Дисципліна: Об’єктно-орієнтоване конструювання програм
Лабораторна робота № 4 (4 год.)

Тема: Мережеве програмування

Метою цієї лабораторної роботи є створення двох застосунків – клієнтського та серверного, які мають обмінюватися короткими текстовими повідомленнями різних типів.

Теоретичні відомості

У Visual Studio сукупність кількох проєктів називається рішенням. Спочатку потрібно створити рішення та проєкт для сервера. Потім одразу створити всередині цього рішення другий проєкт для клієнта. Для цього треба відкрити браузер рішень, правою кнопкою миші клацнути на рядку з рішенням і в меню вибрати команду Додати/Створити проєкт.

Якщо все буде зроблено правильно, то в браузері рішень можна буде побачити обидва проєкти. Один із проєктів є активним, а інші пасивні. Щоб змінити активний проєкт треба в браузері рішень натиснути правою кнопкою на рядку з проєктом та в меню вибрати пункт «Призначити проєктом, що запускається».

Для тестування програми потрібно спочатку проєкт для серверу зробити активним та запустити його в режимі «без налагодження». Потім треба проєкт для клієнта зробити активним та запустити його в режимі «без налагодження».

У якості приклада використаємо TCP, щоб забезпечити впорядковані, надійні двосторонні потоки байтів. Побудуємо завершену програму, що включає клієнт та сервер. Спочатку розглянемо, як розробити на потокових сокетах TCP сервер, а потім клієнтську програму для тестування нашого сервера.

Нижче наведено програму для сервера, який отримує запити на з’єднання клієнтів. Сервер побудований синхронно, отже виконання потоку блокується, поки сервер не дасть згоди на з’єднання з клієнтом. Ця програма демонструє простий сервер, який відповідає клієнту. Клієнт завершує з’єднання, надсилаючи серверу повідомлення <TheEnd>.

Основні класи для роботи з сокетами в .NET:

- **Socket** – забезпечує базову функціональність сокета для застосунку.
- **TcpClient** – побудований на класі Socket, щоб забезпечити TCP обслуговування на більш високому рівні. TcpClient надає декілька методів для відправки та отримання даних мережею.
- **TcpListener** – побудований на низькорівневому класі Socket. Його основне призначення – серверні застосунки. Він очікує вхідні запити на з’єднання від клієнтів та повідомляє застосунки про будь-які з’єднання.
- **UdpClient** – призначений для реалізації UDP обслуговування.
- **SocketException** – цей виняток генерується, коли у сокеті виникає помилка.

Базовим класом у побудові мережних додатків є клас **System.Net.Sockets.Socket**. Основні властивості цього класу:

- AddressFamily – Сімейство адрес сокета (значення із перерахунку Socket.AddressFamily).
- Available – Об’єм даних, які можна зчитати.
- Blocking – Чи знаходиться сокет в блокуючому режимі.
- Connected – Чи з’єднаний сокет з віддаленим хостом.
- LocalEndPoint – Локальна кінцева точка сокета.
- ProtocolType – Тип протокола сокета.
- RemoteEndPoint – Віддалена кінцева точка сокета.
- SocketType – Тип сокета.

Основні методи цього класу:

- Accept() – Створює новий сокет для обробки вхідного запиту на зєднання.
- Bind() – Звязує сокет з локальною кінцевою точкою для очікування вхідних запитів на зєднання.
- Close() – Закриває сокет.
- Connect() – Встановлює з’єднання з віддаленим хостом.
- Listen() – Переводить сокет в режим прослуховування.
- Receive() – Отримує дані від приєднаного сокета.
- Poll() – Визначає статус сокета.

- Send() – Відправляє дані з’єднаному сокету.
- ShutDown() – Забороняє операції відправки та отримання даних на сокеті.

TCP сервер, побудований з використанням класу Socket

```
// SocketServer.cs
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;

namespace SocketServer
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.OutputEncoding = System.Text.Encoding.Unicode;
         Console.InputEncoding = System.Text.Encoding.Unicode;

         // Встановлюємо для сокету 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 кінцевої точки та слухаємо вхідні сокети
         try
         {
            sListener.Bind(ipEndPoint);
            sListener.Listen(10);

            // Починаємо слухати з’єднання
            while (true)
            {
               Console.WriteLine("Очікуємо з’єднання через порт {0}", ipEndPoint);

               // Програма припиняється, очікуючи на вхідне з’єднання
               Socket handler = sListener.Accept();
               string data = null;

               // Ми дочекалися клієнта, який намагається з нами з’єднатися

               byte[] bytes = new byte[1024];
               int bytesRec = handler.Receive(bytes);

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

               // Показуємо дані на консолі
               Console.Write("Отриманий текст: " + data + "\n\n");

               // Відправляємо відповідь клієнту
               string reply = "Дякуємо за запит у " + data.Length.ToString()
                     + " символів";
```

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

                if (data.IndexOf("<TheEnd>") > -1)
                {
                   Console.WriteLine("Сервер завершив з'єднання з клієнтом.");
                   break;
                }

                handler.Shutdown(SocketShutdown.Both);
                handler.Close();
             }
          }
          catch (Exception ex)
          {
             Console.WriteLine(ex.ToString());
          }
          finally
          {
             Console.ReadLine();
          }
       }
    }
}
```

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

Клас Dns надає методи, що повертають інформацію про мережеві адреси, які підтримуються пристроєм у локальній мережі. Якщо пристрій локальної мережі має більше однієї мережної адреси, клас Dns повертає інформацію про всі мережеві адреси, і програма повинна вибрати з масиву відповідну адресу для обслуговування.

Клас IPEndPoint для сервера комбінує першу IP-адресу хост-комп'ютера, отриману від методу Dns.Resolve(), з номером порту. Тут клас IPEndPoint представляє localhost на порті 11000. Новим екземпляром класу Socket з локальною кінцевою точкою для очікування з'єднань, створюється потоковий сокет.

Метод Bind() пов'язує сокет із локальною кінцевою точкою. Викликати метод Bind() треба до будь-яких спроб звернення до методів Listen() і Accept(). Створивши сокет і зв'язавши з ним ім'я, можна слухати вхідні повідомлення, скориставшись методом Listen(). У стані прослуховування сокет чекатиме на вхідні спроби з'єднання.

Як тільки клієнт і сервер встановили між собою з'єднання, можна надсилати та отримувати повідомлення, використовуючи методи Send() та Receive() класу Socket. Коли обмін даними між сервером та клієнтом завершується, потрібно закрити з'єднання використовуючи методи Shutdown() та Close().

TCP клієнт, побудований з використанням класу Socket

```
// SocketClient.cs
using System;
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());
            }
            finally
            {
                Console.ReadLine();
            }
        }

        static void SendMessageFromSocket(int port)
        {
            Console.OutputEncoding = System.Text.Encoding.Unicode;
            Console.InputEncoding = System.Text.Encoding.Unicode;

            // Буфер для вхідних даних
            byte[] bytes = new byte[1024];

            // З’єднуємося з віддаленим пристроєм

            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);

            // Встановлюємо віддалену точку для сокету
            sender.Connect(ipEndPoint);

            Console.Write("Введіть повідомлення: ");
            string message = Console.ReadLine();

            Console.WriteLine("Сокет з’єднується з {0} ", sender.RemoteEndPoint.ToString());
            byte[] msg = Encoding.UTF8.GetBytes(message);

            // Надсилаємо дані через сокет
            int bytesSent = sender.Send(msg);

            // Отримуємо відповідь від сервера
            int bytesRec = sender.Receive(bytes);

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

            // Використовуємо рекурсію для неодноразового виклику SendMessageFromSocket()
```

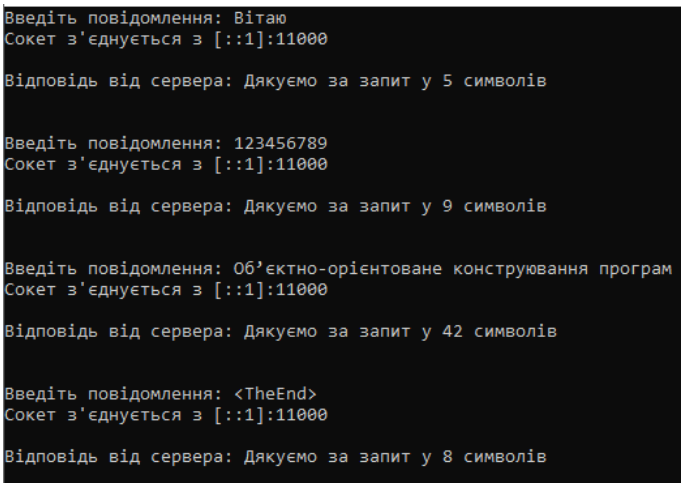
```
            if (message.IndexOf("<TheEnd>") == -1)
                SendMessageFromSocket(port);

            // Звільняємо сокет
            sender.Shutdown(SocketShutdown.Both);
            sender.Close();
        }
    }
}
```

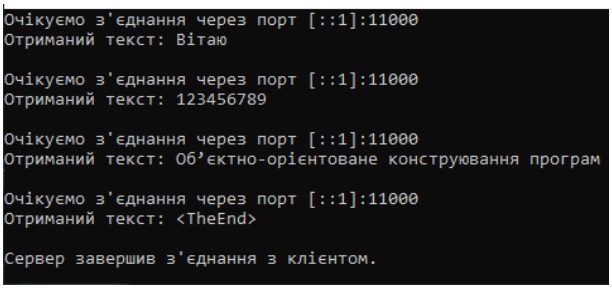
Функції, які використовуються для створення програми-клієнта, нагадують серверну програму. Як і для сервера, використовуються ті ж методи для визначення кінцевої точки, створення екземпляра сокету, надсилання та отримання даних і закриття сокету. Метод Connect() використовується для з'єднання з віддаленим сервером.

Результат тестування програми

Робота клієнта:



Робота сервера:



ЗАВДАННЯ

Реалізувати клієнтський та серверний застосунки на C# для передачі даних із застосуванням протоколу TCP. З'єднання між клієнтом та сервером підтримувати до того часу, поки сервер не отримає від клієнта повідомлення про завершення. Передбачити обробку помилок введення даних користувачем. Реалізувати графічне середовище для користувачів (довільним чином). Виводити повідомлення з підказками на кожному етапі виконання програми.

Варіанти:

1	Клієнт може відправляти на сервер слова, сервер зберігає ці слова. Слова зберігати без повторів: якщо надіслане слово вже є в наборі слів, то сервер повертає клієнту відповідне повідомлення. Передбачити можливість за запитом клієнта: Reset – видалити усі слова, Show – з сервера на клієнт посилається список усіх слів через пробіл, Count – з сервера на клієнт посилається кількість слів, Delete – з клієнта на сервер посилається слово, яке треба видалити.
2	Розробити просту програму-тестування. Сервер має список з 20 тестових питань, на які можна дати відповіді «так» (Y) чи «ні» (N). Після підключення клієнта сервер випадковим чином вибирає 5 питань, по черзі задає питання ці питання клієнту та підраховує кількість правильних відповідей. Після закінчення тестування надсилає на клієнту кількість правильних відповідей.
3	Реалізувати можливість отримання даних від сервера про поточну дату і час з врахуванням часового поясу клієнта (UTC). Реалізувати можливість на вимогу клієнта зміни часового поясу клієнта, вибору формату дати і часу, які надсилає сервер.
4	Розробити просту програму «електронна відомість». Реалізувати можливість за запитом клієнта-викладача: додати ПІП студента та його оцінку в базу, змінити оцінку вказаного студента, видалити студента, повернути список усіх студентів і їх оцінок.
5	Сервер зберігає інформацію про товари: найменування, категорія і ціна. Реалізувати можливість за запитом клієнта: знайти товар за найменуванням і повернути клієнту його ціну, повернути клієнту список усіх товарів з вказаної категорії.
6	Створити елемент скриптової мови. Клієнт відправляє команди серверу: «var = value» – задати змінній var значення value (наприклад, «a = 5»); «var» – повернути клієнту значення змінної var; «var1 + var2» – вивести на консоль суму значень змінних var1 та var2 (аналогічно для різниці, добутку та частки). Всі змінні повинні зберігатися на сервері. Якщо змінної на сервері немає, то повернути відповідне повідомлення про помилку. Після розірвання з'єднання всі змінні з сервера видаляються.
7	Розробити просту програму-опитування. Сервер має список з 3 питань, які по черзі задає клієнту, та збирає відповіді. Якщо клієнт хоче переглянути усі результати опитувань (результати усіх користувачів), то сервер запитує пароль. Якщо пароль правильний, то повертає клієнту результати усіх опитувань. Якщо ні, то розриває з'єднання.
8	Розробити просту програму «нотатки». На сервері зберігається до 5 нотаток користувача (якесь коротке текстове повідомлення). Реалізувати можливість клієнту додати нотатку, переглянути всі свої нотатки, видалити вибрану нотатку, видалити усі свої нотатки. Якщо збережено 5 нотаток, то клієнт не може додати нову нотатку, поки не видалить якусь зі старих.
9	Розробити просту програму для роботи з набором чисел. Відправляти цілі числа з клієнта на сервер, сервер зберігає ці числа. Передбачити можливість за запитом клієнта: Sortup – з сервера на клієнт посилається список усіх чисел через кому в порядку зростання, Sortdown – в порядку спадання, Sum – на клієнт посилається сума, Mean – середнє арифметичне чисел, Reset – усі числа на сервері видаляються, Delete – з клієнта на сервер посилається число, яке треба видалити.

Додаткова література

1. Електронний ресурс. Джерело доступу: https://www.c-sharpcorner.com/article/socket-programming-in-C-Sharp/
2. Електронний ресурс. Джерело доступу: https://metanit.com/sharp/net/