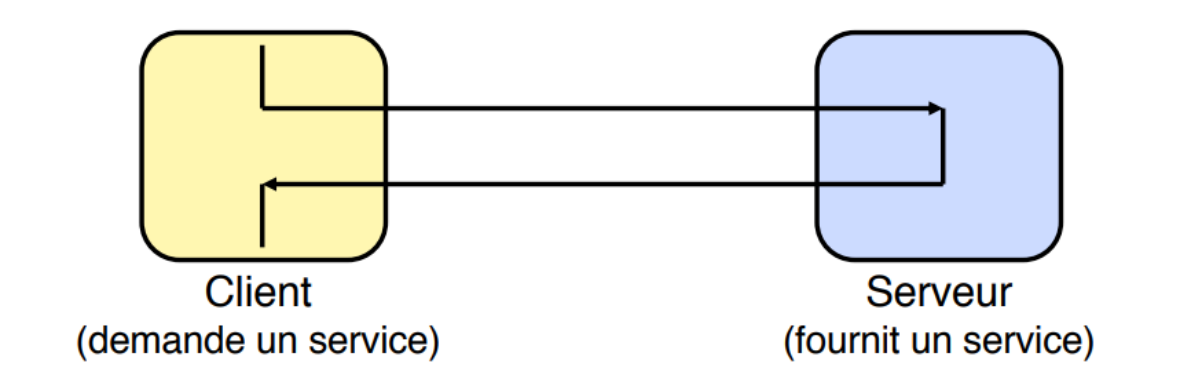
Développement Système

Les Sockets (Linux)

Développement Système: Les sockets

Introduction : le réseau vu par l'utilisateur



- Pour le client, un service est souvent désigné par un nom symbolique. Ce nom doit être converti en une adresse interprétable par les protocoles du réseau.

- La conversion d'un nom symbolique (par ex. www.google.com) en une adresse IP (216.239.39.99) est à la charge du service DNS

Développement Système: Les sockets

Introduction : le réseau vu par l'utilisateur

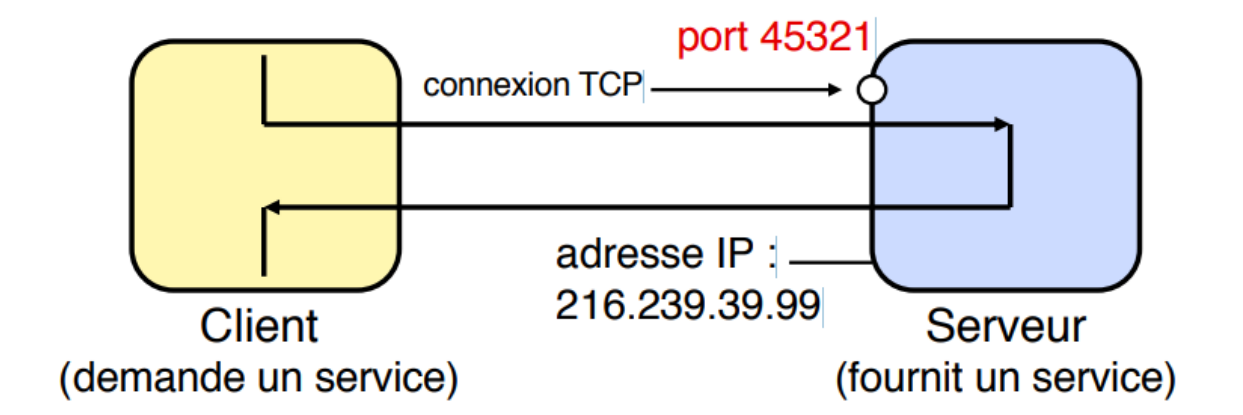
En fait l'adresse IP du serveur ne suffit pas, car le serveur (machine physique) peut comporter différents services; il faut préciser le service demandé au moyen d'un numéro de port, qui permet d'atteindre un processus particulier sur la machine serveur.

Un numéro de port comprend 16 bits (0 à 65 535) et est associé à un protocole de transport donné (le port TCP et le port UDP désignent des objets distincts).

Les numéros de 0 à 1023 sont réservés, par convention, à des services spécifiques et sont identiques sur toutes les machines.

Exemples (avec TCP) :

- 7 : echo
- 25 : SMTP (acheminement mail)
- 443 : HTTPS (HTTP sécurisé)
- 22 : SSH
- 80 : HTTP (serveur web)
- 465 : SMTPS (SMTP sécurisé)



Développement Système: Les sockets

Introduction : histoire et définition

- Les sockets sont apparus en 1983 dans la version BSD (Berkeley Software Distribution) du système Unix et sont à présent disponibles sur tous les systèmes Unix courants.
- C'est un mécanisme de communication entre processus quelconques s'exécutant sur le même système ou sur des systèmes éloignés.
- On peut les assimiler à une extension de la portée des tubes nommés, pour pouvoir faire dialoguer des processus s'exécutant sur des machines différentes.

→ ***Socket*** *désigne à la fois une bibliothèque d'interface réseau et l'extrémité d'un canal de communication (point de communication) par lequel un processus peut émettre et recevoir des données*

C'est un point bidirectionnel qui est désigné par un descripteur comme pour les fichiers, tubes, etc.

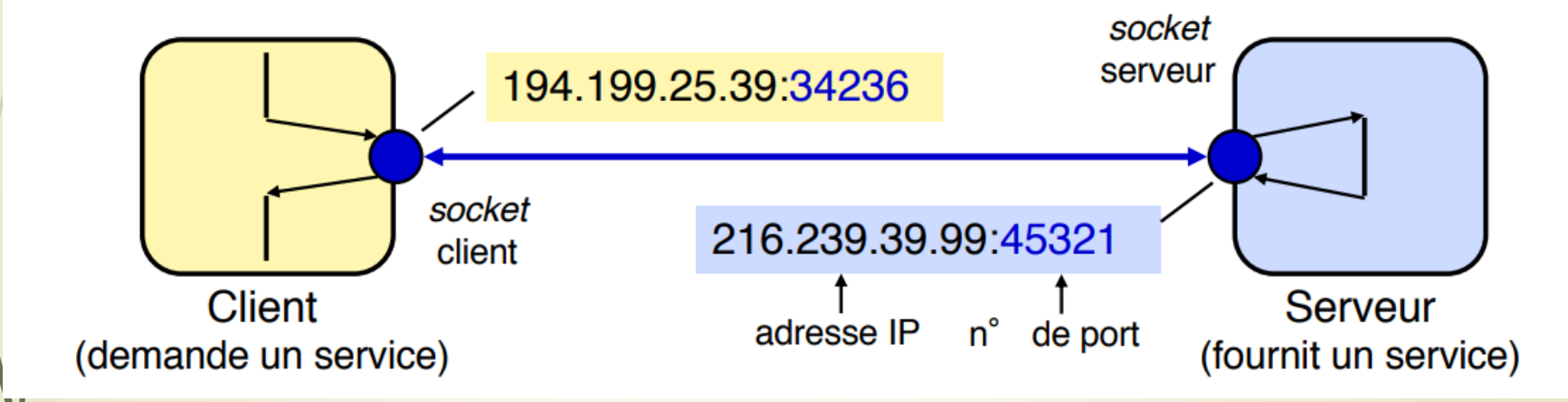
→ *L'Interface socket est un ensemble de primitives qui permettent de gérer l'échange de données entre processus, que ceux-ci soient implantés sur la même machine ou non.*

Développement Système: Les sockets

Introduction : définition

Définition :

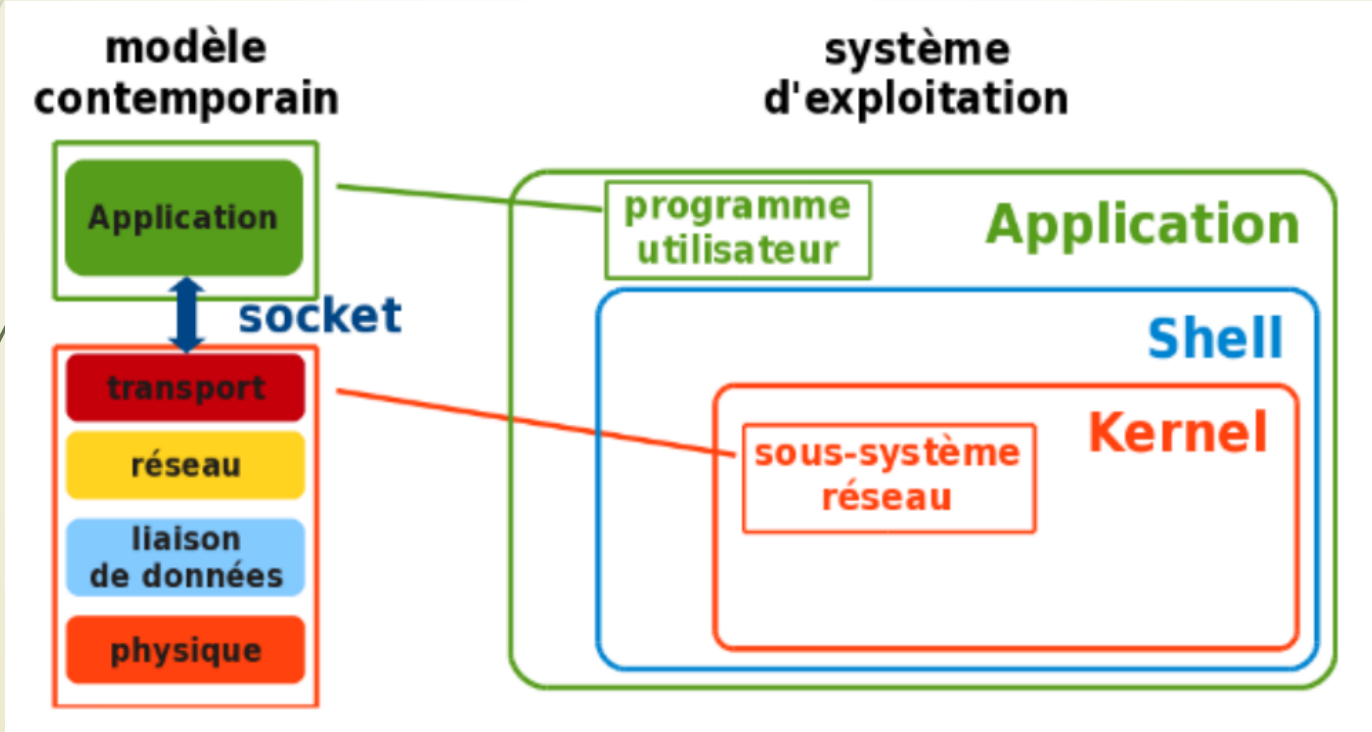
Un socket est un point d'accès aux couches réseau TCP/UDP. Les sockets constituent un mécanisme qui permet la communication sur le réseau Internet ou entre processus locaux tournant sur une même machine;



Développement Système: Les sockets

Introduction : Situation

Un socket est un point d'accès aux couches réseau TCP/UDP Il permet donc d'accéder au réseau comme un fichier.

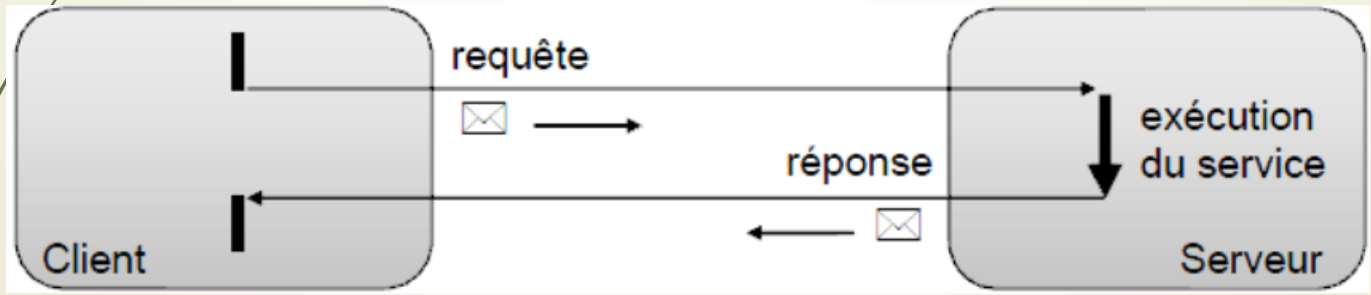


Développement Système: Les sockets

Principe de base d'un socket :

→La communication via sockets s'appuie sur le principe du modèle C/S :

- ✓ Client : processus qui demande l'exécution d'un service à un autre processus
- ✓ Serveur : processus qui accompli le service sur demande d'un client, et lui transmet le résultat



Développement Système: Les sockets

Principe de base d'un socket :

Pour établir une communication avec une machine distante, il faut donc :

1. S'attribuer un N° port;
2. Connaître l'adresse internet @IP de la machine avec laquelle les échanges vont s'effectuer;
3. Connaître le N° port du service distant.

La communication doit se repérée de manière unique, et ce par le quintuple :

1. Protocole transport : TCP ou UDP;
2. Adresse 1 : adresse internet de la machine 1;
3. N° port 1: numéro du port du processus de la machine 1;
4. Adresse 2 : adresse internet de la machine 2;
5. N° port 2: numéro du port du processus de la machine 2

Développement Système: Les sockets

Principe de base d'un socket :

1. chaque machine crée un socket,
2. Chaque socket sera associé à un port de sa machine hôte,
3. Les deux sockets seront explicitement connectés si on utilise un protocole en mode connecté,
4. Chaque machine écrit et/ou lit dans son socket,
5. Les données vont d'un socket à un autre à travers le réseau,
6. Une fois l'échange terminé, chaque machine ferme son socket.

Un socket est défini par:

→ **une famille** (sockets locaux (AF_UNIX); sockets Internet (AF_INET)

→ **un mode de communication** (Datagrammes (SOCK_DGRAM), Flux de données (SOCK_STREAM)

→ **un protocole** (IP,UDP,TCP, …)

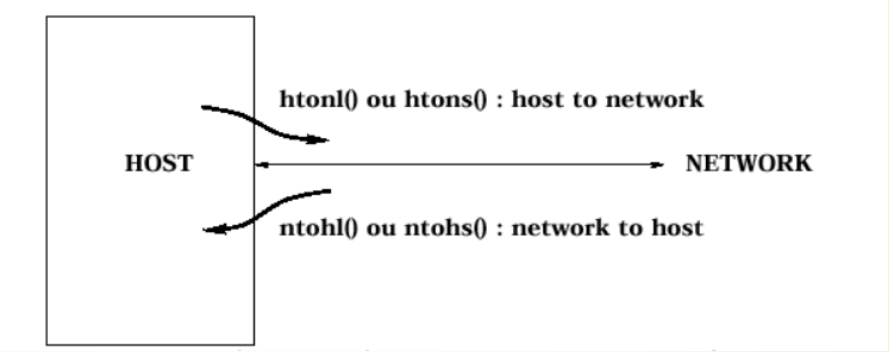
Développement Système: Les sockets

Expression des adresses machine et n° de ports

Une communication met en jeu deux extrémités identifiées par une adresse machine et un port. Ces extrémités peuvent avoir des processeurs différents, des OS différents, des codages différents des nombres, ….

Il existe des fonctions de conversions de valeurs " machine " dans le format " réseau "

- ***htons(…)***
- ***htonl(…)***
- ***ntohl(…)***
- ***ntohs(…)***



Et des fonctions d'information sur le réseau qui répondent dans le format réseau :

- ***gethosbyname(…)***
- ***gethostbyaddr(…)***

Développement Système: Les sockets

Syntaxes des primitives de base des sockets

- *Création d'un socket:*

```
int socket( int domaine /* (famille) ex : AF_UNIX, AF_INET */,
int type /* ex : SOCK_DGRAM,SOCK_STREAM */,
int protocole /* 0 : protocole par défaut, IPPROTO_TCP... */ );
```

Cette primitive retourne le descripteur du socket en cas de réussite

- *Association d'un socket à un port:*

```
int bind( int descripteur /* socket */ ,
struct sockaddr *ptr_adresse /* pointeur sur adresse */,
int lg_adresse /* taille adresse */ );
```

- *Fermeture et suppression d'un socket:*

```
int close( int descripteur /* socket */ );
```

Développement Système: Les sockets

Structures de données des adresses réseaux

Adresse Internet d'une socket :

```
struct sockaddr_in {
short sin_family ; /* AF_INET */
u_short sin_port ; /* Port */
struct in_addr sin_addr; /* Adresse IP */
char sin_zero[8] ;
};
```

Adresse Internet d'une machine

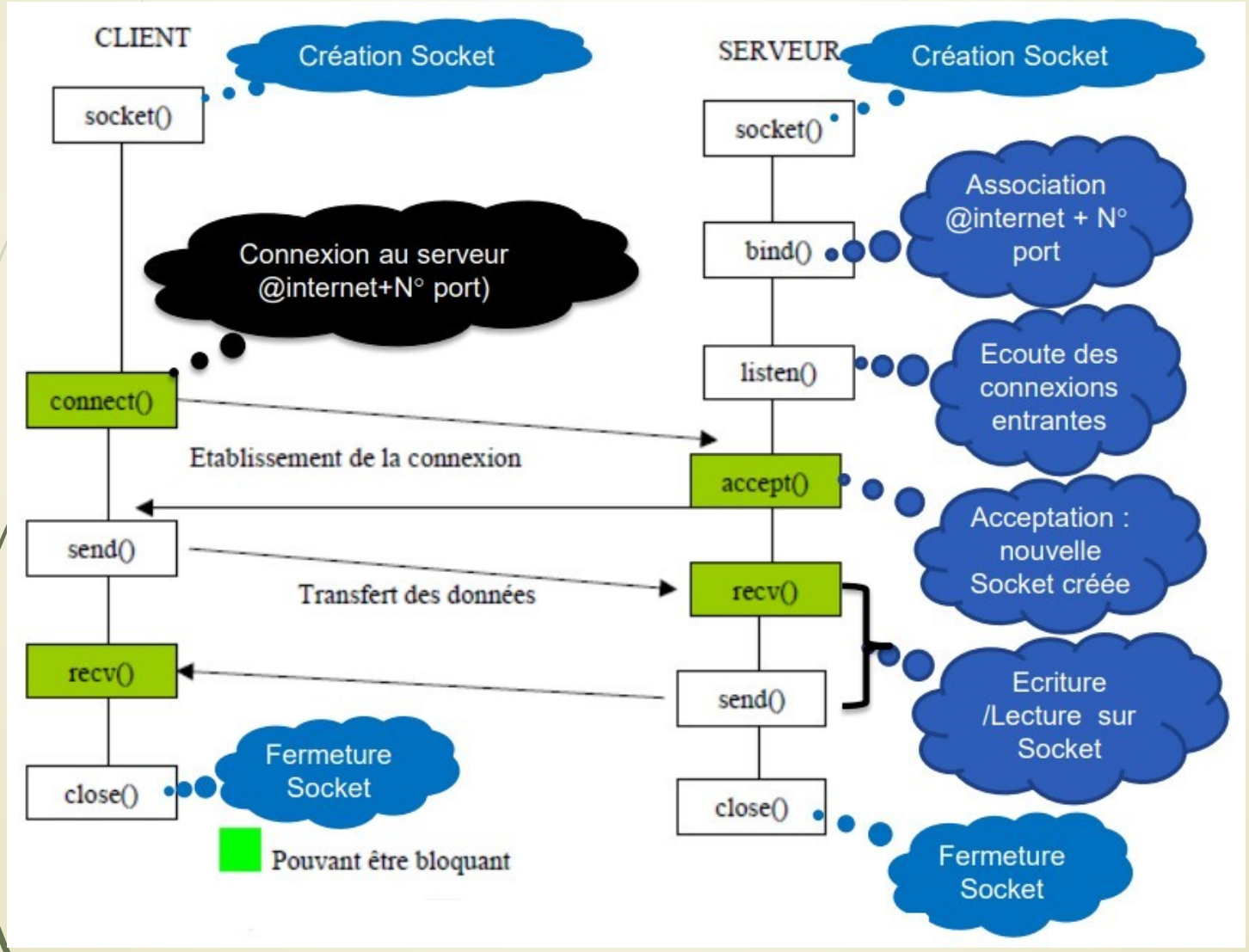
```
struct in_addr {
u_long s_addr ; };
```

Syntaxe :

```
sockaddr_in adr;
 adr.sin_addr.s_addr = … /* adr machine */;
```

Développement Système: Les sockets

Programmation en mode connecté : TCP



Développement Système: Les sockets

Programmation en mode connecté : serveur TCP

Un serveur en mode connecté doit attendre une demande de connexion de la part d'un client, puis traiter la (ou les requêtes) envoyée(s) sur cette connexion par le client.

Les fonctions d'attente et de traitement sont séparées, pour permettre au serveur d'attendre de nouvelles demandes de connexion pendant qu'il traite des requêtes en cours.

→Le socket serveur est associé à un port connu des clients (par ex. port 80 pour un serveur web, etc.)

→Le socket de communication est associé au même numéro de port que le socket serveur, mais avec un descripteur différent.

Développement Système: Les sockets

Programmation en mode connecté : serveur TCP

Description en 4 étapes :

1. Créer un socket:

Internet (IPv4) TCP Protocole

serveurfd= socket(AF_INET,SOCK_STREAM,0);

serveurfd est le descripteur du socket créé avec la primitive ***socket***

2. Associer le socket à un port (local)

bind(serveurfd, &serveuraddr, sizeof(serveuraddr));

Adresse d'une structure sockaddr_in

Longueur de la structure serveuraddr

Développement Système: Les sockets

Programmation en mode connecté : serveur TCP

3. Indiquer que c'est un socket serveur

```
listen(serveurfd , 5 );
```

Taille maximum de la file d'attente des clients

Après un appel à ***listen***, le système d'exploitation de la machine serveur peut commencer à recevoir des demandes de connexion.

Si une demande arrive pendant qu'une autre est en cours de traitement, elle est placée dans une file d'attente.

Si une demande arrive alors que la file est pleine, elle est rejetée

Développement Système: Les sockets

Programmation en mode connecté : serveur TCP

4. Obtention d'un canal de communication par le serveur:

connexionfd = accept(serveurfd, &clientaddr, &addrlen);

Descripteur du socket de communication

Adresse d'une structure sockaddr_in du client dont la demande a été acceptée.

Taille de la structure

La primitive ***accept*** est bloquante

Développement Système: Les sockets

Programmation en mode connecté : serveur TCP

En résumé :

```
#include <sys/types.h>
#include <sys/socket.h>
/* créer un socket serveur, renvoyer le descripteur) */
int socket(int domain, int type, int protocol);
/* associer un socket à l'adresse d'une structure de description */
int bind(int sockfd, struct sockaddr *addr, int addrlen);
/* déclarer (et activer) un socket comme serveur avec taille maxqueuesize */
int listen(int sockfd, int maxqueuesize);
int accept(int sockfd, struct sockaddr *addr, int *addrlen);
/* permet à un processus de prendre connaissance des nouvelles connexions
de clients et d'obtenir un canal de communication pour chacune */
```

Développement Système: Les sockets

Programmation en mode connecté : client TCP

Description en 2 étapes :

Pour demander une connexion, un client est supposé connaître l'adresse d'un serveur et le numéro de port d'un socket serveur sur celui-ci (un processus serveur est en attente sur ce port).

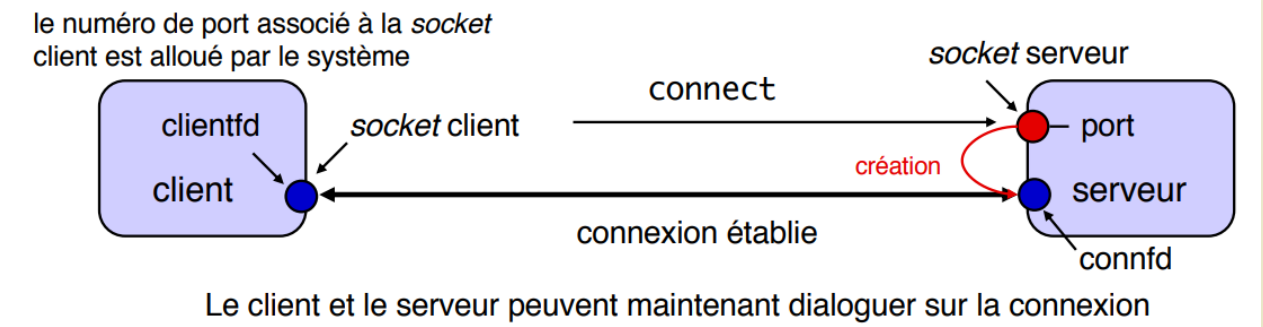
1. Céer un socket : *identique à la création d'un socket côté serveur*

clientfd = socket(AF_INET, SOCK_STREAM, 0);

2. Établir une connexion entre le socket client et le serveur:

connect(clientfd, &serveuraddr, sizeof(serveuraddr));

L'appel connect envoi une demande de connexion au socket serveur, retourne 0 en cas de succès et -1 sinon.



Développement Système: Les sockets

Programmation en mode connecté : client TCP

En résumé :

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
/* crée un socket client , renvoie descripteur) */
int connect(int sockfd, struct sockaddr *addr, int addrlen);
/* envoie une demande de connexion à un serveur */
/* serveur et numéro de port sont spécifiés dans sockaddr */
/* renvoie 0 si succès, -1 si échec

    struct sockaddr { /* structure générique pour sockets */
    unsigned short sa_family; /* famille de protocoles */
    char sa_data[14]; } /* données d'adressage */

    struct sockaddr_in {/* structure pour sockets Internet */
    unsigned short sin_family; /* AF_INET */
    unsigned short sin_port; /* numéro de port */
    struct in_addr sin_addr; /* adresse IPv4, ordre réseau */
    unsigned char sin_zero[8]; } /* remplissage pour sockaddr */
```

Développement Système: Les sockets

Programmation en mode connecté : échanges C/S

Emission et réception de données :

→ Si une connexion est établie

• Un canal de communication directe entre socket côté client et socket de service côté serveur est créé

• Il n'y a pas besoin de préciser l'adresse du destinataire à chaque envoi de données ni de vérifier l'émetteur à la réception

→Les fonctions de communication

• On peut utiliser les fonctions standards pour communiquer

write pour l'émission

read pour la réception

• Si on veut gérer quelques options, on peut utiliser des fonctions spécialisées;

send pour l'émission

recv pour la réception

Développement Système: Les sockets

Programmation en mode connecté : échanges C/S

Syntaxes émission des données:

ssize_t write (int desc_socket, void *ptr_mem, size_t longueur);

ssize_t send (int desc_socket, void *ptr_mem, size_t longueur, int option);

Avec :

- ***desc_socket :*** descripteur du socket
- ***ptr_mem :*** zone mémoire où sont stockées les données à envoyer
- ***longueur:*** nombre d'octets à envoyer
- ***option :*** 0 si envoi normal ou MSG_OOB si envoi de données prioritaires

Ces fonctions retournent le nombre d'octets écrits ou -1 en cas d'erreur.

Développement Système: Les sockets

Programmation en mode connecté : échanges C/S

Syntaxes réception des données:

ssize_t read(int descripteur, void *ptr_mem, size_t longueur);

ssize_t recv(int descripteur, void *ptr_mem, size_t longueur, int option);

Avec :

- ***descristeur :*** descripteur du socket
- ***ptr_mem :*** zone mémoire où seront écrites les données reçues
- ***longueur:*** taille de la zone mémoire
- ***option :***

0 réception normale

MSG_OOB réception de données prioritaires

MSG_PEEK réception des données reçues mais sans les retirer du tampon

Ces fonctions retournent le nombre d'octets reçus ou -1 en cas d'échec.

Développement Système: Les sockets

Programmation en mode connecté : fermeture de connexion

Fermeture de la connexion:

int close (desc_socket);

Le noyau essaie d'envoyer les données non encore émises avant de sortir de close().

Fermeture Partielle:

#include < sys/ioctl.h>

int shutdown (desc_socket, int sens);

Cet appel ferme partiellement le descripteur du socket

- **sens=0** : le socket n' accepte plus de lecture : read / recv, renvoie 0
- **sens=1** : le socket n'accepte plus d'envoi: write/send, provoque SIGPIPE
- **sens=2** : le socket n'accepte plus ni lecture, ni écriture/envoi

Développement Système: Les sockets

Programmation en mode connecté : Exemple

Client.c :

Le client créé un socket, se connecte au serveur, envoie un message et attend une réponse.

Serveur.c :

Le serveur créé un socket, s'attache, se met en attente d'une connexion, reçoit et affiche le message du client et ensuite envoie une réponse.

Exécution du serveur.c :

```
pi@raspberrypi:~/Documents/ExCoursSockets $ ./serveurTCP
Socket créé avec succès ! (3)
Socket attaché avec succès !
Socket placé en écoute passive ...
Attente d'une demande de connexion (quitter avec Ctrl-C)

Message reçu : Hello BUT2 !
 (13 octets)

Message Coucou de R3.05
 envoyé (16 octets)

Attente d'une demande de connexion (quitter avec Ctrl-C)
```

Exécution du client.c :

```
pi@raspberrypi:~/Documents/ExCoursSockets $ ./clientTCP
Socket créée avec succès ! (3)
Connexion au serveur réussie avec succès !
Message Hello BUT2 !
 envoyé avec succès (13 octets)

Message reçu du serveur : Coucou de R3.05
 (16 octets)
```

Développement Système: Les sockets

Programmation en mode non connecté: UDP

Le mode non connecté permet une communication entre plusieurs processus fonctionnant sur un réseau I.P avec le protocole UDP.

Points forts:

+ Rapide et léger en trafic.

+ Souple.

+ Possibilité d'envoyer un paquet à plusieurs destinataires (multicast).

Points Faibles

- Non fiable, possibilité de perdre des paquets sans le savoir.

- Ordre des paquets peut changer à l'arrivée

Développement Système: Les sockets

Programmation en mode non connecté:

Etapes d'une connexion client-serveur en UDP :

- Le serveur et le client ouvrent chacun un socket
- Le serveur l'attache à un de ses ports (un port précis)

Le client ne nomme pas son socket (il sera attaché automatiquement à un port lors de l'émission)

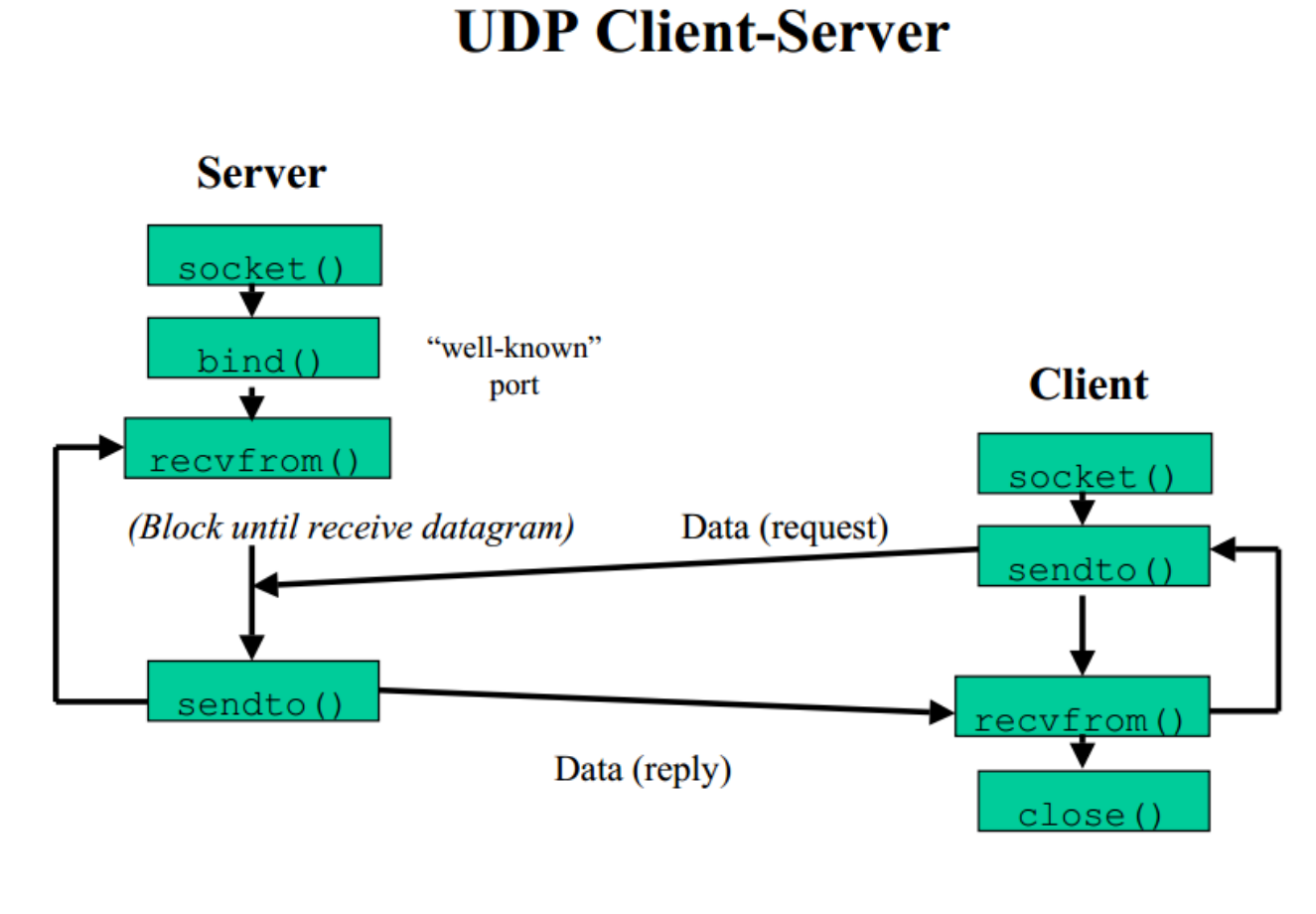
- Le client et le serveur dialoguent : sendto(…) et recvfrom(…)
- A la fin tous les sockets doivent être fermés

Les deux extrémités n'établissent pas une connexion :

- Elles ne mettent pas en œuvre un protocole de maintien de connexion
- Si le processus d'une extrémité meurt l'autre n'en sait rien !

Développement Système: Les sockets

Programmation en mode non connecté: schéma



Développement Système: Les sockets

Programmation en mode non connecté:

Exemple d'un client UDP:

```
int socket_id; /* socket ID */
int count; /* compteur d'octets recus */
struct sockaddr_in serveur_def; /* adresse serveur */
int serveur_lg; /* lg adresse serveur */
struct hostent *serveur_info; /* infos serveur */
char buffer[9000], *message;
socket_id = socket(AF_INET,SOCK_DGRAM,0); /* création socket */
message = …; /* preparation msg */
serveur_def.sin_family = AF_INET; /* envoi par sento(…)*/
serveur_info = gethostbyname(NAME_SERVER);/* (attachement auto)*/
memcpy(&serveur_def.sin_addr, serveur_info->h_addr, serveur_info->h_length);
serveur_def.sin_port = htons(PORT_SERVER);
serveur_lg = sizeof(struct sockaddr_in); /* - taille initiale */
sendto(socket_id,message,lg_message, /* - envoi requete */
0,&serveur_def,serveur_lg);
count = recvfrom(socket_id,buffer,sizeof(buffer), 0,&serveur_def,&serveur_lg);
 /* attente de réponse */
… /* traitement */
close(socket_id);
```

Développement Système: Les sockets

Programmation en mode non connecté:

Exemple d'un serveur UDP:

```
/* declaration de variables */
int socket_id;
struct sockaddr_in serveur_def;
/* création de la socket */
socket_id = socket(AF_INET,SOCK_DGRAM,0);
/* attachement de la socket */
serveur_def.sin_family = AF_INET;
serveur_def.sin_addr.s_addr = htonl(INADDR_ANY);
serveur_def.sin_port = htons(PORT_SERVEUR);
bind(socket_id, &serveur_def, sizeof(struct sockaddr_in));
/* traitement des requetes */
dialog(socket_id);
/* fermeture socket */
close(socket_id);
```

Développement Système: Les sockets

Programmation en mode non connecté:

Exemple d'un serveur UDP: (fonction **dialog**)

```
void dialog(int socket_id){
int count; /* compteur d'octets recus */
struct sockaddr_in client_def;
int client_lg;
char buffer[9000], *reponse;
int lg_reponse;
int fin = FALSE;
/* taille initiale de la definition (@) du client */
client_lg = sizeof(struct sockaddr_in);
/* boucle de traitement des requetes */
while (!fin) {
/* reception requete (bloquante) */
count = recvfrom(socket_id, buffer, sizeof(buffer),0, &client_def, &client_lg);
/* traitement requete */
reponse = …
fin = …
/* emission reponse */
sendto(socket_id, reponse, lg_reponse,0, &client_def, client_lg);}
}
```

Développement Système: Les sockets

Programmation en mode non connecté:

Exécution du serveur UDP:

```
pi@raspberrypi:~/Documents/ExCoursSockets $ ./serveurUDP
Socket créé avec succès ! (3)
Socket attaché avec succès !
Message Hello world !
 reçu avec succès (14 octets)
```

Exécution du client UDP:

```
pi@raspberrypi:~/Documents/ExCoursSockets $ ./clientUDP
Socket créé avec succès ! (3)
Message Hello world !
 envoyé avec succès (14 octets)

pi@raspberrypi:~/Documents/ExCoursSockets $
```