# Les sockets

Il s’agit d’un modèle permettant la communication inter processus afin de permettre à divers processus de communiquer aussi bien sur une même machine qu’à travers un réseau.

On distingue ainsi deux modes de communication :

* -Le mode connecté utilisant le protocole TCP. Dans ce mode de communication, une connexion durable est établie entre les deux processus, de telle façon que l’adresse de destination n’est pas nécessaire à chaque envoi de données.
* -Le mode non connecté utilisant le protocole UDP. Ce mode nécessite l’adresse de destination à chaque envoi, et aucun accusé de réception n’est donné.

En C#, l'espace de nom ***System.Net.Sockets*** contient tout ce qu'il faut pour manipuler des sockets. Il contient la classe ***Socket***.

Le constructeur de cette classe prend trois paramètres qui sont les suivants:

* ***Address.Family.InterNetwork***: une famille d'adresses qui représente un type d'adressage.
* ***SocketType.Stream*** un type de socket qui détermine la façon qu'il utilisera pour communiquer.
* ***ProtocolType.Tcp*** un type de protocole.

Dans ce cas précis, la communication exige qu'une connexion soit établie avant que la communication puisse débuter. La communication pourra alors se faire dans les deux sens, en utilisant un flot d'octets. C'est la méthode TCP.

Dans une communication client/serveur, chacune des deux parties devra se créer un socket. Les deux sockets formeront ultimement les deux bouts d'un tuyau de communication.

**Du côté du serveur,**

***Socket socketServer = new Socket (AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);***

Le socket, une fois créé, doit être attaché à un point de communication ***IPEndPoint*** .Un point de communication correspond à une adresse et un port TCP sur lequel écouter.

***SocketServer.Bind (new IPEndPoint (IPAddress.Parse ("192.168.0.4"), 1212));***

Une fois le socket attaché à un point de communication, on le place en mode "écoute":

***SocketServer.Listen(N);***

***N*** représente le nombre de connexions simultanées maximal

Lorsque le socket est en mode "écoute", il pourra recevoir autant de demandes. Ces demandes seront placées en file et devront être traitées en les acceptants:

***Socket actif = SocketServer.Accept ();***

**Du côté du client,**

On créer un socket de la même façon que pour le serveur.

***Socket SocketClient = new Socket (AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);***

Le socket tentera ensuite une connexion au point de communication correspondant au serveur:

***SocketClient.Connect (new IPEndPoint (IPAddress.Parse ("192.168.0.4"), 1212));***

**Du cote des deux : serveur et client**

On écrire dans le socket en envoyant un tableau d'octets dans le flot.

***byte [] messageAenvoyé = new byte [512];***

***SocketClient.Send (messageAenvoyé);***

On peut lire dans le socket en remplissant un byte[]:

***byte [] messageRecu = new byte [512];***

***SocketClient.Receive (messageRecu);***

On peut convertir un string en byte[] en utilisant la fonction ***System.Text.Encoding.ASCII.GetBytes()*** .

L'inverse peut être fait grâce à sa fonction miroir ***System.Text.ASCIIEncoding.ASCII.GetString().***

On pourra fermer le socket en faisant simplement:

***SocketClient.Close();***

On peut voir un socket comme deux tuyaux entre les interlocuteurs. Chacun d'eux peut envoyer des messages dans le tuyau, messages qui s'accumulent là jusqu'à ce que l'autre les lise. De son côté, l'autre peut faire la même chose. Il n'est pas nécessaire qu'un des deux ait lu tout ce qu'il a reçu avant d'envoyer. Il faut donc évidemment avoir un protocole quelconque d'établi pour une communication harmonieuse.

Ce petit programme de chat se composera de deux applications distinctes : une application serveur qui se mettra en attente d'une connexion et qui l'établira puis recevra des messages du client et une application cliente qui se connectera au serveur et enverra des messages au serveur.