Drukuj Zamknij

LANCHAT - Korzystanie z sieci na przykładzie transmisji UDP

2003-12-14 20:07:51
Maciej Paczewski - Wydział Informatyki, Politechnika Szczecińska

Do komunikacji w sieci LAN (Local Area Network) pomiędzy dwoma użytkownikami potrzebowalibyśmy programu, który prześle interesujące nas dane drugiemu użytkownikowi. Może to być program, który utworzy „sztywne" połączenie pomiędzy portem naszego komputera a portem innego komputera w sieci. Korzystalibyśmy z protokołu TCP/IP. Dzięki IP nasze dane trafią do komputera o odpowiednim numerze IP. TCP (Trasmission Control Protocol) dba o to, czy aby na pewno nasze dane dotarły do odbiorcy oraz o to by po odebraniu ich nadal były w odpowiedniej kolejności. Wszystko byłoby świetnie, gdyby nie to, iż w zwykle rozmawia ze sobą duża ilość użytkowników i ustanawianie połączenia każdy z każdym w tym przypadku nas nie interesuje. Oczywiście, do przeprowadzenia rozmowy prywatnej należałoby takie połączenie nawiązać, a dla zwiększenia bezpieczeństwa strumień danych zaszyfrować. Lecz w tym artykule chciałbym poruszyć problem nawiązania połączenia pomiędzy wieloma użytkownikami w jak najprostszy i najszybszy sposób.

Dlaczego UDP i multicasting ?

UDP (User Datagram Protocol) jest protokołem sieciowym pozbawionym mechanizmu, który pozwala na kontrolowanie doręczenia czy powielania informacji. Ma to swe wady, ale i zalety. UDP z założenia korzysta z protokołu IP. Poza tym, w dużym uproszczeniu jest to protokół, który nie wymaga ustanowienia „sztywnego" połączenia pomiędzy użytkownikami, co więcej w protokole tym nie ma mowy o czymś takim jak stan połączenia użytkowników. Pakiety danych są małe (nagłówek ma 8 bajtów), a szybkość doręczenia zależy przede wszystkim od szybkości wygenerowania datagramu oraz dostępu do sieci. Umożliwia nam to po podłączeniu się do sieci (w naszym przypadku LAN) nadanie datagramu, który może zostać odebrany przez wszystkich użytkowników nasłuchujących na określonym przez nas porcie. Stosujemy tutaj rozwiązanie zwane transmisją *rozsiewczą* (multicasting), która to likwiduje potrzebę powielania strumienia danych podczas, np. wysyłania informacji do wielu użytkowników.

Program, opis i działanie

Jak już wcześniej wspomniałem, program ten będzie bardzo prostym chatem, dlatego też jest to aplikacja na konsolę. Rozwiązanie pozwala na skupieniu się na kodzie obsługującym sieć. W tym paragrafie omówię najważniejsze części programu. Nieomówione części kodu źródłowego są opatrzone komentarzami w pliku UDPtalk.cs.

Korzystać będziemy m.in. z przestrzeni nazw:

```
using System.Net;
using System.Threading;
```

BCL (Base Class Library) oferuje bogaty zbiór klas ułatwiających dostęp do sieci. Jest nią przestrzeń nazw **System.Net**, która zawiera wiele klas umożliwiających korzystanie z Cookies, Http, Proxy, Dns`ów... i wielu innych, szeroko używanych w aplikacjach sieciowych. Nie muszę chyba dodawać, że z tej przestrzeni nazw korzystać będziemy najwięcej. Przestrzeń nazw **System.Threading** udostępnia wiele klas umożliwiających zaimplementowanie programów, które będą wykonywały się wielowątkowo. W moim programie użyję wątków w najprostszy z możliwych sposobów, mianowicie w programie działać będzie wątek obsługujący nasłuchiwanie sieci na odpowiednim porcie (odbiór informacji) oraz wątek obsługujący wysyłanie wiadomości do innych użytkowników.

```
private static int odbior = 2048;
```

Jest to numer portu, z którego będziemy korzystać podczas wysyłania i odbierania naszych wiadomości (datagramów). Użyłem 2048, gdyż zgodnie z materiałami organizacji IANA ten port nie jest wykorzystywany przez usługi standardowe.

```
private static IPEndPoint ipep;
```

IPEndPoint zawiera adres grupy „multicast" oraz numer portu, na który wysyłać będziemy nasze datagramy.

```
private static string broadcast = "224.0.0.1";
```

W zasadzie jest to tylko string... lecz po konwersji jest to numer IP klasy D, który służy nam jako adres do transmisji multicasting. W zasadzie wszystkie adresy 224.0.0.1 – 239.255.255.255 do tego służą, ale warto podkreślić, iż te z zakresu 239.253.0.0 - 239.255.255.255 służą do transmisji lokalnej. Niestety w niektórych konfiguracjach systemowych czy sieciowych komunikacja dla podanego zakresu adresów do transmisji lokalnej nie działa zgodnie z naszymi oczekiwaniami.

Metoda Start:

```
public static void Start()
{
```

tutaj konstruujemy nowego UdpClient`a na porcie numer nadaj=2048:

```
udpcl = new UdpClient(nadaj);
```

także tworzymy IPEndPoint, który zawierać będzie adres naszej grupy oraz numer portu

```
ipep = new IPEndPoint(adresgrupy,odbior);
```

i dołączamy do MulticastGroup – kod umieszczony w bloku try-catch. Argumentem metody dołączającej naszego UdpClient`a do grupy multicast jest adres tej grupy (224.0.0.1)

```
udpcl.JoinMulticastGroup(adresgrupy);
}
```

Kolejna metoda jest metodą, która wykonywana jest w jednym z wątków i odpowiedzialna jest za nasłuchiwanie datagramów UDP na odpowiednim porcie:

```
public static void Nasluch()
{
```

najważniejszy element metody Nasłuch(), bezpośrednio odbierający wiadomość z portu i przekazujący ją do tablicy byte[]. Argument przekazywany przez referencję endpoint. Jest on ustawiony wiersz wcześniej na null, tak więc akceptuje wszystkie datagramy odbierane przez niego. Gdybyśmy chcieli się dowiedzieć kto dokładnie i z jakiego portu wysłał odebraną przez nas informację wystarczy wywołać po odebraniu wiadomości endpoint.ToString();

```
     byte[] data = udpcl.Receive(ref endpoint);
     po otrzymaniu tablicy byte[] należy dokonac konwersji do łańcucha
     string Sdata = Encoding.ASCII.GetString(data);
}
```

Metoda MojIP() służy do uzyskania nazwy hosta, oraz numeru IP hosta:

```
public static void MojIP()
{
```

uzyskujemy nazwę hosta korzystając z metody:

```
nazwahosta = Dns.GetHostName();
```

natomiast korzystając z następujących metod otrzymujemy adres IP naszego host`a:

```
    IPHostEntry h = Dns.Resolve(nazwahosta);
    IPAddress adres = h.AddressList[0];
}
```

W metodzie static void Main(string[] args)wywołujemy najpierw metody MojIP() oraz Start(). Teraz nasz program gotów jest na coś nowego, czyli stworzenie i uruchomienie drugiego wątku, który będzie nasłuchiwał na porcie 2048:

Korzystamy z konstruktora wątku (Thread Constructor) w celu zainicjowania nowego wątku. Jedynym parametrem konstruktora nowego wątku jest delegat ThreadStart.

```
Thread watek = new Thread(new ThreadStart(Nasluch));
```

Po uruchomieniu wykonywania wątku metodą watek.Start(); rozpoczyna się wykonywanie metody Nasluch ().

Dalej w pętli while(!zamykanie) umieszczony jest kod odpowiedzialny za odczytywanie tekstu z konsoli, konwertowanie go do datagramu i wysyłanie do grupy odbiorców. Odbywa się to odwrotnie do algorytmu zawartego w metodzie Listen(), a mianowicie:

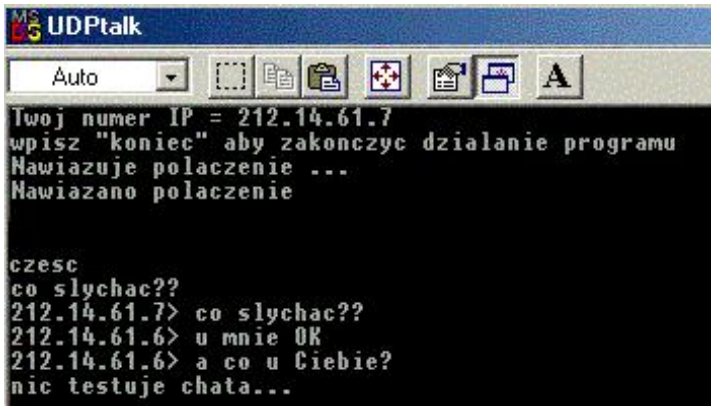
```
datagram = s.ToCharArray();// 1
int dlugosc = Encoding.ASCII.GetBytes(datagram,0,s.Length,bufores,0); // 2
udpcl.Send(bufores,dlugosc,ipep); // 3
```

Metody kolejno służą do: 1. konwersja do tablicy znaków 2. konwersja tablicy znaków datagram do tablicy byte bufores oraz obliczenie jej długości 3. metoda ta stanowi kluczowy moment wysłania naszej wiadomości. Korzystając z IPEndPoint`a ipep, gdzie zawarty jest numer portu oraz adres nadawania „ubieramy" nasze dane z buforu w nagłówek i wysyłamy.

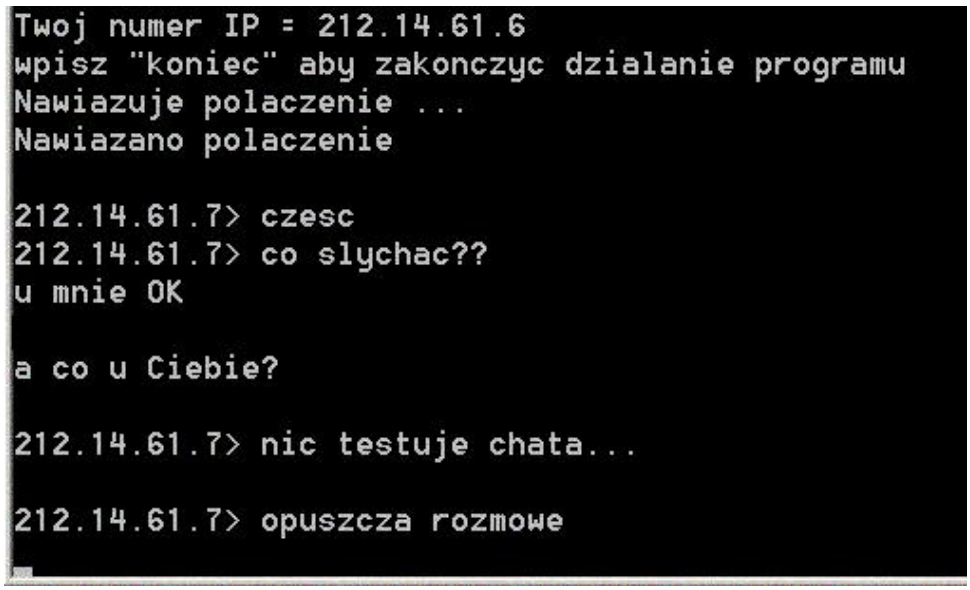
Po zakończeniu wykonywania pętli while(!zamykanie) następuje wywołanie metody watek.Abort(); Wywołanie jej powoduje rozpoczęcie procesu kończenia działania wątku i prowadzi do jego zakończenia. A następnie watek.Join(); czyli zablokowanie wątku wywołującego do czasu zakończenia wątku Thread watek.

PRZYKŁADOWA SESJA

użytkownik nr 1:



użytkownik nr 2:



PODSUMOWANIE

Jak widać Base Class Library platformy .NET umożliwia w prosty sposób napisanie programu sieciowego. Ten omówiony przeze mnie jest to tylko prosty przykład jak łatwo uzyskać dostęp do sieci korzystając z BCL. Co więcej w przestrzeni nazw System.Net jest 37 klas oferujących różne poziomy abstrakcji od implementacji schematu żądanie-odpowiedź do bezpośredniego dostępu do gniazd.

Drukuj Zamknij