API Socket

Programmation

Présentation

L'API (Application Programme Interface) Socket est une interface entre les programmes d'applications et les couches réseau. (Origine noyau Unix BSD)

Ensemble de primitives destiné aux programmeurs pour la communications :

* inter-processus locaux.

* processus distants à travers un réseau.

Les sockets permettent d'accéder au réseau, via un modèle client-serveur, de la même manière qu'on accède à un fichier.

Communication client-serveur

La communication entre deux processus, qu'ils soient sur la même machine ou sur différentes machines, se fait sur le modèle client/serveur.

Le rôle de chaque processus sera dissymétrique; le processus client aura le rôle actif de "demandeur" de service tandis que le processus serveur se contentera de rester en "écoute" et de répondre aux différents services qui lui seront demandés.

Voir cours Annexe p1 & p2

Définition d'une socket

Une socket est un point de connexion servant d'élément de référence dans les échanges entre processus.

But: faire communiquer des processus via la pile TCP/IP à partir d'un descripteur à la manière des fonctions open(), read(), write() comme pour les fichiers classiques.

Remarques:

- Le fait qu'une socket possède un descripteur au même titre qu'un fichier fait qu'on pourra rediriger les entrées/sorties standards sur une socket.
- Tout nouveau processus créé par un fork() hérite des descripteurs, et donc des sockets du processus père .

Création d'une socket -fonction socket()-

La primitive socket() crée une socket. Celle-ci n'est rattachée à rien d'autre que la machine qui la crée. L'établissement de la connexion avec le réseau se fait plus tard.

```
#include <sys/socket.h>
int socket (int dom, int type, int proto);
```

int **dom**: domaine de la socket (AF_UNIX, AF_INET, etc.)
int **type**: type de la socket (SOCK_STREAM ou SOCK_DGRAM)
int **proto**: protocole de communication.
Mettre "0" dans ce paramètre indique au système de se baser sur le type défini dans le paramètre précédent (type) pour définir automatiquement son protocole de communication.

Création d'une socket -domaine-

Les sockets peuvent utiliser plusieurs protocoles de communication mais il faut la même convention d'@.
On définit ainsi des domaines de communications qui doivent être spécifiés lors de la création de la socket :
AF_UNIX: domaine UNIX (même machine Unix)
AF_INET:domaine INTERNET (via TCP/IP)
AF_IPX: domaine IPX
AF_X25: domaine X25, etc.

Voir cours p2

Il existe une structure générique "sockaddr " (définie dans <sys/socket.h>) mais pour le programmeur, chaque domaine d'utilisation nécessite une structure d'un type précis :
AF_INET: la structure est **struct sockaddr_in** définie dans <netinet/in.h>
AF_UNIX: la structure est struct sockaddr_un et est définie dans <sys/un.h>

Création d'une socket -structures-

La structure " sockaddr_in " définie dans <netinet/in.h> est à employer domaine Internet (AF_INET). Elle permet d'indiquer quelle sera l'adresse d'une machine distante et le numéro de port du service associés à la connexion.

```
#include <sys/types.h>
#include <netinet/in.h>
struct sockaddr_in {
short sin_family; /* famille de l'adresse */
u_short sin_port;    /* numéro de port */
struct in_addr sin_addr; /* @ machine */
char sin_zero[8];   /* champ à zéro */
};
```

Voir cours Annexe p3

Création d'une socket -structure in_addr-

La structure in_addr définie dans <netinet/in.h> définit le format de la variable sin_addr de la structure sockaddr_in. Ce découpage permet une plus grande souplesse d'évolution possible (IP V6par exemple).

```
#include <sys/types.h>
#include <netinet/in.h>
struct in_addr {
u_long s_addr;          / * adresse IP */
};
```

Voir cours Annexe p3

Exemple : @IP dotnet : 172.16.79.211
u_long s_adrr = 0xAC104FD3

Création d'une socket -type-

Précise la nature de la communication supportée par la socket. Les principales propriétés recherchées sont :

- fiabilité de la transmission;
- préservation de l'ordre de transmission des données;
- non-duplication des données émises;
- communication en mode connecté.
- route inchangé durant toute la durée de l'émission
- envoi de messages urgents;
- ...

Création d'une socket -type- (suite)

Les différents types (définies dans <sys/socket.h>)

SOCK_DGRAM: mode non-connecté envoi de datagrammes de taille bornée, non fiable dans le domaine INTERNET, cela correspond au protocole **UDP**).

SOCK_STREAM: communications fiables en mode connecté orienté flots d'octets (dans le domaine INTERNET, cela correspond au protocole **TCP**).

SOCK_RAW: accès aux protocoles de plus bas niveau dans le domaine INTERNET, cela correspond au protocole Internet IP, ICMP.

Création d'une socket -signaux-

Trois signaux sont rattachés aux sockets :

SIGIO: Indique qu'une socket est prête pour une entrée/sortie asynchrone. Le signal est envoyé au processus (ou au groupe de processus) associé au signal.

SIGURG: Indique que des données express sont arrivées sur une socket. Il est envoyé au processus (ou au groupe de processus) associé au signal.

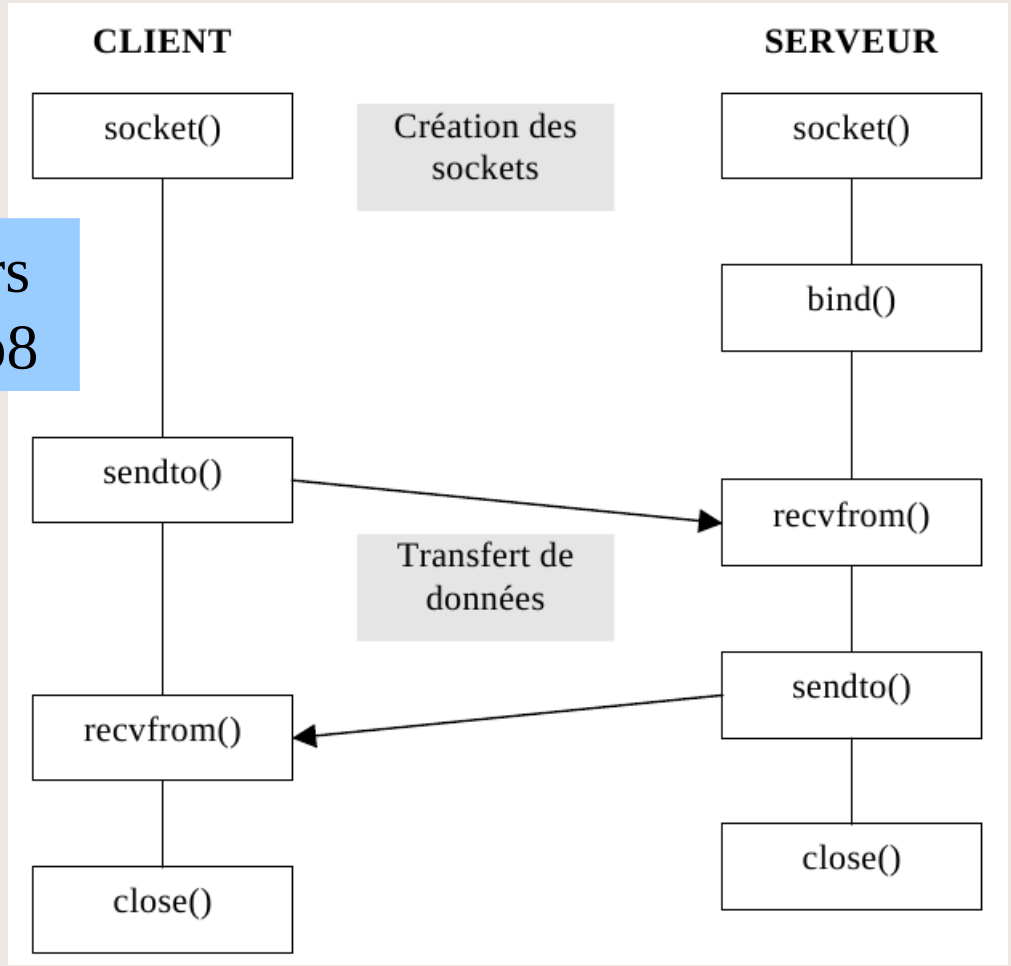
SIGPIPE: Indique qu'il n'est plus possible d'écrire sur une socket. Il est envoyé au processus associé à la socket.

Utilisation d'une socket en mode non connecté

- Un processus voulant émettre un message à destination d'un autre doit disposer d'une socket locale.
- Il doit en outre connaître une adresse sur le système distant auquel appartient son interlocuteur.
- Dans ce type de communication, rien n'indique au processus demandeur que son interlocuteur dispose d'une socket attachée à l'adresse détenue.
- En mode non-connecté, toute demande d'envoi d'un message doit comporter l'adresse de la socket destinataire.

fonctionnement en mode non connecté

Voir cours
Annexe p8



Utilisation d'une socket en mode connecté

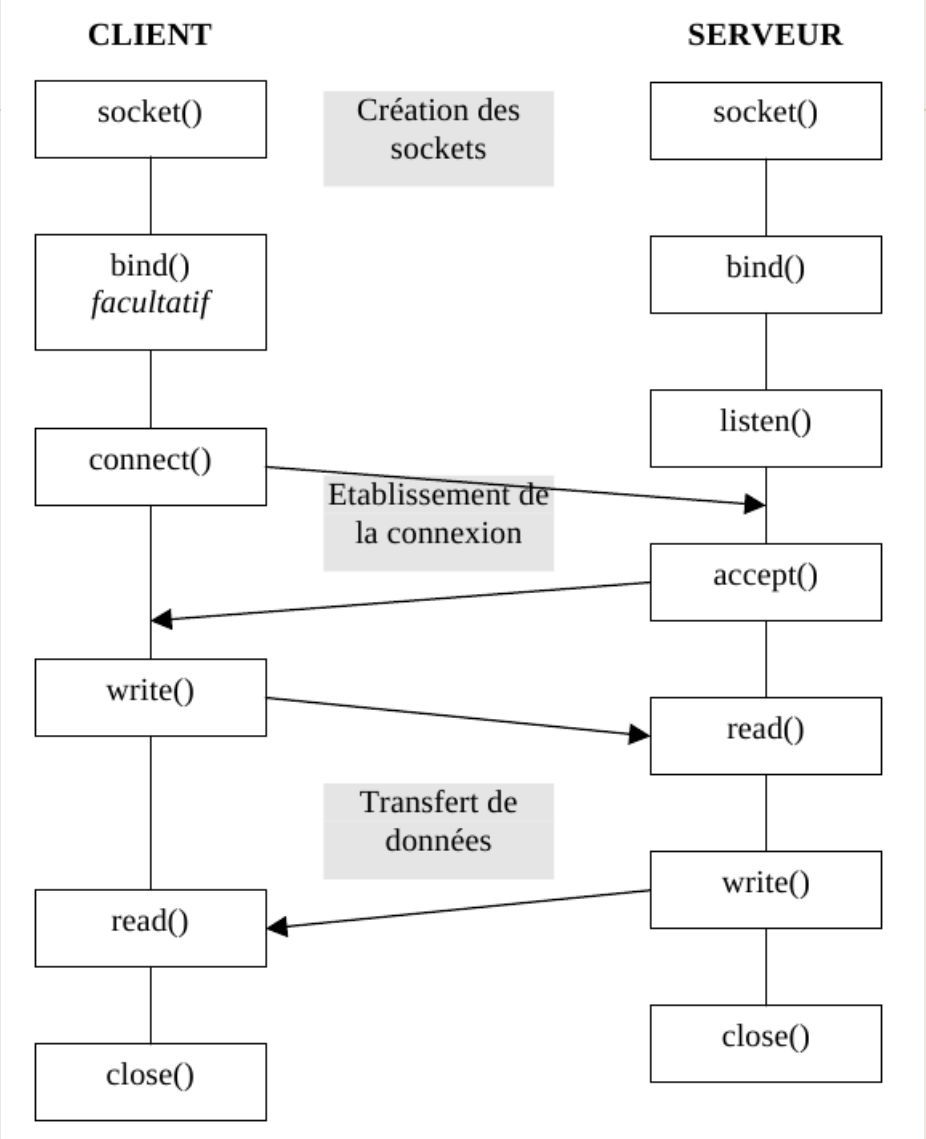
C'est le mode de communication utilisé par la plupart des applications standard utilisant le protocole "Internet" (telnet, ftp, etc.) ou les applications Unix (rlogin, rsh, rcp). Ce mode apporte une grande fiabilité dans les échanges de données mais cela se traduit par un accroissement du volume total des données à transmettre.

Principe

Pour fonctionner dans ce mode, il faut au préalable réaliser une connexion entre deux points, ce qui revient à établir un "**circuit virtuel**". Une fois celle-ci créée, le transfert de données se fait d'une manière continue (notion de "flot de données").

Fonctionnement en mode connecté

Voir cours
Annexe p8

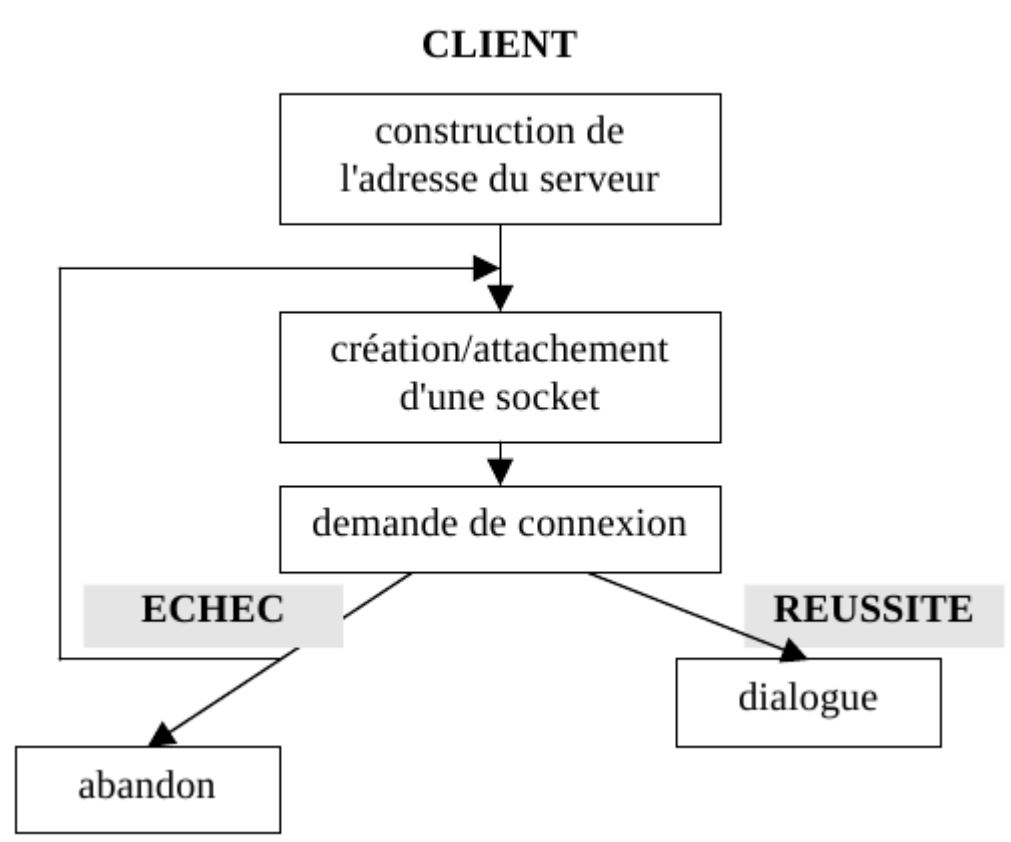


Fonctions bloquantes

Pour le client:
connect() jusqu'à ce que le serveur effectue un accept()
write() si le tampon d'émission est plein
read() jusqu'à ce qu'un caractère au moins soit reçu

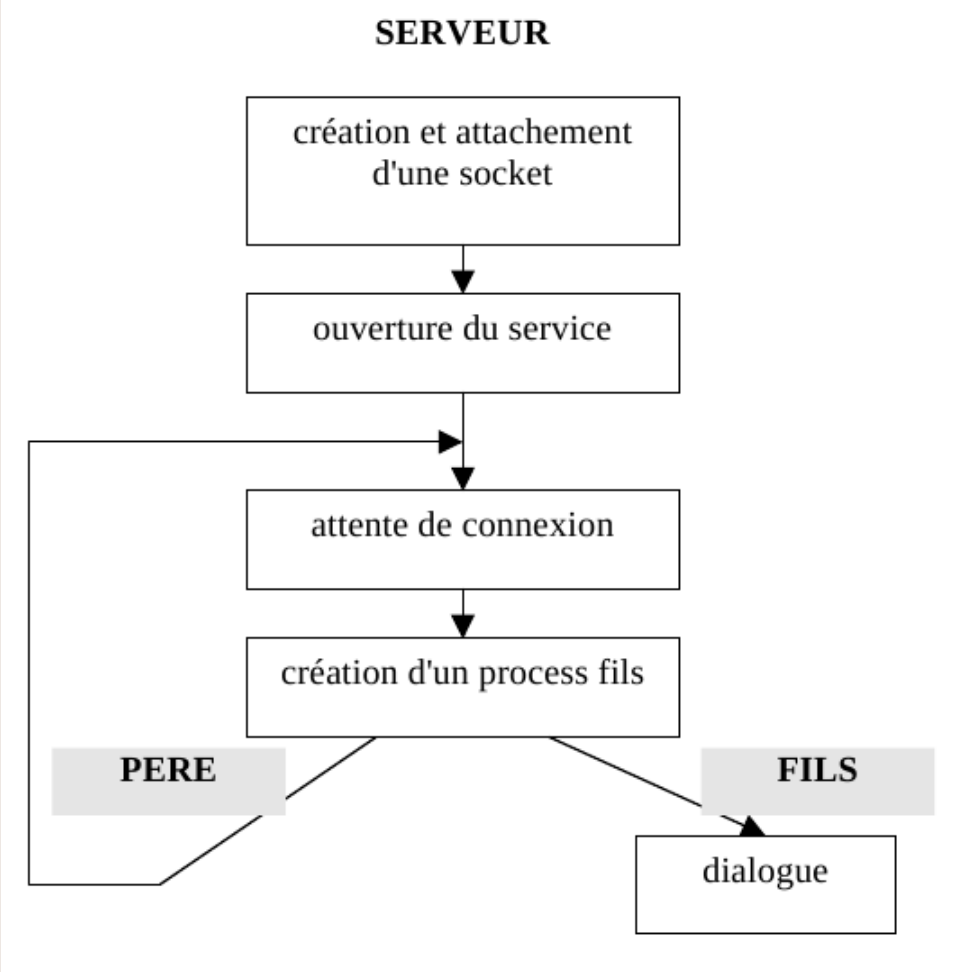
Pour le serveur:
accept() jusqu'à ce que le client effectue un connect()
read() jusqu'à ce qu'un caractère au moins soit reçu
write() si le tampon d'émission est plein

Principe d'un dialogue Client / Serveur



Voir cours p4

Principe d'un dialogue Client / Serveur



Voir cours p4

Principe du mono-client ou multi-clients ?

Homogénéisation des formats de données

Le codage d'un nombre sur plus d'un octet diffère selon l'architecture d'une machine ou d'une autre (octet de poids fort ou de poids faible en premier ou en dernier).

Les sockets devant pouvoir transmettre des informations en milieu hétérogène impliquent une représentation de ces nombres dans un format compréhensible quel que soit l'architecture de la machine.

Ce format est appelé "forme réseau" ou "big-endian".

Homogénéisation des formats de données

Les primitives htonl(), htons(), ntohl et ntohs() permettent de transformer respectivement un entier long ou court en entier "représentation réseau", et un entier "représentation réseau" en entier long ou court.

```
#include <sys/types.h>
#include <netinet/in.h>
u_long htonl (u_long hostlong);
u_short htons (u_short hostshort);
u_long ntohl (u_long netlong);
u_short ntohs (u_short netshort);
```

Voir cours
Annexe p5

Récupération d'informations sur le réseau

Le langage C met à la disposition du programmeur un ensemble de fonctions et de structures prédéfinies qui permettent l'accès aux renseignements contenus dans les différents fichiers de configuration réseau.

Les informations récupérées sont stockées dans des variables d'un type prédéfini. C'est au programmeur de déclarer ces variables et leurs noms est donc libre... mais pas les membres des structures associées.

Avoir des informations sur une machine

Les primitives gethostbyname() et gethostbyaddr() vont chercher dans le fichier "/etc/hosts" les informations sur une machine dont on connaît le nom ou l'@.

```
#include <netdb.h>
struct hostent {
char *h_name; /* nom de la machine */
char **h_aliases; /* liste des alias */
int h_addrtype; /* type d'adresse */
int h_length; /* longueur de l'adresse */
char **h_addr_list; /* liste d'adresses */
#define h_addr h_addr_list[0] /* 1er @ de la liste */
};
```

Voir cours
Annexe p5

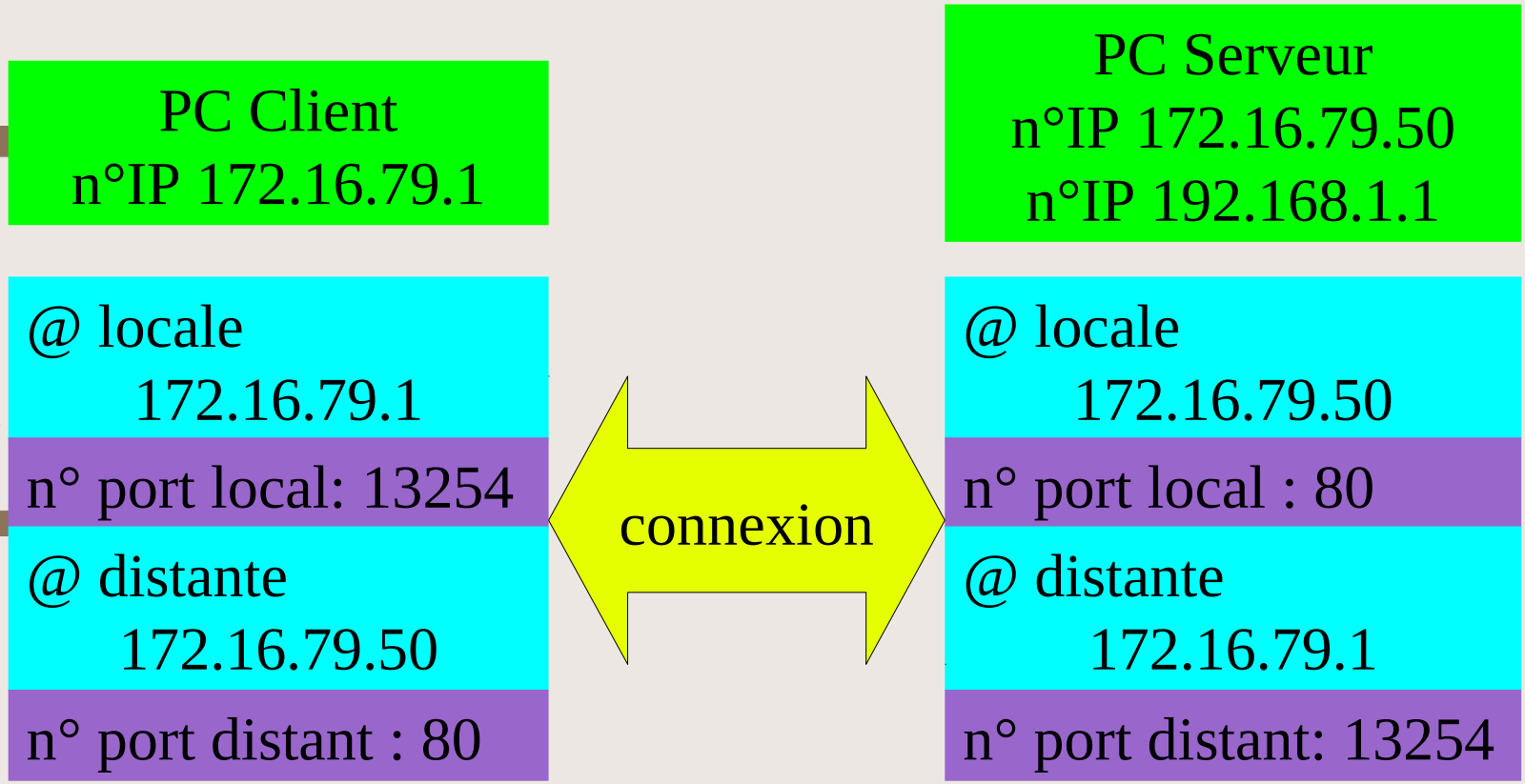
Avoir des informations sur les services et n° de port

Les primitives getservbyname() et getservbyport() vont chercher dans le fichier "/etc/services" les informations sur les services disponibles dont on connaît le n° de port ou le nom.

```
#include <netdb.h>
struct servent {
char *s_name; /* nom du service */
char **s_aliases; /* liste des alias */
int s_port; /* numéro de port */
char *s_proto; /* protocole utilisé */
};
```

Voir cours
Annexe p5

Résumé



Contenu essentiel des sockets Client & Serveur

Exemples de programme

Info.c

Squelettes Client/Serveur UDP

Squelettes Client/Serveur TCP