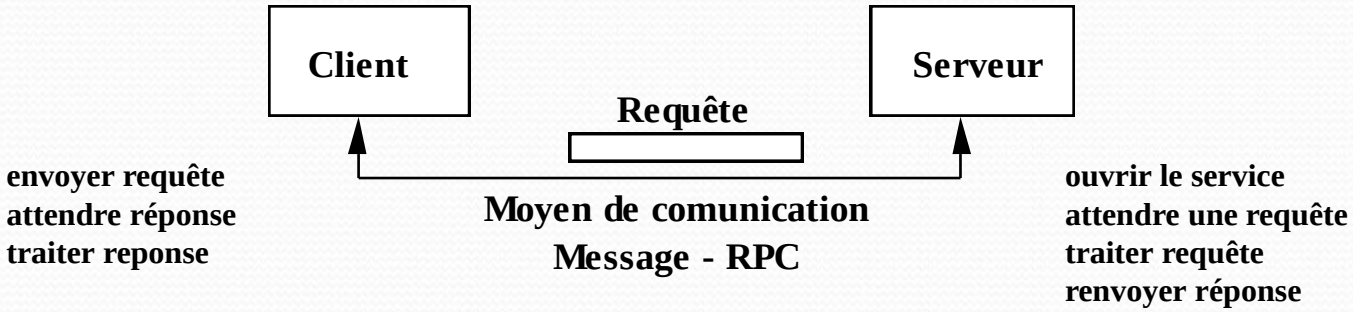
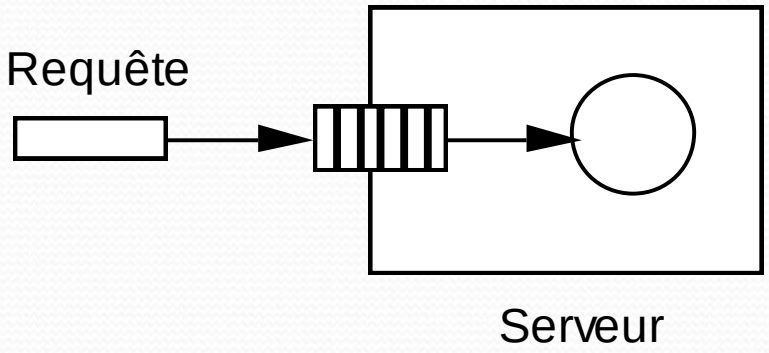
Programmation clients serveurs distants

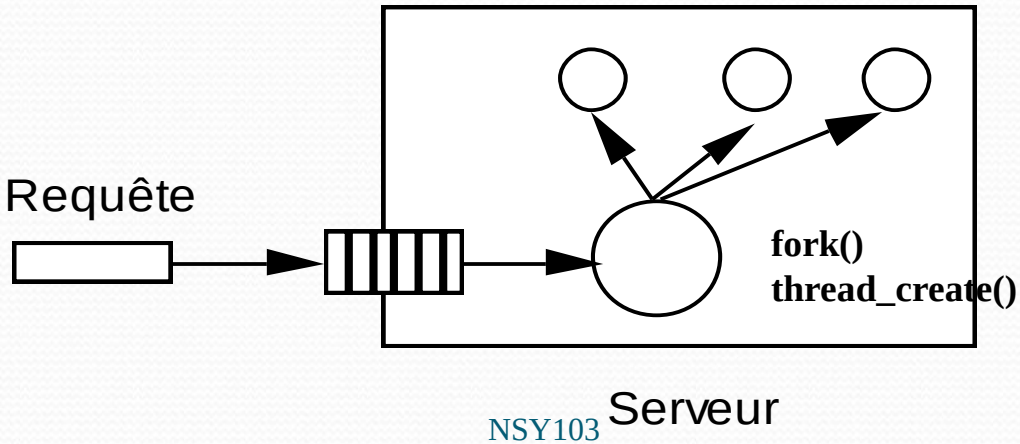
- Modèle clients serveurs
- Interconnexion de réseaux
- Programmation socket

Le modèle client-serveur



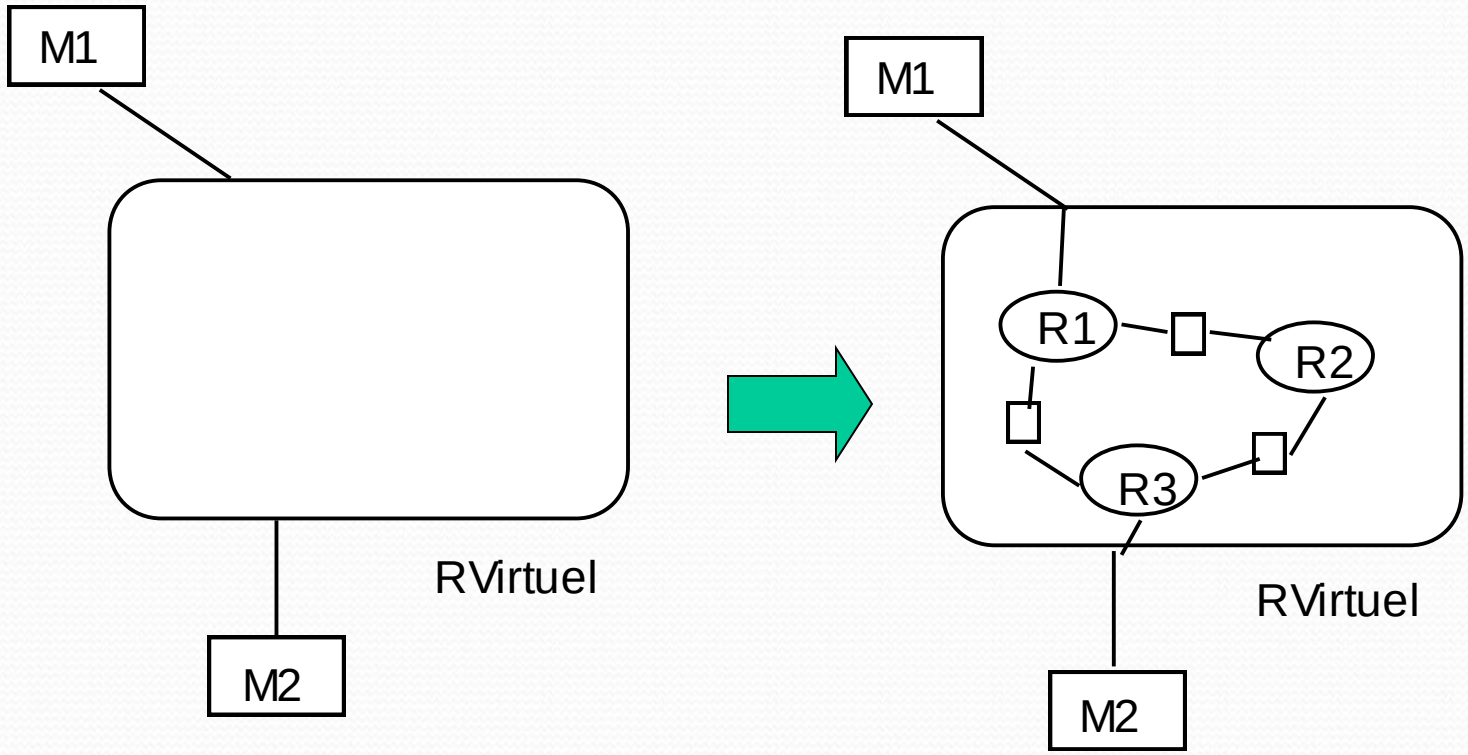


Serveur Itératif
Un seul processus effectue la réception, le traitement et l'émission



Serveur Parallèle
Le processus père effectue la réception. Il crée un fils pour réaliser le traitement et l'émission de la réponse.

L'interconnexion de réseau : quelques principes

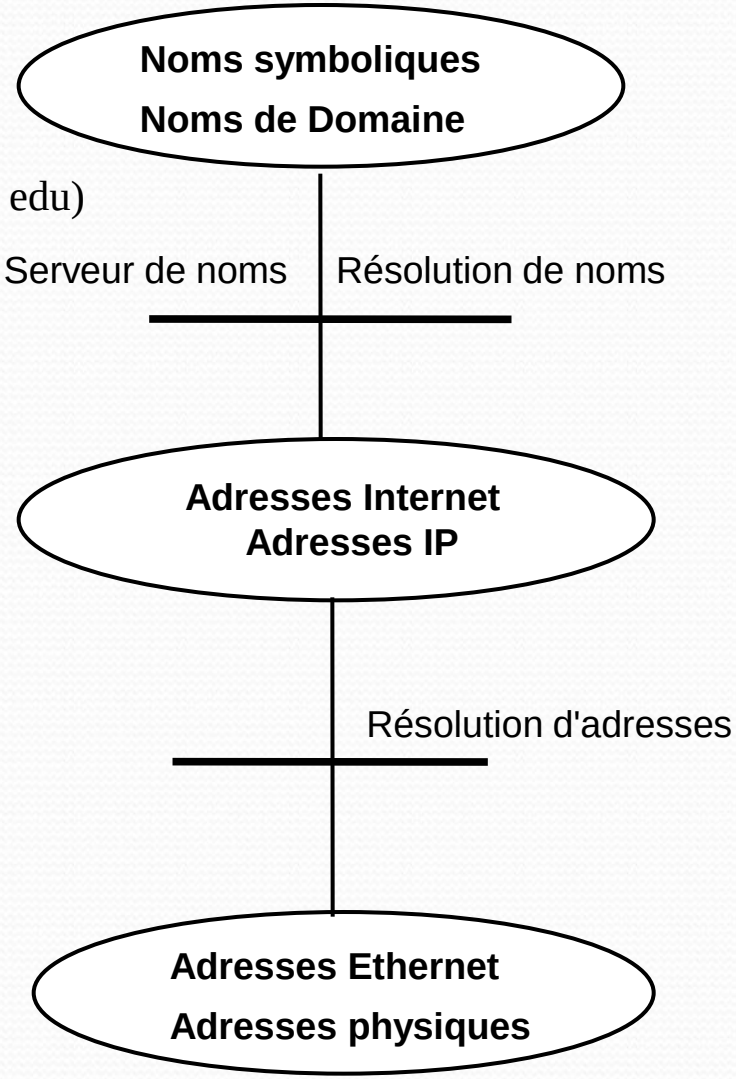


- Construire un réseau virtuel
  - adressage (adresse IP, noms symboliques)
  - transport de bout en bout (protocole TCP/IP)

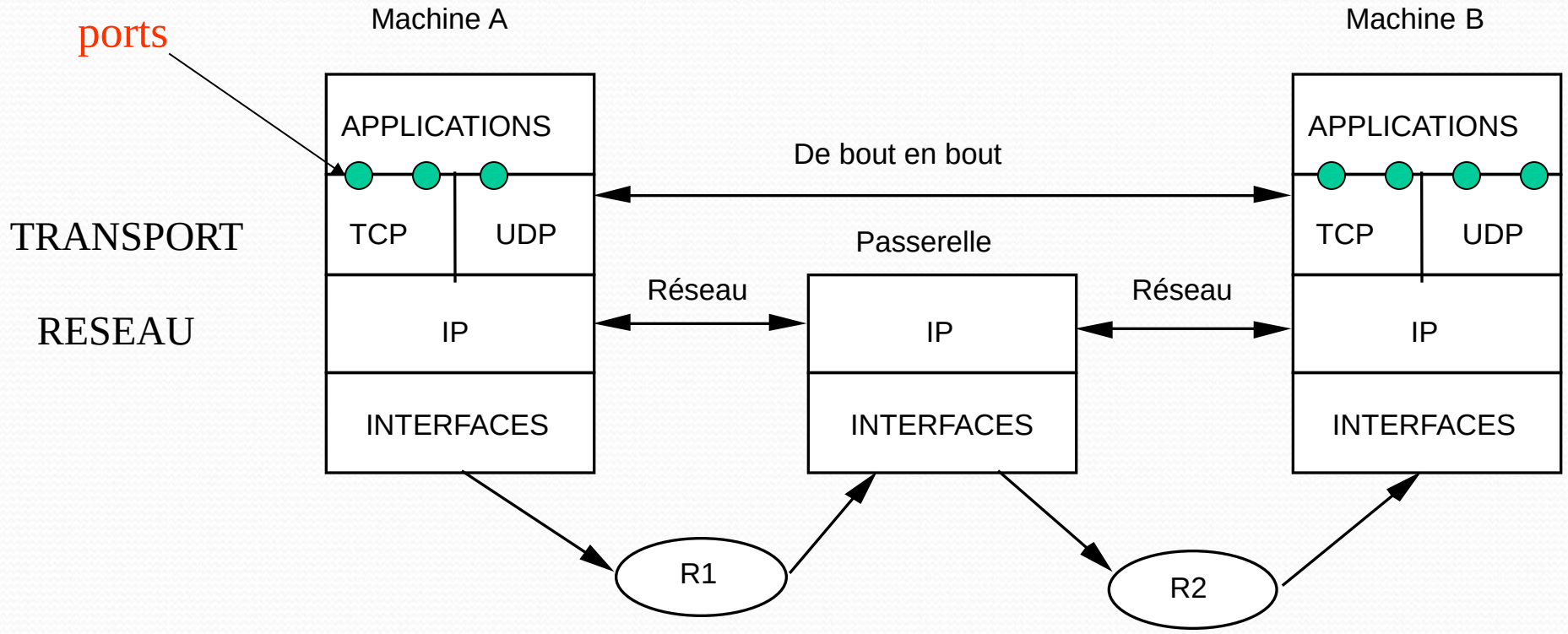
L'interconnexion de réseau : quelques principes

Structure hiérarchique fondée sur le domaine (DNS)
objet.sous-domaine.domaine fermi.cnam.fr
objet : nom d'une machine.
domaine : géographique (fr, jp) ou institutionnel (com, mil, edu)

Adresses IP : 32 bits
Paire (adresse réseau, adresse machine dans le réseau)
Forme pointée : ***10000000 00001010 00000010 00011110***
128.10.2.30



L'interconnexion de réseau : quelques principes



- IP : adressage de machine à machine via adresse IP
- TCP/UDP : adressage d'applications à applications
  - ➢ notion de ports (entier 16 bits)

L'interconnexion de réseau : identification réseau

- Chaque machine a une identification unique sur un réseau (e.g., internet)
  - Adresse IP : série de 4 nombres entre 0 et 255
    - 163.215.82.55 (notation pointée)
  - Plusieurs adresses par machine (fonction destination)
    - Adresse interne (localhost ou boucle locale) : 127.0.0.1
    - Adresse Internet (adresse du routeur vers extérieur) : 85.79.27.105
  - Adresses découpées en deux parties
    - Adresse du réseau et adresse de la machine

L'interconnexion de réseau : identification réseau

- Adresses classiques des machines utilisent IPv4
  - Adresse IP sur 32 bits : 4 nombres entre 0 et 255
    - 163.215.82.55 (notation pointée)
  - Milliards d'adresses possibles
- Problème : le nombre d'adresses avec IPv4 n'est plus suffisant !
  - Nécessité d'avoir une adresse par personne
- Migration de plus en plus vers IPv6
  - Adresse IP sur 128 bits : caractères en hexa, paire octets séparés par « : »
    - 2005:0db8:c9d2:aee5:73e3:924a:a5ae:9238
  - *Nb adresses supérieur au nb d'étoiles de l'univers*

L'interconnexion de réseau : identification réseau

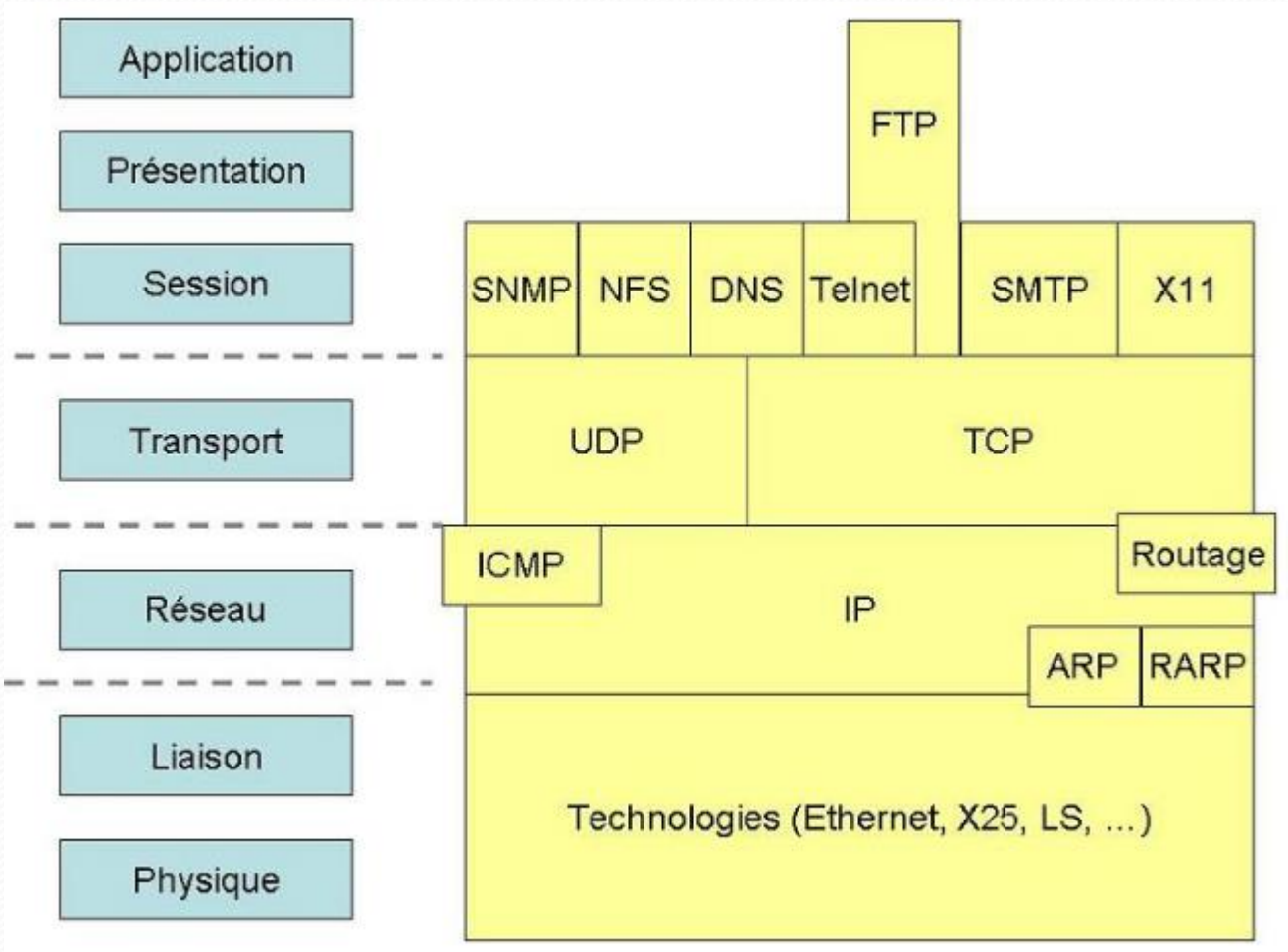
- Plusieurs processus/services destinataires sur même machine
  - Données simultanément pour navigateur / client mail / jeux …
- Besoin d'indiquer le processus destinataire des informations
  - Utilisation **numéro de port** sur 16 bits (entier 1 à 65 536)
    - Ex : 21 pour FTP, 80 pour HTTP …
  - Protocoles/services fréquemment utilisés ont numéro de port dédié (indiqué dans */etc/services* sous Linux)
  - Utiliser un numéro de port libre (***supérieur à 1024***)

L'interconnexion de réseau : protocole réseau

- Protocole de communication
  - Ensemble de règles communes à 2 machines leur permettant d'échanger de l'information
- Certains nombre de protocoles existant
  - Haut niveau : manipulation spécifiée des données, documentation disponible (FTP, SMTP …)
  - Bas niveau : manipulation des données à définir (octet par octet), difficile mais grande liberté (TCP/IP, UDP/IP)
- **Tous protocoles haut niveau utilisent soit TCP/IP ou UDP/IP**

L'interconnexion de réseau : protocole réseau

Modèle Internet



Couche Application

Couche Hôte à Hôte

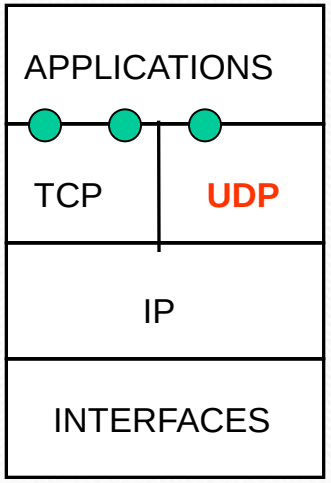
Couche Internet

Couche accès réseau

L'interconnexion de réseau : protocole réseau

- Echanges d'informations réalisés par morceaux
  - Niveau TCP/UDP : Envoie/réception de paquets de données
  - Niveau physique : paquets découpés en trames
  - Taille dépend de la couche MAC utilisée
- Deux méthodes d'échanges d'informations
  - TCP (Transmission Control Protocol) :
    - Etablie une connexion entre les 2 machines (**mode connecté**)
    - Système de contrôle des paquets, renvoie des paquets perdus
    - Echange plus coûteux et lent -> ajout d'informations de contrôle
  - UDP (User Datagram Protocol) :
    - Echange de données sans connexion (**mode non-connecté**)
    - Aucun contrôle des paquets (possibilité de perte ou désordonné)
    - Transmission des informations plus rapide (peu d'infos de contrôle)

L'interconnexion de réseau : protocole réseau



La couche UDP (User datagram Protocol)

- Protocole transport de bout en bout

adressage d'application à application via les ports UDP
Ports UDP réservés
exemple : port 513 application who

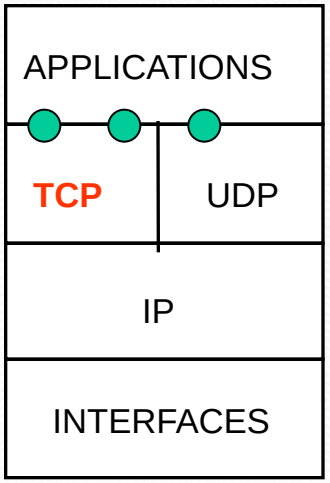
- Protocole transport non fiable basé sur IP

Datagrammes indépendants
Perte de datagrammes, datagrammes dupliqués, pas de remise dans l'ordre d'émission

Analogie : le courrier

L'interconnexion de réseau : protocole réseau

La couche TCP (Transport Control Protocol)



- Protocole transport de bout en bout

  adressage d'application à application via les ports TCP
  Ports TCP réservés
  exemples : port 21 application ftp

- Protocole transport fiable orienté connexion basé sur IP

  Connexion / Flux d'octets
  Pas de perte de messages, pas de duplication, remise dans l'ordre d'émission

  Analogie : le téléphone

L'interconnexion de réseau : protocole réseau

- Quand utiliser TCP ?
  - Si perte d'informations gênante pour application
  - Contrôle des paquets
  - Ex : transfert de fichiers, de texte, …
- Quand utiliser UDP ?
  - Si perte d'informations non gênante et besoin de communications rapides
  - Aucun contrôle des paquets
    - Sinon utiliser TCP ou à la charge de l'application
  - Ex : jeux en ligne, contenu en streaming …

L'interconnexion de réseau : protocole réseau

Protocole TCP / UDP

UDP

Echange de datagrammes

TCP

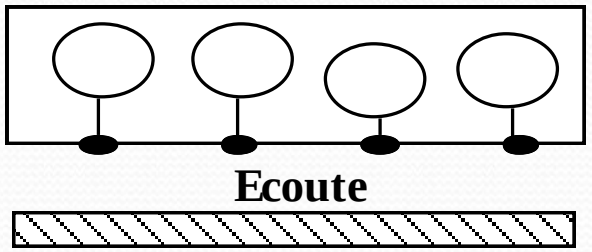
Etablissement connexion

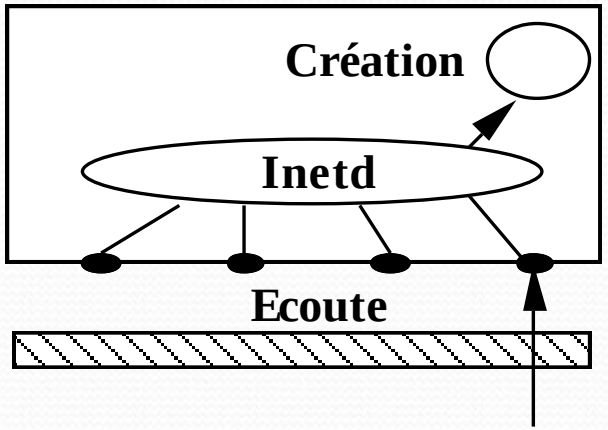
Echange de flux d'octets

Fermeture connexion

L'interconnexion de réseau : Inetd

La couche Application



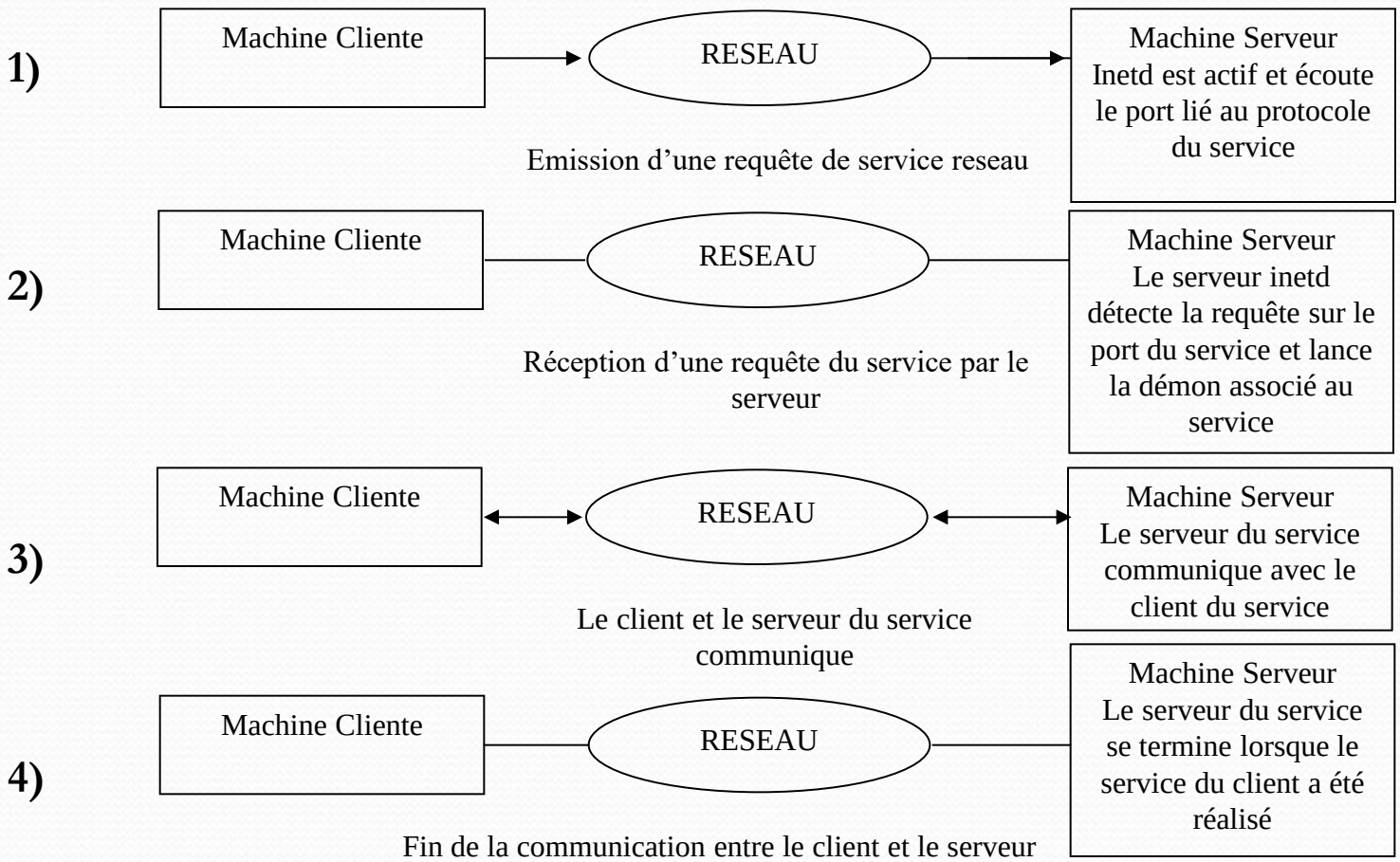


Un démon par service
Encombrement de la table des processus

Le démon inetd
en écoute sur les différents ports
crée le serveur sollicité

L'interconnexion de réseau : Inetd

Exemple :

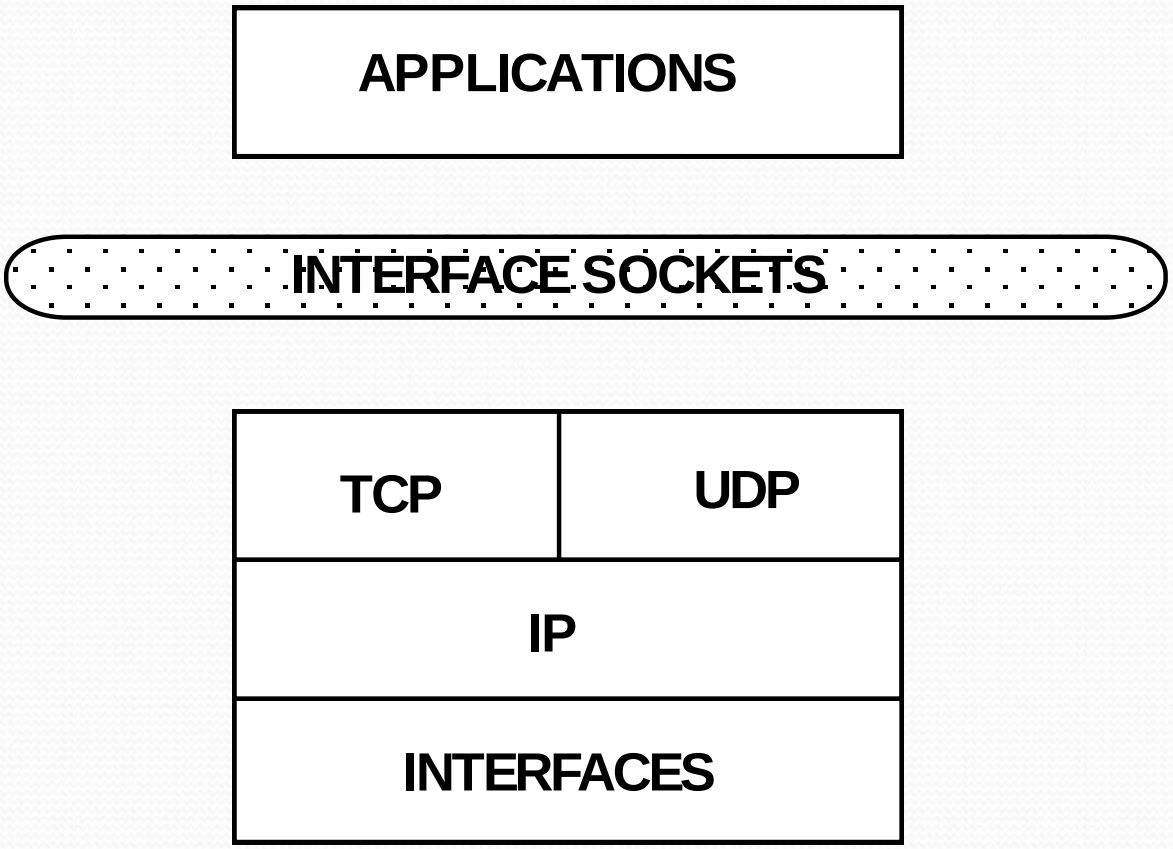


Pendant l'ensemble de ces étapes, le processus inetd est toujours actif.

Programmation clients serveurs distants

- Interface de programmation socket

L'interface socket



- Interface de programmation (ensemble de primitives)
- Point de communication (adresse d'application)
- Compatible SGF

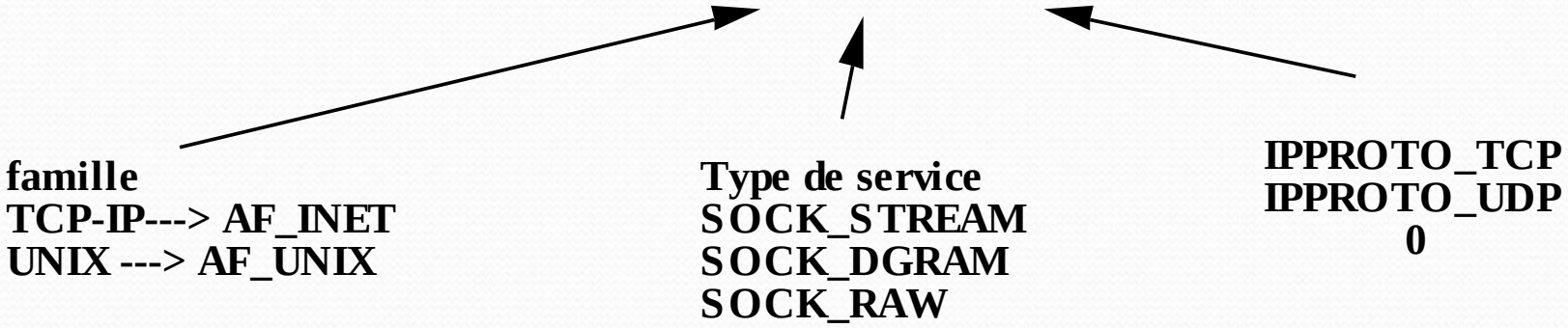
L'interface socket

Types de socket

- Plusieurs types de socket
- 3 trois types principaux
  - Stream sockets : communication connectée
    - Utilise le protocole TCP, dénommé SOCK_STREAM (http://tools.ietf.org/html/rfc793)
  - Datagram sockets : communication non-connectée
    - Utilise le protocole UDP, dénommé SOCK_DGRAM (http://tools.ietf.org/html/rfc791)
  - Raw sockets :
    - Permet d'accéder directement à la couche IP (droit super-utilisateur), dénommé SOCK_RAW
- Les deux premiers sont les plus utilisés

L'interface socket : création d'une socket

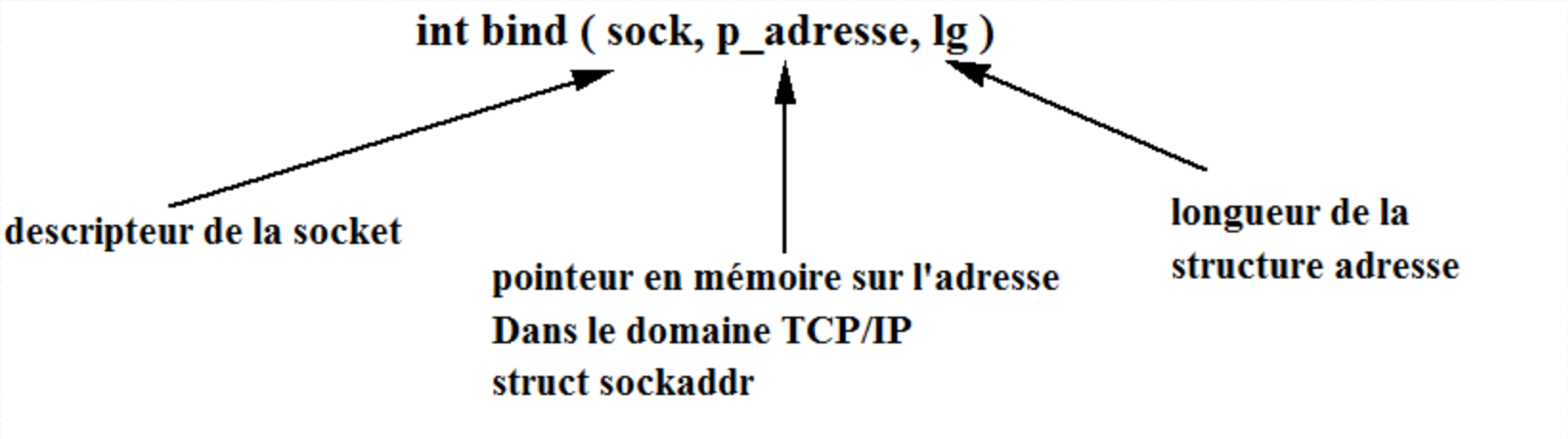
Sock = socket (af, type, protocole)



int socket (af, type, protocole)
int af;
int type;
int protocole;

Retourne un descripteur de socket ayant les mêmes propriétés qu'un descripteur de fichier (héritage)
Accessible par le créateur et les fils de celui-ci

L'interface socket : Attachement d'une adresse d'application



int bind (sock, p_adresse, lg)
int sock;
struct sockaddr *p_adresse; -- adresse d'application
int lg;

L'opération d'attachement permet d'étendre le groupe des processus pouvant accéder à la socket

L'interface socket : Attachement d'une adresse d'application

```
int bind (sock, p_adresse, lg)
int sock;
struct sockaddr *p_adresse;  -- adresse d'application
int lg;


struct sockaddr  {
                        short sin_family; -- AF-INET
                        char sa_data[14]; --adresse Ip et num port
                        }
```

La fonction getaddrinfo (nom de machine, numero de port, struct addrinfo hints) permet de récupérer une liste d'adresse sockaddr pour un service donné qualifié par un nom de machine et un numéro de port, en spécifiant des critères dans la structure addrinfo hints (protocole, type de service, famille…).

L'interface socket : Attachement d'une adresse d'application

- Initialisation de la structure ***addrinfo*** :

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

int getaddrinfo(const char *node,          // e.g. "www.example.com" or IP
                const char *service,     // e.g. "http" or port number
                const struct addrinfo *hints, // additional infos on socket type
                struct addrinfo **res);
```

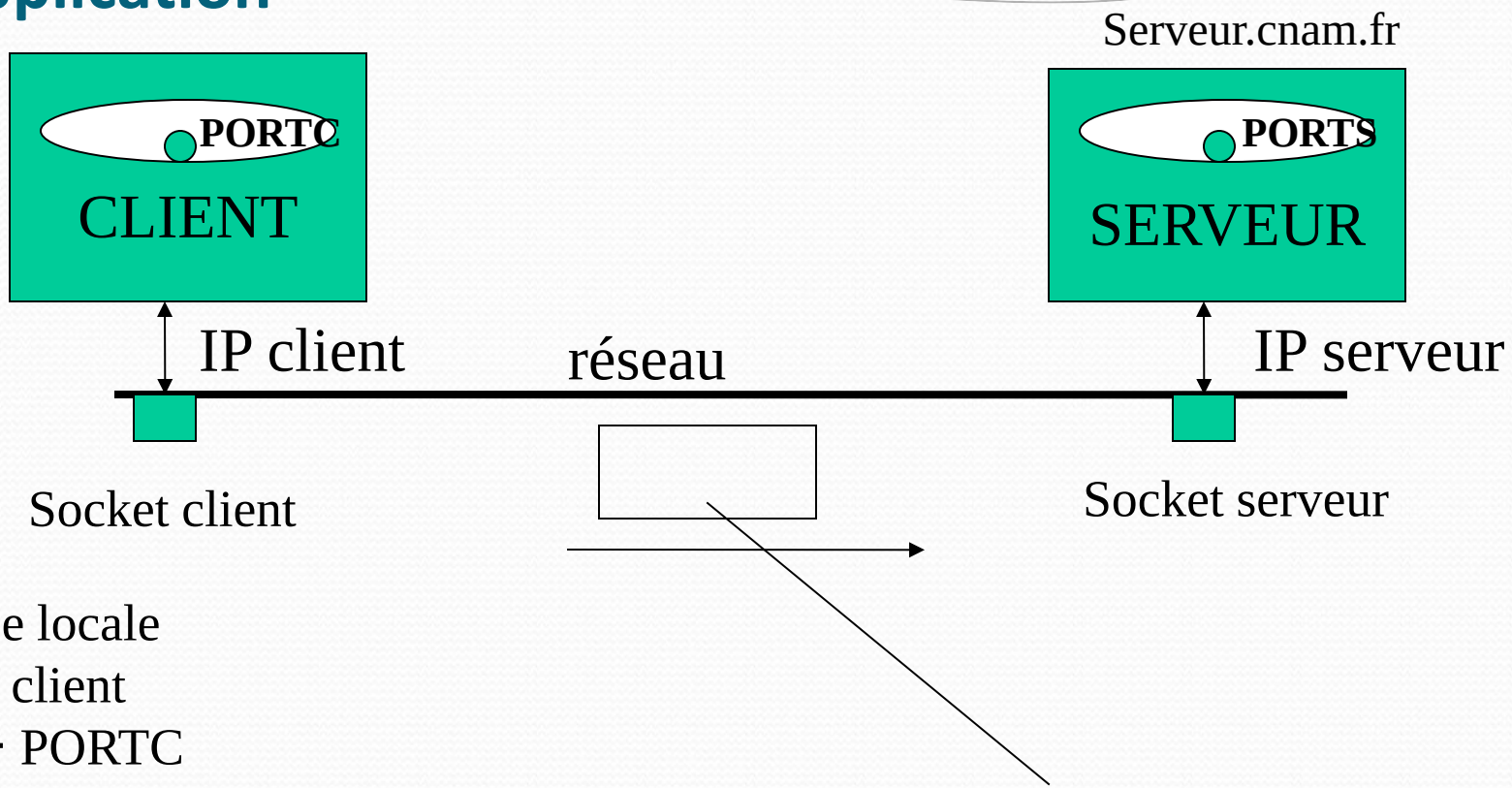
- Retour : 0 si succès,
- Résultat de la fonction est retournée dans champ ***res***

L'interface socket : Attachement d'une adresse d'application

- Définition de la structure de données stockant adresse(s) de l'hôte
  - Chaînage entre structures (via pointeur ***ai_next***)
  - Permet d'utiliser adresses IPv4 ou IPv6 (indiqué par ***ai_family***)

```
struct addrinfo {
        int ai_flags;         // AI_PASSIVE, AI_CANONNAME, etc.
        int ai_family;              // AF_INET, AF_INET6, AF_UNSPEC
        int ai_socktype;            // SOCK_STREAM, SOCK_DGRAM
        int ai_protocol;            // use 0 for "any"
        size_t ai_addrlen;          // size of ai_addr in bytes
        struct sockaddr *ai_addr;  // struct sockaddr_in or _in6
        char *ai_canonname; // full canonical hostname

        struct addrinfo *ai_next;    // linked list, next node
};
```

L'interface socket : Attachement d'une adresse d'application



Adresse locale
Socket client
Port ← PORTC
Adresse IP locale
getaddrinfo (client.cnam.fr, PORTC)

Adresse distante
Port ← PORTS
Adresse IP ← serveur.cnam.fr
getaddrinfo (serveur.cnam.fr, PORTS)

Exemple création socket

```
int socket (af, type, protocole)
int bind (sock, p_adresse, lg)
struct addrinfo {
        int ai_flags;           // AI_PASSIVE, AI_CANONNAME, etc.
        int ai_family;          // AF_INET, AF_INET6, AF_UNSPEC
        int ai_socktype;        // SOCK_STREAM, SOCK_DGRAM
        int ai_protocol;        // use 0 for "any"
        size_t ai_addrlen;      // size of ai_addr in bytes
        struct sockaddr *ai_addr; // struct sockaddr_in or _in6
        char *ai_canonname;     // full canonical hostname

        struct addrinfo *ai_next;  // linked list, next node };
int getaddrinfo (const char *node,                  // e.g. "www.example.com" or IP
                 const char *service,          // e.g. "http" or port number
                 const struct addrinfo *hints, // additional infos on socket type
                 struct addrinfo **res);
```

#define PORT “3490”

int status,sockfd;
struct addrinfo hints;
struct addrinfo *res;

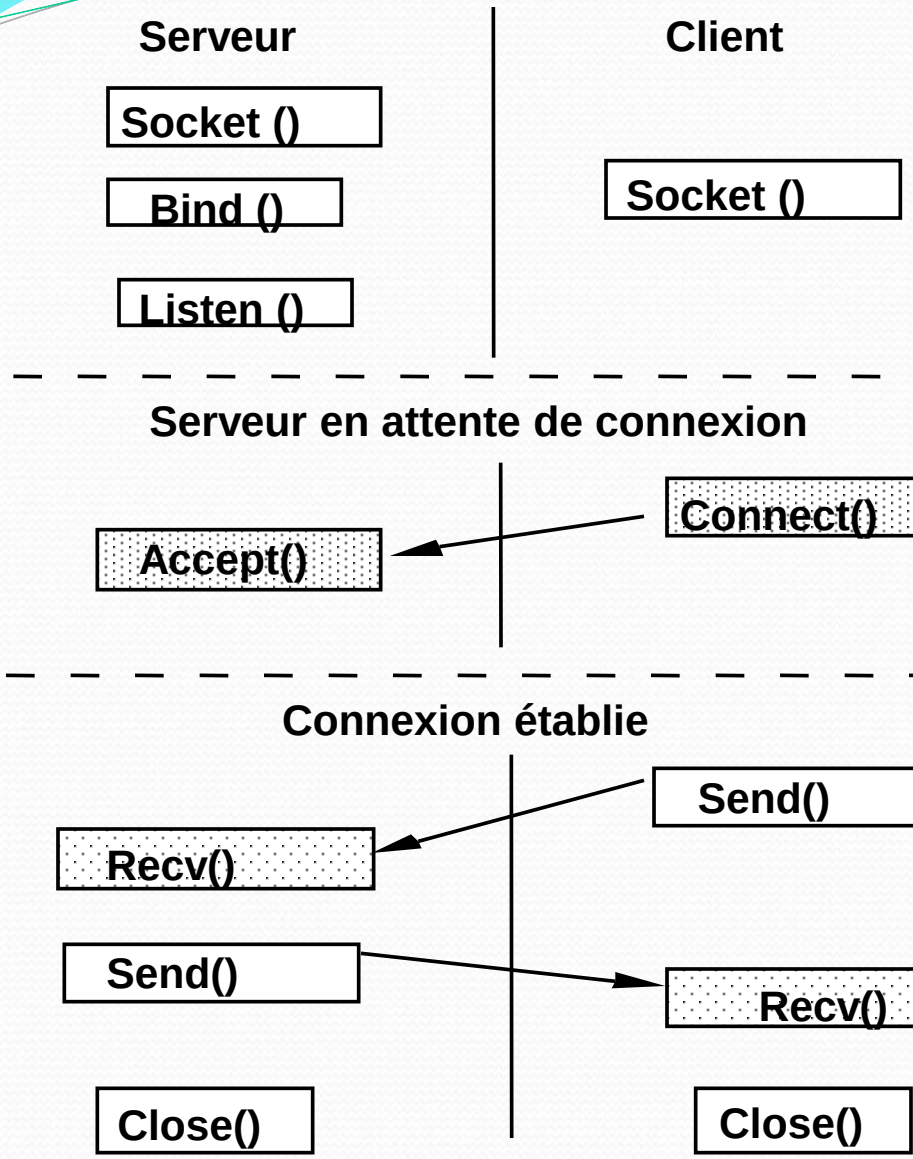
memset(&hints, 0, sizeof hints); // make sure the struct is empty
hints.ai_family = AF_UNSPEC; // don't care IPv4 or IPv6, otherwise use AF_INET or AF_INET6
hints.ai_socktype = SOCK_STREAM; // TCP stream sockets
hints.ai_flags = AI_PASSIVE; // fill in my IP, or use IP address (frist parameter of getaddrinfo())

status = getaddrinfo(NULL, PORT, &hints, &res); /*Si l'attribut **AI_PASSIVE** est indiqué dans *hints.ai_flags*, et si *node* est NULL, les adresses de socket renvoyées sont pertinentes pour lier une socket */

sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);

bind (sockfd, res->ai_addr, res->ai_addrlen);

L'interface socket : Communication en mode connecté



Sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)

Etablissement de connexion avec échange des adresses participantes

→ Les messages échangés ne contiennent pas l'adresse du destinataire

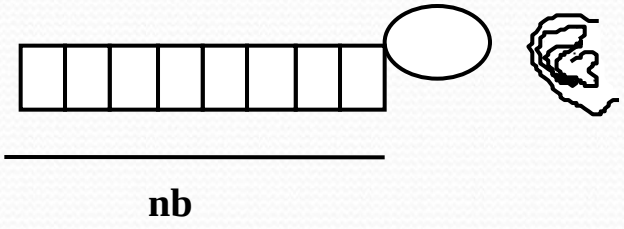
Appels bloquants

L'interface socket : Communication en mode connecté

Serveur | **Client**

Sock_ecoute = socket() -- création de la socket d'écoute

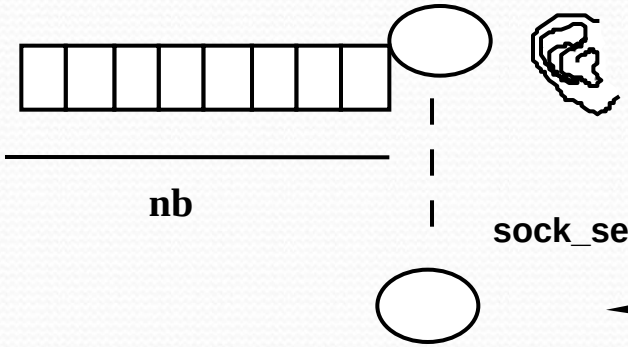
Listen (sock_ecoute, nb)



sock = socket() -- création de la socket

connect (sock, ...)

sock_service = accept(sock_ecoute,....)



read | **write**

Création d'une socket de service sur laquelle s'effectue les échanges de données

Communication en mode connecté

Connexion du client sur une application serveur

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

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

- Client essaie de se connecter à une application serveur
- Retourne 0 si succès, -1 sinon

Communication en mode connecté

Ecoute et acceptation connexion de clients

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

```
int listen(int sockfd, int backlog);
```

Socket d'écoute côté serveur

Nb max de connexion en attente de traitement (20 max système, en pratique entre 5 et 10)

- Serveur écoute et est en attente d'une demande de connexion d'un client
- Retourne 0 si succès, -1 sinon

```
newfd = accept(int sockfd, struct sockaddr *client_addr, socklen_t *addrlen);
```

Socket d'écoute côté serveur

Structure avec adresse et port du client

Taille structure client_addr

- Accepte une connexion client et retourne nouveau descripteur de socket de service (newfd) pour traiter requête du client
- Retourne -1 si echec

Communication en mode connecté

Echange de données

```
int send(int sockfd, const void *msg, int len, int flags);
```

Socket où envoyer les données	Pointeur sur les données à envoyer	Taille des données à envoyer (en octets)	Par défaut à 0, données normales

- Envoie des données sur sockfd indiquées par pointeur msg
- Retourne nombre d'octets envoyés si succès (***peut être*** < ***len***), -1 sinon

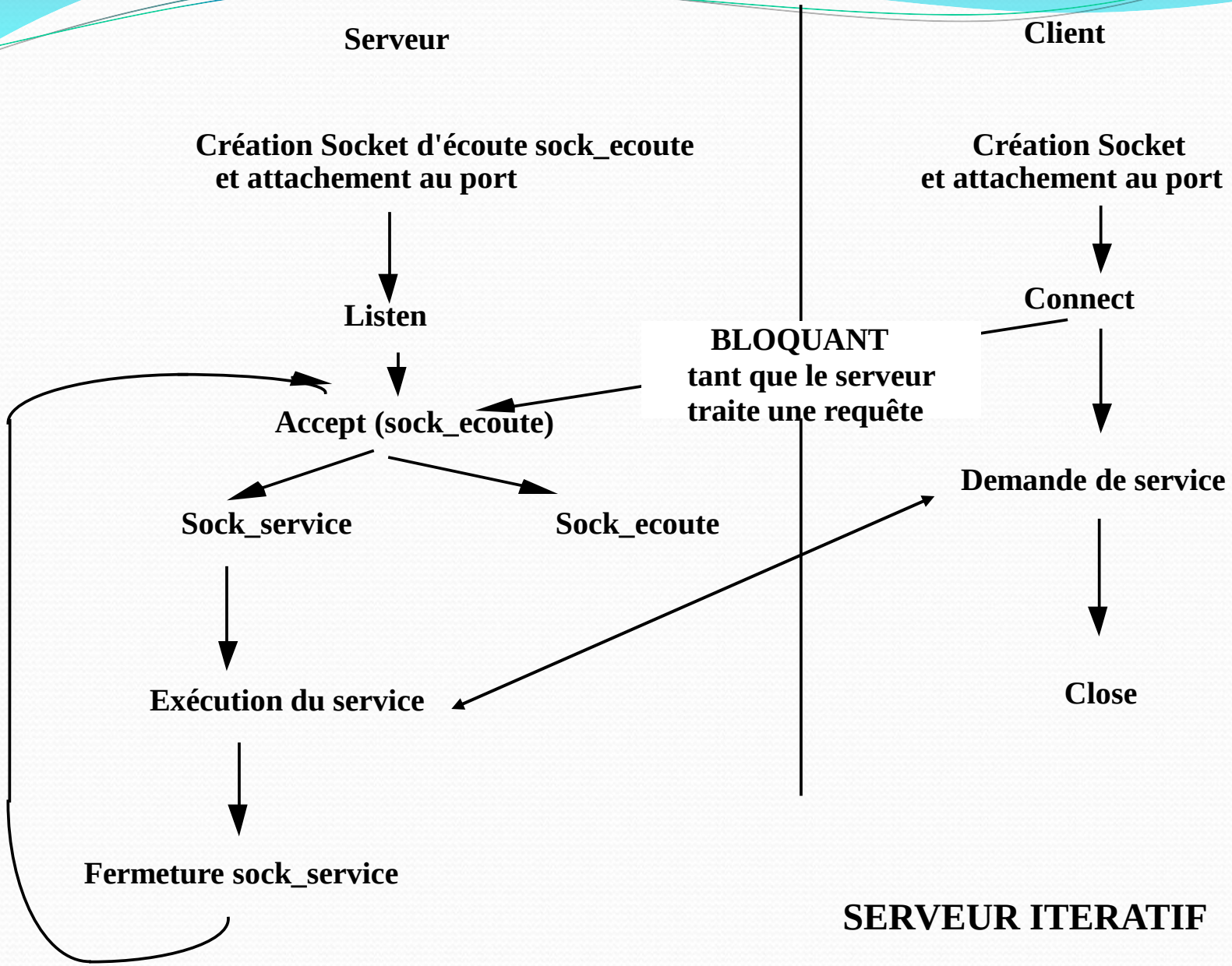
---

```
int recv(int sockfd, void *buf, int len, int flags);
```

Socket de réception de données	Pointeur buffer stocker données	Taille maximale du buffer (en octet)	Par défaut à 0 données normales

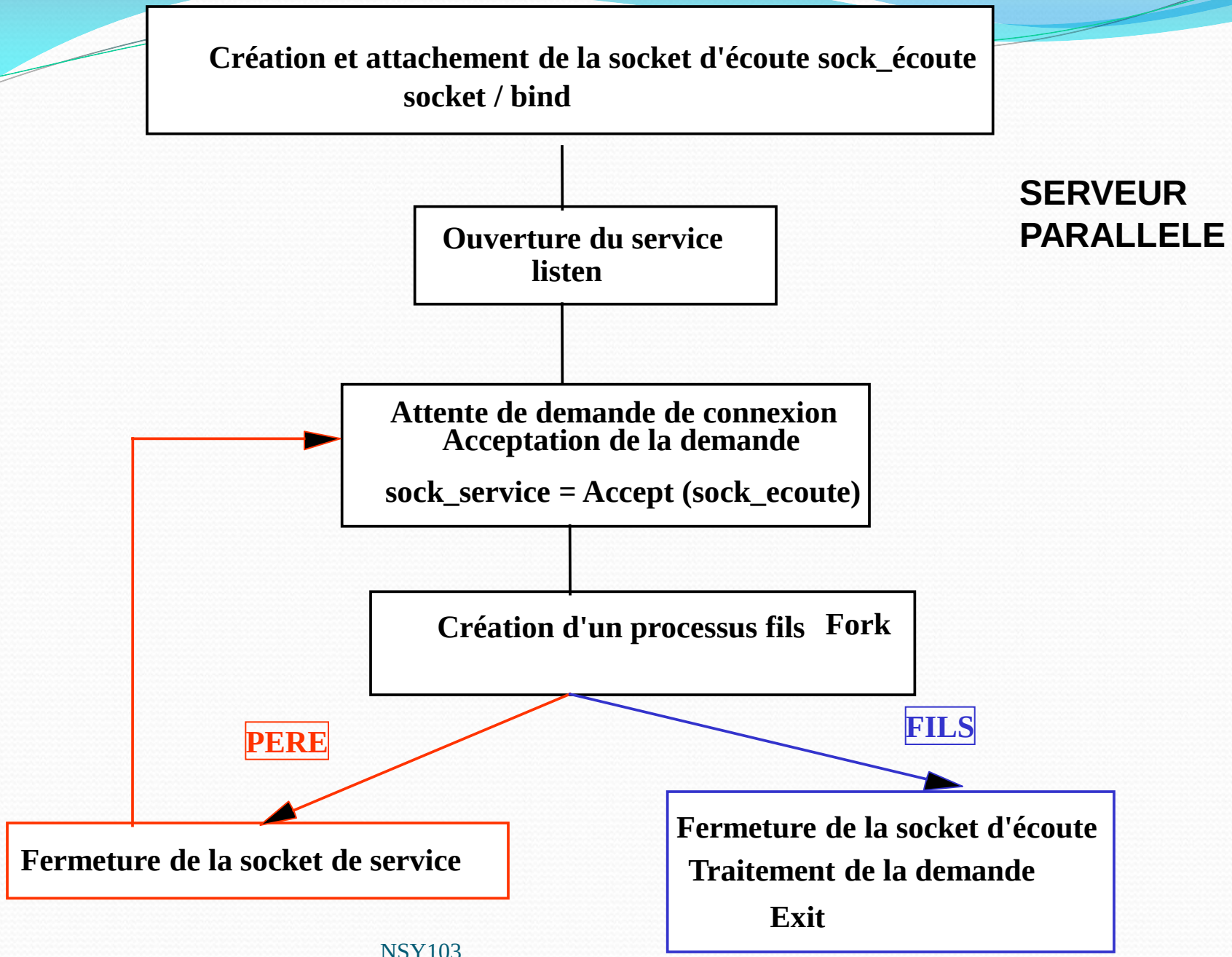
- Réceptionne des données d'une socket et les stocke dans un buffer (***appel bloquant***)
- Retourne nb octets lu si succès, 0 indique socket a été fermée, -1 sinon

L'interface socket : Communication en mode connecté



SERVEUR ITERATIF

L'interface socket : Communication en mode connecté



SERVEUR PARALLELE

Communication en mode connecté

```
/******* SERVEUR TCP  **************/

#include     <stdio.h>
#include     <sys/types.h>
#include     <sys/socket.h>
#include     <netinet/in.h>
#include     <netdb.h>
#include     <netdb.h>

#define PORTS “2058”

main()
{
int sockfd, new_fd, rv, sin_size;
pid_t numpid;
struct addrinfo hints, *svinfo;
struct sockaddr their_adr;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;          // use my IP
rv = getaddrinfo(NULL, PORTS, &hints, & svinfo);
```

```
// Création  socket  et  attachement
sockfd = socket(svinfo->ai_family, svinfo->ai_socktype,
   svinfo->ai_protocol))
bind(sockfd, svinfo ->ai_addr, svinfo ->ai_addrlen);

listen(sockfd, 5);

while(TRUE)
{   sin_size = sizeof(their_addr);
   new_fd = accept(sockfd, &their_addr, &sin_size);
   numpid= fork();
   if (numpid==0) {
   {  close(sockfd);
      send(new_fd, ‘’Hello!‘’, 6, 0);
      close(new_fd);  exit(0);  }
   else
       close (new_fd) } }}
```

```
int socket (af, type, protocole)
int bind (sock, p_adresse, lg)
struct addrinfo {
        int ai_flags;                   // AI_PASSIVE, AI_CANONNAME, etc.
        int ai_family;                  // AF_INET, AF_INET6, AF_UNSPEC
        int ai_socktype;                // SOCK_STREAM, SOCK_DGRAM
        int ai_protocol;                // use 0 for "any"
        size_t ai_addrlen;      // size of ai_addr in bytes
        struct sockaddr *ai_addr;  // struct sockaddr_in or _in6
        char *ai_canonname; // full canonical hostname

        struct addrinfo *ai_next;       // linked list, next node };
int getaddrinfo (const char *node,                  // e.g. "www.example.com" or IP
                const char *service,        // e.g. "http" or port number
                const struct addrinfo *hints, // additional infos on socket type
                struct addrinfo **res);
```

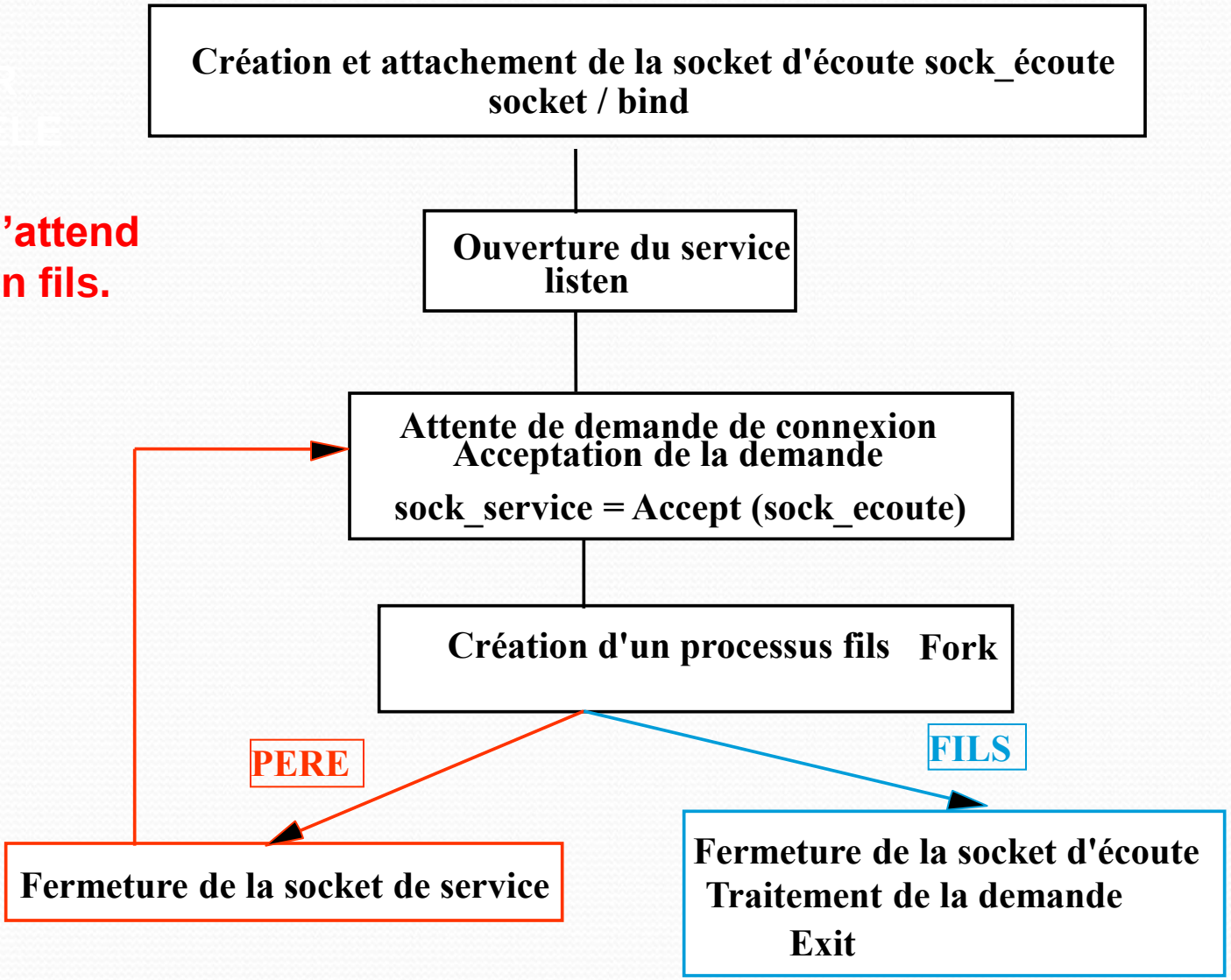
Communication en mode connecté

```
/*********** CLIENT TCP  **************/

#include     <stdio.h>
#include     <sys/types.h>
#include     <sys/socket.h>
#include     <netinet/in.h>
#include     <netdb.h>
#include     <netdb.h>
#define SERVEUR "163.215.82.55"
#define PORTS "2058"
#define PORTC "4096"


main()
{
int sockfd, new_fd, rv, sin_size;
struct addrinfo hints, *svinfo,  *res;
Char buf[100];

// Construction adresse serveur pour connect//
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
rv = getaddrinfo(SERVEUR, PORTS, &hints,
&svinfo);
```

```
// Création  socket  et  attachement
memset(&hints, 0, sizeof (hints);
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM);
hints.ai_flags = AI_PASSIVE;

status = getaddrinfo(NULL, PORTC, &hints, &res);

sockfd = socket(res->ai_family, res->ai_socktype, res-
>ai_protocol);

bind (sockfd, res->ai_addr, res->ai_addrlen);

connect(sockfd, svinfo ->ai_addr, svinfo ->ai_addrlen);
numbytes = recv(sockfd, buf, 100, 0);
printf(‘’Message reçu : %s\n”,buf);
close(sockfd);
}
```

```
int socket (af, type, protocole)
int bind (sock, p_adresse, lg)
struct addrinfo {
        int ai_flags;                   // AI_PASSIVE, AI_CANONNAME, etc.
        int ai_family;                  // AF_INET, AF_INET6, AF_UNSPEC
        int ai_socktype;                // SOCK_STREAM, SOCK_DGRAM
        int ai_protocol;                // use 0 for "any"
        size_t ai_addrlen;      // size of ai_addr in bytes
        struct sockaddr *ai_addr;  // struct sockaddr_in or _in6
        char *ai_canonname; // full canonical hostname

        struct addrinfo *ai_next;       // linked list, next node };
int getaddrinfo (const char *node,                      // e.g. "www.example.com" or IP
                const char *service,    // e.g. "http" or port number
                const struct addrinfo *hints, // additional infos on socket type
                struct addrinfo **res);
```

Communication en mode connecté



Le père n'attend pas son fils.

Communication en mode connecté

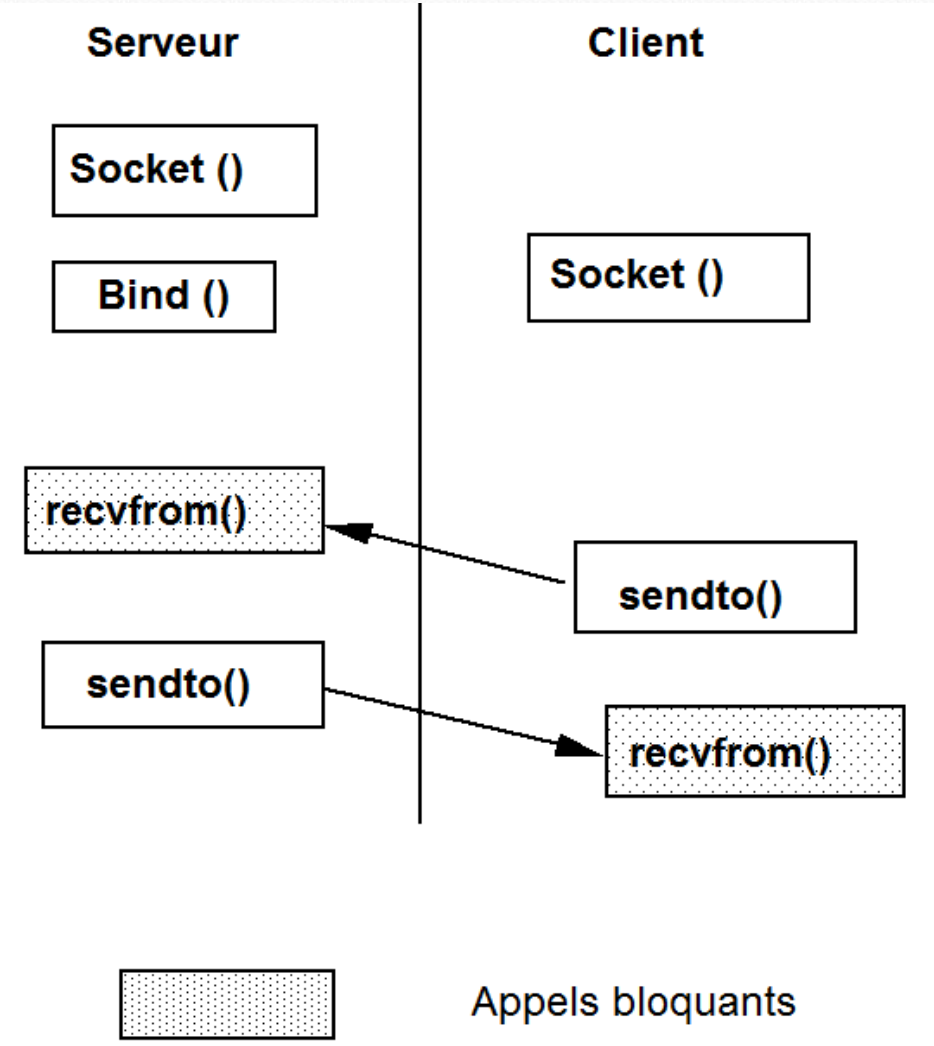
Le père n'attend pas ses fils → handler

Signal(SIGCHLD, SIG_IGN);

```
/******* SERVEUR TCP  **************/

#include     <stdio.h>
#include     <sys/types.h>
#include     <sys/socket.h>
#include     <netinet/in.h>
#include     <netdb.h>
#include     <netdb.h>

#define PORTS “2058”

main()
{
int sockfd, new_fd, rv, sin_size;
pid_t numpid;
struct addrinfo hints, *svinfo;
struct sockaddr their_adr;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;          // use my IP
rv = getaddrinfo(NULL, PORTS, &hints, & svinfo);
```

```
// Création  socket  et  attachement
sockfd = socket(svinfo->ai_family, svinfo->ai_socktype,
   svinfo->ai_protocol))
bind(sockfd, svinfo ->ai_addr, svinfo ->ai_addrlen);

signal(SIGCHLD, SIG_IGN);

listen(sockfd, 5);

while(TRUE)
{   sin_size = sizeof(their_addr);
   new_fd = accept(sockfd, &their_addr, &sin_size);
   numpid= fork();
    if (numpid==0) {
   {  close(sockfd);
      send(new_fd, ‘’Hello!‘’, 6, 0);
      close(new_fd);  exit(0);  }
    else
       close (new_fd) } }}
```

Communication en mode datagramme



Sock = socket (AF_INET, SOCK_DGRAM, IPPROTO_UDP)

Pas d'établissement de connexion

→ Chaque message échangé contient l'adresse du destinataire

Communication en mode non-connecté

Envoie et réception de données sur une socket

```
int sendto(int sockfd, void *msg, int len, int flags, struct sockaddr *to, socklen_t tolen);
```

Socket où envoyer les données

Pointeur données à envoyer

Taille données (en octets)

Par défaut à 0, données normales

Structure adresse destinataire et Taille structure

- Envoie des données sur sockfd indiquées par pointeur msg
- Retourne nombre d'octets envoyés si succès (***peut être*** < ***len***), -1 sinon

---

```
int recvfrom(int sockfd, void *buf, int len, int flags, struct sockaddr *from, int *fromlen);
```

Socket de réception de données

Pointeur buffer stocker données

Taille maximale (en octet)

Par défaut à 0 données normales

Structure adresse émetteur et Taille structure

- Réceptionne des données d'une socket et les stocke dans un buffer (***appel bloquant***)
- Retourne nb octets lu si succès, 0 indique socket a été fermée, -1 sinon

Communication en mode non-connecté

```
/******* SERVEUR UDP  **************/
#include     <stdio.h>
#include     <sys/types.h>
#include     <sys/socket.h>
#include     <netinet/in.h>
#include     <netdb.h>
#include     <netdb.h>

#define PORTS “2058”

main()
{
int sockfd, new_fd, rv, addr_len;
struct addrinfo hints, *svinfo;
struct sockaddr their_adr;
char buf[100];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_flags = AI_PASSIVE;         // use my IP
rv = getaddrinfo(NULL, PORTS, &hints, &svinfo);
```

```
// Création  socket  et  attachement
sockfd = socket(svinfo ->ai_family, svinfo ->ai_socktype,
 svinfo ->ai_protocol));
bind(sockfd, svinfo ->ai_addr, svinfo ->ai_addrlen);

while(TRUE)
{
numbytes = recvfrom(sockfd, buf, 100, 0,
        &their_addr, &addr_len);
printf("Chaine reçue %s\n", buf);
}
close(sockfd);  } }
```

```
int socket (af, type, protocole)
int bind (sock, p_adresse, lg)
struct addrinfo {
        int ai_flags;           // AI_PASSIVE, AI_CANONNAME, etc.
        int ai_family;          // AF_INET, AF_INET6, AF_UNSPEC
        int ai_socktype;        // SOCK_STREAM, SOCK_DGRAM
        int ai_protocol;        // use 0 for "any"
        size_t ai_addrlen;      // size of ai_addr in bytes
        struct sockaddr *ai_addr; // struct sockaddr_in or _in6
        char *ai_canonname;     // full canonical hostname

        struct addrinfo *ai_next;  // linked list, next node };
int getaddrinfo (const char *node,                  // e.g. "www.example.com" or IP
                const char *service,          // e.g. "http" or port number
                const struct addrinfo *hints, // additional infos on socket type
                struct addrinfo **res);
```

Communication en mode non-connecté

```
/********** CLIENT UDP **************/
#include     <stdio.h>
#include     <sys/types.h>
#include     <sys/socket.h>
#include     <netinet/in.h>
#include     <netdb.h>
#include     <netdb.h>
#define SERVEUR "163.215.82.55"
#define PORTS "2058"
#define PORTC "4096"

main()
{
int sockfd, new_fd, rv, sin_size;
struct addrinfo hints, *svinfo, *p;
struct sockaddr their_adr;
Char buf[100];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
rv = getaddrinfo(SERVEUR, PORTS, &hints,
&svinfo);
```

```
// Création socket et attachement
memset(&hints, 0, sizeof (hints);)    )
hints.ai_family = AF_UNSPEC;          )
hints.ai_socktype = SOCK_DGRAM)
hints.ai_flags = AI_PASSIVE;          )

status = getaddrinfo(NULL, PORTC, &hints, &res);

sockfd = socket(res->ai_family, res->ai_socktype, res-
>ai_protocol);

bind (sockfd, res->ai_addr, res->ai_addrlen);

sendto(sockfd, ''HELLO!", 6, 0, p->ai_addr, p->ai_addrlen);

close(sockfd);
return 0;
}
```

```
int socket (af, type, protocole)
int bind (sock, p_adresse, lg)
struct addrinfo {
        int ai_flags;                   // AI_PASSIVE, AI_CANONNAME, etc.
        int ai_family;                  // AF_INET, AF_INET6, AF_UNSPEC
        int ai_socktype;                // SOCK_STREAM, SOCK_DGRAM
        int ai_protocol;                // use 0 for "any"
        size_t ai_addrlen;      // size of ai_addr in bytes
        struct sockaddr *ai_addr;  // struct sockaddr_in or _in6
        char *ai_canonname; // full canonical hostname

        struct addrinfo *ai_next;       // linked list, next node };
int getaddrinfo (const char *node,                  // e.g. "www.example.com" or IP
                const char *service,        // e.g. "http" or port number
                const struct addrinfo *hints, // additional infos on socket type
                struct addrinfo **res);
```

Sockets avancés

Fonctions bloquantes

- Certaines fonctions de l'API Socket sont bloquantes
  - Attente de l'arrivée d'un évènement
  - Exemple de fonctions
    - *listen, connect, accept, recv/recvfrom*
- Besoin parfois de socket non bloquant
  - Attente d'évènements sur plusieurs sockets
- Possibilité de rendre socket non bloquant
  - Fonction *fcntl()*
  - Pb : utilisation attente active pour recevoir données
    - Utilisation inutile du CPU

Utilisation de plusieurs Sockets

- Fonction donnant l'état de plusieurs sockets

```
int select(int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
```

Numéro plus grand Socket descriptor +1

Ensembles de sockets en attente de :
- Lecture
- Ecriture
- Exceptions

Timer attente, Désactivé si NULL

- Fonction attendant évènement sur l'un des sockets des ensembles (ou attente période indiquée par ***timeout***)
- Retourne nb sockets avec évènements, 0 si timeout, -1 si erreur

```
struct timeval {
        int tv_sec; // seconds
        int tv_usec; // microseconds
};
```

Utilisation des ensembles de Sockets

- Plusieurs macro utiles
  - Ajout d'un descripteur de socket
    - **FD_SET(int fd, fd_set *set);**
  - Suppression d'un descripteur de socket
    - **FD_CLR(int fd, fd_set *set);**
  - Test si descripteur socket est dans l'ensemble
    - **FD_ISSET(int fd, fd_set *set);**
  - Supprime tous les éléments de l'ensemble
    - **FD_ZERO(fd_set *set);**

Exemple utilisation Select()

```
/******* SERVEUR TCP  **************/
#include     <stdio.h>
#include     <sys/types.h>
#include     <sys/socket.h>
#include     <netinet/in.h>
#include     <netdb.h>

#define PORTS “2058”

main()
{
int sockfd, new_fd, rv,;
struct addrinfo hints, *servinfo, *p;
struct sockaddr their_adr;
socklen_t addrlen;
char buf[256];  // buffer données client

fd_set master;  // master file descriptor list
fd_set read_fds;  // temp file descriptor list for select()
int fdmax;  // maximum file descriptor number

FD_ZERO(&master);  // clear the master and temp
sets
FD_ZERO(&read_fds);

// Création  socket  et  attachement
        …….
listen(sockfd, 5);

FD_SET(sockfd, &master);   // Ajout sockfd à ensemble
fdmax = sockfd;   // Garde valeur max socket

while(TRUE)
{  read_fds = master;     // ensemble socket attente lecture
   if (select(fdmax+1, &read_fds, NULL, NULL, NULL)
== -1)
   {  perror("select"); exit(4);}

   for(i = 0; i <= fdmax; i++)
   {
       if (FD_ISSET(i, &read_fds))
       {   if (i == sockfd)
           {     addrlen = sizeof(their_adr);
        new_fd = accept(sockfd,
                &their_adr, &addrlen);
         if (new_fd == -1)
        {  perror("accept");}
        else
        {  // Ajout new_fd  à ensemble
           FD_SET(new_fd, &master);
           if (new_fd > fdmax)
           {  fdmax = new_fd; }
           printf(”Nouvelle connexion au serveur.\n‘’);
        }
           }
```

Exemple utilisation Select()

```
else
{   // gestion données client i
   if ((nbytes = recv(i, buf, sizeof buf, 0)) <= 0)
   {   // erreur ou connexion fermée par client
      if (nbytes == 0)
      { printf(« Connexion %d fermée.\n", i);}
      else
      {  perror("recv"); }
      close(i);
      FD_CLR(i, &master); // Supprime ensemble
    }
    else
    {  // Données reçu du client
       for(j = 0; j <= fdmax; j++)
       {  // Envoie données à tous les autres clients
j
          if (FD_ISSET(j, &master))
          {
    // Sauf serveur et client source données
    if (j != sockfd && j != i)
    {
        send(j, buf, nbytes, 0);
    }
          }
      }
    }
        } // Fin bloc ELSE client
      } // Fin bloc IF  FD_ISSET
   } // Fin bloucle FOR sur i
} // Fin boucle WHILE

 return 0;
}
```