# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**ЗВІТ**

**Про виконання лабораторної роботи №5**

**«Створення клієнт/серверних програм за допомогою бібліотек C#»**

**з дисципліни «Програмне забеспечення мережевих технологій»**

Виконав:

слухач групи ПЗС-11

Гринчук Тарас

Прийняв:

ст. викл.Сенів М.М.

« »\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2014 р.

∑ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛЬВІВ – 2014

**Тема роботи**: Створення клієнт/серверних програм за допомогою бібліотек C#

**Мета роботи:** Навчитись концепціям програмування мовою C#, а також засвоїти основні стандартні класи і методи бібліотеки .net.

## 1. Хід роботи

**Завдання:**

Розробити сценарій взаємодії двох програм. Вибрати необхідні класи та методи. Вибрати структури даних. Запрограмувати два мережних модуля обміну файлами, а саме:

1. Розробити механізм читання вхідного файлу та його рзміщенння;

2. Спроектувати структури для зберіганння даних;

3. Формалізувати алгоритм. Вибрати класи та методи для реалізації;

4. Створити візуальні засоби.

Сервер побудуємо синхронно, так щоб, виконання потоку блокувалося, поки сервер не дасть згоди на з'єднання з клієнтом. Клієнт завершує з'єднання, відправляючи серверу повідомлення *<TheEnd>.*

**Сервер TCP**

Перший крок полягає у встановленні для сокета локальної кінцевої точки . Перш ніж відкривати сокет для очікування з'єднань, потрібно підготувати для нього адресу локальної кінцевої точки. Унікальний адресу для обслуговування TCP / IP визначається комбінацією IP- адреси хоста з номером порту обслуговування, яка створює кінцеву точку для обслуговування.

Клас Dns надає методи, які повертають інформацію про мережевих адресах, підтримуваних пристроєм в локальній мережі. Якщо у пристрої локальної мережі є більше одного мережевого адреси, клас Dns повертає інформацію про всі мережевих адресах, і додаток повинен вибрати з масиву відповідну адресу для обслуговування.

Створимо IPEndPoint для сервера, комбінуючи перший IP-адреса хост- комп'ютера, отриманий від методу Dns.Resolve ( ) , з номером порту :

*IPHostEntry ipHost = Dns.GetHostEntry("localhost");*

*IPAddress ipAddr = ipHost.AddressList[0];*

*IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 11000);*

Тут клас IPEndPoint представляє localhost на порту 11000. Далі новим екземпляром класу Socket створюємо потоковий сокет. Встановивши локальну кінцеву точку для очікування з'єднань, можна створити сокет:

*Socket sListener = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);*

Перерахування AddressFamily вказує схеми адресації , які екземпляр класу Socket може використовувати для дозволу адреси.

У параметрі SocketType розрізняються сокети TCP і UDP. У ньому можна визначити в тому числі такі значення:

*Dgram*

Підтримує дейтаграми. Значення Dgram вимагає вказати Udp для типу протоколу і InterNetwork в параметрі сімейства адрес.

*Raw*

Підтримує доступ до базового транспортного протоколу .

*Stream*

Підтримує потокові сокети . Значення Stream вимагає вказати Tcp для типу протоколу .

Третій і останній параметр визначає тип протоколу , необхідний для сокета. У параметрі РrotocolType можна вказати наступні найбільш важливі значення - Tcp , Udp , Ip , Raw .

Наступним кроком має бути призначення сокета за допомогою методу Bind ( ) . Коли сокет відкривається конструктором , йому не призначається ім'я , а тільки резервується дескриптор . Для призначення імені сокету сервера викликається метод Bind() . Щоб сокет клієнта міг ідентифікувати потоковий сокет TCP , серверна програма повинна дати ім'я своїй сокету :

*sListener.Bind(ipEndPoint);*

Метод Bind () пов'язує сокет з локальною кінцевою точкою. Викликати метод Bind () треба до будь-яких спроб звернення до методів Listen () і Accept ().

Тепер, створивши сокет і зв'язавши з ним ім'я, можна слухати вхідні повідомлення, скориставшись методом Listen(). У стані прослуховування сокет буде очікувати вхідної спроби з'єднання:

*sListener.Listen(10);*

В параметрі визначається заділ (backlog) , який вказує максимальне число з'єднань, що очікують обробки в черзі.

У стані прослуховування треба бути готовим дати згоду на з'єднання з клієнтом , для чого використовується метод Accept(). За допомогою цього методу виходить з'єднання клієнта і завершується встановлення зв'язку імен клієнта і сервера. Метод Accept( ) блокує потік викликає програми до надходження з'єднання.

Метод Accept( ) витягує з черги чекають запитів перший запит на з'єднання і створює для його обробки новий сокет . Хоча новий сокет створений, первісний сокет продовжує слухати і може використовуватися з багатопотокової обробкою для прийому декількох запитів на з'єднання від клієнтів. Ніяке серверний додаток не повинно закривати слухає сокет . Він повинен продовжувати працювати поряд з сокетами , створеними методом Accept для обробки вхідних запитів клієнтів.

Як тільки клієнт і сервер встановили між собою з'єднання , можна відправляти і отримувати повідомлення , використовуючи методи Send ( ) і Receive ( ) класу Socket .

Метод Send ( ) записує вихідні дані сокету , з яким встановлено з'єднання . Метод Receive ( ) зчитує вхідні дані в потоковий сокет . При використанні системи , заснованої на TCP , перед виконанням методів Send ( ) і Receive ( ) між сокетами повинно бути встановлене з'єднання . Точний протокол між двома взаємодіючими сутностями має бути визначений завчасно, щоб клієнтське і серверне додатки не блокували один одного , не знаючи , хто повинен відправити свої дані першого .

Коли обмін даними між сервером і клієнтом завершується , потрібно закрити з'єднання використовуючи методи Shutdown ( ) і Close ( ) .

SocketShutdown - це перерахування , що містить три значення для зупинки: Both - зупиняє відправку та отримання даних сокетом , Receive - зупиняє отримання даних сокетом і Send - зупиняє відправлення даних сокетом .

Сокет закривається при виклику методу Close ( ) , який також встановлює у властивості Connected сокета значення false .

**Клієнт на TCP**

Функції , які використовуються для створення програми-клієнта , більш-менш нагадують серверний додаток . Як і для сервера , використовуються ті ж методи для визначення кінцевої точки , створення екземпляра сокета , відправки та отримання даних і закриття сокета :

**Текст програми сервера**

using System;

using System.Text;

using System.Net;

using System.Net.Sockets;

using System.IO;

namespace SocketServer

{

class Program

{

static void Main(string[] args)

{

// Встановлюємо для сокета локальну кінцеву точку

IPHostEntry ipHost = Dns.GetHostEntry("localhost");

IPAddress ipAddr = ipHost.AddressList[0];

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 11000);

// Створюємо сокет Tcp / Ip

Socket sListener = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

// Призначаємо сокет локальної кінцевій точці і слухаємо вхідні сокети

try

{

sListener.Bind(ipEndPoint);

sListener.Listen(10);

// Починаємо слухати з'єднання

while (true)

{

Console.WriteLine("Очiкуємо з'єднання через порт {0}", ipEndPoint);

// Програма призупиняється, очікуючи вхідне з'єднання

Socket handler = sListener.Accept();

string data = null;

// Ми дочекалися клієнта, що намагається з нами з'єднатися

byte[] bytes = new byte[1024];

int bytesRec = handler.Receive(bytes);

data += Encoding.UTF8.GetString(bytes, 0, bytesRec);

int k = data.IndexOf("#");

string fileName = "";

if (k > -1)

{

fileName = data.Substring(0, k);

data = data.Replace(fileName + "#", "");

// Показуємо дані на консолі

Console.WriteLine("\nОтриманий файл: " + fileName);

Console.WriteLine("============================================");

Console.WriteLine(data);

Console.WriteLine("--------------------------------------------");

string fullName = @"D:\\_Server\" + fileName;

File.WriteAllText(@fullName, data, Encoding.Default);

Console.WriteLine("Файл збережено: " + fullName);

Console.WriteLine("============================================");

// Відправляємо відповідь клієнту

string reply = "Сервер отримав файл. Довжина: " + data.Length.ToString()

+ " символiв";

byte[] msg = Encoding.UTF8.GetBytes(reply);

handler.Send(msg);

}

if (data.IndexOf("<TheEnd>") > -1)

{

Console.WriteLine("Сервер завершив з'єднання з клiєнтом.");

break;

}

handler.Shutdown(SocketShutdown.Both);

handler.Close();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

finally

{

Console.ReadLine();

}

}

}

}

**Текст програми клієнта**

using System;

using System.IO;

using System.Text;

using System.Net;

using System.Net.Sockets;

namespace SocketClient

{

class Program

{

static void Main(string[] args)

{

try

{

SendMessageFromSocket(11000);

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

Console.ReadLine();

}

}

static void SendMessageFromSocket(int port)

{

// Буфер для вхідних даних

byte[] bytes = new byte[1024];

// Встановлюємо віддалену точку для сокета

IPHostEntry ipHost = Dns.GetHostEntry("localhost");

IPAddress ipAddr = ipHost.AddressList[0];

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, port);

Socket sender = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

// З'єднуємо сокет з віддаленою точкою

sender.Connect(ipEndPoint);

int index = 0;

DirectoryInfo dir = new DirectoryInfo(@"D:\\_Client");

Console.WriteLine("\n===================================================\nКаталог: D:\\\_Client\n");

foreach (var item in dir.GetFiles())

Console.WriteLine(" " + index++ + ": " + item.Name);

Console.Write("\nВведiть номер файлу (-1 - вихiд): ");

index = Convert.ToInt32(Console.ReadLine());

if (index < 0 || index > dir.GetFiles().GetUpperBound(0))

{

sender.Send(Encoding.UTF8.GetBytes("<TheEnd>"));

//звільняємо сокет

sender.Shutdown(SocketShutdown.Both);

sender.Close();

return;

}

string message = System.IO.File.ReadAllText(@dir.GetFiles()[index].FullName);

message = dir.GetFiles()[index].Name + '#' + message;

Console.Clear();

Console.WriteLine("Сокет з'єднується з {0} ", sender.RemoteEndPoint.ToString());

byte[] msg = Encoding.UTF8.GetBytes(message);

// Відправляємо дані через сокет

int bytesSent = sender.Send(msg);

// Отримуємо відповідь від сервера

int bytesRec = sender.Receive(bytes);

Console.WriteLine("\nВiдповiдь з сервера: {0}\n\n", Encoding.UTF8.GetString(bytes, 0, bytesRec));

// Використовуємо рекурсію для неодноразового виклику SendMessageFromSocket()

SendMessageFromSocket(port);

}

}

}

Запустимо дану програму **Сервера** на виконання (рис. 1):

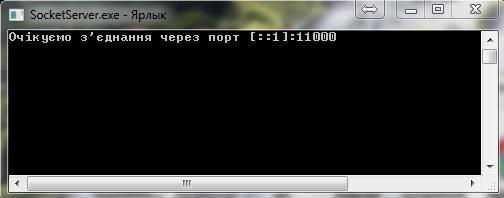


Рис. 1. Вікно програми Сервера

Запустимо дану програму **Клієнта** на виконання (рис. 2):

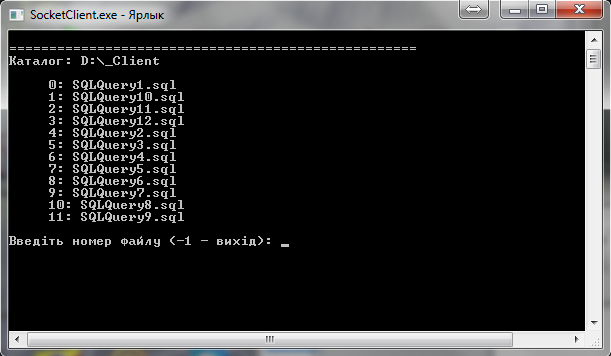


Рис. 2. Вікно програми Клієнта

У вікні клієнта виберемо номер файла, який хочемо переслати на сервер (рис. 3, 4):

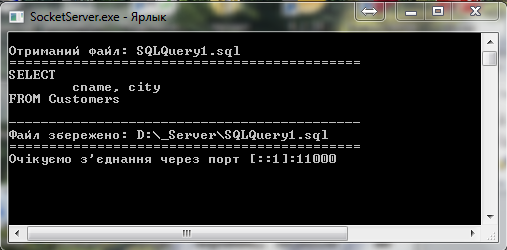


Рис. 3. Вікно програми Сервера після отримання файлу

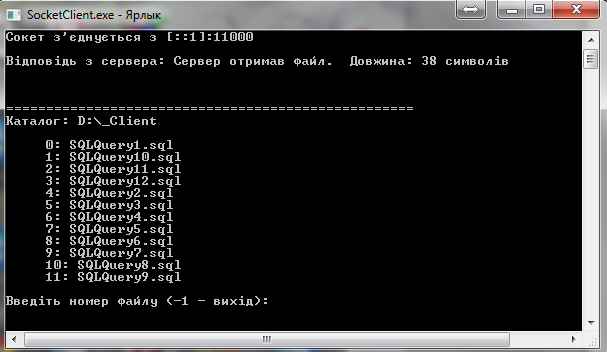


Рис. 4. Вікно програми Клієнта після пересилання файлу

Відправимо також, ще файли під номерами: 10 та 11 і потім у діалозі вибору номеру файлу вкажемо число: -1 – програми завершать роботу сокетів. Папка з файлами сервера набуде вигляду (рис. 5):

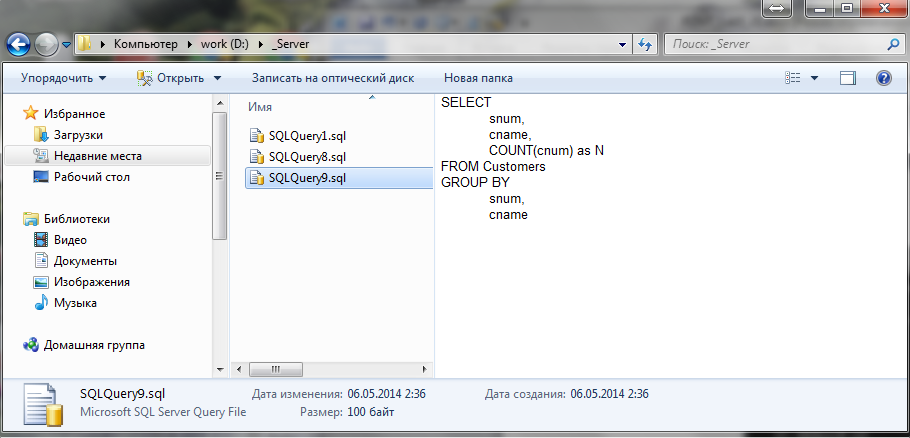


Рис. 5. Вигляд папки з отриманими файлами

## ВИСНОВКИ

На даній лабораторній роботі я навчився концепціям програмування мовою C#, а також засвоїв основні стандартні класи і методи бібліотеки .net.