Primo esempio di trasmissione/ricezione via TCP con i Socket

L'argomento che verrà trattato ora è un po' più complesso ma la maggior parte delle comunicazioni in rete avviene utilizzando i famosi

Socket

Il SOCKET è un oggetto software che consente la trasmissione o ricezione in rete di informazioni. Esso viene sempre utilizzato attraverso la coppia

indirizzo IP ed una porta

Come già sappiamo, un indirizzo ip identifica un host in una rete ed è composto da quattro ottetti ognuno di 8 bit (*32 bit*), esempio:

192.168.200.30
oppure
172.16.1.2

il primo identifica un host all'interno di una rete di classe **C**, mentre il secondo un host all'interno di una rete di classe **B**.

Ma se volessi comunicare (*trasmettere o ricevere*) dati con uno di questi hosts, ho la necessità di aprire un canale di comunicazione con esso, cioè dovrò comunicare attraverso la cosiddetta

porta

che è identificata da un numero intero di 16 bits, quindi avrò a disposizione ben

65536 porte.

In realtà non possiamo utilizzare tutte le porte, infatti le porte da 0 a 1023 sono chiamate

'Well-known ports'

cioè porte 'ben conosciute' perché utilizzate già da altri software (o servizi).

Ad esempio, quando ci connettiamo ad un sito web con il browser, non facciamo altro che connetterci ad un server web che ha un indirizzo IP preciso e che risponderà alla richiesta di una pagina web sulla sua porta

80 se il sito usa il protocollo http o
443 se il sito usa il protocollo https

Quindi queste porte, cosi come tante altre, sono riservate e non possono essere utilizzate.

*[1] Approfondimento vedi nota a piè pagina

Invece se volessimo trasmettere dei messaggi con una nostra applicazione, abbiamo la necessità di utilizzare una porta di comunicazione superiore alla 1024 e sapere l'IP dell'host destinatario dei messaggi.

Questo è il codice della applicazione a console che riceverà i messaggi trasmessi da un'altra applicazione:

```
namespace SocketServer
{
    class Program
    {
        static void Main(string[] args)
        {
            EseguiServer();
            Console.ReadKey();
        }

        public static void EseguiServer()
        {
            Socket listener = new Socket(AddressFamily.InterNetwork,SocketType.Stream, ProtocolType.Tcp);
            try
            {
               listener.Bind(new IPEndPoint(IPAddress.Any, 3306));
               listener.Listen(10);
               Console.WriteLine("Attesa di connessioni ... ");

               byte[] bytes = new Byte[1024];
               string data = null;
               Socket clientSocket = listener.Accept();

               while (true)
                {
                     int numByte = clientSocket.Receive(bytes);
                     data = Encoding.ASCII.GetString(bytes,0, numByte);

                     if (data.IndexOf("@") > -1)
                            break;

                     Console.WriteLine("Testo ricevuto -> {0} ", data);
                     byte[] message = Encoding.ASCII.GetBytes("OK messaggio ricevuto!!!");
                     clientSocket.Send(message);
                     numByte = 0;
                }
                clientSocket.Shutdown(SocketShutdown.Both);
                clientSocket.Close();
                listener.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }
    }
}
```

Nel metodo **`Main`** (entry point) compaiono solo due istruzioni, la prima richiama il metodo

EseguiServer()

che conterrà tutta la logica del programma e rimane in ascolto e visualizza tutti i messaggi provenenti dall'applicazione trasmittente fino a quando il client non trasmette il carattere '**@**' per indicare la fine della trasmissione. Di seguito la spiegazione.

[1] https://it.wikipedia.org/wiki/Porte_TCP_e_UDP_standard#Da_0_a_1023_(Well-known_ports)

Spiegazione:

`Socket listener = new Socket(`
`AddressFamily.InterNetwork,`
`SocketType.Stream,`
`ProtocolType.Tcp`
`);`
	Creo il primo oggetto di tipo Socket al cui costruttore passo 3 parametri di tipo enumeratore[2]:
	1. AddressFamily.InterNetwork
	2. SocketType.Stream
	3. ProtocolType.Tcp
	Il primo rappresenta la tipologia di indirizzi che riguardano le reti[3] e nella fattispecie quelli relativi all'IPV4.
	Il secondo indica che vogliamo usufruire di un flusso sulla rete[4].
	Il terzo indica il tipo di protocollo utilizzato[5].

`try`
`{`
`....`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e.ToString());`
`}`
	Blocco try…catch per gestire le eventuali eccezioni.

`listener.Bind(new IPEndPoint(IPAddress.Any, 3306));`	Associa (**Bind**) al socket un indirizzo IP locale passato come oggetto nel costruttore. L'oggetto di tipo IPEndPoint rappresenta un indirizzo IP ed una porta. Il primo parametro del costruttore di tipo enumeratore, significa che viene associato un qualsiasi (**Any**) indirizzo IP delle eventuali schede di rete (NIC) presenti sulla macchina a seguito di una richiesta di connessione su una di queste. Il secondo parametro è la porta di ascolto.
`listener.Listen(10);`	Mette il socket in stato di attesa. Il parametro numerico specifica il numero di connessioni in ingresso che possono essere accodate per l'accettazione di una connessione.
`byte[] bytes = new Byte[1024];` `string data = null;`	Rappresenta un array di bytes come buffer per la ricezione dei messaggi. Creo una variabile string 'vuota'.
`Socket clientSocket = listener.Accept();`	Creo un secondo socket attraverso l'invocazione del metodo **Accept()** sul primo socket che ha accettato una richiesta di connessione e che ritorna proprio un oggetto Socket. Ora **clientSocket** sarà connesso all'applicazione client.
`while (true)`	Ciclo infinito che consente al server di attendere sempre un messaggio dal client.
`int numByte = clientSocket.Receive(bytes);`	Appena il client trasmette un messaggio, il metodo **Receive(bytes)**, invocato sull'oggetto socket '*clientSocket*', memorizza automaticamente il messaggio, sottoforma di singoli bytes, nell'array '**bytes**' e ritorna il numero esatto di bytes (***numByte***) ricevuti.
`data = Encoding.ASCII.GetString(bytes,0, numByte);`	Una volta ricevuto il messaggio dal client, esso verrà convertito, byte per byte, nel corrispondente carattere ASCII e memorizzato nella variabile '*data*' di tipo string.
`if (data.IndexOf("@") > -1)` `break;`	Controllo se nel messaggio appena ricevuto c'è il carattere '**@**' di fine trasmissione. Se affermativo, cioè l'indice del carattere all'interno della stringa è maggiore di -1, forzo l'uscita dal ciclo infinito while.

[2] https://docs.microsoft.com/it-it/dotnet/csharp/language-reference/builtin-types/enum
[3] https://docs.microsoft.com/it-it/dotnet/api/system.net.sockets.addressfamily?view=netframework-4.8
[4] https://docs.microsoft.com/it-it/dotnet/api/system.net.sockets.sockettype?view=netframework-4.8
[5] https://docs.microsoft.com/it-it/dotnet/api/system.net.sockets.protocoltype?view=netframework-4.8

`Console.WriteLine("Testo ricevuto -> {0} da {1} ", data, clientSocket.RemoteEndPoint.ToString());`	Visualizzo il messaggio ricevuto e l'indirizzo IP del client.
`byte[] message = Encoding.ASCII.GetBytes("OK messaggio ricevuto!!! ");`	Converto byte per byte la stringa di caratteri e memorizzo i byte risultanti nell'array di tipo byte[] **message**.
`clientSocket.Send(message);` `numByte = 0;`	Trasmetto al client l'array contenente i byte della stringa appena convertita, invocando il metodo Send(..) sull'oggetto **clientSocket** che rappresenta la connessione al client. Azzero la variabile ***numByte*** per contenere il numero di byte di un eventuale nuovo messaggio da parte del client.

`clientSocket.Shutdown(SocketShutdown.Both);`
`clientSocket.Close();`
`listener.Close();`
	Appena si esce dal ciclo infinito while, chiudo tutte le connessioni aperte per l'ascolto e la trasmissione.