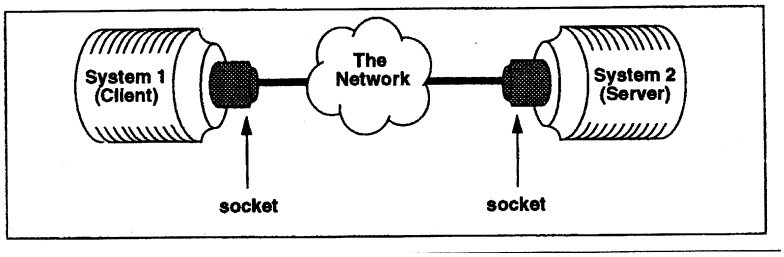
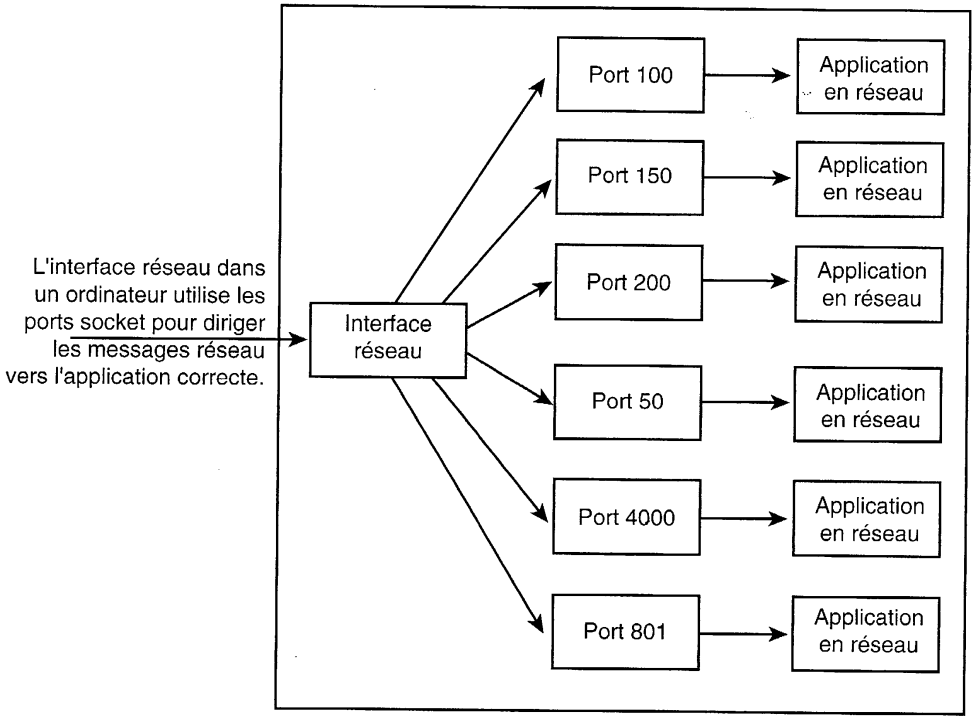
Communication de deux applications par le réseau (socles de communication).

L'université de Berkeley en Californie a mis au point des fonctions (écrites en C à l'origine) de communication sur un réseau TCP-IP. Ces fonctions de base forment ce qu'on appelle "la programmation Socket". La classe Socket (.net) encapsule ces fonctions.

D'une façon générale, les **Berkeley sockets**, que l'on pourrait traduire par "connecteurs réseau de Berkeley" représentent une interface de programmation qui fournit aux applications un accès aux protocoles de communication réseau.



- Un socle de communication désigne la combinaison d'une adresse IP et d'un port.
- Les socles permettent de faire communiquer des systèmes hétérogènes.
- pour accepter plusieurs communications avec plusieurs client, le serveur utilise un socle par client.
- Les ports sont utilisés pour "router" les communications réseau vers l'application correcte. Chaque application utilise un port pour dialoguer sur le réseau.



:

Avant toute chose le programme client et le programme serveur doivent créer un socle, dans notre cas un objet de la classe Socket dans lequel est spécifié:

- Le type d'adresse: On va utiliser des adresses codées sur 4 octets (IPV4), mais beaucoup d'autres types d'adresses sont possibles.
- Le type de socket (utilisation de datagrammes ou non)
- Le protocole utilisé (TCP de TCP/IP dans notre cas, ce qui garantit la fiabilité des transmissions)

Classe `Socket`

```
using System.Net;
using System.Net.Sockets;
```

Constructeur de la classe `Socket`

`Socket(AddressFamily, SocketType, ProtocolType);`	Constructeur de la classe. On y spécifie : — un type d'adresse, ce qui implique généralement l'utilisation d'un protocole particulier. On peut spécifier l'une des valeurs suivantes de l'énumération `AddressFamily` : `AppleTalk`, `Ipx`, `Banyan`, `DecNet`, `NetBios`, `Sna`, `Unix` mais surtout `InterNetwork` (adresses IP codées sur quatre octets) et, dans une mesure moindre encore pour le moment, `InterNetworkV6` (nouvelle génération d'adresses IP codées sur seize octets) ; — le type de socket (`Stream` dans notre cas). On peut y spécifier une des valeurs de l'énumération `SocketType` (`Dgram`, `Row`, `Stream`, etc.) ; — le protocole : une dizaine de protocoles sont supportés. Nous ne nous intéresserons qu'à TCP bien qu'il soit possible de travailler au niveau UDP et même IP. On peut spécifier une des valeurs de l'énumération `ProtocolType` (`Icmp`, `IP`, `Ipx`, `Tcp`, `Udp`, etc.).

Méthodes de la classe `Socket`

`void Bind(EndPoint localEP);`	Opération effectuée sur le serveur qui associe un socket avec ce que l'on appelle un *end-point* (formé d'une adresse IP et d'un numéro de port). L'exception `SocketException` est générée si l'opération ne peut être exécutée correctement (généralement parce qu'un programme travaille déjà sur ce port).
`void Listen(int n);`	Place le socket en attente de connexions. L'argument indique la taille de la file de connexions qui pourraient être en attente de traitement. Cette opération est effectuée sur le serveur.
`Socket Accept();`	Renvoie un socket à utiliser pour la communication avec le client qui vient de se manifester. Cette opération est effectuée sur le serveur.

`void Connect(` `  EndPoint remoteEP);`	Établit (à partir du client) une liaison avec un ordinateur distant. Le serveur doit avoir exécuté `Bind` avant que le client n'exécute `Connect` (sinon l'exception `SocketException` est générée).

`int Send(byte[]b);`
`int Send(byte[] b, int n,`
`  SocketFlags);`
`int Send(byte[] b, int offset,`
`  int n, SocketFlags);`
	Envoie des caractères au correspondant. Les données proviennent du tableau `b` d'octets.
	
	La deuxième forme permet de spécifier le nombre d'octets à envoyer. `SocketFlags` peut être une des valeurs suivantes de l'énumération `SocketFlags` : `None` (ne pas retenir cet argument) ou `Partial` (envoi d'une partie de message). L'argument sera généralement `SocketFlags.None`.
	
	Dans la troisième forme, on spécifie un déplacement à l'intérieur de `b` (c'est à partir de ce déplacement que les données sont puisées dans ce tableau).
	
	`Send` renvoie le nombre d'octets envoyés.

`int Receive(byte[] b);`
`int Receive(byte[] b, int n,`
`  SocketType);`
`int Receive(byte[] b,`
`  int offset, int n,`
`  SocketType);`
	Reçoit des caractères provenant du correspondant et les stocke dans le tableau `b`. La limite de `b` ne sera jamais dépassée et on ne risque donc pas de dépasser la capacité du buffer.
	
	Dans la deuxième forme, on spécifie le nombre maximum d'octets à copier dans `b`.
	
	Dans la troisième forme, on spécifie un déplacement dans `b` et c'est à partir de ce déplacement que les données sont copiées dans le tableau.
	
	`Receive` renvoie le nombre d'octets copiés dans le tableau.

`void Close();`	Ferme la liaison par socket. Exécutez `Shutdown` avant `Close` pour que les données en attente dans les buffers soient traitées.
`void Shutdown(int how);`	Met fin aux envois et réceptions sur le socket. `how` peut prendre l'une des valeurs suivantes de l'énumération `SocketShutdown` : `Both` (les deux parties cessent de communiquer), `Receive` (arrêt en réception) et `Send` (arrêt en émission). Les données en attente de traitement dans les buffers sont encore traitées.

I) Principe de communication par Socket en C#

Dans l'application cliente et serveur on ajoute les directives:

using System.Net;
using System.Net.Sockets;

On déclare en attribut de la classe (du formulaire par exemple):

Socket sock;

1°) Création du socle coté serveur et coté client (par exemple dans le constructeur de la classe formulaire)

Utilisation du protocole TCP de TCP/IP, avec adresses codées sur 4 octets

```
//création de l'objet socle de communication
sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

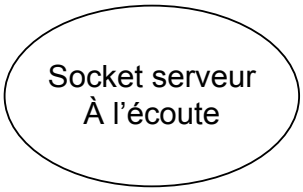
2°) Le socle coté serveur est associé à l'adresse IP de la machine serveur et à un numéro de port grâce à la méthode Bind()

```
//Association du socle à l'adresse IP de la machine et à un numéro de port.
sock.Bind(new IPEndPoint(IPAddress.Parse("192.168.0.11"),4000));
```

- Pour cela, il existe une classe IPEndPoint qui permet de créer un objet qui contient une adresse IP et un n° de port. On parle d'objet de connexion.
- En effet l'argument de la méthode Bind est un objet de la classe IPEndPOint

3°) Le socle serveur se met à l'écoute grâce à la méthode Listen(). L'argument de Listen() est le nombre de connexions pouvant être en attente de traitement.

```
//Serveur à l'écoute
sock.Listen(1);
```

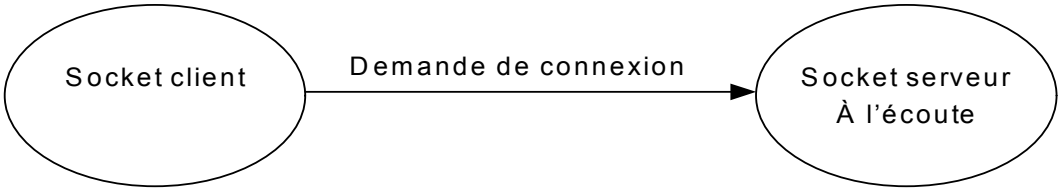


4°) Le socle serveur se met en attente d'une demande de connexion provenant d'un client. La méthode Accept() est bloquante (opération synchrone)

```
//Le serveur attend la demande de connexion du client (Accept bloquant)
sockClient = sock.Accept();
```

- Accept() renverra un objet de type Socket (ici sockClient) qui sera utilisé pour dialoguer avec le client.

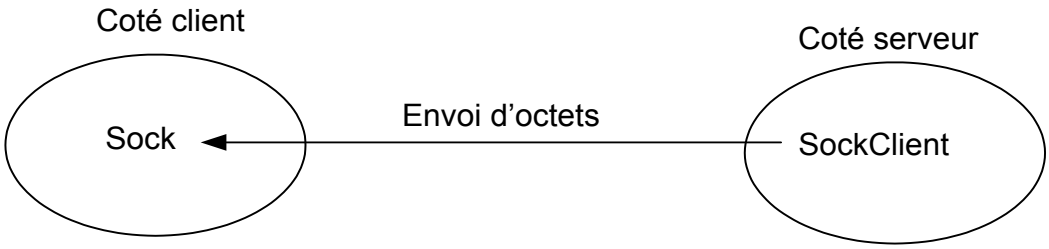
5°) Le client demande à se connecter au socle du serveur grâce à la méthode Connect()



- Pour cela le client doit connaître l'adresse IP sur serveur et le n°de port de l'application serveur.

```
//demande de connexion au serveur
sock.Connect(new IPEndPoint(IPAddress.Parse("192.168.0.11"),4000));
```

6°) En réponse à la demande de connexion du client, la méthode Accept() du serveur se termine, et renvoie un objet de type Socket (ici sockClient).
sockClient du serveur est connectée au socle client, ainsi le serveur peut par exemple envoyer un message au client:

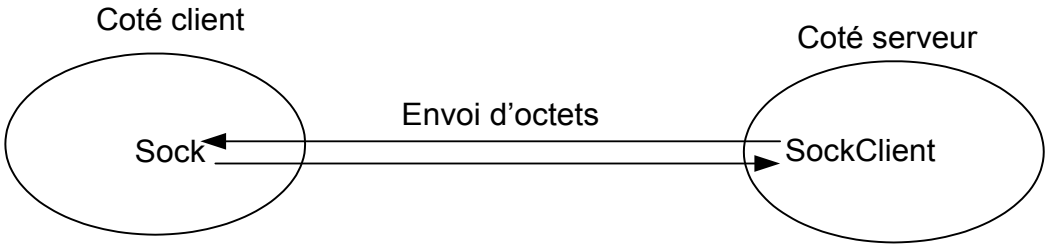



```
//le client est connecté, La variable sockClient permet de dialoguer avec le client
//envoi d'un message au client
string reponse = "Vous êtes connecté au serveur";
byte[] tab = ASCIIEncoding.Default.GetBytes(reponse);
sockClient.Send(tab);
```

7°) le client se met en attente de réception du message du serveur grâce à la méthode Receive() qui est bloquante:

```
byte[] buffer = new byte[50];
//receive lit les données dans une mémoire tampon de réception
//si le nombre de caractères envoyés par le serveur est > à 50, il faut
//appeler de nouveau Receive().Receive()est bloquant (lecture synchrone)
int n = sock.Receive(buffer);
string recu = ASCIIEncoding.Default.GetString(buffer);
textBoxReception.Text = recu;
```

8°) Un dialogue (synchronisé) peut s'établir entre le client et le serveur grâce aux méthodes Receive() et Send()



9°) Fermeture de la connexion du coté serveur et/ou du coté client:

```
//fin de l'émission et reception
sock.Shutdown(SocketShutdown.Both);
//fermeture la liaison par socle
sock.Close();
//ferme l'application
```

II) Utilisation de la classe dns

- **Rappel:**

La classe IPAdress correspond à une adresse IP.
La classe IPEndPoint correspond à un ensemble (adresse IP, numéro de port).

- La classe dns

La classe dns de l'espace de noms System.Net permet d'effectuer des conversions entre un nom (ex: www.google.fr) ou un nom de machine sur le réseau local et une adresse IP.
La classe IPHostEntry contient principalement les attributs suivants:

Classe Dns	
Dns ← Object	
Méthodes de la classe Dns	
`string GetHostName();`	Renvoie le nom de la machine locale.
`IPHostEntry GetHostEntry(string);`	Effectue une conversion à partir d'un nom de domaine ou d'un nom de machine. `GetHostEntry` renvoie un objet `IPHostEntry` : `IPHostEntry he = Dns.GetHostEntry("www.xyz.com");` Ce dernier objet `IPHostEntry` peut fournir une liste d'adresses IP associées à cet ordinateur (voir ci-après). Une exception est générée si la résolution de nom ne peut être effectuée.
`IPHostEntry GetHostByAddr(string);`	Renvoie un objet `IPHostEntry` correspondant à une adresse IP (passée sous forme d'une chaîne de caractères).

- Hostname contient le nom de domaine.
- AddressList est un tableau d'adresses IP associées au nom (ce tableau peut ne contenir qu'une seule adresse).

- Pour retrouver automatiquement l'adresse IP de sa machine, l'application serveur peut utiliser le code suivant:

```
//Pour retrouver le nom de la machine et son adresse IP.
string NomPc = Dns.GetHostName(); //Obtient le nom de l'ordinateur
IPHostEntry he = Dns.GetHostEntry(NomPc);
IPAddress[] TabIP = he.AddressList;
//L'adresse IP est dans TabIP[0]
//Association du socle à l'adresse IP de la machine et à un numéro de port.
sock.Bind(new IPEndPoint(TabIP[0],4000));
```

Le client peut retrouver automatiquement l'IP du serveur à partir de son nom avec le code suivant:

```
//Retrouve l'IP du serveur à partir de son nom
IPHostEntry he = Dns.GetHostEntry("mobile");
IPAddress[] TabIP = he.AddressList;

//demande de connexion au serveur
sock.Connect(new IPEndPoint(TabIP[0],4000));
```