String path)

{

string content = "";

try

{

using (StreamReader sr = new StreamReader(path))

{

content = sr.ReadToEnd();

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

return content;

}

public static void Main(string[] args)

{

// Create a Http server and start listening for incoming connections

listener = new HttpListener();

listener.Prefixes.Add(url);

listener.Start();

Console.WriteLine("Listening for connections on {0}\n", url);

// Handle requests

Task listenTask = HandleIncomingConnections();

listenTask.GetAwaiter().GetResult();

// Close the listener

listener.Close();

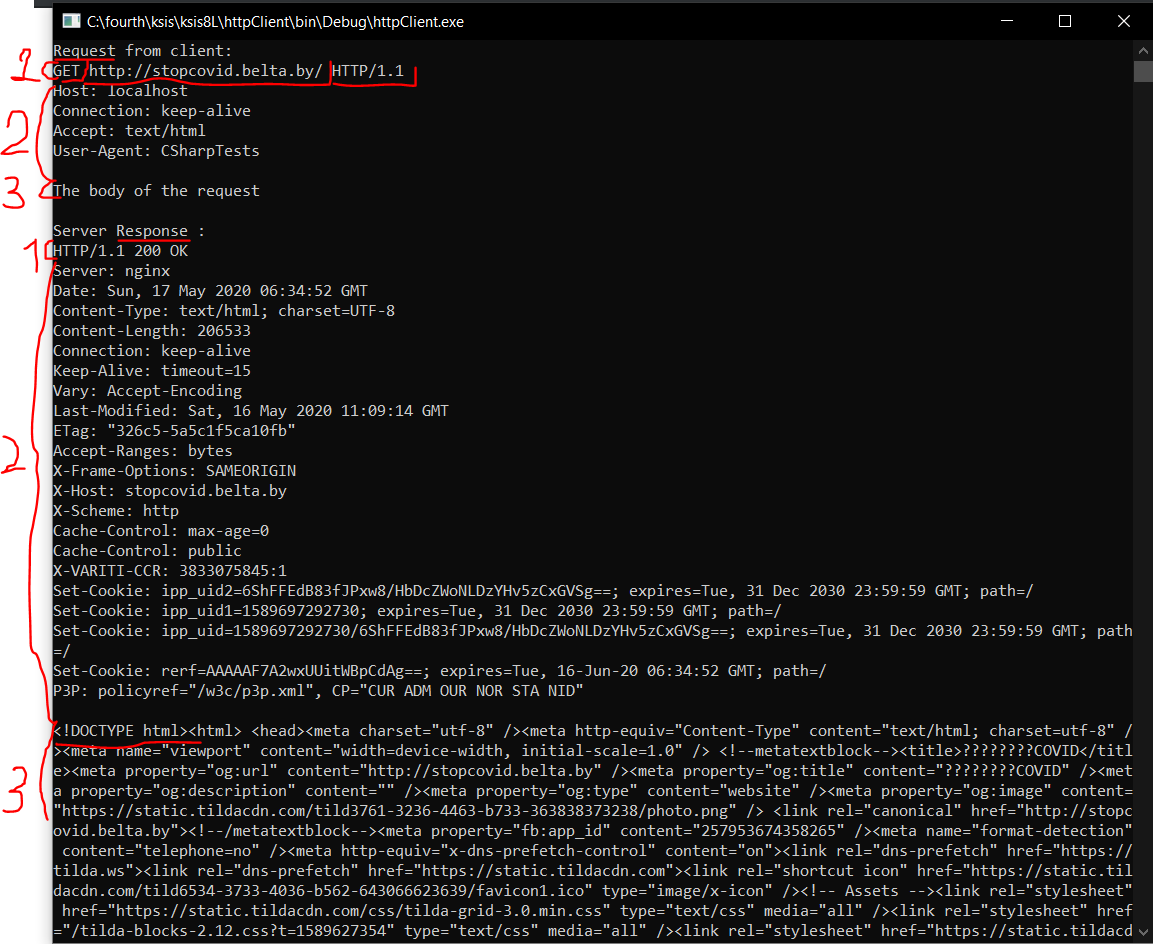
Console.Read();

}

}

}

**Задание 2**

Написать консольное приложение, реализующее функции простого HTTP-клиента. Получение Web-страницы от HTTP-сервера должно осуществляться по заданному URI адресу. В HTTP-запрос добавлять заголовок User-Agent и другие заголовки. Должен поддерживаться запрос GET, а также наиболее распространенные коды ответов. Отображать на экране весь протокол общения HTTP клиента с HTTP сервером. Полученные от сервера данные выводить в виде HTML-кода на экран или в файл. Отображение полученных данных в форматированном виде не обязательно. Тестирование и демонстрацию HTTP клиента производить при помощи запроса к реальному web-серверу, расположенному в Internet, установленному в локальной сети или при помощи запроса к web-серверу, написанному в предыдущем задании.

Я написал клиент и продемонстрировал общение клиента и удаленного сервера на рисунке 3. А так же общение «моего» клиента с «моим» сервером на рисунке 4.

Рис. 3 – общение клиента с удаленным сервером

Рис. 4 – общение «моих» клиента и сервера

using System;

using System.Text;

using System.Net.Sockets;

using System.Text.RegularExpressions;

namespace httpClient

{

class Program

{

static string targetIP;//= "185.165.123.36";

static string url;//= "http://stopcovid.belta.by/";

static int port;//= 80;

static void Main(string[] args)

{

initialize();

Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

socket.Connect(targetIP, port);

string GETrequest = "GET " + url + " HTTP/1.1\r\n" +

"Host: localhost\r\nConnection: keep-alive\r\nAccept: text/html\r\n" +

"User-Agent: CSharpTests\r\n\r\n";

socket.Send(Encoding.ASCII.GetBytes(GETrequest));

Console.WriteLine("Request from client:\n" + GETrequest);

bool flag = true; // just so we know we are still reading

string headerString = ""; // to store header information

byte[] bodyBuff = new byte[0]; // to later hold the body content

while (flag)

{

// read the header byte by byte, until \r\n\r\n

byte[] buffer = new byte[1];

socket.Receive(buffer, 0, 1, 0);

headerString += Encoding.ASCII.GetString(buffer);

if (headerString.Contains("\r\n\r\n"))

{

Regex reg = new Regex("\\\r\nContent-Length: (.\*?)\\\r\n");

Match m = reg.Match(headerString);

int contentLength = int.Parse(m.Groups[1].ToString());

flag = false;

// read the body

bodyBuff = new byte[contentLength];

socket.Receive(bodyBuff, 0, contentLength, 0);

}

}

Console.WriteLine("Server Response :\n" + headerString+ Encoding.ASCII.GetString(bodyBuff));

socket.Close();

Console.Read();

}

static void initialize()

{

Console.Write("Enter target IP:");

targetIP = Console.ReadLine();

Console.Write("Enter target url:");

url = Console.ReadLine();

Console.Write("Enter target port:");

port = int.Parse(Console.ReadLine());

Console.WriteLine();

}

}

}

**Контрольные вопросы**

1. На каком уровне стека протоколов работает HTTP?

Прикладной.

2. Для чего предназначен протокол HTTP?

Для передачи гипертекста .

3. Какой номер порта закреплен за HTTP по умолчанию?

80

https - 443

4. Какова структура запроса HTTP-клиента? Что такое метод? Перечислите основные методы.

<метод> <URI запрашиваемого\_ресурса> HTTP/1.1<\n>

<заголовочное\_поле>: <значение><\n>

<заголовочное\_поле>: <значение><\n>

[..заголовочных полей может быть много..]<\n>

<\n>

<тело содержимого>

Метод - это HTTP-команда, с которой начинается первая строка запроса клиента. Метод сообщает серверу о цели запроса. Для HTTP определены три основных метода: GET, HEAD и POST.

5. Какова структура ответа HTTP-сервера? Что такое код состояния? Перечислите наиболее распространенные коды состояний?

HTTP/1.1 <код\_ответа> <сообщение><\n>

<заголовочное\_поле>: <значение><\n>

<заголовочное\_поле>: <значение><\n>

[..заголовочных полей может быть много..]<\n>

<\n>

<тело\_документа>

Код состояния - это трехразрядное число, обозначающее результат обработки сервером запроса клиента. Описание, следующее за кодом состояния, представляет собой просто понятный для человека текст, поясняющий код состояния.

200 OK- запрос был получен и обработан

301 Ресурс перемещен постоянно

302 Ресурс перемещен временно

400 Неверный запрос – сообщение с запросом имеет некорректный формат

401 Несанкционированный доступ – у пользователя нет прав для доступа к

запрошенному документу

402 Ресурс доступен за плату

408 Тайм-аут запроса

500 Внутренняя ошибка сервера – ошибка помешала серверу обработать

запрос

6. Какие заголовочные поля могут содержаться в HTTP-запросе и HTTP-ответе?

Request from client:

GET http://stopcovid.belta.by/ HTTP/1.1

Host: localhost

Connection: keep-alive

Accept: text/html

User-Agent: CSharpTests

The body of the request

Server Response :

HTTP/1.1 200 OK

Server: nginx

Date: Sat, 16 May 2020 21:45:47 GMT

Content-Type: text/html; charset=UTF-8

Content-Length: 206533

Connection: keep-alive

Keep-Alive: timeout=15

Vary: Accept-Encoding

Last-Modified: Sat, 16 May 2020 11:09:14 GMT

ETag: "326c5-5a5c1f5ca10fb"

Accept-Ranges: bytes

X-Frame-Options: SAMEORIGIN

X-Host: stopcovid.belta.by

7. Какие классы .NET можно использовать для работы с протоколом HTTP?

HttpListener

Socket

TcpListener

8. Какие ресурсы можно адресовать с помощью URI? Приведите примеры широко распространенных схем, используемых в URI.

FTP, Gopher, WAIS, Usenet, E-mail, Prospero, Telnet, HTTP

9. Какова структура URI?

Схема(имя\_протокола)://имя\_польз:пароль@доменное\_имя (или IP-адрес): номер\_порта / путь/имя\_файла?запрос ( или #фрагмент)

10. Как осуществляется работа с URI адресами в .NET?

Класс Uri из пространства имен System.Net инкапсулирует универсальный идентификатор ресурсов. Он содержит свойства и методы для анализа, сравнения и комбинирования URI.

**Выводы** :

* Изучил структуру URI;
* Изучил структуру запроса к http серверу;
* Изучил структуру ответа сервера;
* Написал свой http client & http server .