СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 1](#_Toc514694437)

[1 ОБЗОР СОСТОЯНИЯ ВОПРОСА 2](#_Toc514694438)

[1.1 Протокол UDP 3](#_Toc514694439)

[1.2 Понятие сокета 3](#_Toc514694440)

[1.1 Типы сокетов 4](#_Toc514694441)

[2 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc514694442)

[3 МОДЕЛИРОВАНИЕ И РЕАЛИЗАЦИЯ ПО 5](#_Toc514694443)

[1.1 Архитектура ПО и протокол взаимодействия клиента и сервера 6](#_Toc514694444)

[1.2 Алгоритмы работы модулей 6](#_Toc514694445)

[3.3 Интерфейс пользователя 6](#_Toc514694446)

[3.4 Библиотечные классы и методы 6](#_Toc514694447)

[3.5 Разработанные классы и методы 7](#_Toc514694448)

[4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 8](#_Toc514694449)

[ЗАКЛЮЧЕНИЕ 9](#_Toc514694450)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 10](#_Toc514694451)

[ПРИЛОЖЕНИЕ A 11](#_Toc514694452)

[ПРИЛОЖЕНИЕ Б 20](#_Toc514694453)

# ВВЕДЕНИЕ

Цель курсового проекта – закрепление и углубление знаний, полученных при изучении курса “Компьютерные системы и сети” посредством разработки программного обеспечения на языке С#.

В теоретической части курсового проекта рассматриваются протокол UDP, а также основные понятия технологии сокетов и клиент-серверного взаимодействия.

В практической части курсового проекта осуществляется разработка программы по теме: “Сетевая игра”.

# 1 ОБЗОР СОСТОЯНИЯ ВОПРОСА

## 1.1 Протокол UDP

UDP (англ. User Datagram Protocol — протокол пользовательских датаграмм) — один из ключевых элементов TCP/IP, набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые датаграммами) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных. Протокол был разработан Дэвидом П. Ридом в 1980 году и официально определён в RFC 768.

UDP использует простую модель передачи, без неявных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных. Таким образом, UDP предоставляет ненадёжный сервис, и датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа. UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении. Чувствительные ко времени приложения часто используют UDP, так как предпочтительнее сбросить пакеты, чем ждать задержавшиеся пакеты, что может оказаться невозможным в системах реального времени. При необходимости исправления ошибок на сетевом уровне интерфейса приложение может задействовать TCP или SCTP, разработанные для этой цели.

Природа UDP как протокола без сохранения состояния также полезна для серверов, отвечающих на небольшие запросы от огромного числа клиентов, например DNS и потоковые мультимедийные приложения вроде IPTV, Voice over IP, протоколы туннелирования IP и многие онлайн-игры.

## 1.2 Понятие сокета

Технология сокетов лежит в основе современного сетевого программирования. Основные операционные среды (Unix или Windows) базируются в настоящее время на идеологии сокетов (socket). Эта технология была разработана в университете г. Беркли (США) для системы Unix, поэтому их иногда называют сокетами Беркли (berkeley sockets).

Сокеты представляют собой конечные точки связи, обращение к которым осуществляется при помощи соответствующих дескрипторов сокетов, описывающих связь сокета с определенной машиной или приложением. Соединение (или пара сокетов) состоит из пары IP-адресов, обладатели которых общаются друг с другом и пары номеров портов, где номер порта представляет собой 16-разрядное целое без знака, как правило, в десятичной системе счисления.

## Типы сокетов

Существуют три основных типа сокетов: потоковые, дейтаграммые и сырые.

Потоковые сокеты– это сокеты с установлением соединения, состоящие из потока байтов, который может быть двунаправленным. Т.е. через такую конечную точку приложение может и передавать, и получать данные. Потоковый сокет гарантирует обнаружение и исправление ошибок, обрабатывает доставку и сохраняет последовательность данных. Он подходит для передачи больших объемов данных, поскольку в этом случае накладные расходы, связанные с установлением соединения, незначительны по сравнению со временем передачи самого сообщения. Качество передачи достигается за счет использования протокола TCP.

Дейтаграммные сокеты– это сокеты без установления соединения, не обеспечивающие надежность при передаче. Применяются для приложений, когда неприемлемы затраты времени, связанные с установлением явного соединения. Для передачи данных используется протокол UDP.

Сырые сокеты(raw sockets - необрабатываемые, простые) – это сокеты, которые взаимодействуют с протоколами сетевого уровня в обход протоколов транспортного уровня. Используются для непосредственного доступа приложения к IP-пакетам сетевого уровня. Использование сырых сокетов возможно при разработке низкоуровневого системного ПО. Например, сырые сокеты используют различные программы-анализаторы пакетов, сниферы, утилиты TCP/IP ping, tracert и т д..

# 2 ПОСТАНОВКА ЗАДАЧИ

Написать программы, реализующие функции игрового клиента и игрового сервера. В окне сервера должен отображаться весь протокол общения клиента с сервером.

# 3 МОДЕЛИРОВАНИЕ И РЕАЛИЗАЦИЯ ПО

## Архитектура ПО и протокол взаимодействия клиента и сервера

Приложение реализовано по технологии клиент-сервер. После запуска сервера происходит ожидание подключений от клиентов. Процесс обмена данными игроков происходит через сервер.

## Алгоритмы работы модулей

Алгоритм работы приложения сервер:

* Запуск сервера;
* Ожидание подключения двух клиентов;
* Получение, обработка и отправление данных клиентов;
* Закрытие соединения

Алгоритм работы приложения клиент:

* Запуск клиента;
* Подключение к серверу;
* Отправка данных на сервер и получение данных от сервера;

## Интерфейс пользователя

Серверное приложение представляет собой консольное приложение.

Клиентское приложение представлено графическим интерфейсом (см. рис. 1-5 графической части).

## Библиотечные классы и методы

Для работы с протоколом UDP в .NET предназначены классы HttpWebRequest и HttpWebResponse. Эти классы принадлежат к пространству имен System.Net.

Класс HttpWebRequest позволяет отправлять HTTP-запросы, а класс HttpWebResponse позволяет обрабатывать полученные ответы.

HttpWebRequest request =

(HttpWebRequest)WebRequest.Create(URI);

HttpWebResponse response =

(HttpWebResponse)request.GetResponse();

Чтобы взаимодействовать с сервером HttpWebResponse определяет метод GetResponseStream(). Который возвращает поток, используемый для чтения основного текста ответа с сервера.

dataStream = response.GetResponseStream();

После окончания работы с HttpWebResponse его надо закрыть методом Close().

response.Close();

Для отправки данных у потока используется метод Write():

byte[] byteArray = Encoding.Unicode.GetBytes(postData);

Stream dataStream = request.GetRequestStream();

dataStream.Write(byteArray, 0, byteArray.Length);

Остальные используемые библиотечные классы и методы можно посмотреть в приложении (см. ниже приложение).

## Разработанные классы и методы

Класс CGet, имеет свои поля и свойства, а так же содержит 3 метода.

* metodGet
* getClient
* getServer

Метод metodGet реализует запрос HTTP GET для запроса содержимого указанного ресурса.

Метод getClient реализует протокол общения клиента при запросе GET.

Метод getServer реализует протокол общения сервера при запросе GET.

Класс CHead имеет свои поля и свойства, а так же содержит 3 метода:

* metodHead
* headClient
* headServer

Метод metodHead реализует запрос HTTP HEAD для получения метаинформации о ресурсе без передачи по сети самого ресурса.

Метод headClient реализует протокол общения клиента при запросе HEAD.

Метод headServer реализует протокол общения сервера при запросе HEAD.

Класс CPost имеет свои поля и свойства, а так же содержит 3 метода:

* metodPost
* postClient
* postServer

Метод metodPost реализует запрос HTTP POST для передачи пользовательских данных заданному ресурсу.

Метод postClient реализует протокол общения клиента при запросе POST.

Метод postServer реализует протокол общения сервера POST.

# 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Пользователь сначала открывает программу Http\_Client с расширением \*.exe (см. рисунок 1 графической части). Для подключения к серверу требуется ввести URI-адрес в строку вверху приложения и нажать кнопку “Start”, предварительно выбрав один из предложенных запросов (см. рисунки 2-4 графической части).

В окне “Результат запросов” клиента будет отображен ответ сервера на запрос. (см. рисунки 1-4 графической части)

В окнах “Клиент”, “Сервер” будет отображен протокол общения между сервером и клиентом.

При выборе запроса POST, в левом нижнем углу приложения нужно ввести сообщение, передаваемое серверу (см. рисунок 3 графической части).

# ЗАКЛЮЧЕНИЕ

В теоретической части курсового проекта рассматривается протокол HTTP, основные понятия технологии сокетов и URL-адреса.

HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов.

Сокет(Socket - гнездо, разъем) - абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки сетевого соединения.

Технология (интерфейс) сокетов – название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут выполняться как на одном компьютере, так и на разных, связанных между собой сетью.

URI (Uniform Resource Identifiers - Универсальный идентификатор ресурса) — последовательность символов, идентифицирующая абстрактный или физический ресурс.

В практической части курсового проекта осуществляется разработка программы по теме: “Приложение HTTP-клиента”.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Олифер В. Г., Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов / Олифер В. Г., Олифер Н. А. 4-е изд. — Санкт-Петербург: Питер, 2010. — 944 е.: ил.
2. https://msdn.microsoft.com/
3. http://cdo.bseu.by/library/ibs1/applic\_l/www/http.htm

# ПРИЛОЖЕНИЕ A

**Графическая часть**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Рисунок 1 – Клиент ожидающий ввод URI адреса    Рисунок 2 – Клиент посылает запрос GET на сервер | | | | | | | | | | | | | | | | |
|  |  |  | |  | |  | | КП—10701214 03–ДО–201 | | | | | | | | |
|  |  |  | |  | |  | |
| Изм. | Лист | № документа | | Подпись | | Дата | |
| Разраб. | | Веселов | |  | |  | | Архитектура приложения | | Лит | | | | Лист | | Листов |
| Руковод. | | Белова | |  | |  | |  | | Д |  | **1** | | **8** |
| Консульт. | | Белова | |  | |  | | 1 40 01 02 БНТУ  г.Минск | | | | | | |
| Н.контр. | | Белова | |  | |  | |
| Зав.каф. | | . | |  | |  | |
| Рисунок 3 – Клиент посылает запрос POST на сервер    Рисунок 4 – Клиент посылает запрос HEAD на сервер | | | | | | | | | | | | | | | | |
|  | | | Фамилия | | Подпись | | Дата | | Архитектура ПО | | Лист | | | | Листов | |
| Студентка | | | Веселов | |  | |  | |
| Руководитель | | | Белова | |  | |  | | 2 | | | | 8 | |
| Рисунок 5 – Сообщение об ошибке подключения | | | | | | | | | | | | | | | | |
|  | | | Фамилия | | Подпись | | Дата | | Архитектура ПО | | Лист | | | | Листов | |
| Студентка | | | Веселов | |  | |  | |
| Руководитель | | | Белова | |  | |  | | 3 | | | | 8 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Рисунок 6 – Алгоритм работы метода metodGet | | | | | | |
|  | Фамилия | Подпись | Дата | Архитектура ПО | Лист | Листов |
| Студентка | Веселов |  |  |
| Руководитель | Белова |  |  | 4 | 8 |
| Рисунок 7 – Алгоритм работы метода metodHead | | | | | | |
|  | Фамилия | Подпись | Дата | Архитектура ПО | Лист | Листов |
| Студентка | Веселов |  |  |
| Руководитель | Белова |  |  | 5 | 8 |
| Рисунок 8 – Алгоритм работы метода metodPost | | | | | | |
|  | Фамилия | Подпись | Дата | Архитектура ПО | Лист | Листов |
| Студентка | Веселов |  |  |
| Руководитель | Белова |  |  | 6 | 8 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Рисунок 9 – Алгоритм работы метода SendM | | | | | | |
|  | Фамилия | Подпись | Дата | Архитектура ПО | Лист | Листов |
| Студентка | Веселов |  |  |
| Руководитель | Белова |  |  | 7 | 8 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Рисунок 10 – Диаграмма классов клиента | | | | | | |
|  | Фамилия | Подпись | Дата | Архитектура ПО | Лист | Листов |
| Студентка | Веселов |  |  |
| Руководитель | Белова |  |  | 8 | 8 |

# ПРИЛОЖЕНИЕ Б

/\*Проект клиента\*/

/\*Form1.cs\*/

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.IO;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Http\_Client

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

string sURI = textBox2.Text;

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(sURI); //создание объекта класса HttpWebRequest

if (radioButton1.Checked)

{

request.Method = "GET"; //свойство задает метод запроса

request.Accept = "text"; //свойство задания заголовка

CGet get = new CGet(); //создание объекта класса CGet

HttpWebResponse response = (HttpWebResponse)request.GetResponse();

textBox1.Text += get.metodGet(request, response, sURI); //вывод в textBox тела сообщения полученное запросом

textBox3.Text += get.getClient(request, response, sURI); //

textBox4.Text += get.getServer(request, response); //вывод в textBox протокол общения сервера с клиентом

response.Close();

}

if (radioButton2.Checked)

{

request.Method = "POST"; //свойство задает метод запроса

string postData = textBox5.Text;

CPost post = new CPost(); //создание объекта класса CPost

textBox1.Text += post.metodPost(request, sURI, postData);

HttpWebResponse response = (HttpWebResponse)request.GetResponse();

textBox3.Text += post.postClient(request, response, sURI); //

textBox4.Text += post.postServer(request, response); //вывод в textBox протокол общения сервера с клиентом

response.Close();

}

if (radioButton3.Checked)

{

request.Method = "HEAD"; //свойство задает метод запроса

request.Accept = "text"; //свойство задания заголовка

CHead head = new CHead(); //создание объекта класса CHead

HttpWebResponse response = (HttpWebResponse)request.GetResponse();

textBox1.Text += head.metodHead(request, response, sURI); //вывод в textBox заголовков полученное запросом

textBox3.Text += head.headClient(request, response, sURI); //

textBox4.Text += head.headServer(request, response); //вывод в textBox протокол общения сервера с клиентом

response.Close();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message); //выводит сообщение об ошибке

}

}

private void button2\_Click(object sender, EventArgs e)

{

textBox1.Text = ""; //очищение окна запросов

}

private void button3\_Click(object sender, EventArgs e)

{

Environment.Exit(0); //выход из программы

}

}

}

/\* CGet.cs\*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.IO;

using System.Text;

using System.Threading.Tasks;

namespace Http\_Client

{

class CGet

{

private string output { get; set; } //поле для записи ответа

public string metodGet(HttpWebRequest request, HttpWebResponse response, string sURI)

{

StreamReader reader = new StreamReader(response.GetResponseStream());

output = reader.ReadToEnd(); //считывание потока с сервера

reader.Close();

return "Выполнение запроса " + request.Method + " по адресу " + sURI + Environment.NewLine

+ output + Environment.NewLine;

}

//протокол общения клиента с сервером

public string getClient(HttpWebRequest request, HttpWebResponse response, string sURI)

{

return request.Method + " " + sURI + " HTTP/" + request.ProtocolVersion + Environment.NewLine

+ "Connection: " + request.Connection + Environment.NewLine

+ "Host: " + request.Host + Environment.NewLine

+ "Accept: " + request.Accept + Environment.NewLine + Environment.NewLine;

}

public string getServer(HttpWebRequest request, HttpWebResponse response)

{

return "HTTP/" + request.ProtocolVersion + " " + response.StatusCode.GetHashCode() + " "

+ response.StatusCode + Environment.NewLine

+ "Date: " + DateTime.Now.DayOfWeek + " " + DateTime.Now + Environment.NewLine

+ "Server: " + response.Server + Environment.NewLine

+ "LastModified: " + response.LastModified.DayOfWeek + " " + response.LastModified + Environment.NewLine

+ "ContentType: " + response.ContentType + Environment.NewLine

+ "ContentLength: " + response.ContentLength + Environment.NewLine + Environment.NewLine;

}

}

}

/\* CHead.cs\*

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

namespace Http\_Client

{

class CHead

{

private string output { get; set; } //поле для записи ответа

public string metodHead(HttpWebRequest request, HttpWebResponse response, string sURI)

{

for (int i = 0; i < response.Headers.Count; ++i) //считывание массива заголовков

{

output += "Имя: " + response.Headers.Keys[i] + Environment.NewLine

+ "Значение : " + response.Headers[i] + Environment.NewLine

+ Environment.NewLine;

}

return "Выполнение запроса " + request.Method + " по адресу " + sURI + Environment.NewLine

+ output + Environment.NewLine;

}

//протокол общения клиента с сервером

public string headClient(HttpWebRequest request, HttpWebResponse response, string sURI)

{

return request.Method + " " + sURI + " HTTP/" + request.ProtocolVersion + Environment.NewLine

+ "Connection: " + request.Connection + Environment.NewLine

+ "Host: " + request.Host + Environment.NewLine

+ "Accept: " + request.Accept + Environment.NewLine + Environment.NewLine;

}

public string headServer(HttpWebRequest request, HttpWebResponse response)

{

return "HTTP/" + request.ProtocolVersion + " " + response.StatusCode.GetHashCode() + " "

+ response.StatusCode + Environment.NewLine

+ "Date: " + DateTime.Now.DayOfWeek + " " + DateTime.Now + Environment.NewLine

+ "Server: " + response.Server + Environment.NewLine

+ "LastModified: " + response.LastModified.DayOfWeek + " " + response.LastModified + Environment.NewLine

+ "ContentType: " + response.ContentType + Environment.NewLine

+ "ContentLength: " + response.ContentLength + Environment.NewLine + Environment.NewLine;

}

}

}

/\* CPost.cs\*

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

namespace Http\_Client

{

class CPost

{

private string output { get; set; } //поле для записи ответа

public string metodPost(HttpWebRequest request, string sURI, string postData)

{

byte[] byteArray = Encoding.Unicode.GetBytes(postData); //заполнение масива байт для отправки на сервер

request.ContentType = "text/plain";

request.ContentLength = byteArray.Length;

//отправка на сервер потока данных

Stream dataStream = request.GetRequestStream();

dataStream.Write(byteArray, 0, byteArray.Length);

dataStream.Close();

//запрос данных с сервера

HttpWebResponse response = (HttpWebResponse)request.GetResponse();

dataStream = response.GetResponseStream();

StreamReader reader = new StreamReader(dataStream);

output = reader.ReadToEnd();

reader.Close();

dataStream.Close();

response.Close();

return "Выполнение запроса " + request.Method + " по адресу " + sURI + Environment.NewLine

+ output + Environment.NewLine;

}

//протокол общения клиента с сервером

public string postClient(HttpWebRequest request, HttpWebResponse response, string sURI)

{

return request.Method + " " + sURI + " HTTP/" + request.ProtocolVersion + Environment.NewLine

+ "Host: " + request.Host + Environment.NewLine

+ "Accept: " + request.Accept + Environment.NewLine

+ "ContentType: " + request.ContentType + Environment.NewLine

+ "ContentLength: " + request.ContentLength + Environment.NewLine + Environment.NewLine;

}

public string postServer(HttpWebRequest request, HttpWebResponse response)

{

return "HTTP/" + request.ProtocolVersion + " " + response.StatusCode.GetHashCode() + " "

+ response.StatusCode + Environment.NewLine

+ "Date: " + DateTime.Now.DayOfWeek + " " + DateTime.Now + Environment.NewLine

+ "Server: " + response.Server + Environment.NewLine

+ "LastModified: " + response.LastModified.DayOfWeek + " " + response.LastModified + Environment.NewLine

+ "ContentType: " + response.ContentType + Environment.NewLine

+ "ContentLength: " + response.ContentLength + Environment.NewLine + Environment.NewLine;

}

}

}