BEÁGYAZOTT RENDSZEREK ALAPJAI

Socket programozás alapjai .NET MF platformon

A laborfoglalkozás célja

- A foglalkozás célja bemutatni a .NET Microframework hálózati programozását emulátor segítségével:
  - Socket osztály
  - TCP és UDP kapcsolat

Socket osztály

- A System.Net névtér osztályainak célja egyszerűsíteni a hálózati programozást
- Maga a Socket osztály a Socket API-t implementálja, melyet az 1980-as években specifikáltak
- A Socket egy kapcsolati végpontot reprezentál, biztosítva az adatok küldését és fogadását szerver és kliens oldalról is
- Az alkalmazások közti hálózati kapcsolat létrehozásához szükségünk van a szerver IP címére és egy előre definiált, egyezményes port számra
- A .NET Micro Framework Socket osztálya az eredeti .NET Framework-ben található implementáció csonkított változata (pl. az aszinkron adatcserét biztosító metódusok kimaradtak)

Socket osztály használata

- A Socket inicializálását a konstruktor segítségével lehet elvégezni a példányosítás során:

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

  - AddressFamily.InterNetwork: a címfeloldás módszerét definiálja. Internet, illetve LAN esetén InterNetwork-öt kell megadni (IPv4 támogatás).
  - SocketType.Stream: kétirányú, összeköttetéses adatfolyam kapcsolat
  - ProtocolType.Tcp: A használni kívánt protokoll (jelen esetben TCP)
- Használat után mindig zárjuk le a Socket-et

Socket osztály használata

- Csatlakozás:

```
Socket clientSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp)

IPHostEntry entry = Dns.GetHostEntry("192.168.100.1");
IPAddress ipAddress = entry.AddressList[0];
IPEndPoint serverEndPoint = new IPEndPoint(ipAddress, 2000);

clientSocket.Connect(serverEndPoint);
```

- A Dns.GetHostEntry metódust használjuk a hoszt név/ip cím feloldásra, mely visszaad legalább egy IPAddress objektumot az AddressList tömb segítségével. Ha a paraméterben üres string-et adunk meg, akkor a lokális ip címet kapjuk vissza.
- Az IpAddress objektum segítségével létrehozható egy IPEndPoint objektum, a port szám definiálásával. Ez már felhasználható a csatlakozás megkezdéséhez.
- A Connect metódus hívásával lehet csatakozni a paraméterben megadott végponthoz

Socket osztály használata

- A küldés és fogadás során bájt tömböket kell használni
- Szöveges adatok küldése:

```
byte[] messageBytes = Encoding.UTF8.GetBytes("Hello World!");
clientSocket.Send(messageBytes);
```

- Szöveges adatok fogadása:

```
byte[] inBuffer = new byte[100];
int count = clientSocket.Receive(inBuffer);
char[] chars = Encoding.UTF8.GetChars(inBuffer);
string str = new string(chars, 0, count);
```

- A fogadás egy megfelelő méretű buffer-be történhet. A Receive metódus visszatérési értékként megadja a fogadott bájtok számát.
- A Send és a Receive metódusok is blokkolnak, aszinkron adatküldés nincs

Socket osztály használata

- Ha nem akarunk fix méretű tömböt alkalmazni a fogadásnál, akkor a Socket osztály Available tulajdonságát használhatjuk, mely megadja, hogy pillanatnyilag hány bájt van a buffer-ben
- Olvasás előtt lehetőség van a socket státuszát lekérdezni („polling”):

```
if (communicationSocket.Poll(-1, //timeout in microseconds (-1 = infinite)
SelectMode.SelectRead))
{
    // A socket olvasható, adatok olvasása...
}
```

- Az első paraméterben megadható, hogy maximálisan mennyi időt hagyunk a válaszra (-1 = végtelen), a második paraméterben pedig azt a tulajdonságot adjuk meg, amelynek az állapotát le akarjuk kérdezni:

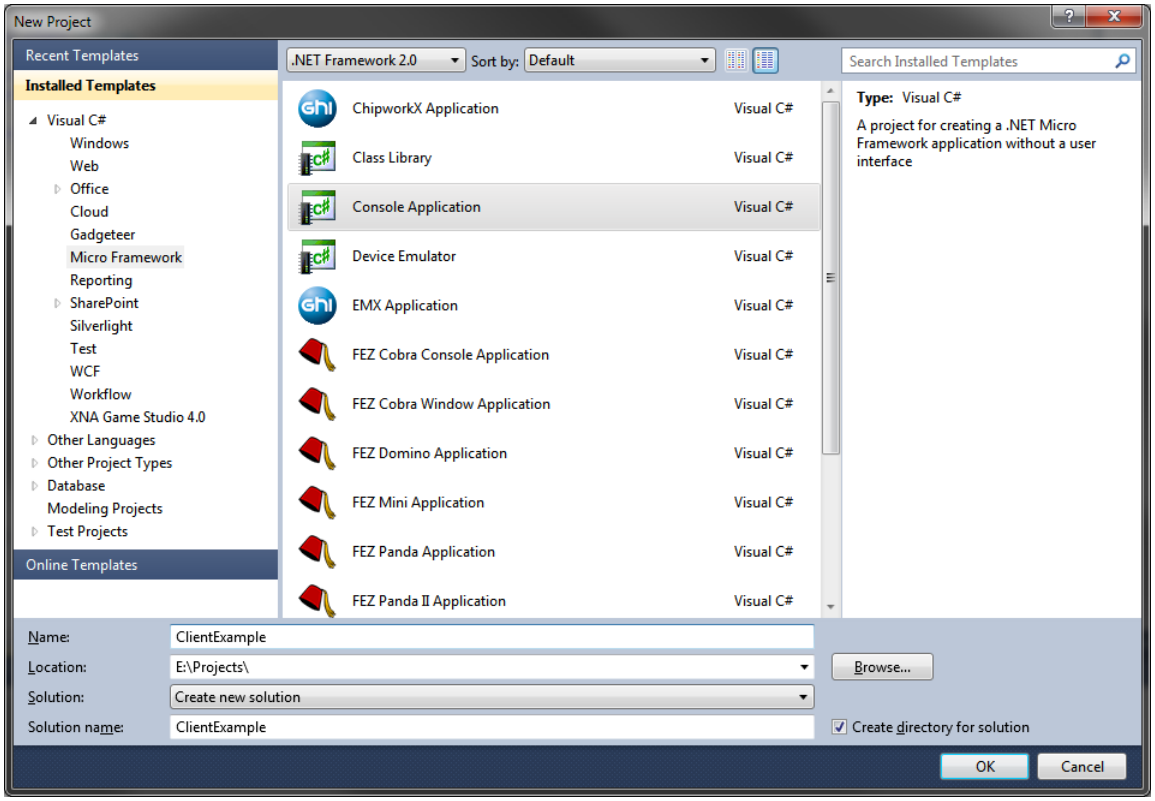
```
public enum SelectMode
{
    SelectRead = 0,
    SelectWrite = 1,
    SelectError = 2,
}
```

I. Feladat

- Készítsünk egy egyszerű kliens és egy szerver alkalmazást Socket használatával:
  - A kliens küldjön szöveges adatot a várakozó szerver számára
  - Használjunk TCP/IP protokollt

Kliens alkalmazás készítése

- Hozzunk létre egy új .NET MF konzol alkalmazást a kliens alkalmazás számára:

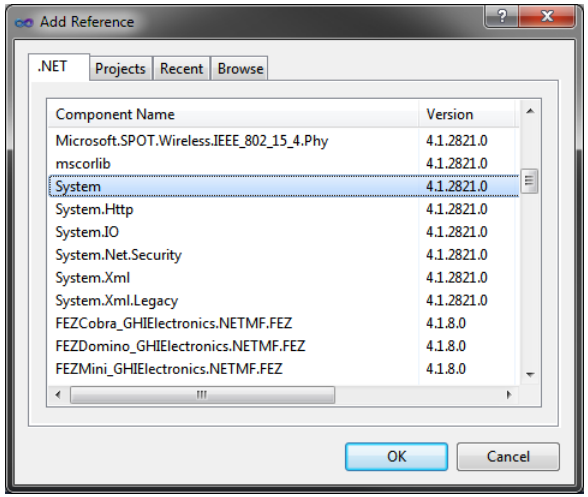


Kliens alkalmazás készítése

- A következő program keretet kapjuk:

```
public class Program
{
    public static void Main()
    {
        Debug.Print(
            Resources.GetString(Resources.StringResources.String1));
    }

}
```

- A System.dll-t fel kell venni a referenciák közé, mert ebben található a System.Net névtér:
  - References/Add Reference…



Kliens alkalmazás készítése

```
public class Program
{
    private const string dottedServerIPAddress = "127.0.0.1";
    private const int port = 2000;

    public static void Main()
    {
        // A socket példányosítása
        using (Socket clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
               ProtocolType.Tcp))
        {
            // Cím beállítása DNS feloldással
            IPHostEntry entry = Dns.GetHostEntry(dottedServerIPAddress);
            IPAddress ipAddress = entry.AddressList[0];
            IPEndPoint serverEndPoint = new IPEndPoint(ipAddress, port);
            // Csatlakozás
            Debug.Print("Connecting to server " + serverEndPoint + ".");
            clientSocket.Connect(serverEndPoint);
            Debug.Print("Connected to server.");
            // Adatfolyam küldése
            byte[] messageBytes = Encoding.UTF8.GetBytes("Hello World!");
            clientSocket.Send(messageBytes);
        }
    // Itt meghívodik a Dispose, ami lezárja a Socket-et, és felszabadítja az erőforrásokat.
    }
}
```

Socket osztály használata

- Szerver oldali socket létrehozása esetén meg kell adnunk, hogy mely kliensek csatlakozhatnak ahhoz, erre szolgál a Bind metódus:

```
public void Bind(EndPoint localEP);
```

- Egy IPEndPoint objektum átadásával megadható a fogadni kívánt kliens(ek) IP címe, illetve az a portszám, amin a szerver „figyelni” fog
- Amennyiben címnek IPAddress.Any-t adunk meg, bármely klienst fogadja a szerver a megadott porton

```
public class IPAddress
{
    public static readonly IPAddress Any;
    public static readonly IPAddress Loopback;

    public IPAddress(byte[] newAddressBytes);
    public IPAddress(long newAddress);

    public override bool Equals(object obj);
    public byte[] GetAddressBytes();
    public static IPAddress Parse(string ipString);
    public override string ToString();
}
```

Socket osztály használata

- A Listen metódus hívását követően a szerver figyelni fog a csatlakozni kívánó kliensekre (listen mode). Ez a metódus a háttérben, aszinkron módon fut.
- A Listen után meg kell hívni az Accept metódust, amely blokkolja a program futását (szinkron) egy kliens csatlakozásáig. Ennek backLog paraméterével szabályozható a kliensek maximális száma. A csatlakozást követően a metódus visszatér a klienst reprezentáló Socket objektummal.

```
Socket listeningSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                                    ProtocolType.Tcp);
EndPoint endPoint = new IPEndPoint(IPAddress.Any, 80);
listeningSocket.Bind(endPoint);
listeningSocket.Listen(1);
Socket communicationSocket = listeningSocket.Accept();
```

- Ha több klienst szeretnénk kezelni, akkor az Accept többször is hívható

Szerver alkalmazás készítése

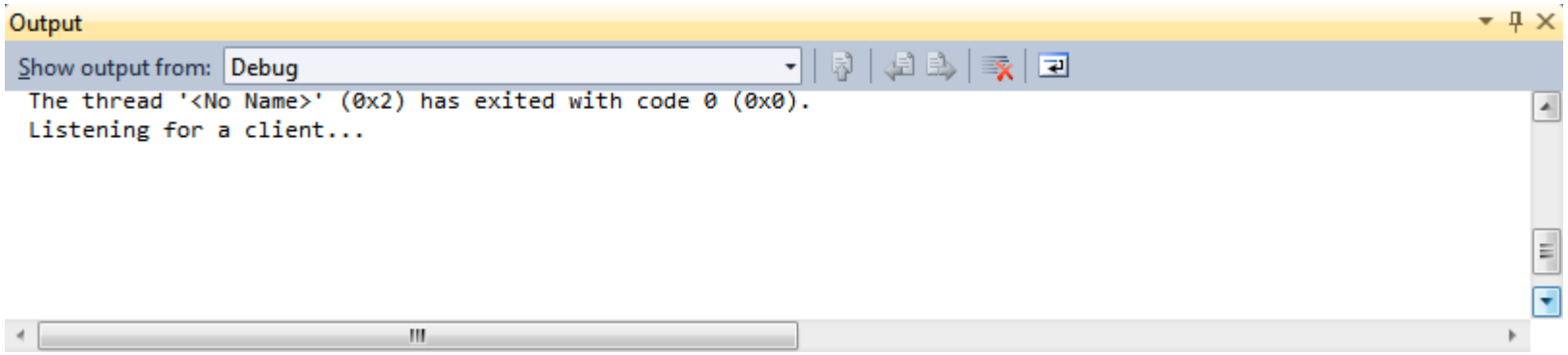
- Az előzőekben bemutatott módon készítsünk egy új „Console Application” projektet a szerver számára
- Ne felejtsük el hozzáadni a System.dll referenciát a projekthez!

Szerver alkalmazás készítése

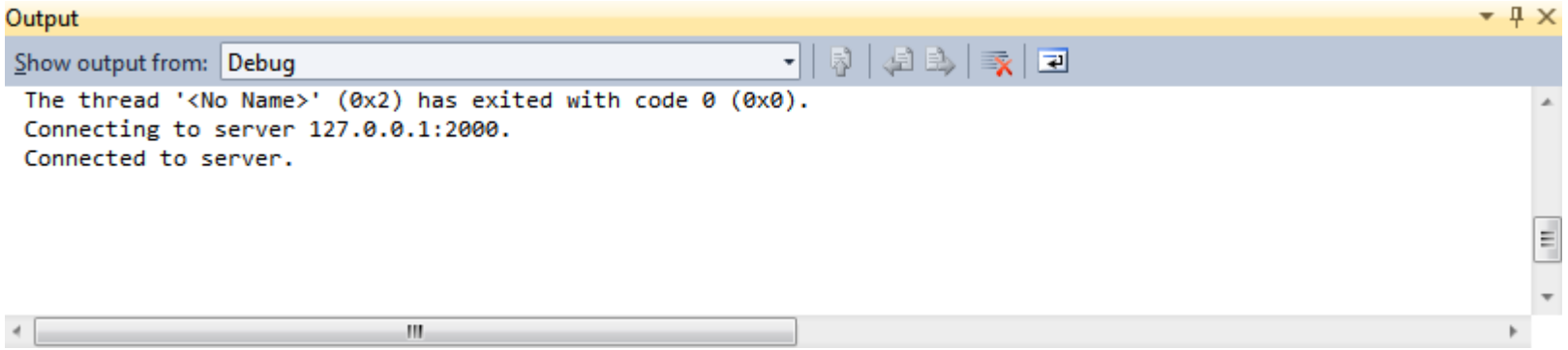
```
public class Program
{
    private const int port = 2000;

    public static void Main()
    {
        using (Socket listeningSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
               ProtocolType.Tcp))
        {
            // Végpont hozzárendelése a Socket-hez (bármely cím elfogadása).
            listeningSocket.Bind(new IPEndPoint(IPAddress.Any, port));
            Debug.Print("Listening for a client...");
            // Várakozás maximum 1 kliensre.
            listeningSocket.Listen(1);
            using (Socket communicationSocket = listeningSocket.Accept())
            {
                Debug.Print("Connected to client.");
                // Várakozás válaszra a klienstől.
                if (communicationSocket.Poll(-1, SelectMode.SelectRead))
                {
                    byte[] inBuffer = new byte[communicationSocket.Available];
                    int count = communicationSocket.Receive(inBuffer);
                    string message = new string(Encoding.UTF8.GetChars(inBuffer));
                    Debug.Print("Received '" + message + "'.");
                }
            }
        }
    }
}
```

Tesztelés

- Indítsuk el először a szerver alkalmazást!
  - Az Output ablakban a következőnek kell megjelennie:

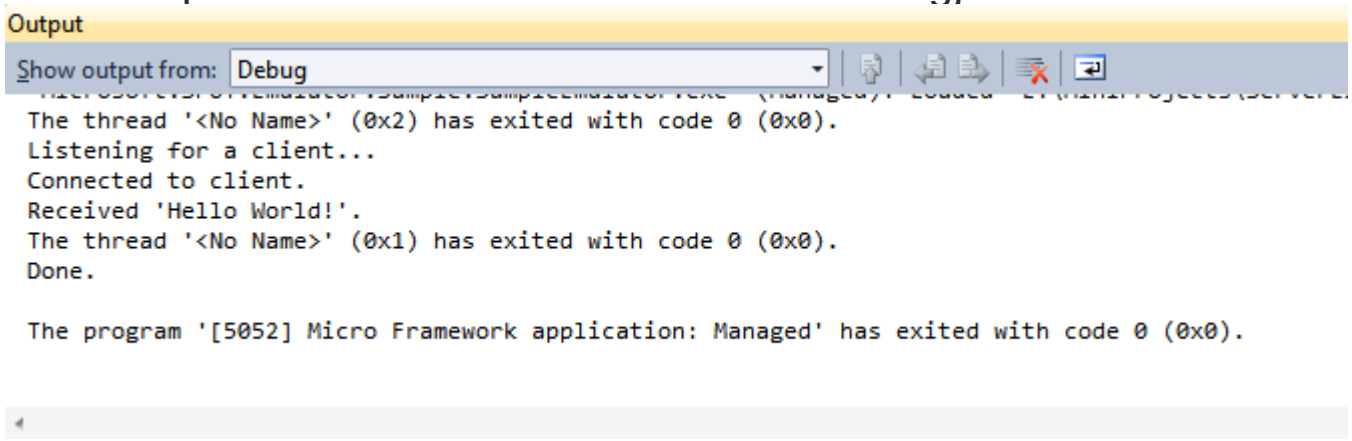


- Ezután indítsuk el a klienst!
  - Az Output ablakban a következőnek kell megjelennie:



Tesztelés

- Váltsunk át a szerver Output ablakára!
  - Az Output ablakban a következőnek kell megjelennie:



II. Feladat

- Az előző feladatot készítsük el UDP protokoll használatával!

UDP protokoll

- Összeköttetés mentes protokoll
- Egy csomag többször is megérkezhet, illetve azok el is veszhetnek, a protokoll nem biztosítja a sorrendtartó, hibamentes átvitelt
- Valósidejű alkalmazásoknál használják (Voice over IP, közvetítés, stb.)
- Broadcast, illetve multicast címzési mód

Kliens alkalmazás (UDP)

```
public class Program
{
    private const string dottedServerIPAddress = "127.0.0.1";
    private const int port = 2000;

    public static void Main()
    {
        using (Socket clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
               ProtocolType.Udp))
        {
            // Cím beállítása DNS feloldással
            IPHostEntry entry = Dns.GetHostEntry(dottedServerIPAddress);
            IPAddress ipAddress = entry.AddressList[0];
            IPEndPoint serverEndPoint = new IPEndPoint(ipAddress, port);
            // Adatok küldése
            byte[] messageBytes = Encoding.UTF8.GetBytes("Hello World!");
            clientSocket.SendTo(messageBytes, serverEndPoint);
        }
    }
}
```

Szerver alkalmazás (UDP)

```
public class Program
{
    private const int port = 2000;

    public static void Main()
    {
        using (Socket serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
               ProtocolType.Udp))
        {
            EndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, port);
            serverSocket.Bind(remoteEndPoint);
            if (serverSocket.Poll(-1, SelectMode.SelectRead))
            {
                byte[] inBuffer = new byte[serverSocket.Available];
                int count = serverSocket.ReceiveFrom(inBuffer, ref remoteEndPoint);
                string message = new string(Encoding.UTF8.GetChars(inBuffer));
                Debug.Print("Received '" + message + "'.");
            }
        }
    }
}
```

TCP vs. UDP

- Összeköttetés mentesség:
  - Nem volt szükséges a kapcsolódás (Connect metódus) kliens oldalról
  - Nem kellett Listen és Accept metódust hívni szerver oldalon
- Működésbeli eltérés:
  - A Socket példányosításakor SocketType.Dgram-ot adtunk át paraméterként SocketType.Stream helyett (ez az összeköttetés mentességből is következik)
  - Küldéshez a SendTo, fogadáshoz pedig a ReceiveFrom metódust használtuk. A ReceiveFrom metódus eltárolja a távoli végponthoz rendelt Socket-et a megadott paraméter segítségével (ref remoteEndPoint).

III. Feladat

1. Készítsünk egy szervert, amelyik két számot vár egy klienstől, majd a két szám összegét visszaküldi annak. A kliens oldali konzolon (Output window) jelenjen meg az eredmény. Az adatcseréhez használjunk szöveges formátumot, pl. a két számot küldjük vesszővel elválasztva.

Irodalomjegyzék

- Jens Kühner: Expert .NET Micro Framework, Apress, April 28, 2008, ISBN-10: 159059973X , ISBN-13: 978-1590599730