

Client/Server Socket

Qu'est-ce qu'un serveur ?



- un « logiciel serveur » **offre un service** sur le réseau,
- le « serveur » est la machine sur laquelle s'exécute le logiciel serveur,
- le serveur doit être sur un site avec **accès permanent** et s'exécuter en permanence. Un site peut offrir plusieurs services.



*Le serveur **accepte des requêtes**, les traite et **envoie le résultat** au demandeur.*

- Un service est fourni sur un **Port de communication identifié par un numéro**.
- Certains numéros de Port (internationalement définis) identifient le service quelque soit le site
ex :
 - le service FTP est offert sur les ports numéros 21 (contrôle) et 20 (données),
 - le service TELNET (émulation terminal) sur le port 23,
 - le service SMTP (mail) sur le port 25,
 - etc.
- Pour **accéder à un service**, il faut l'**adresse du site** et le **numéro du port**.

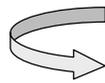


Qu'est-ce qu'un client ?

Qu'est-ce qu'un client ?



- un « logiciel client » **utilise le service** offert par un serveur,
- le « client » est la machine sur laquelle s'exécute le logiciel client,
- Le client est raccordé par une **liaison temporaire**.



*Le client envoie des requêtes
et reçoit des réponses.*



client



serveur

Architecture client/serveur ?



description du **fonctionnement coopératif** entre le serveur et le client.

Un serveur peut être utilisé par plusieurs clients,
ceux-ci pouvant être développés sur différents systèmes : Unix, Mac ou PC par ex.



Mais **obligation de respecter le protocole** entre les deux processus communicants.

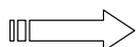


Pile de protocoles TCP/IP très souvent utilisée (Internet par ex.)



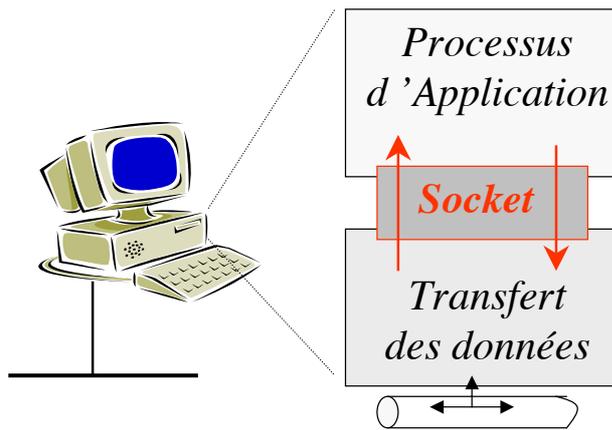
L'Application Program Interface (API) utilisée est l'**API Sockets**.

(dans notre cas) API = un ensemble de primitives pour l'accès à la pile de protocoles TCP/IP.



Notion de Sockets (et principales primitives)

Notion de Sockets (prises de raccordement)



Ce n'est **ni** une norme de communication
ni une couche de protocole

C'est une **interface** entre
le programme d'application et
les protocoles de communication

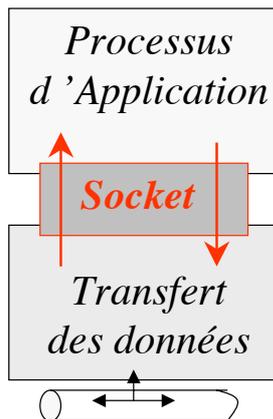


mécanisme de communication bidirectionnel interprocessus
(dans un environnement distribué)

Socket

{
API (ensemble de primitives de programmation)
et
Extrémités de la communication {Add IP, numéro Port}

Principe de programmation



Un socket s'utilise comme un fichier :

1. Création/Définition/Ouverture
2. Communication
3. Fermeture/Libération.

• Création/Définition/Ouverture

- réservation des ressources pour l'interface de communication
- création d'un descripteur du socket (similaire descripteur de fichier)



seulement une extrémité de connexion

• Communication

- utilisation des primitives *read* (réception), *write* (émission) ... (idem fichier)
- plusieurs variantes de primitive d'émission et de réception

• Fermeture/Libération

- utilisation de la primitive *close* (fermeture de la communication)

Algorithmes

Programme serveur pour un seul client

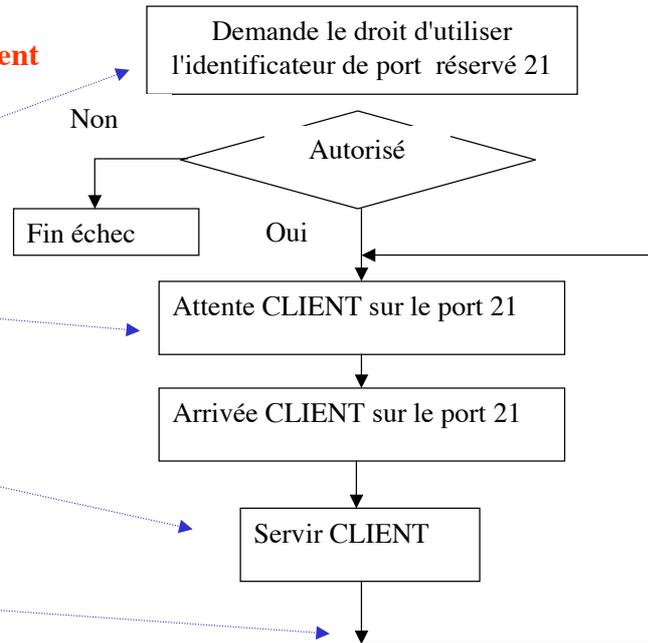
(illustré pour FTP)

Initialisation du serveur

Attente connexion

Service rendu au client

Fin connexion
(pour ce client)



Algorithmes

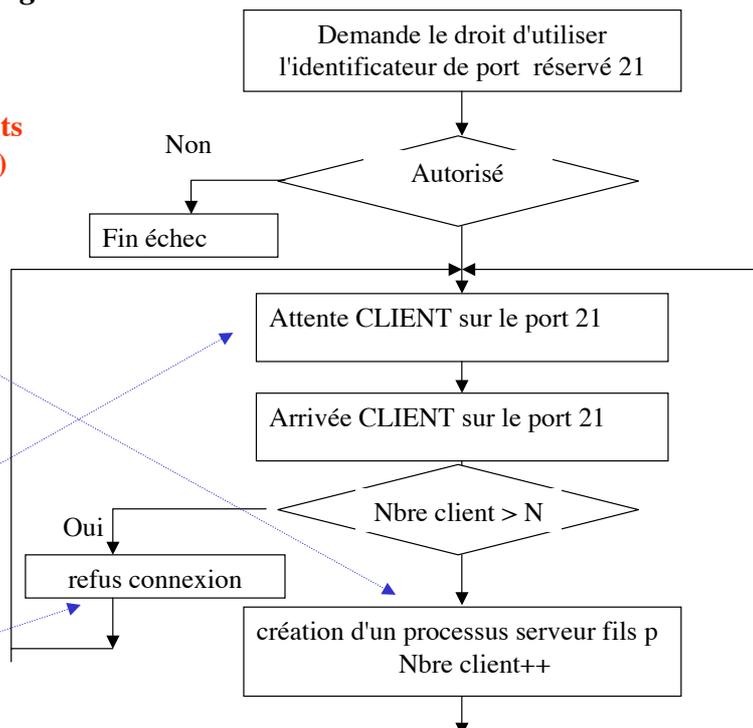
Concept de serveur multi-clients (programmation concurrente)

(illustré pour FTP)

Connexion d'un client C_i , le serveur génère un processus fils (duplication du prog. serveur) pour répondre à la demande du client C_i .

Ainsi, le programme serveur peut-il attendre un autre client.

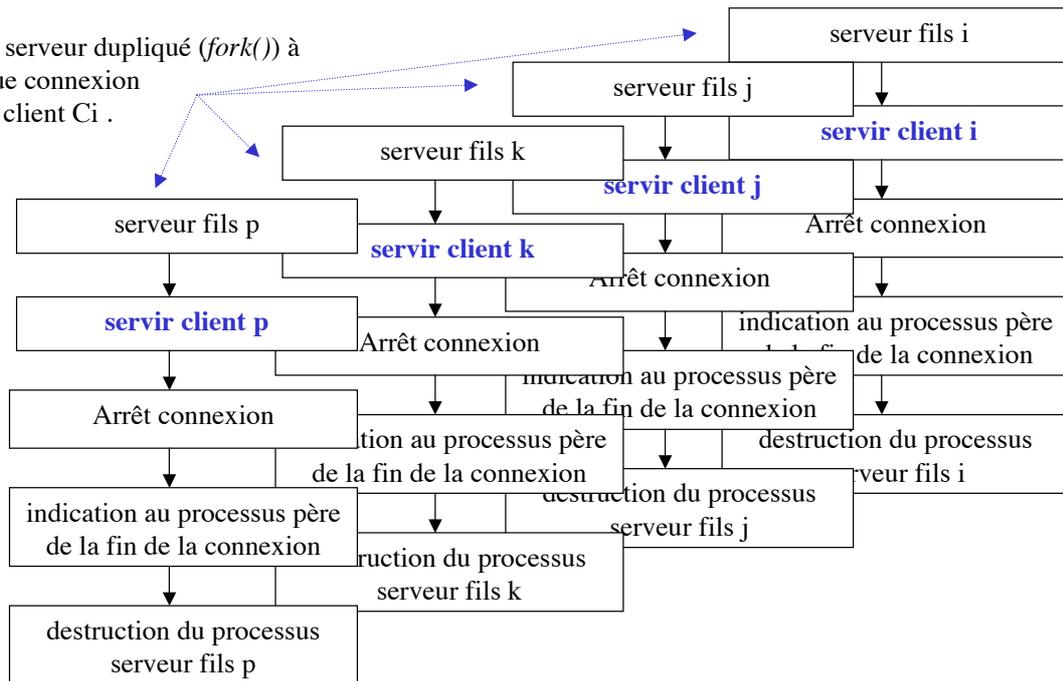
connexion refusée pour cause de saturation du serveur (nombre max de clients atteint)



**Concept de serveur multi-clients
(programmation concurrente)**

Algorithmes

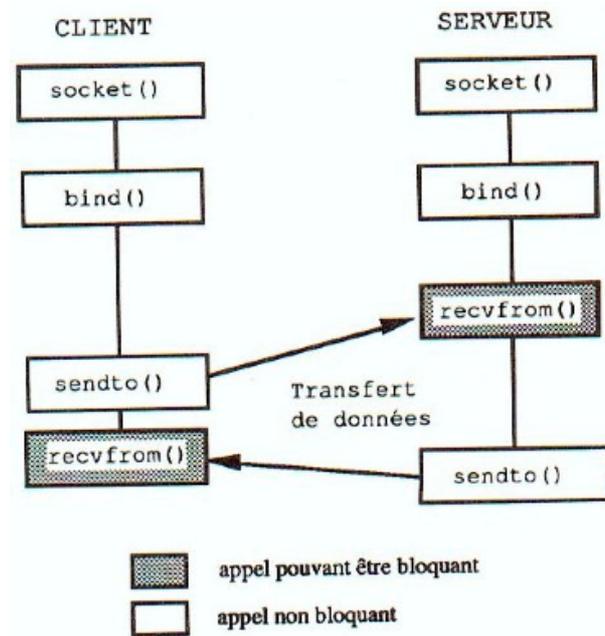
Prog. serveur dupliqué (*fork()*) à chaque connexion d'un client C_i .



Introduction aux Sockets

- IPC : Inter Process Communication
 - UNIX
- Bibliothèque
 - `#include <sys/socket.h>`
- La communication par socket utilise un **descripteur** pour désigner la connexion sur laquelle on **envoie/reçoit** les données

Communication locales

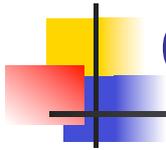


11

Communication locales

- Dans le domaine UNIX
- Communication locale à une machine
- Les identificateurs sont des noms de fichiers
- Utilise une modalité de communication non connecté
 - Le destinataire reçoit le message petit à petit
 - Taille du message indéterminée

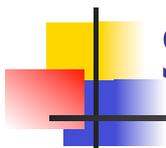
12



Communication locales

- L'ouverture d'un socket se fait en 2 étapes :
 - Création d'un socket et de son descripteur
 - Fonction `socket ()`
 - Spécifier le type de communication
 - Fonction `bind ()`
 - Un server écoute
 - Un client envoie

13



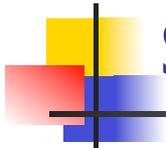
Socket data structure

- Structure générique

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

```
struct sockaddr {  
    ushort sa_family; /* famille d'adresse AF_... */  
    char sa_data[14]; /* données */  
}
```

14



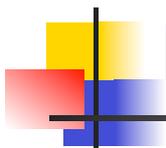
Socket data structure

- Famille AF_UNIX
 - Plus grand que la structure générique
 - Il faut spécifier la taille

```
#include <sys/un.h>

struct sockaddr_un {
    ushort sun_family; /* AF_UNIX */
    char sun_path[108]; /* chemin + nom fichier */
}
```

15

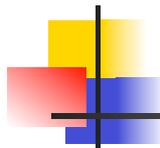


Fonction socket ()

- `int socket (int af, int type, int protocol)`
 - af: AF_UNIX, AF_INET
 - Type: SOCK_STREAM, SOCK_DGRAM, SOCK_STREAM
 - Protocol: 0 (car af et type suffit)
- Retourne le descripteur
- Aucun transfert pour le moment

type	AF_UNIX	AF_INET
SOCK_STREAM	oui, protocole 0	oui, protocole 0 ou IPPROTO_TCP
SOCK_DGRAM	oui, protocole 0	oui, protocole 0 ou IPPROTO_UDP
SOCK_RAW	non	oui, deux possibilités : IPPROTO_ICMP : accès ICMP IPPROTO_RAW : accès IP

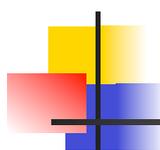
16



Fonction bind ()

- Assignation d'une adresse/numéro de port locaux à un socket
- `int bind(int sockfd, struct sockaddr *mon_adr, int lg_adr)`

17



Fonctions sendto () recvfrom ()

- Transmission/reception des données

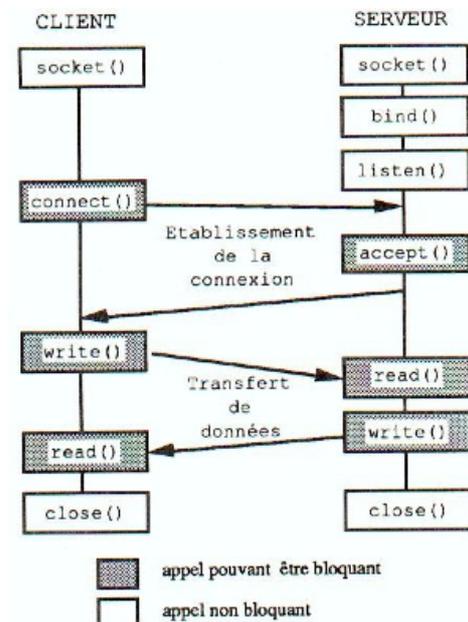
```
int sendto (int sockfd, char *buf, int noctets, int flags,  
           struct sockaddr *to, int lg_adr);
```

```
int recvfrom (int sockfd, char *buf, int noctets, int flags,  
            struct sockaddr *to, int *lg_adr);
```

- `Flags = 0`

18

Communication en mode connecté



19

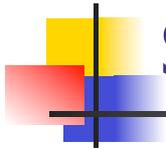
Socket data structure

■ Structure générique

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

```
struct sockaddr {  
    ushort sa_family; /* famille d'adresse AF_... */  
    char sa_data[14]; /* données */  
}
```

20



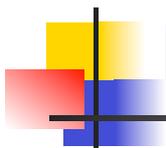
Socket data structure

- Famille AF_INET
 - IPv4 AF_INET sockets:

```
struct sockaddr_in {
    short  sin_family;
    unsigned short sin_port; // 2 bytes
    struct in_addr  sin_addr;
    char sin_zero[8];
};

struct in_addr {
    unsigned long s_addr; // 4 bytes
};
```

21

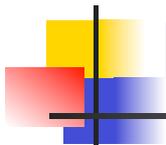


Fonction socket ()

- `int socket (int af, int type, int protocol)`
 - af: AF_UNIX, AF_INET
 - Type: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
 - Protocol: 0 (car af et type suffit)
- Retourne le descripteur
- Aucun transfert pour le moment

type	AF_UNIX	AF_INET
SOCK_STREAM	oui, protocole 0	oui, protocole 0 ou IPPROTO_TCP
SOCK_DGRAM	oui, protocole 0	oui, protocole 0 ou IPPROTO_UDP
SOCK_RAW	non	oui, deux possibilités : IPPROTO_ICMP : accès ICMP IPPROTO_RAW : accès IP

22



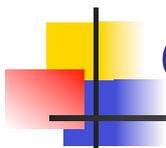
Fonction bind ()

- Assignation d'une adresse/numéro de port locaux à un socket

```
int bind( int sockfd,
          struct sockaddr *mon_adr,
          int lg_adr)
```

- Serveur: enregistre son adresse publique
 - IP + port
- Client: pas besoin en général

23

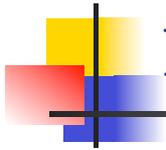


exemple

```
sockaddr_in localaddr ;
localaddr.sin_family = AF_INET; /* Protocole internet */
/* Toutes les adresses IP de la station */
localaddr.sin_addr.s_addr = htonl(INADDR_ANY);
/* port d'écoute par défaut au dessus des ports réservés */
localaddr.sin_port = htons(port);

if (bind(listen_socket, (struct sockaddr*)&localaddr,
        sizeof(localaddr) ) == SOCKET_ERROR) {
// Traitement de l'erreur;
}
```

24



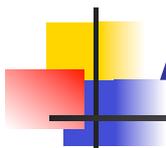
htonl (); htons ();

```
unsigned long int htonl (unsigned long int hostlong);
```

```
unsigned short int htons (unsigned short int hostshort);
```

- La fonction **htonl()** convertit un **entier long hostlong** depuis l'ordre des octets de l'hôte vers celui du réseau.
- La fonction **htons()** convertit un **entier court (short) hostshort** depuis l'ordre des octets de l'hôte vers celui du réseau.

25

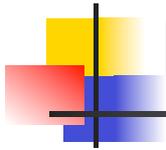


Adresse IP

- **INADDR_ANY** le socket est associé à n'importe quelle adresse IP de la machine
- La fonction `inet_addr ()` spécifie une adresse IP

```
inet_addr("127.0.0.1");  
/* utilisation de l'adresse de boucle locale */
```

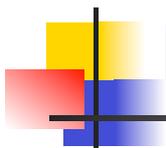
26



Port

- Le socket peut être reliée à un port libre quelconque en utilisant le numéro 0

27

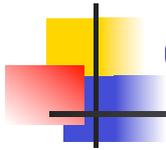


Port

- Vérifier la port utilise par le serveur:

```
getsockname (hServerSocket, (struct sockaddr *) &Address,  
                                                     (socklen_t *)&nAddressSize);  
printf("opened socket as fd (%d) on port (%d) for stream i/o\n",hServerSocket,  
                                             ntohs(Address.sin_port) );  
printf("Server\n\  
      sin_family      = %d\n\  
      sin_addr.s_addr = %d\n\  
      sin_port        = %d\n\  
      , Address.sin_family  
      , Address.sin_addr.s_addr  
      , ntohs(Address.sin_port)  
);
```

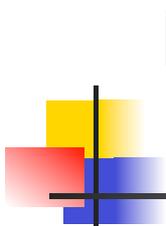
28



Connect ()

- `int connect(int socket, const struct sockaddr *address, socklen_t address_len);`
- Débuter une connexion sur une socket

31



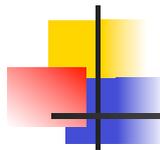
Fonctions

read () write ()

- Réception des données
- `int read(int sockfd, char *buf, int count);`
 - read lit jusqu'à count octets depuis le descripteur de socket dans le buffer pointé par buf.
 - read renvoie -1 s'il échoue, sinon read renvoie le nombre d'octets lus

32

Fonctions



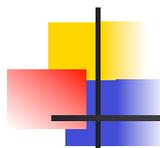
read () write ()

- Transmission des données

- `int write(int sockfd, char *buf, int count);`

- write écrit jusqu'à count octets dans le fichier associé au descripteur sockfd depuis le buffer pointé par buf
- write renvoie le nombre d'octets écrits (0 signifiant aucune écriture), ou -1 s'il échoue.

33



Close ()

- `int close(int sockfd);`

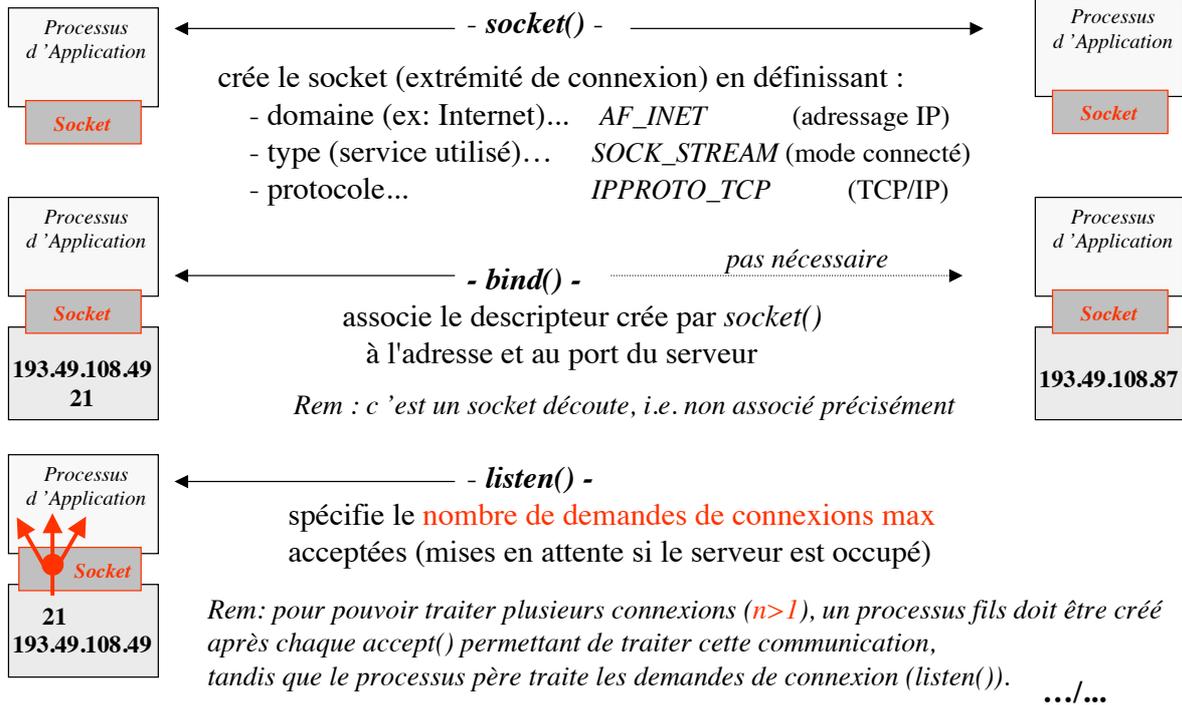
- close ferme le descripteur sockfd,
- close renvoie 0 s'il réussit, ou -1 en cas d'échec

34

Primitives de programmation (API) Berkeley Sockets (UNIX)

SERVEUR

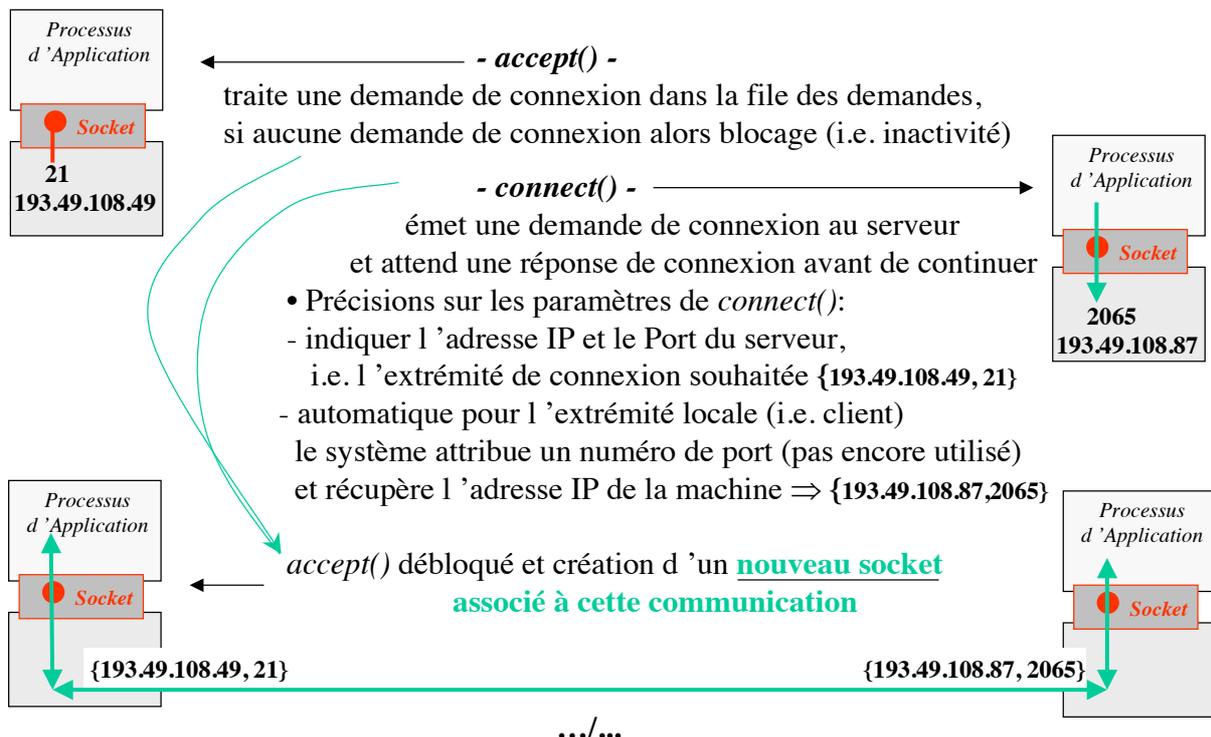
CLIENT

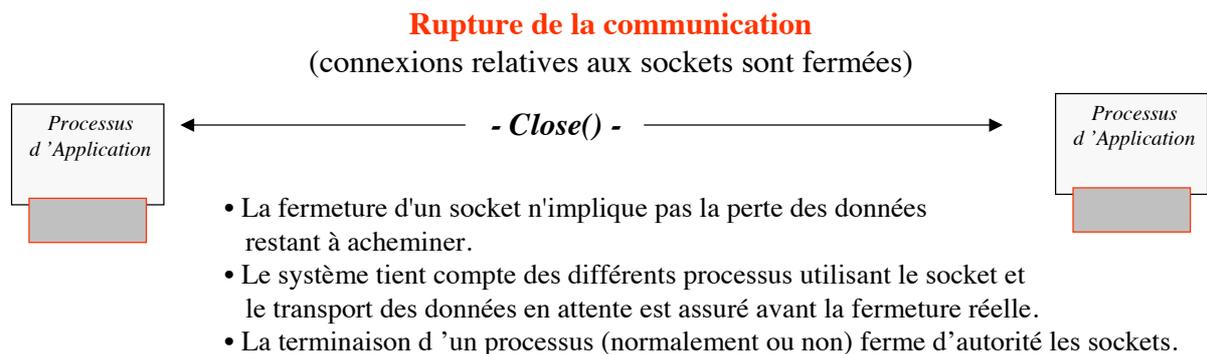
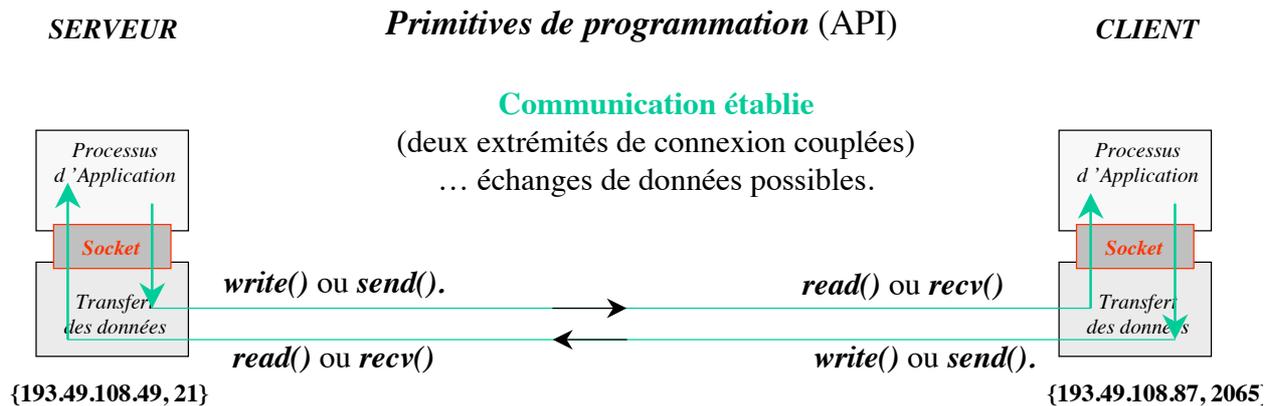


Primitives de programmation (API)

SERVEUR

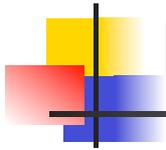
CLIENT





Debug

- Commande `netstat`
 - Affiche toutes les sockets réseaux ouvertes
 - Permet de vérifier les connexion ouvertes



Debug

- Commande telnet
 - telnet <server> <port>
 - Ouvre une connexion TCP vers le <server><port>
 - Permet de recevoir/transmettre chaîne de caractère
 - Fin chaîne de caractère avec 1310