|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  **«Московский технологический университет»**  **МИРЭА** | | |

Институт Информационных Технологий

Кафедра Корпоративных Информационных Систем

**ОТЧЕТ**

по Лабораторной Работе №5

на тему

«Cокеты Беркли»

по дисциплине

«Технологии программирования»

Выполнил студент группы ИСБО-11-16 Шайхуллин С.В.

Принял Cтарший преподаватель Мирзоян Д.И.

Выполнено «\_\_\_» \_\_\_\_\_\_\_\_\_2017 г.

Зачтено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2017г.

Москва, 2017

**Теоретическая часть**

Сокеты Беркли — это стандартный интерфейс программирования межпроцессного и сетевого взаимодействия, разработанный в Калифорнийском университете в Беркли. Интерфейс предназначен для обеспечения максимально легкой портируемости приложений на уровне исходного кода и поддерживается большинством существующих операционных систем. Интерфейс определяет некоторый базовый набор функций, предназначенных в основном для работы в сетях TCP/IP.

Интерфейс является процедурным (как и большинство системных API), состоит из набора функций обслуживания сокета и вспомогательных подпрограмм.

Перед использованием библиотеку сокетов необходимо загрузить и проинициализировать, вызвав соответствующую функцию.

Для создания сокета используется функция socket. В качестве аргументов ей передается тип создаваемого сокета (протокол транспортного уровня) и семейство адресов (протокол сетевого уровня). Возвращается хэндл сокета, для дальнейшего использования. Некоторые виды сокетов могут использоваться многократно. Для уничтожения сокета и освобождения связанных с ним ресурсов используется функция closesocket.

Сокеты делятся по типу используемого протокола транспортного уровня на потоковые и датаграммные. Первые работают с протоколом TCP, с потоковым механизмом и установкой виртуального соединения, вторые работают с протоколом UDP, механизмом отдельных сообщений (датаграмм). От выбора типа сокета зависит набор используемых для работы с ним функций.

Также сокеты по характеру использования делятся на клиентские и серверные. Клиентские сокеты служат для передачи данных, они инкапсулируют виртуальное соединение. Серверные сокеты принимают запросы на подключение, приходящие по сети.

Рассмотрим основные функции для работы с сокетами.

Функция connect. Устанавливает виртуальное соединение для сокетов потокового типа. Без установки соединения работа такого сокета невозможна. Для сокетов датаграммного типа устанавливает адрес назначения «по-умолчанию».

Функция bind. Устанавливает локальный адрес сокета. Необходима для инициализации серверного сокета — задает адрес и порт, на котором сокет будет ожидать соединения. Для клиентских сокетов задает адрес и порт отправителя. Если в клиентском сокете эта функция не была использована, адрес и порт отправителя назначаются автоматически операционной системой. Обратите внимание, что использование этой функцией основного диапазона портов (обычно до 1024) требует, чтобы приложение было запущено с привилегиями администратора.

Функция listen. Переключает серверный сокет в «слушающий» режим. Именно после вызова этой функции сокет и становится серверным — т.е. способным принимать соединения.

Функция accept. Принимает входящее соединение к серверному сокету. Возвращает адрес и порт отправителя для его идентификации. Самое главное — эта функция устанавливает логическое соединение со стороны сервера и создает клиентский сокет для работы с данным подключением. Все взаимодействие с клиентом ведется через созданный клиентский сокет. Передача данных через серверный сокет невозможна.

Функция recv. Считывает данные из сокета.

Функция send. Отправляет данные в сокет.

Функция recvfrom. Считывает данные и адрес отправителя. Применима только для датаграммных сокетов.

Функция sendto. Отправляет данные конкретному получателю. Применима только для датаграммных сокетов.

Датаграммные сокеты, поскольку они не требуют установки виртуального соединения, можно использовать и как серверные и как клиентские одновременно.

Функция setsockopt. Устанавливает параметры сокета.

Функция ioctlsocket. Устанавливает параметры сокета.

Вспомогательные функции

Hton\* преобразуют порядок байт из машинного в сетевой. В случае x86 меняют порядок байт на противоположный (сетевой порядок big-endian, машинный little-endian). В случае big-endian машин не делают ничего.

Ntoh\* обратные функции, преобразуют порядок байт из сетевого в машинный.

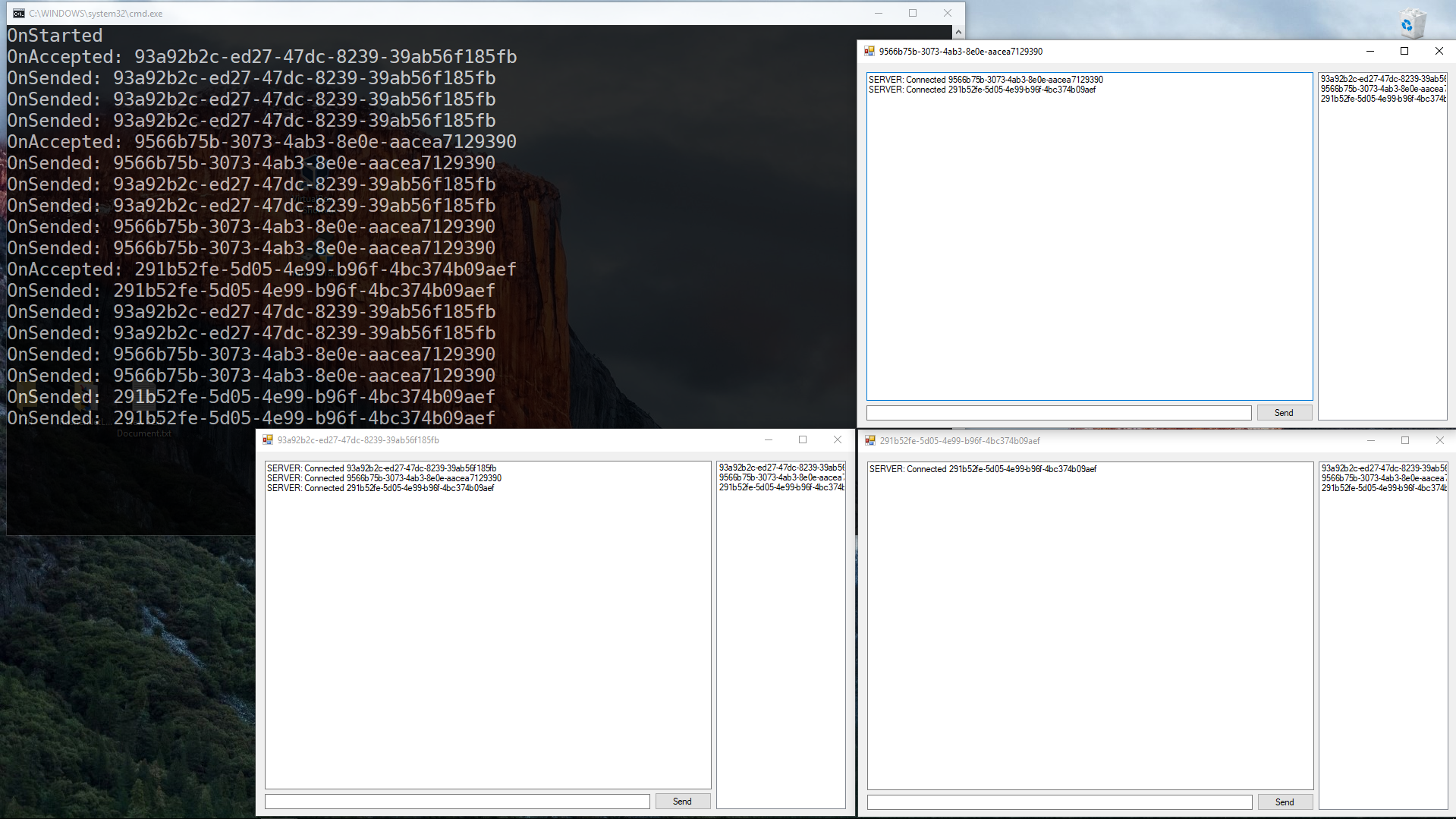
Функция gethostbyname. Получить адрес (IP) по имени узла. Используются все доступные ОС службы — DNS, WINS, SAMBA, hosts, AD и прочие.

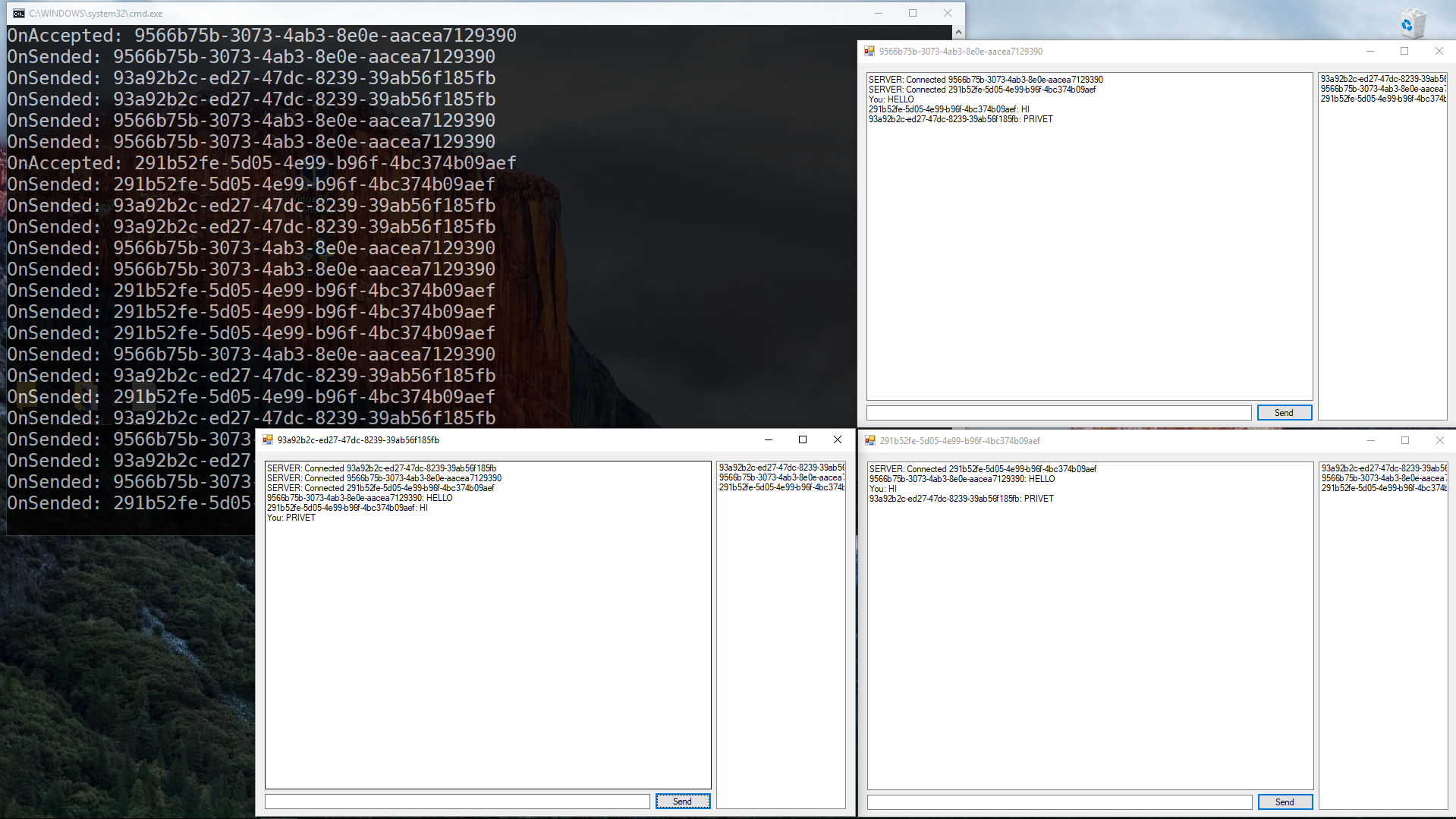
Функция gethostbyaddr. Получить имя узла по его адресу.

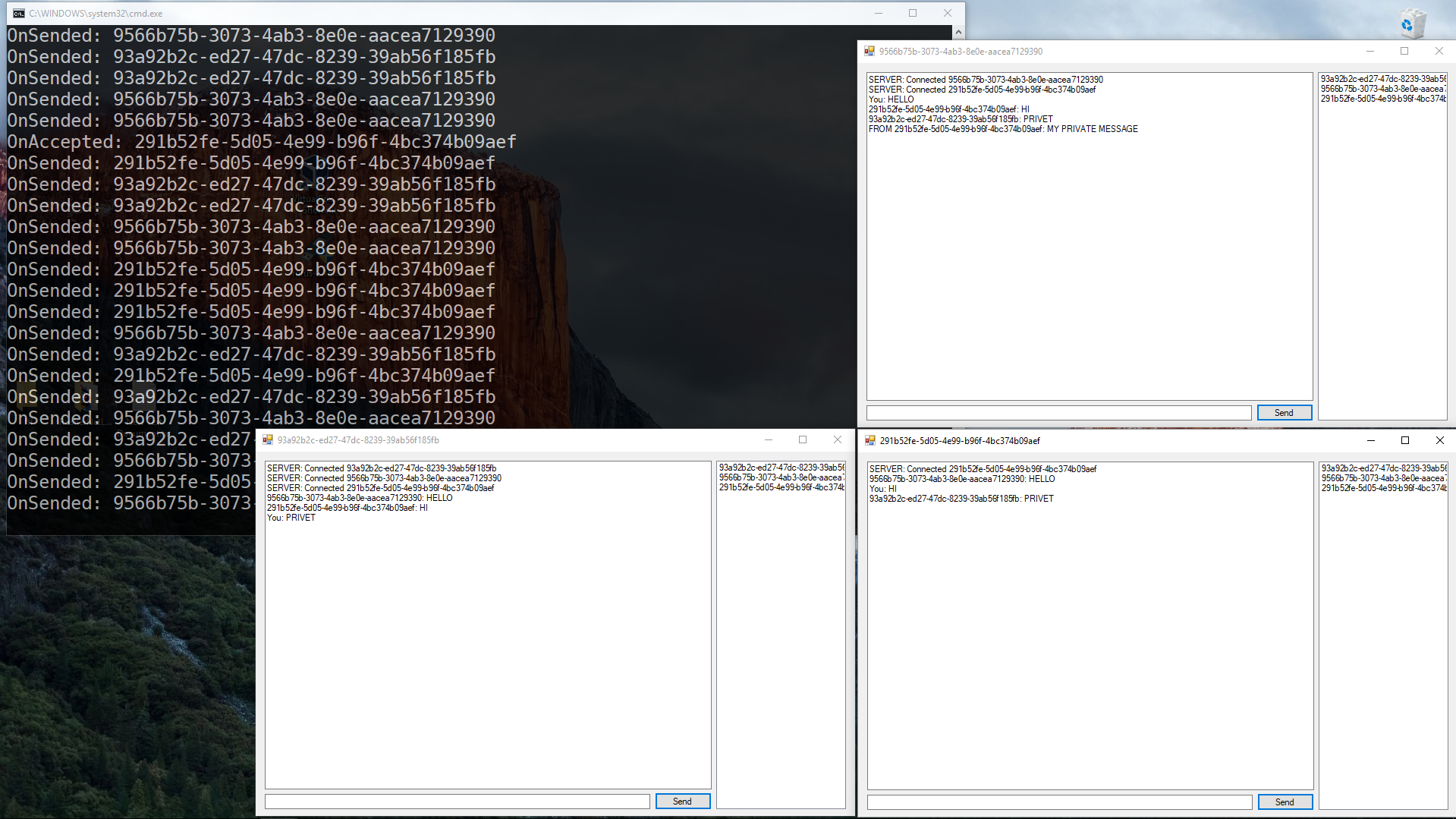
**Задание**

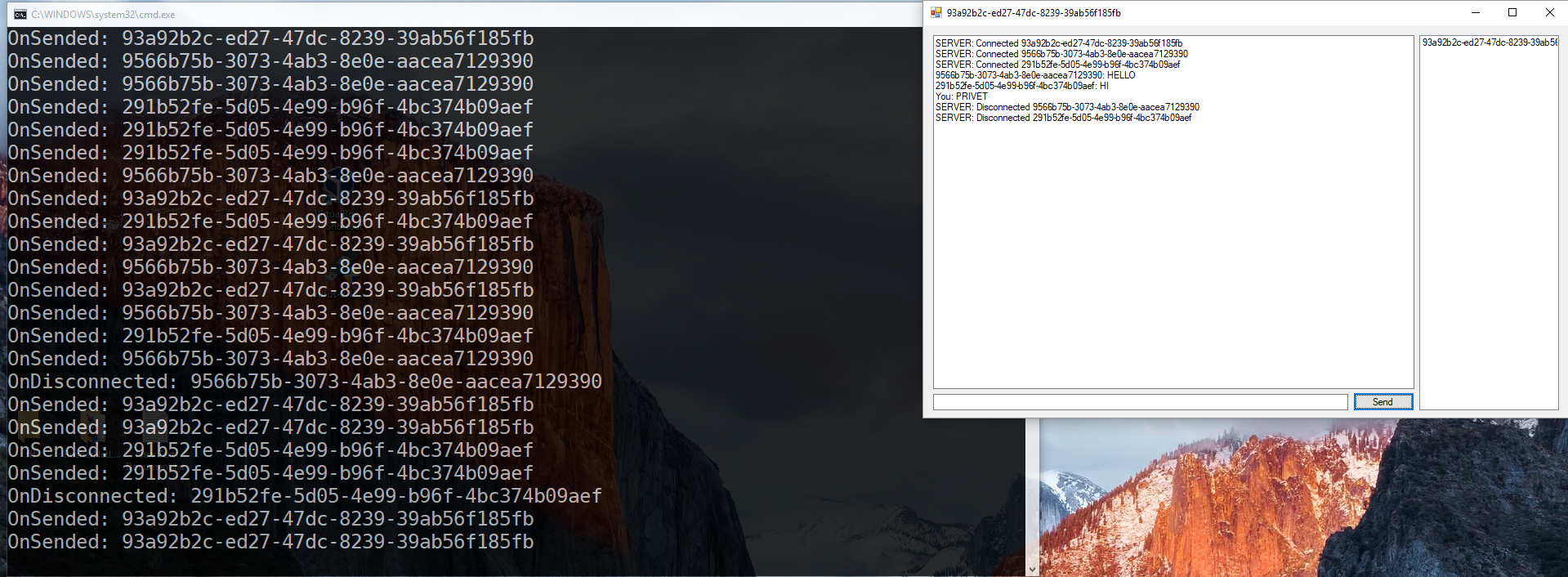
Разработать сетевое приложение, позволяющее передавать текстовые сообщения в масштабах локальной сети. Все варианты реализации должны быть совместимы между собой (использовать один протокол).

**Пример работы**









**Заключение**

В данной лабораторной работе мы научились работать с АПИ сокетов, отправлять данные другим клиентам, принимать и обрабатывать данные. А так же проектировать протокол передачи данных поверх стека TCP/IP. С исходных кодом библиотеки можно ознакомиться в репозитории https://github.com/yolorusher/RusherNetLib

**Исходный код**

using System;

using System.Data;

using System.Linq;

using System.Windows.Forms;

using System.Threading.Tasks;

using System.Collections.Generic;

using RusherNetLib.Core;

using RusherNetLib.NetClient;

namespace TestClient

{

public partial class MainWindow : Form

{

private readonly IClient client;

public MainWindow()

{

InitializeComponent();

client = new Client()

.AddHandler(ClientType.Connected, OnConnected)

.AddHandler(ClientType.Received, OnReceived)

.AddHandler(ClientType.Disconnected, OnDisconnected)

.Connect("127.0.0.12", 4000);

}

private void OnDisconnected(IConnection conn, IMessage msg)

{

MessageBox.Show("Connection lost!");

Application.Exit();

}

private void OnReceived(IConnection conn, IMessage msg)

{

if (msg["Type"] == "List")

{

loop.Enqueue(() =>

{

listBox1.Items.Clear();

listBox1.Items.AddRange(msg["List"].Split(','));

});

}

else if (msg["Type"] == "Name")

{

loop.Enqueue(() =>

{

Text = msg["Name"];

});

}

else if (msg["Type"] == "Send")

{

loop.Enqueue(() =>

{

messages.AppendText($"{msg["Author"]}: {msg["Text"]}\n");

});

}

else if(msg["Type"] == "PrivateSend")

{

loop.Enqueue(() =>

{

messages.AppendText($"FROM {msg["Author"]}: {msg["Text"]}\n");

});

}

}

private void OnConnected(IConnection conn, IMessage msg)

{

MessageBox.Show("Connected! " + conn.SocketError);

}

private void SendClicked(object sender, EventArgs e)

{

if (listBox1.SelectedItems.Count == 0)

{

var msg = client.CreateMessage();

msg["Type"] = "Send";

msg["Text"] = inputTextBox.Text;

client.Send(msg);

}

else

{

var msg = client.CreateMessage();

msg["Type"] = "PrivateSend";

msg["Persons"] = string.Join(",", listBox1.SelectedItems.Cast<string>().Where(x => x != Text));

msg["Text"] = inputTextBox.Text;

client.Send(msg);

}

inputTextBox.Text = "";

listBox1.SelectedItems.Clear();

}

private Queue<Action> loop = new Queue<Action>();

private async void MainWindow\_Load(object sender, EventArgs e)

{

while (true)

{

while (loop.Any())

{

loop.Dequeue()();

}

await Task.Delay(100);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using RusherNetLib.Core;

using RusherNetLib.NetServer;

namespace TestServer

{

class Program

{

private static List<Person> persons = new List<Person>();

static void Main(string[] args)

{

var server = new Server()

.AddHandler(ServerType.Started, OnStarted)

.AddHandler(ServerType.Accepted, OnAccepted)

.AddHandler(ServerType.Sended, OnSended)

.AddHandler(ServerType.Received, OnReceived)

.AddHandler(ServerType.Disconnected, OnDisconnected)

.AddHandler(ServerType.Stopped, OnStopped)

.Start("127.0.0.12", 4000);

Console.ReadLine();

server.Stop();

Console.ReadLine();

}

//Когда сервер стартовал

private static void OnStarted(IConnection conn, IMessage msg)

{

Console.WriteLine("OnStarted");

}

//Когда сервер принял клиент

private static void OnAccepted(IConnection conn, IMessage msg)

{

var p = new Person

{

Name = Guid.NewGuid().ToString(),

Connection = conn

};

persons.Add(p);

Console.WriteLine($"OnAccepted: {p.Name}");

var name = conn.CreateMessage();

name["Type"] = "Name";

name["Name"] = p.Name;

conn.Send(name);

Thread.Sleep(100);

foreach (var person in persons)

{

var message = person.Connection.CreateMessage();

message["Type"] = "List";

message["List"] = string.Join(",", persons.Select(x => x.Name).ToArray());

person.Connection.Send(message);

Thread.Sleep(100);

message["Type"] = "Send";

message["Author"] = "SERVER";

message["Text"] = $"Connected {p.Name}";

person.Connection.Send(message);

}

}

private static void OnSended(IConnection conn, IMessage msg)

{

var person = persons.First(x => x.Connection == conn);

Console.WriteLine($"OnSended: {person.Name}");

}

private static void OnReceived(IConnection conn, IMessage msg)

{

if (msg["Type"] == "Send")

{

var person = persons.First(x => x.Connection == conn);

msg["Author"] = person.Name;

foreach (var p in persons.Where(x => x != person))

{

p.Connection.Send(msg);

}

msg["Author"] = "You";

person.Connection.Send(msg);

}

else if (msg["Type"] == "PrivateSend")

{

var person = persons.First(x => x.Connection == conn);

var message = conn.CreateMessage();

message["Type"] = "PrivateSend";

message["Author"] = person.Name;

message["Text"] = msg["Text"];

var names = msg["Persons"].Split(',');

foreach (var name in names)

{

var psn = persons.First(x => x.Name == name);

psn.Connection.Send(message);

}

}

}

private static void OnDisconnected(IConnection conn, IMessage msg)

{

var p = persons.First(x => x.Connection == conn);

persons.Remove(p);

Console.WriteLine($"OnDisconnected: {p.Name}");

Thread.Sleep(100);

foreach (var person in persons)

{

var message = person.Connection.CreateMessage();

message["Type"] = "List";

message["List"] = string.Join(",", persons.Select(x => x.Name).ToArray());

person.Connection.Send(message);

Thread.Sleep(100);

message["Type"] = "Send";

message["Author"] = "SERVER";

message["Text"] = $"Disconnected {p.Name}";

person.Connection.Send(message);

}

}

private static void OnStopped(IConnection conn, IMessage msg)

{

Console.WriteLine("OnStopped");

}

}

class Person

{

public string Name { get; set; }

public IConnection Connection { get; set; }

}

}