

M3102

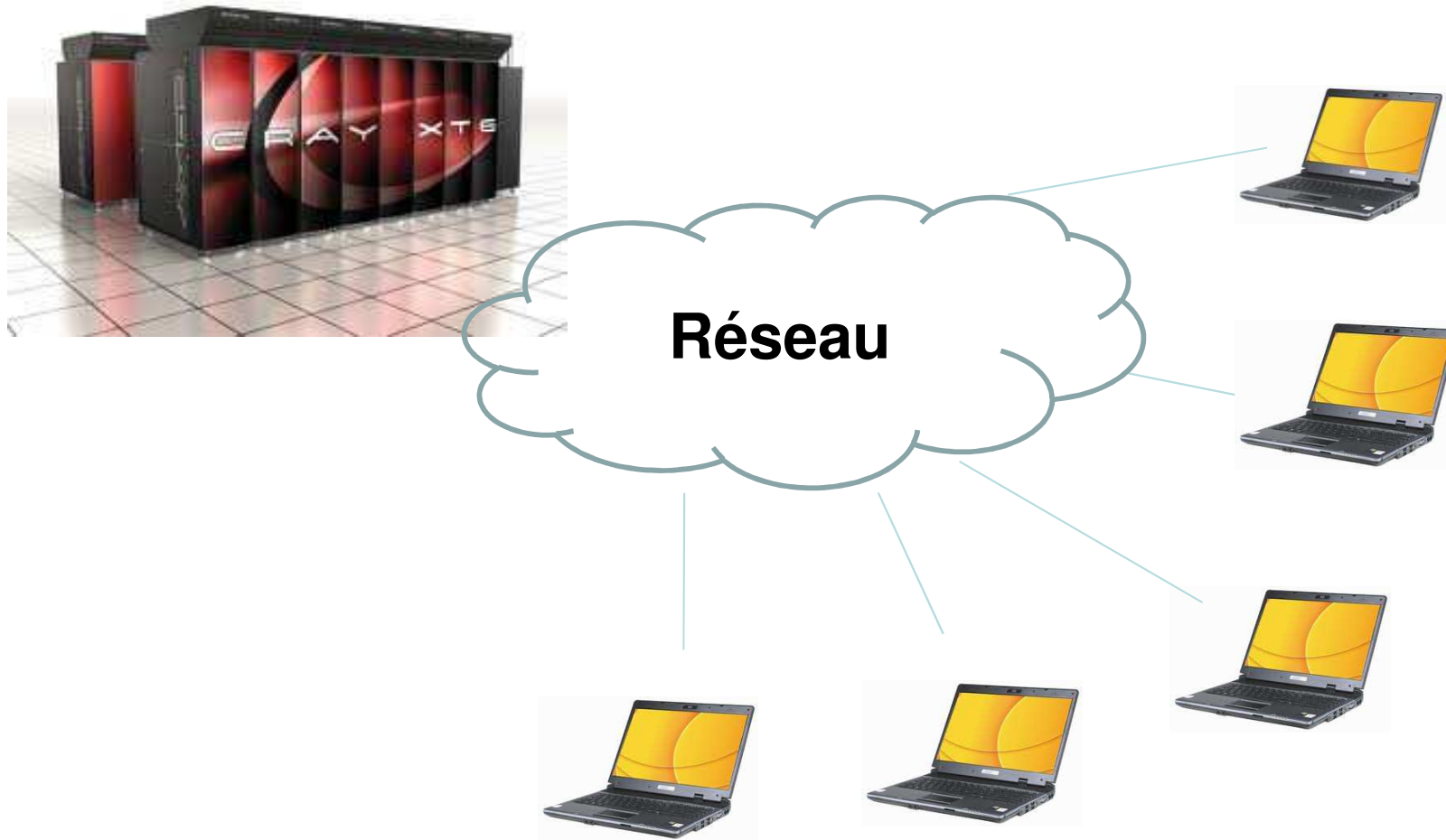
Services réseaux

Services réseaux

L'API Socket

L'Api Socket

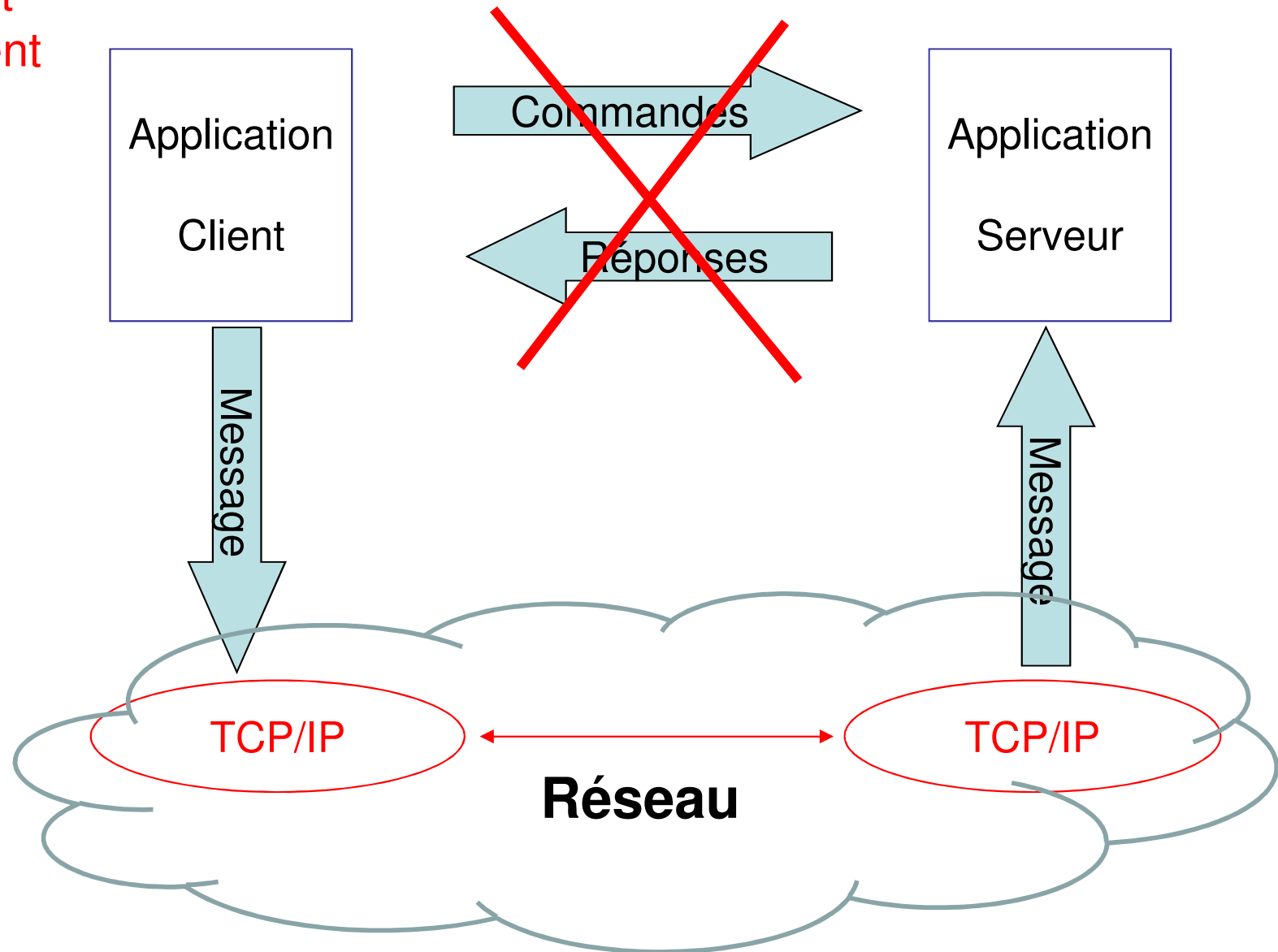
L'objectif des réseaux est de mettre en relation des machines afin que ces dernières se partagent leurs ressources. Il faut, pour cela, créer des applications, capables de dialoguer entre elles via un réseau.



→ On parle d'application Client/Serveur ou de services

L'Api Socket

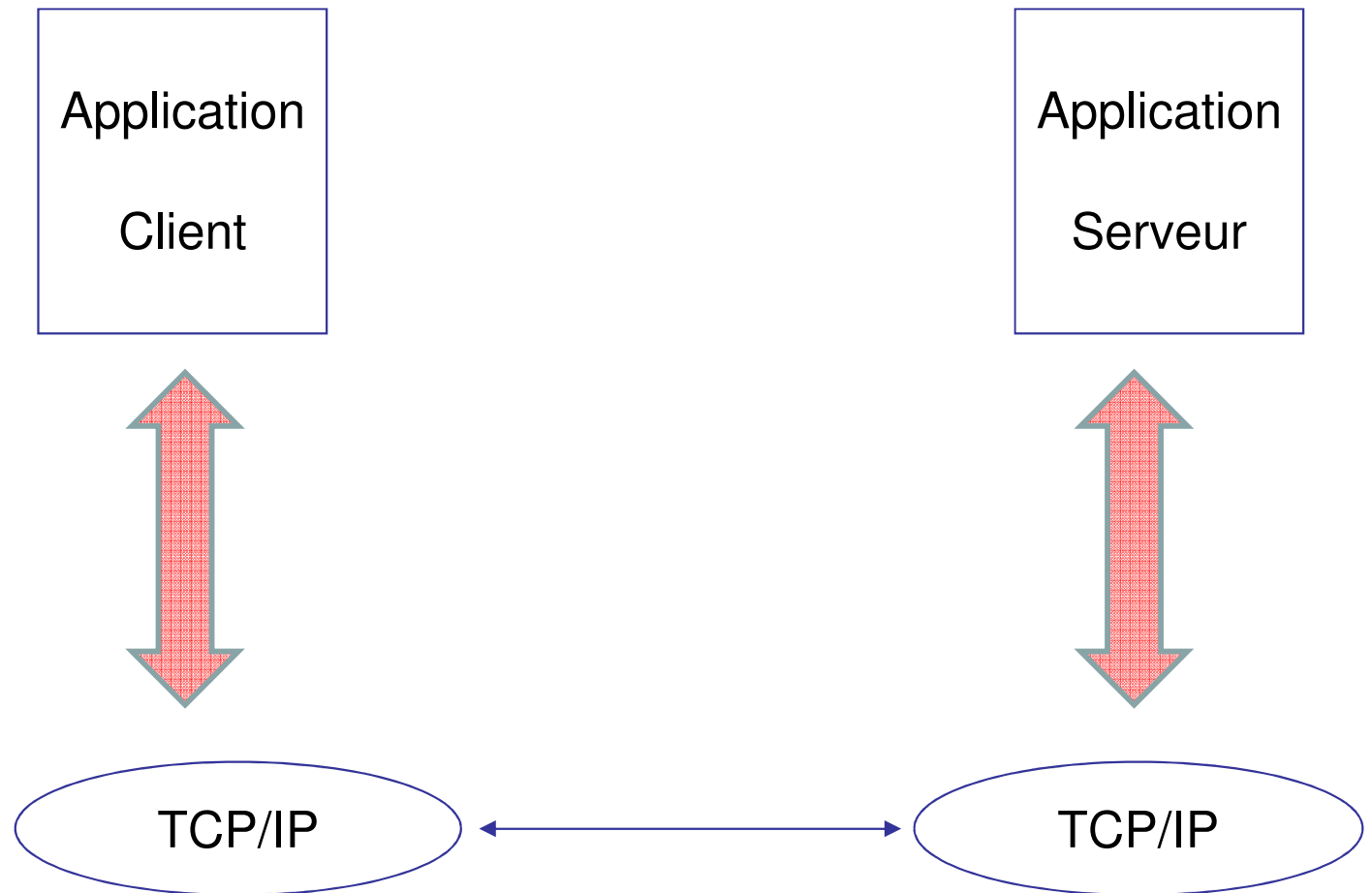
Les applications client-serveur, communiquent grâce aux couches transport TCP/IP ou UDP/IP.



L'Api Socket

Les applications client serveur, communiquent grâce aux couches transport TCP/IP ou UDP/IP.

Il est nécessaire de mettre en relation ces deux fonctions.



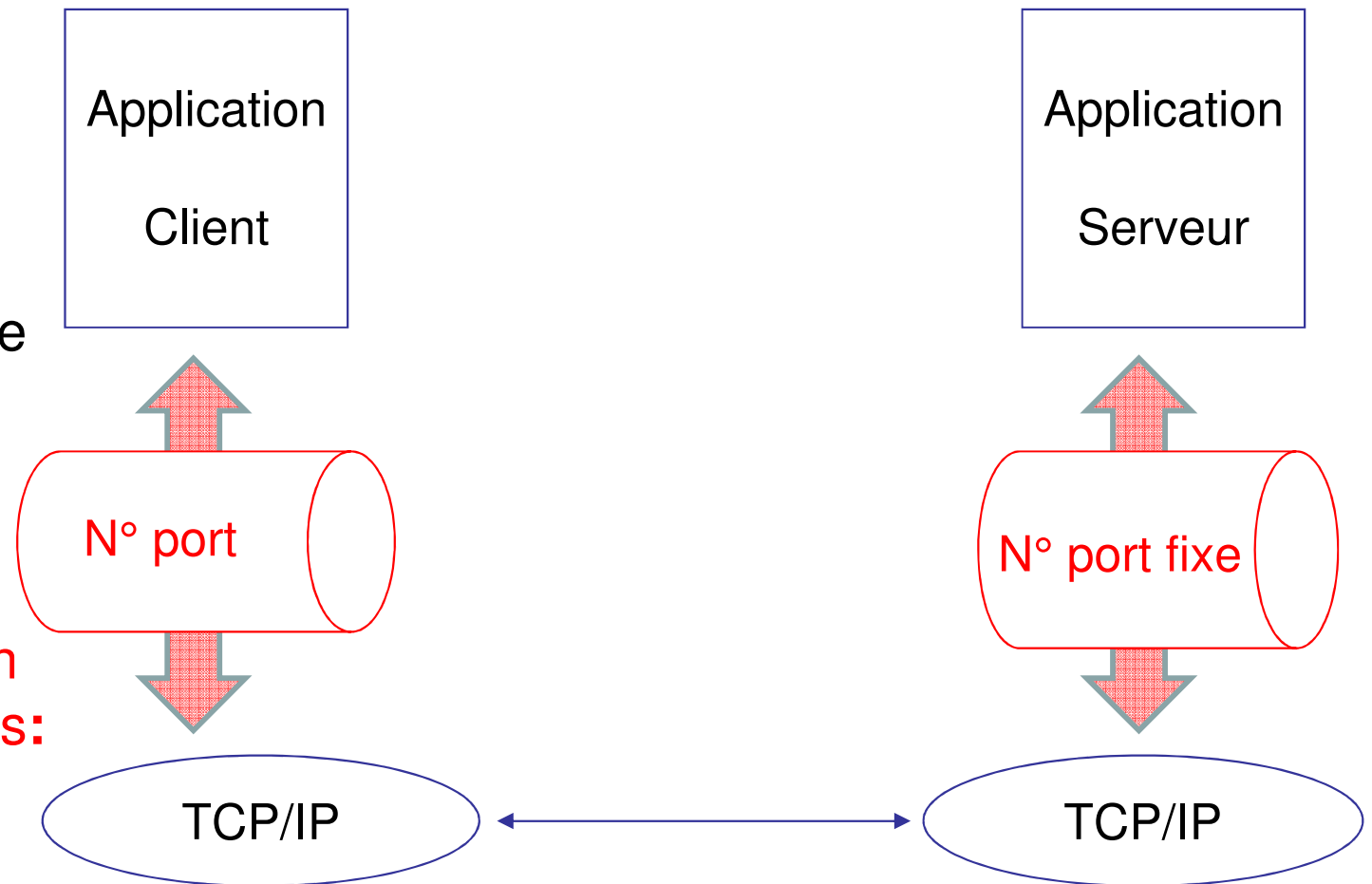
L'Api Socket

Les applications client serveur, communiquent grâce aux couches transport TCP/IP ou UDP/IP.

Il est nécessaire de mettre en relation ces deux fonctions.

Pour cela on utilise des fichiers de communication identifiés par des numéros:

le numéro de port.



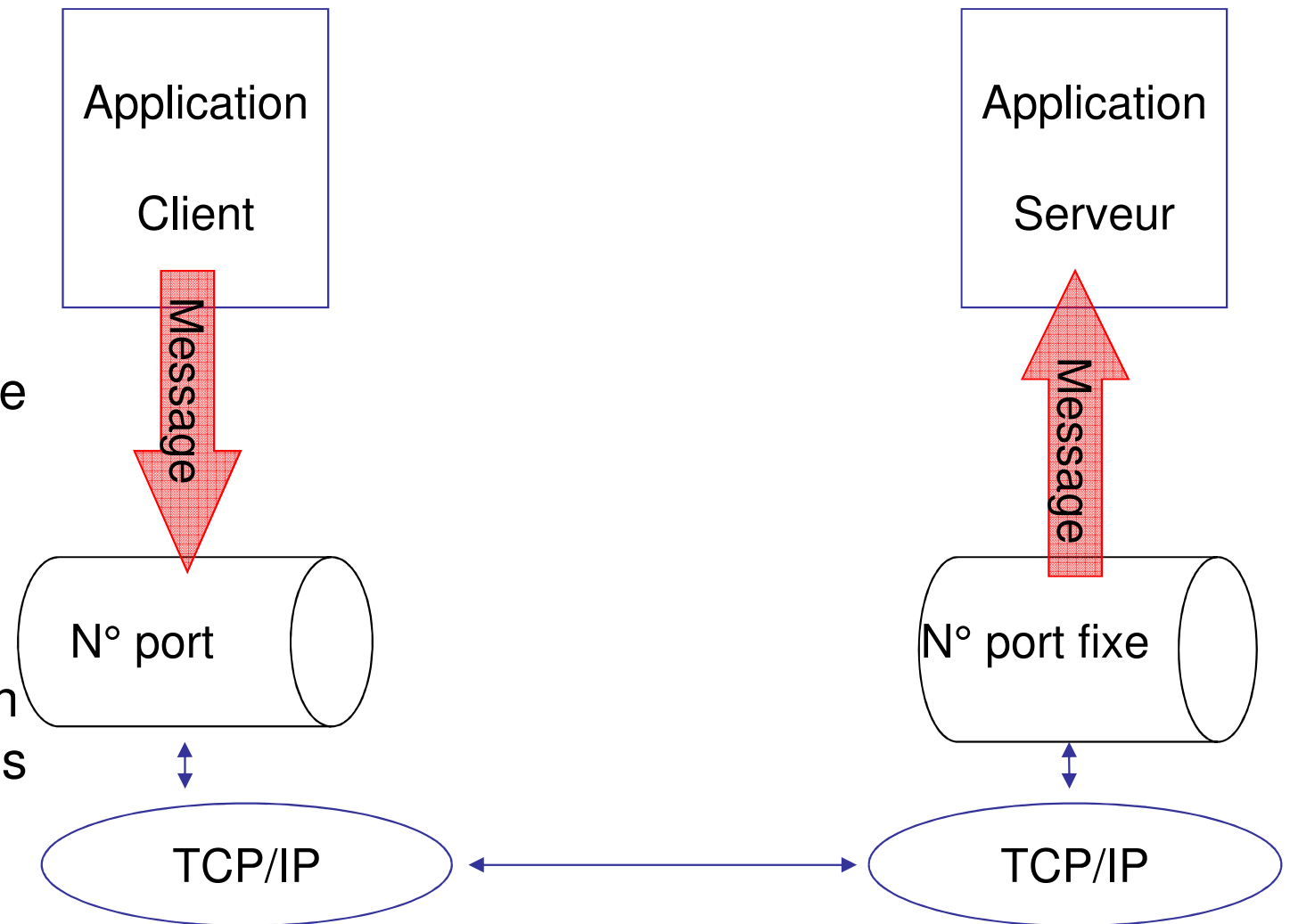
L'Api Socket

Les applications client serveur, communiquent grâce aux couches transport TCP/IP ou UDP/IP.

Il est nécessaire de mettre en relation ces deux fonctions.

Pour cela on utilise des fichiers de communication identifiés par des numéros de port.

Il faudra fournir , aux applications des fonctions pour accéder à ces fichiers.



On parle d'interface de programmation (API)

L'Api Socket

Interfaces de programmation

Sockets

- Interface de programmation pour TCP/IP ou UDP/IP. 1981. Système UNIX Berkeley BSD.
- Le standard UNIX de facto.
- L'API sockets sous Windows baptisée WinSock.
- Disponibles en C, Java, php, visual basic, ...

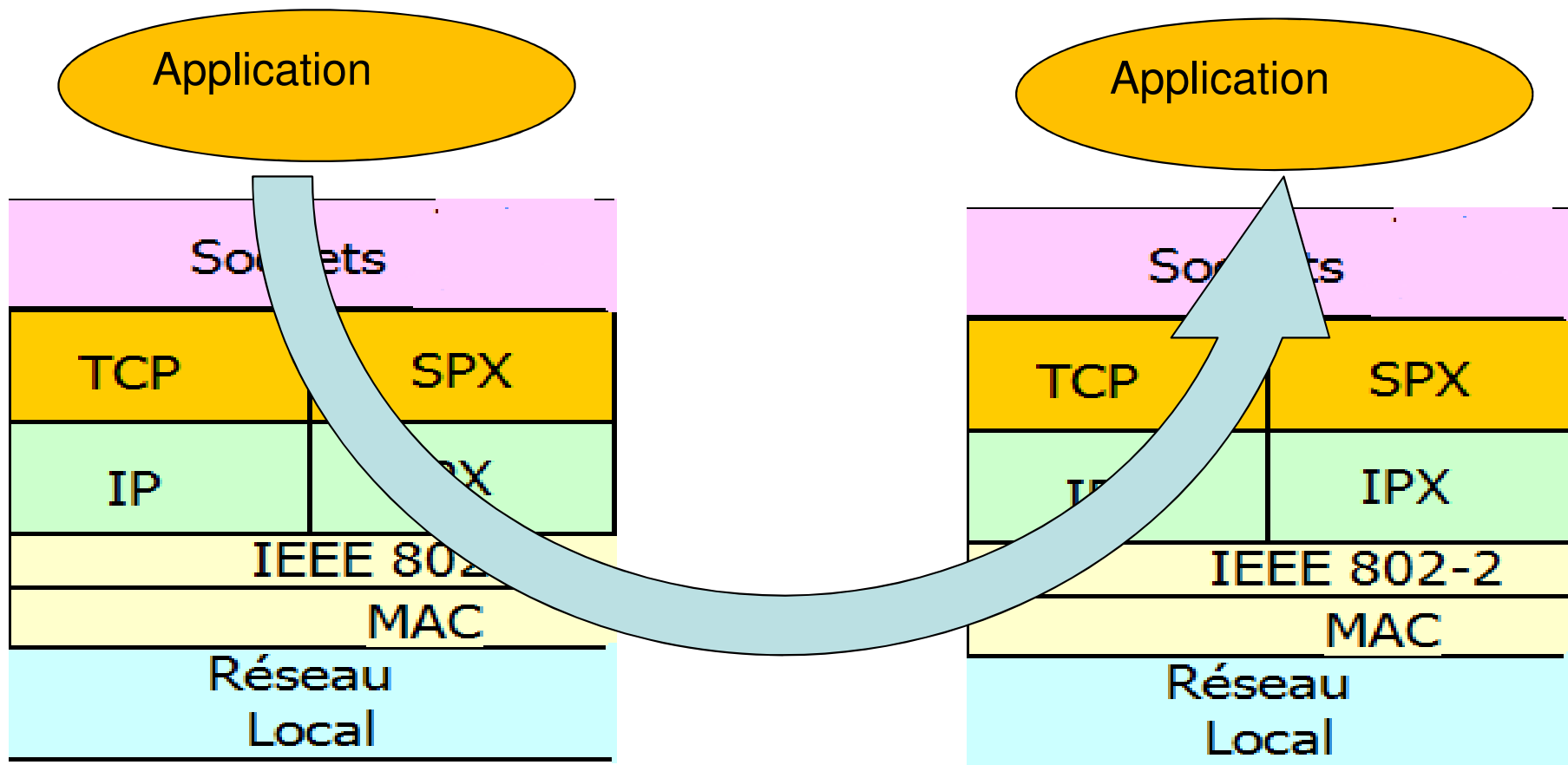
The diagram illustrates the seven layers of the OSI model, each with associated protocols and programming interfaces. A red bracket at the top groups the top three layers (Transport, Réseau, Liaison) under the heading 'Interfaces de programmation'. A red circle highlights the 'Sockets' interface in the Transport layer.

Transport	NetBEUI	Sockets	Tubes nommés	CPI-C APPC
Réseau	NetBIOS	TCP	SPX	LU 6.2 APPN
		IP	IPX	
Liaison		IEEE 802-2 - PPP		
		MAC		
Physique	Réseau	Réseau Local	Voie point à point	

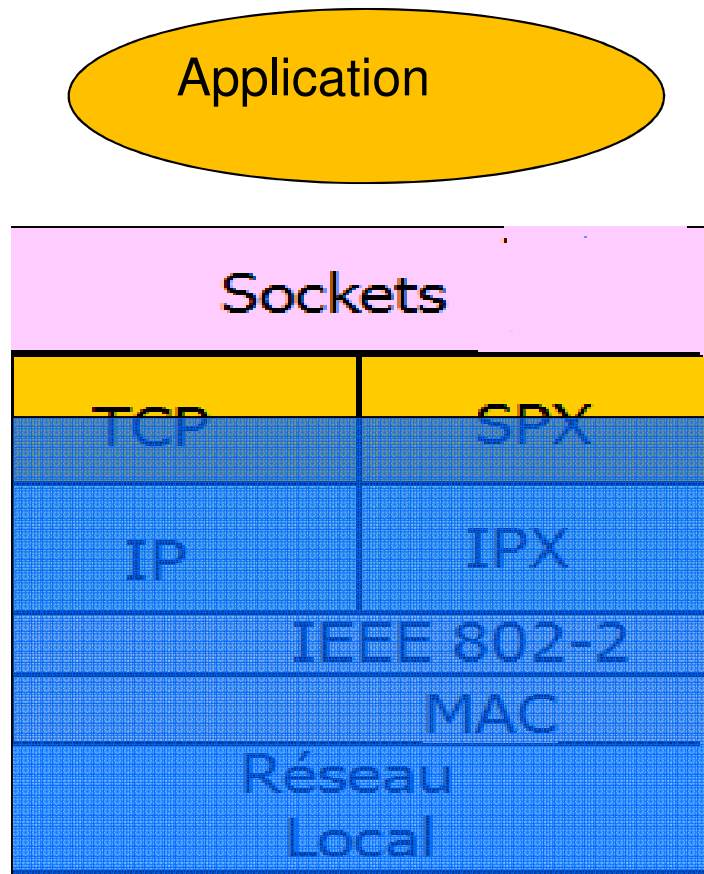
L'Api Socket

Objectif 1

Fournir des moyens de communications entre processus (IPC ou Communication Inter-Processus) **utilisables en toutes circonstances**: échanges locaux ou réseaux.



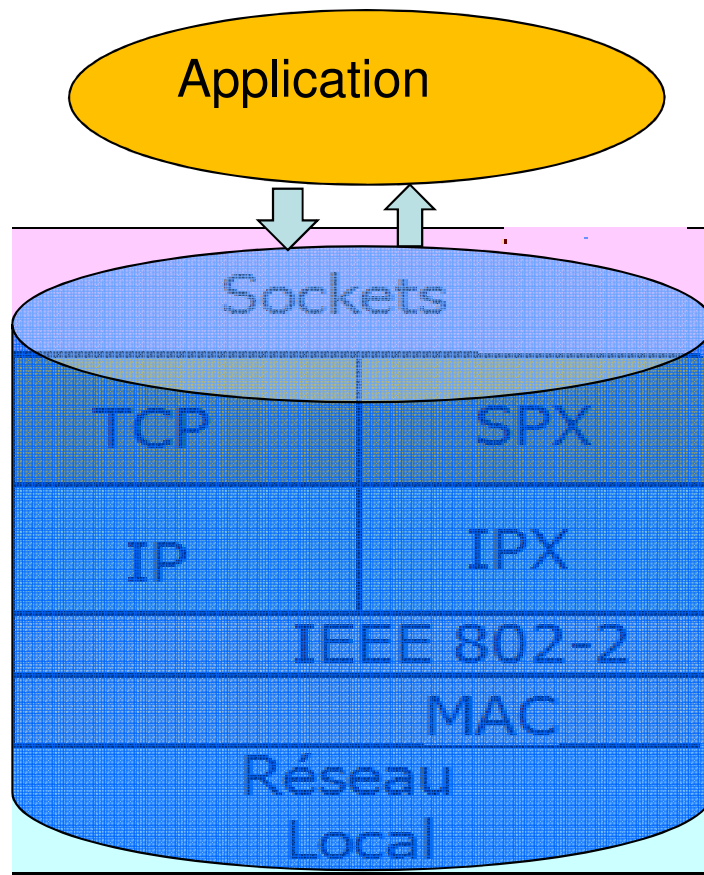
L'Api Socket



Objectif 2

Cacher **les détails d'implantation** des couches de transport aux usagers.

L'Api Socket



Objectif3

Fournir une interface d'accès qui se rapproche **des accès fichiers (read, write, ...)** lorsque la **connexion est établie entre les applications.**

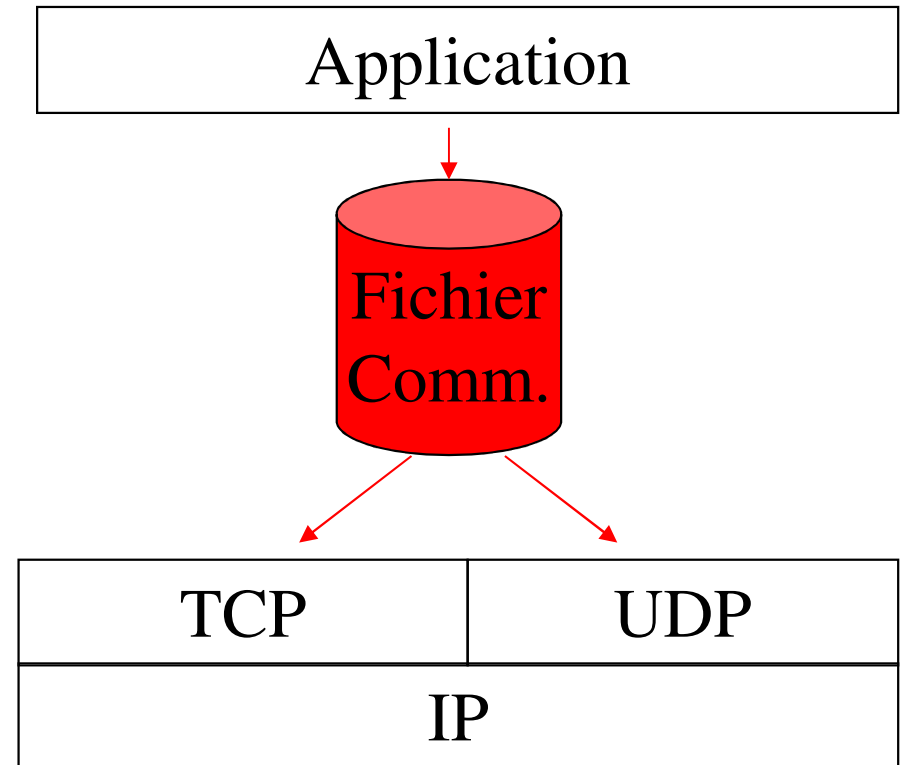
L'Api Socket

Une socket est analogue à un fichier de communication (PIPE).

- Traitement des données en mode FiFo
- Lecture destructive
- E/S synchronisées

→ Les mêmes qu'un tube

mais visible par toutes les machines d'un réseau



L'Api Socket

L'interface socket définit :

■ **Un type**: Pour quel protocole de transport est-elle un point d'accès de service?

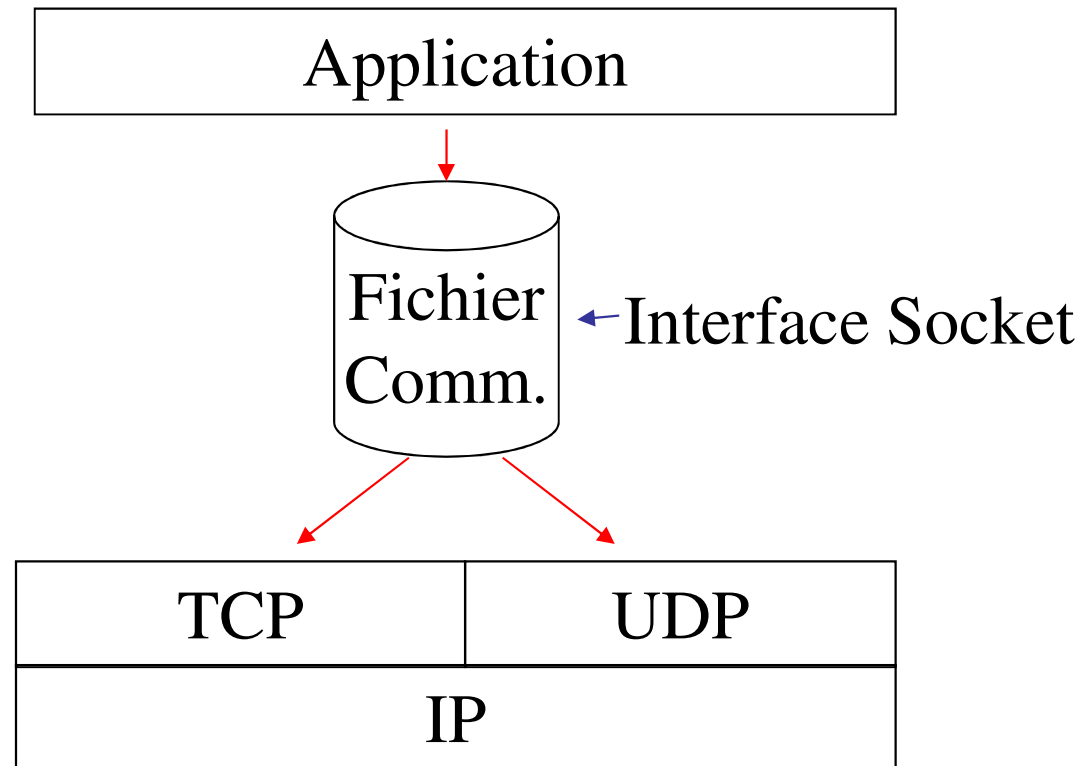
■ **Un nom (ou port)**: identifiant unique sur chaque site (en fait un entier 16 bits).

■ **Un ensemble de primitives** : un service pour l'accès aux fonctions de transport.

■ **Des données encapsulées** :

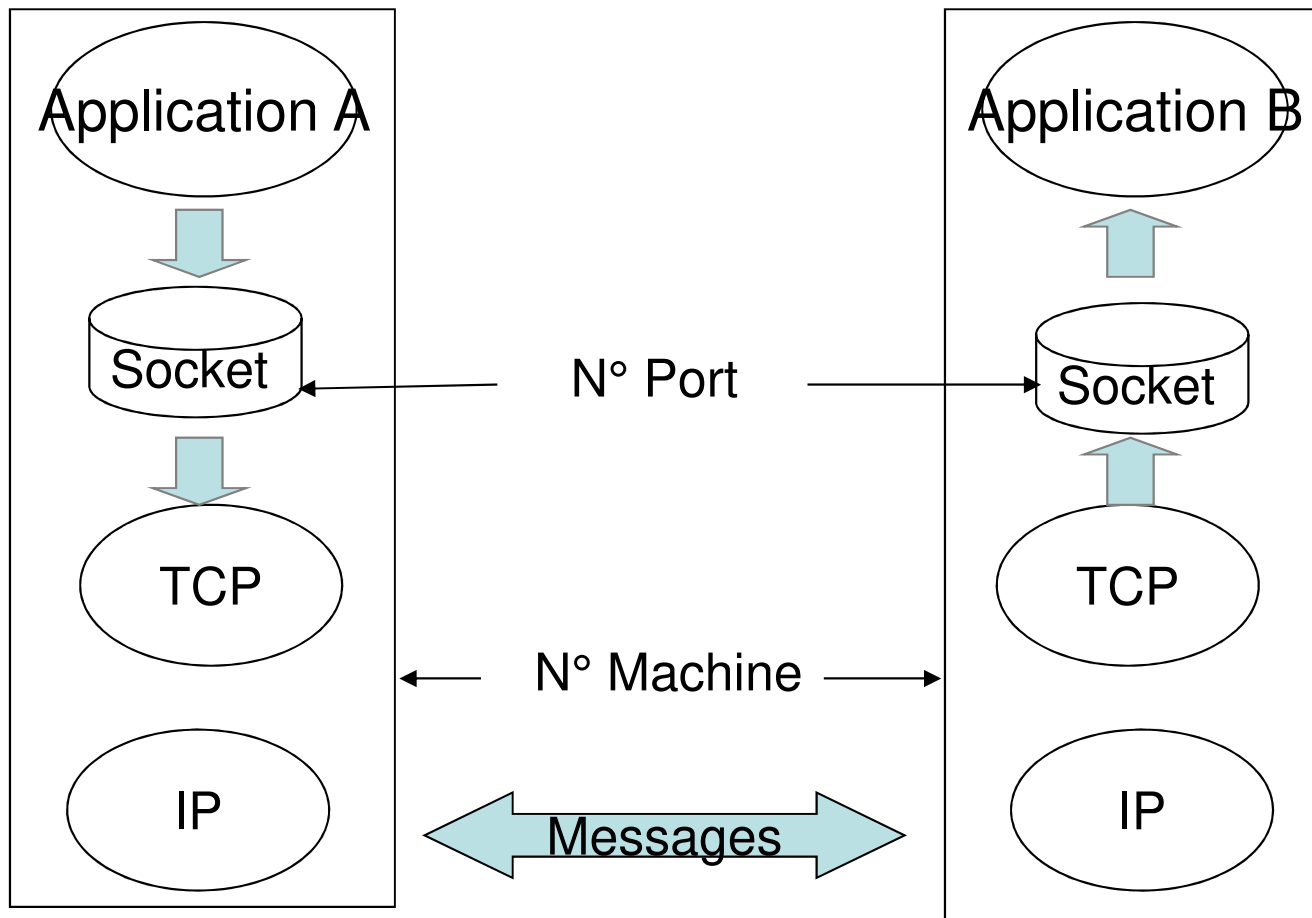
■ **un descriptif** (pour sa désignation et sa gestion)

■ **des files d'attente** de messages en entrée et en sortie.



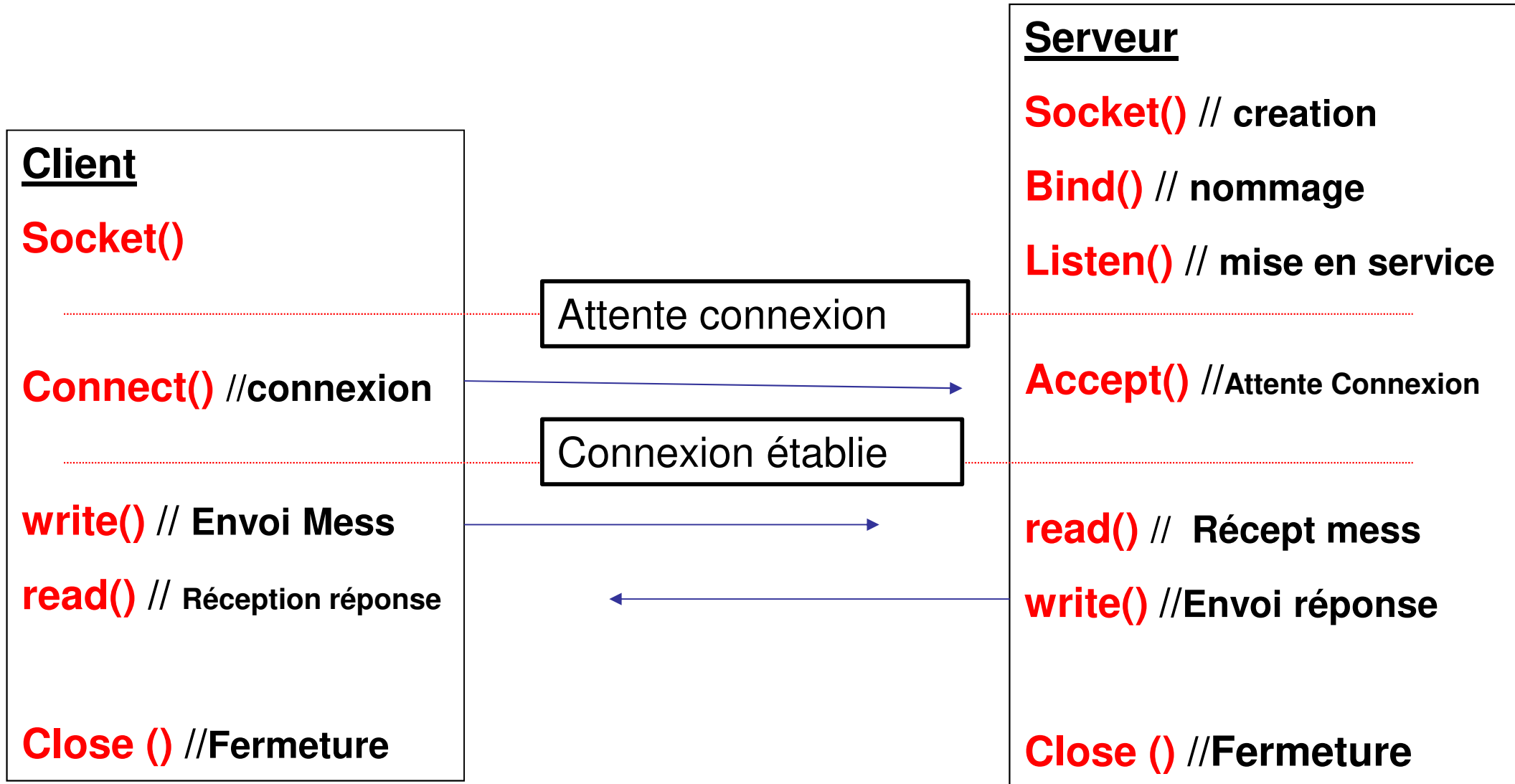
L'Api Socket

Ainsi la communication entre deux applications, dans un réseau, se fait sur la base des : **Numéro de port** et **Adresse IP**



L'Api Socket

Primitives de l'API socket - mode TCP



Les services réseaux

Interface Socket – Langage C

Interface Socket - Bibliothèques

- `#include <sys/socket.h>`

→ fonctions d'accès

- `#include <netinet/in.h>`

→ définition des familles de protocoles Internet

- `#include <sys/types.h>`

→ format des données

Interface Socket

Client

Socket()

Connect() // connexion

Send() // Envoi Mess

Recv() // Réception
réponse

Close () //Fermeture

Attente connexion

Connexion établie

Serveur

Socket() // creation

Bind() // nommage

Listen() // mise en service

Accept() //Attente Connexion

Recv() // Récept mess

Send() //Envoi réponse

Close () //Fermeture



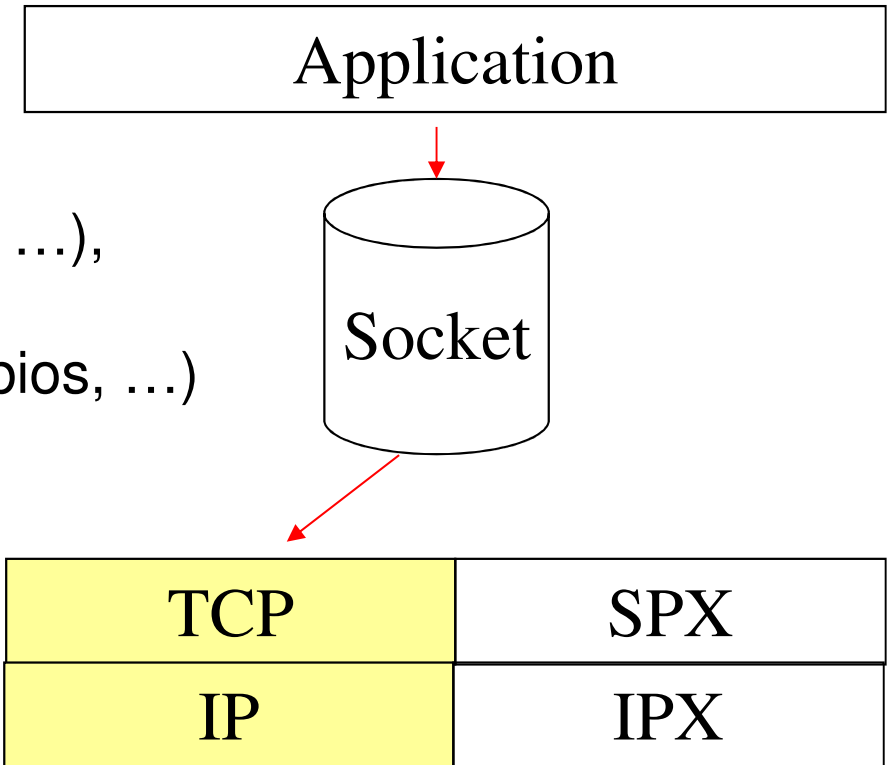
Interface Socket - Création

A la création d'une socket il faut indiquer :

- Le format des adresses utilisées (internet, sna, ...),
- Le type de service utilisé (connecté ou non),
- Le nom effectif du protocole (tcp, udp, spx, netbios, ...)

Dans le cas ci-joint :

- Adresses internet (pile TCP/IP),
- Service en mode connecté,
- Protocole TCP.



Sock = socket (af, type, protocole)

Famille d'adresses

Type de service

Nom du protocole

Interface Socket - Création

Sock = socket (af, type, protocole)

Famille d'adresses

TCP-IP---> **AF_INET**

SPX-IPX → AF_IPX

Type de service

SOCK_STREAM

→ connecté

SOCK_DGRAM

→ non connecté

Nom du protocole

IPPROTO_TCP → TCP

IPPROTO_UDP → UDP

0 (défaut)

Interface Socket - Création

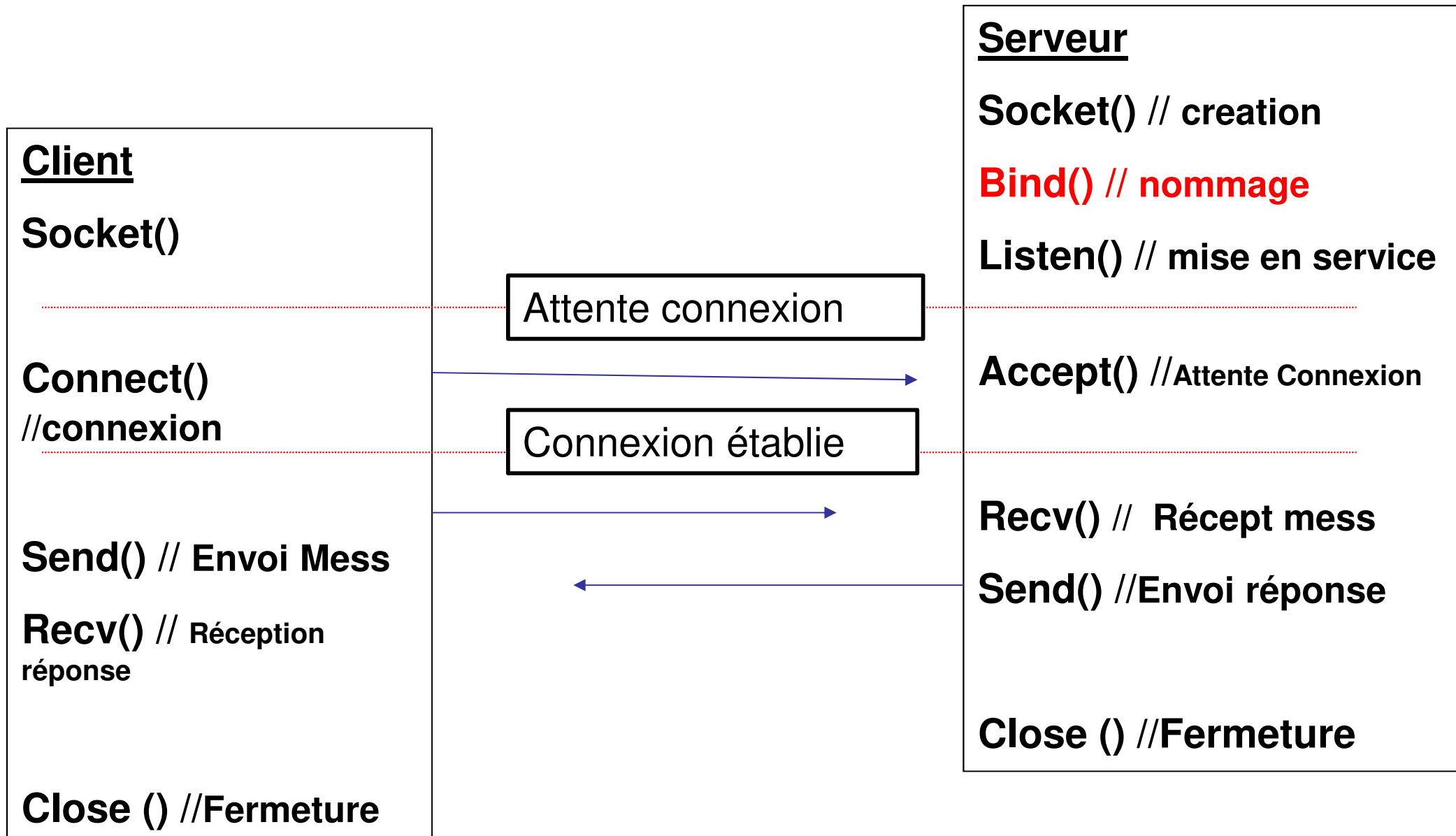
Exemple :

```
void main()  
{ int sock,  
...  
sock=socket(AF_INET, SOCK_STREAM, 0);  
...  
}
```

Résultat : Création d'une socket

- Pour internet (AF_INET)
- Mode connecté (SOCK_STREAM)
- Utilisant le protocole « mode connecté » par défaut (0), en principe TCP

Interface Socket



Interface Socket - Affectation Numéro

Cette fonction va attribuer un numéro de port spécifique à une socket.
→ On parlera de **socket d'écoute**.

```
int bind ( sock, p_struct_adress, long_struct_adress )
```

descripteur de la socket

Pointeur champs adresse

longueur de la
structure adresse

Famille adr.	N° Port	@ IP	Options
--------------	---------	------	---------

Structure adresse

Interface Socket - Affectation Numéro

La structure du champs adresse est générique et peut être utilisée de plusieurs façons:

struct sockaddr



Définition en C

struct sockaddr

```
{ u_short sa_family;  
  char    sa_data[14];};
```


Interface Socket - Affectation Numéro

La structure **sockaddr_in** est une structure particulière de **sockaddr** utilisée pour communiquer sous internet.

La famille AF_INET (internet) utilise la structure sockaddr_in définie de la façon suivante:

Structure sockaddr_in

Famille adr.	N° Port	@ IP	Options
--------------	---------	------	---------

Définition en C

struct sockaddr_in

```
{ u_short  AF_INET;           // famille d'adresse internet (2 octets)
  u_short  sin_port;          // n° de port (2 octets)
  struct in_addr sin_addr;     // adresse IP V.4 ou INADDR_ANY (4 octets)
  char     sin_zero[8];        // options possibles (8 octets)
};
```

Interface Socket - Affectation Numéro

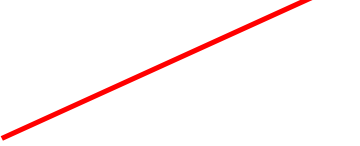
Exemple :

```
#define PORT 12345
```

```
void main()
```

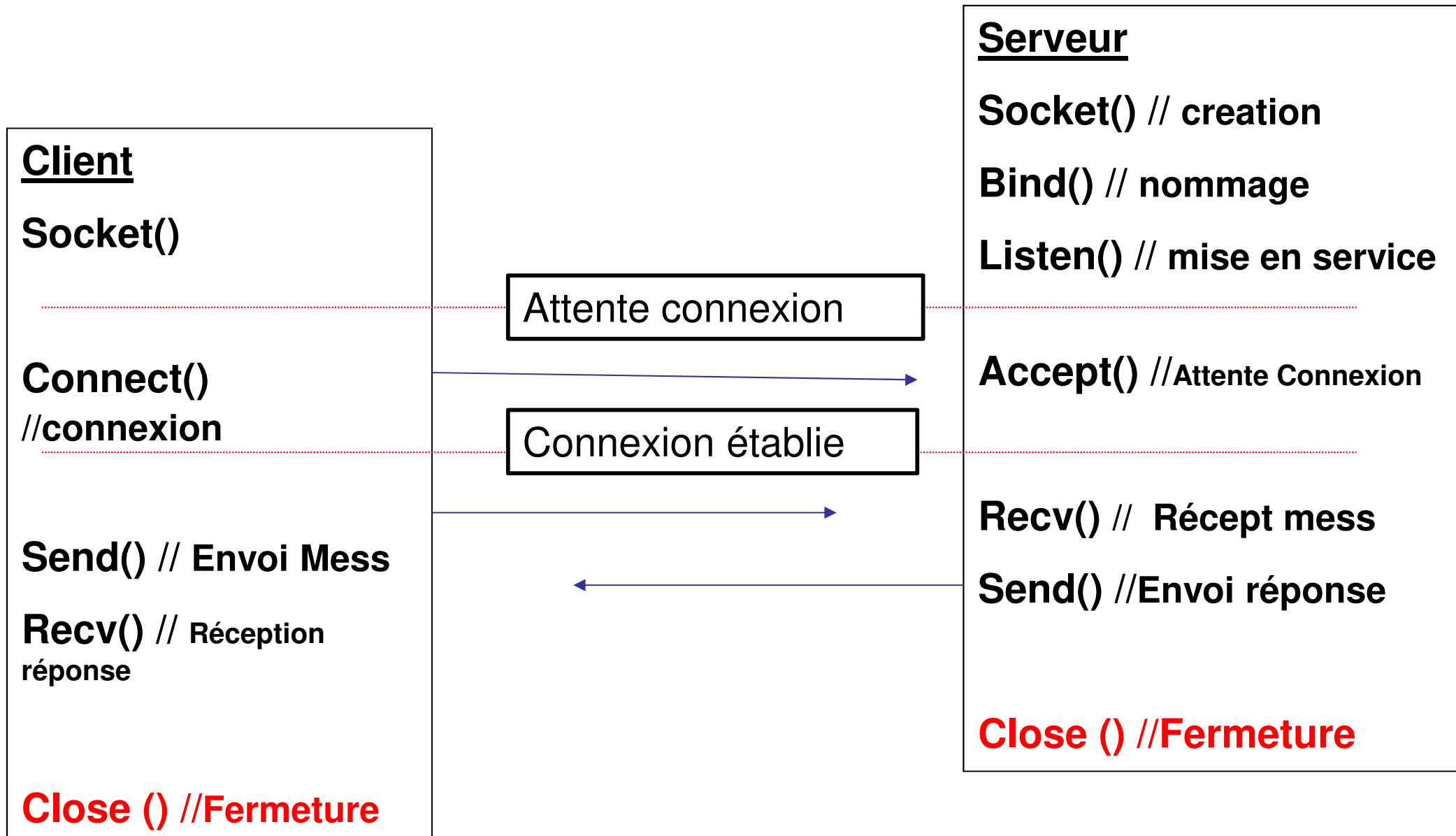
AF_INET	12345	127.0.0.1	00000000
---------	-------	-----------	----------

```
{ int sock, lg;  
  struct sockaddr_in local;  
  
  bzero(&local, sizeof(local)); // Mise à zéro du champs adresse  
  local.sin_family = AF_INET; // C'est un champs internet  
  local.sin_port = htons(PORT); // le port = 12345  
  local.sin_addr.s_addr = INADDR_ANY; // l'adresse IP de la machine  
  ...  
  bind(sock, (struct sockaddr *)&local, sizeof(struct sockaddr));  
  ...  
}
```



Remarque : htons() convertit la variable PORT en un entier sur 2 octets

Interface Socket



Interface Socket - Fermeture

int close (sock)

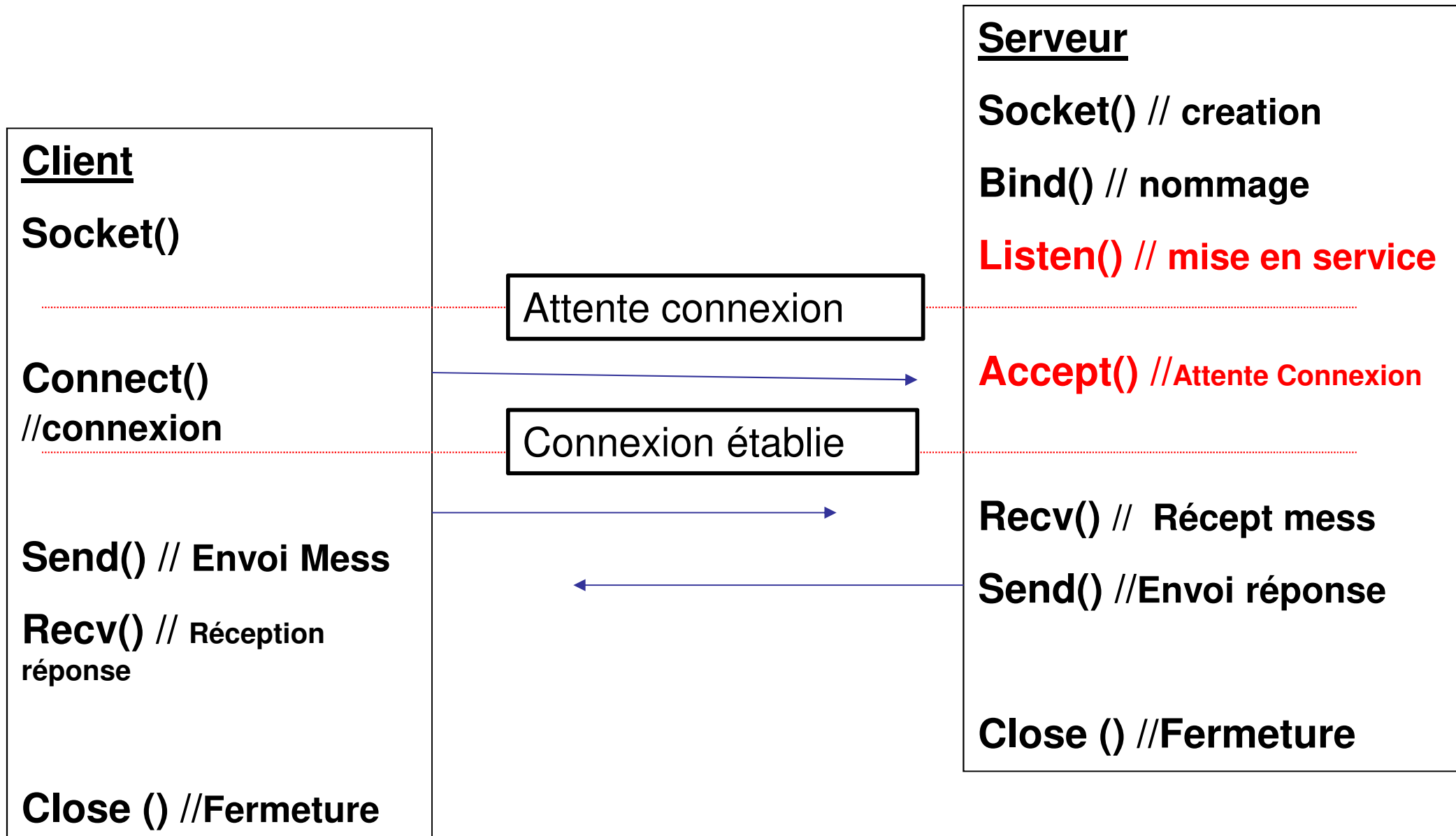


descripteur de la socket

Remarque : Il est important de fermer chaque socket après utilisation, faute de quoi, dans le cas d'un serveur, il ne sera pas possible de réutiliser le même n° de port.

Attention : il faut arrêter le client avant le serveur, faute de quoi le port restera inutilisable un certain temps.

Interface Socket



Interface Socket – Mise à l'écoute

int listen (sock, nb)

Le paramètre **nb** définit une longueur maximale pour la file d'attente des connexions.

Si une nouvelle connexion arrive alors que la file est pleine, deux cas sont possibles :

- 1 - le client reçoit une erreur indiquant **ECONNREFUSED**,
- 2 - si le protocole supporte les retransmissions (ce qui est souvent le cas), la requête est ignorée afin qu'une nouvelle tentative réussisse.

Interface Socket – Attente connexion

int accept (sock, p_struct_adress, socklen_t)

La fonction « accept » est bloquante, elle attend la demande de connexion d'un client.

Les paramètres :

sock = socket d'écoute

p_struct_adress = est un champs de type sockaddr_in, après la connexion il contiendra les n° de port et @IP de la machine cliente.

→ La fonction retourne un entier qui correspond au port utilisé pour l'échange des données (différent du port d'écoute).

→ **Ce port devra être utilisé par la suite pour les échanges de données.**

Interface Socket – Attente connexion

Exemple :

```
void main()
```

```
{int sock, socket2;
```

```
struct sockaddr_in local; // champs d entete local
```

```
struct sockaddr_in distant; // champs d entete distant
```

```
...
```

```
listen(sock, 5) ;
```

```
socket2=accept(sock, (struct sockaddr *)&distant, sizeof(struct sockaddr);
```

```
...
```

```
}
```

AF_INET	12345	127.0.0.1	000000
---------	-------	-----------	--------



Local

N° port et @IP du serveur
à initialiser avant le “bind”

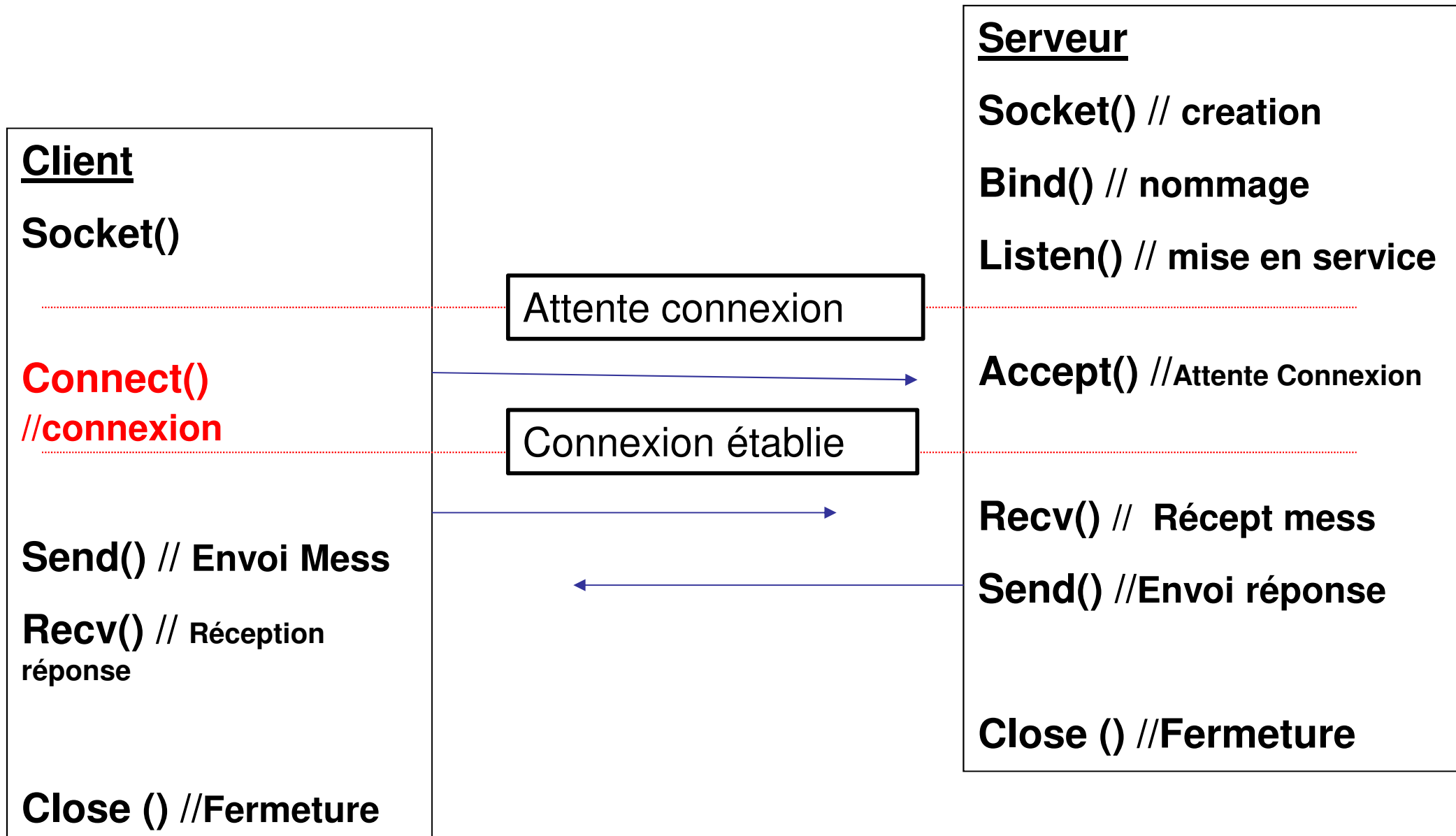
AF_INET	25123	127.0.0.1	000000
---------	-------	-----------	--------



Distant

N° port et @IP du client
initialisé par “accept”

Interface Socket



Interface Socket – Connexion client

int connect (sock, struct_adr, lgadr)

La fonction « connect » est bloquante, elle attend la réponse du serveur.

Les paramètres :

sock = socket locale

struct_adr = est un champs de type sockaddr_in, qui contiendra les n° de port et @IP du serveur

Interface Socket – Connexion client

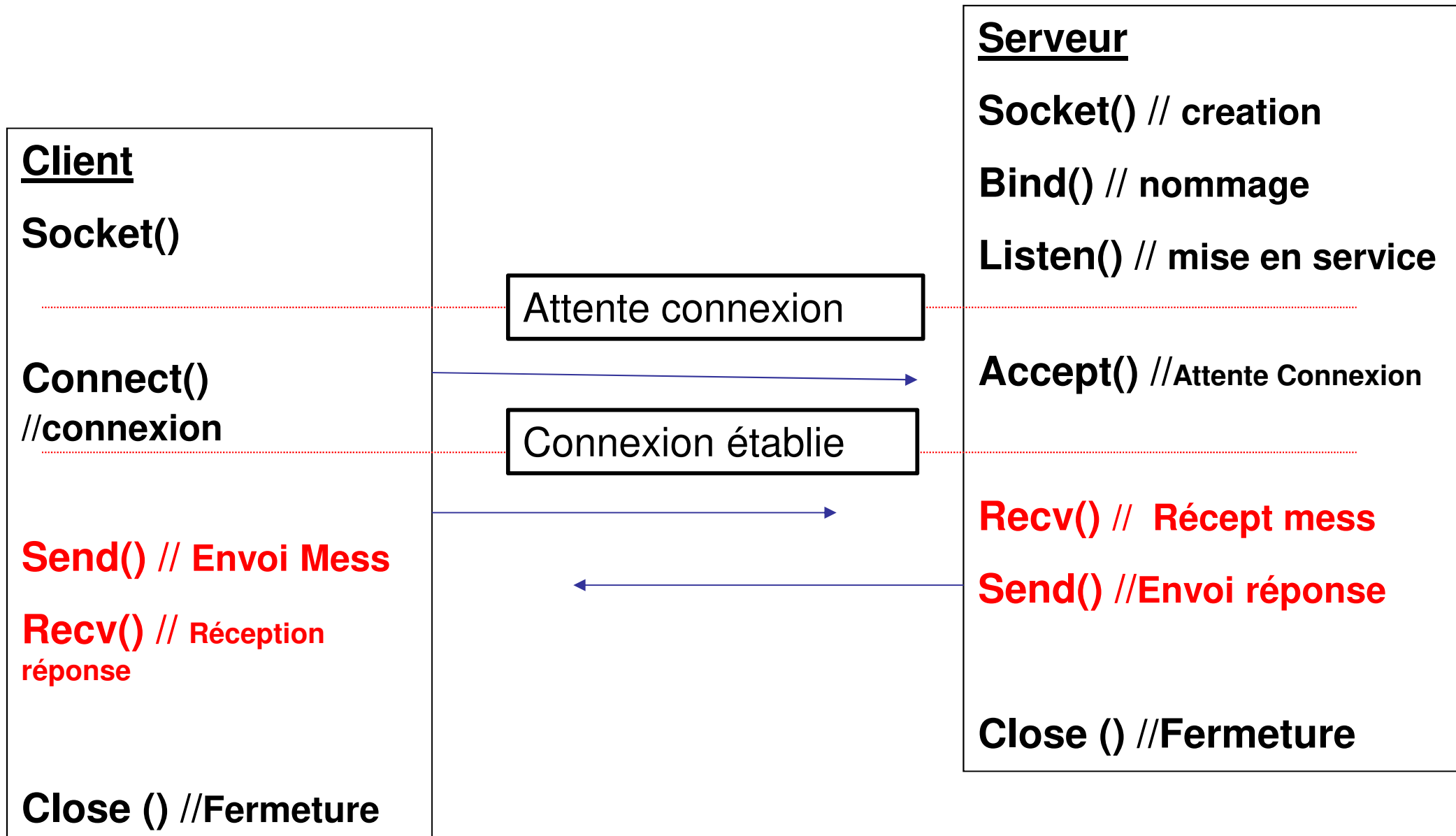
Exemple :

```
#define SERV "127.0.0.1"           // adresse IP = boucle locale
#define PORT 12345                // port d'ecoute serveur

struct sockaddr_in  serv_addr;    // zone adresse
struct hostent      *server;      // nom serveur

main()
{ server = gethostbyname(SERV);    // verification existence adresse
  bzero(&serv_addr, sizeof(serv_addr)); // preparation champs entete
  serv_addr.sin_family = AF_INET;  // Type d'adresses
  serv_addr.sin_port = htons(PORT); // port de connexion du serveur
  bcopy(server->h_addr, &serv_addr.sin_addr.s_addr, server->h_length); // @ IP
  ...
  connect (sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr) );
  ...
}
```

Interface Socket



Interface Socket - échange

Fonctions de lecture / écriture

int send (sock, msg, lg,0); // envoi de la chaine msg

int recv (sock, msg, lg,0) // réception du message dans msg

Remarques :

sock = socket locale (client) ou socket d'échange (serveur)

msg = chaîne de caractère

lg = longueur de la chaîne msg

0 = options (ici pas d'option)

Remarque : il n'est pas nécessaire de préciser les adresses des émetteurs et destinataires

Interface Socket - échange

Autres fonctions de lecture / écriture

int write (sock, msg, lg); // envoi de la chaine msg

int read (sock, msg, lg); // réception du message dans msg

Remarques :

sock = socket locale (client) ou socket d'échange (serveur)

msg = chaîne de caractère

lg = longueur de la chaîne msg

Remarque : il n'est pas nécessaire de préciser les adresses des émetteurs et destinataires

Interface Socket - échange

Notion de flux de données

En mode connecté les processus client et serveur peuvent traiter des messages de taille différente.

Par exemple la longueur des caractères lus peuvent correspondre à plusieurs messages déposés.

La lecture est destructive.

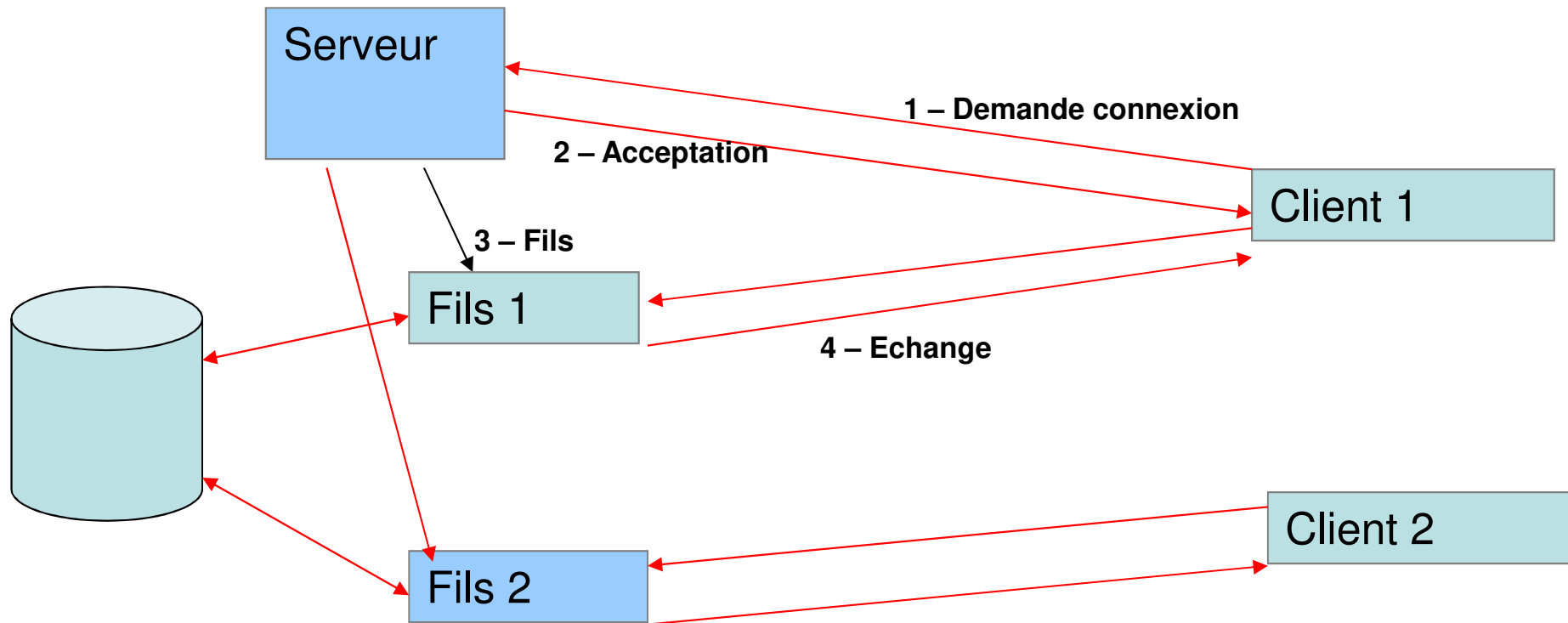
- `#include <netinet/in.h>`
- `#include <sys/socket.h>`
- `#include <sys/types.h>`
- `#define PORT 12345`
- `void main()`
- `{ int sock, sock2, lg;`
- `struct sockaddr_in local;`
- `struct sockaddr_in distant;`
- `local.sin_family = AF_INET;`
- `local.sin_port = htons(PORT);`
- `local.sin_addr.s_addr = INADDR_ANY;`
- `bzero(&(local.sin_zero), 8);`
- `lg = sizeof(struct sockaddr_in);`
- `sock=socket(AF_INET, SOCK_STREAM, 0);`
- `bind(sock, (struct sockaddr *)&local, sizeof(struct sockaddr));`
- `listen(sock, 5);`
- `while(1)`
- `{sock2=accept(sock, (struct sockaddr *)&distant, &lg);`
- `close(sock2);`
- `}`
- `}`

Serveur

Client

- `#define SERV "127.0.0.1"`
- `#define PORT 12345`
- `void main()`
- `{ int port,sock;`
-
- `struct sockaddr_in serv_addr;`
- `struct hostent *serveur;`
-
- `port = PORT;`
- `serveur = gethostbyname(SERV);`
- `if (!serveur){fprintf(stderr, "Problème serveur \"%s\"\n",SERV);exit(1);}`
-
- `sock = socket(AF_INET, SOCK_STREAM, 0);`
- `bzero(&serv_addr, sizeof(serv_addr));`
- `serv_addr.sin_family = AF_INET;`
- `bcopy(serveur->h_addr, &serv_addr.sin_addr.s_addr,serveur->h_length);`
- `serv_addr.sin_port = htons(port);`
-
- `if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)`
- `{perror("Connexion impossible");exit(1);}`
- `}`

Structure générale application (Client/Serveur)

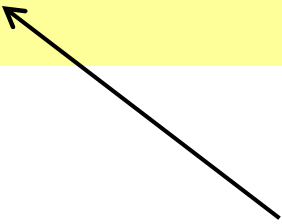


En règle générale un serveur doit être capable de dialoguer avec plusieurs clients en même temps

Structure générale application (Client/Serveur)

Il sera nécessaire de dupliquer autant de fois que nécessaire, les instructions chargées du dialogue avec le client, juste après la connexion.

```
while(1)
{ socket2=accept(sock, (struct sockaddr *)&distant, &lg);
  {
    ....
    // instructions chargées du dialogue avec le client
    ....
    close(socket);
  }
}
```



Instructions à dupliquer

Structure générale application (Client/Serveur)

Deux techniques sont possibles pour cela :

- La duplication du processus via une fonction : `fork()`,
- La création d'un Thread, contenant les instructions du dialogue.

Exemple :

```
while(1)
{ socket2=accept(sock, (struct sockaddr *)&distant, &lg);
  if (fork()==0)
  {....
    // instructions chargées du dialogue avec le client
    ....
    close(socket);
  }
}
```