

Communications sous UNIX

programmation des sockets

Thierry Grandpierre
ESIEE Paris

Plan du cours

I. Introduction

- Rappel, définitions

II. Mode non connecté

- Unix, UDP
- Scrutation
- Pseudo connexions

III. Mode connecté :

- Modèle client/serveur
- Messages urgents

IV. Compléments

- Fonctions de paramétrage, options, paramètres
- Gestion des erreurs et des signaux (asynchronisme)
- Inetd

Communications entre applications

- Différentes techniques de communication entre des processus appartenant à un même système :
 - Signaux
 - Tubes
 - Fichiers...

Les sockets

- Comm. entre appli. sur même système ou diff. syst.
- Sockets : points de communication bidirectionnels
processus : émettre et/ou recevoir informations
 - Le type des informations dépend du type des sockets
 - Domaine de la socket : définit l'ensemble des sockets atteignables + protocoles utilisables (influe sur type)
 - Sécurisé ou non
 - Orienté connexions ou sans connexions

Structure socket

short so_type	type socket
short so_options	options d'utilisation
short so_linger	durée de persistance après fermeture
short so_state	état de la socket
caddr_t so_pcb	bloc de ctrl. protocole
struct protosw *so_proto	description du protocole
struct socket *so_head	pointeur chaînage arrière
struct socket *so_q0	pointeur cnx. en cours d'établis.
struct socket *so_q	pointeur cnx. établies pendantes
short so_q0len	nombre de cnx. partielles
short so_qlen	nombre de cnx. pendantes
short so_qlimit	nombre max. de cnx. pendantes
short so_timeo	temporisation pour cnx.
u_short so_error	erreur en cours de cnx.
short so_pgrp	groupe de processus (gestion signaux)
struct sockbuf so_rcv, so_snd	tampons émission/réception

Type d'une socket

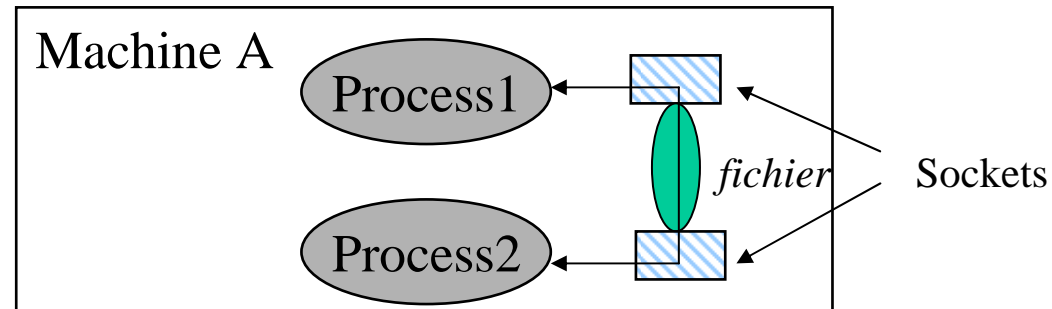
- **SOCK_DGRAM** : socket de transmission de datagrammes structurés (messages structurés)
- **SOCK_STREAM** : socket de transmission de séquences continues de caractères
- Autres types : SOCK_RAW, SOCK_RDM...

Domaine d'une socket

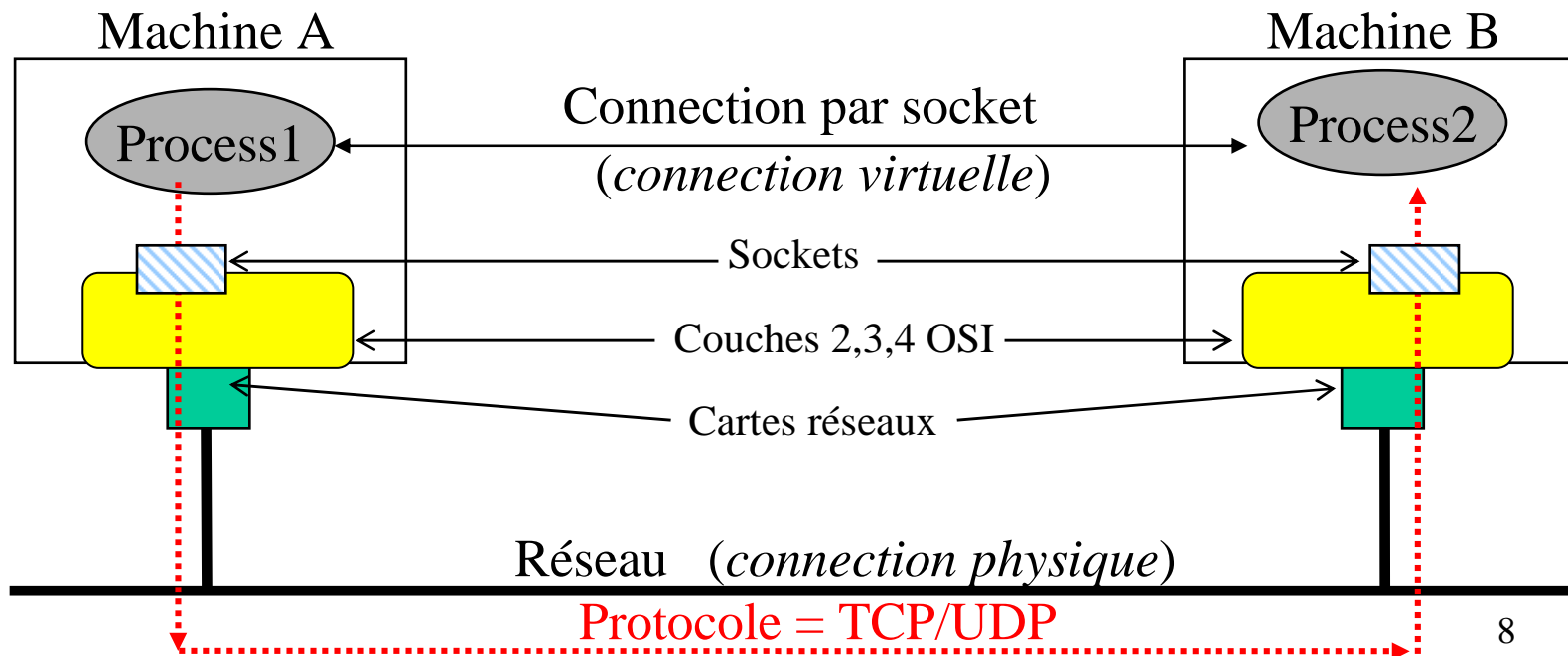
- Domaine **AF_UNIX** :
 - Comm. entre applications sur le même système
 - Socket \Leftrightarrow simple fichier
- Domaine **AF_INET** :
 - Communication entre applications sur systèmes différents connectés par réseau internet
 - Protocoles : UDP et TCP
- Autres : AF_NS (Xerox), AF_APPLETALK...

Domaines AF_UNIX et AF_INET

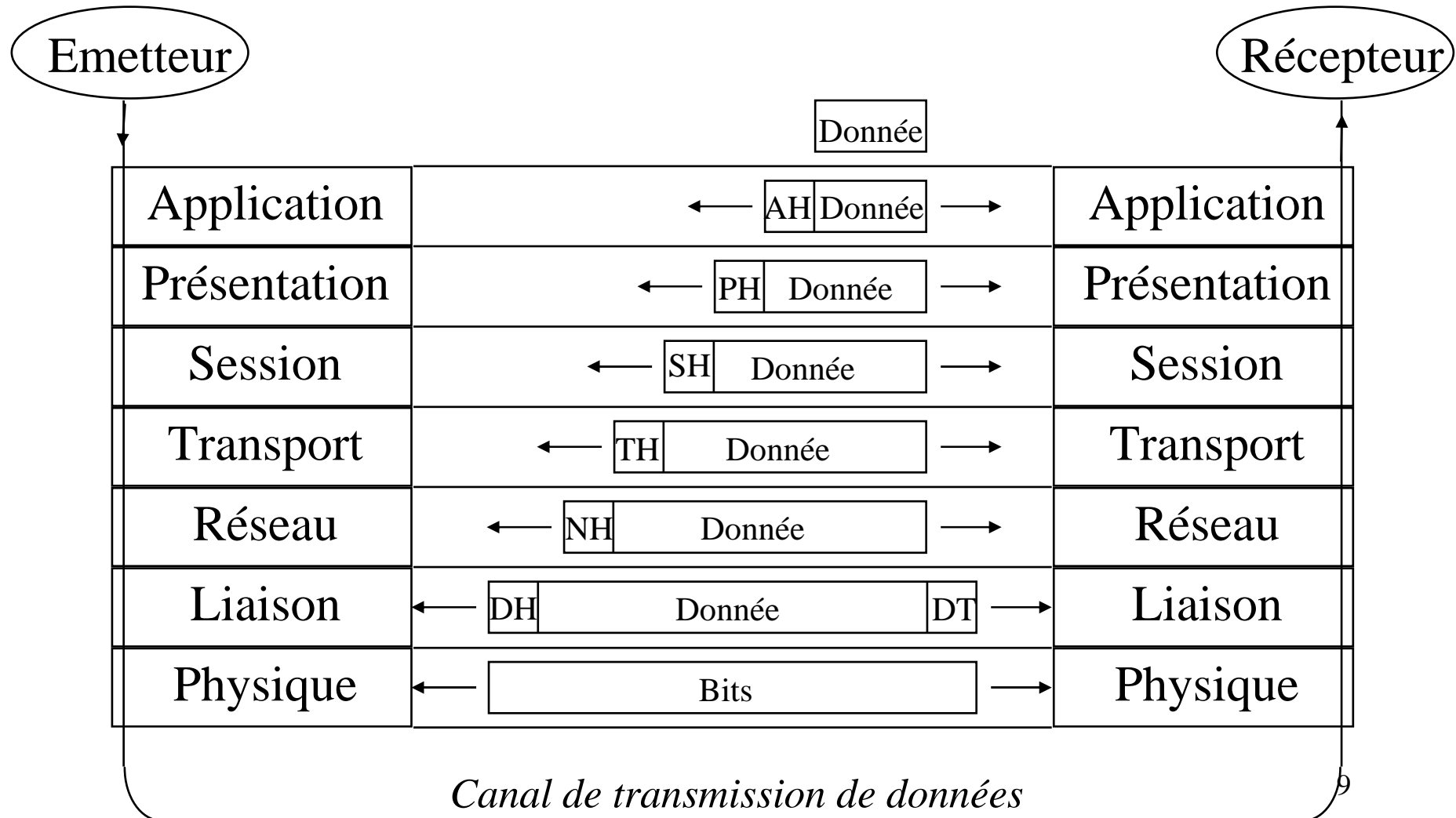
AF_UNIX :



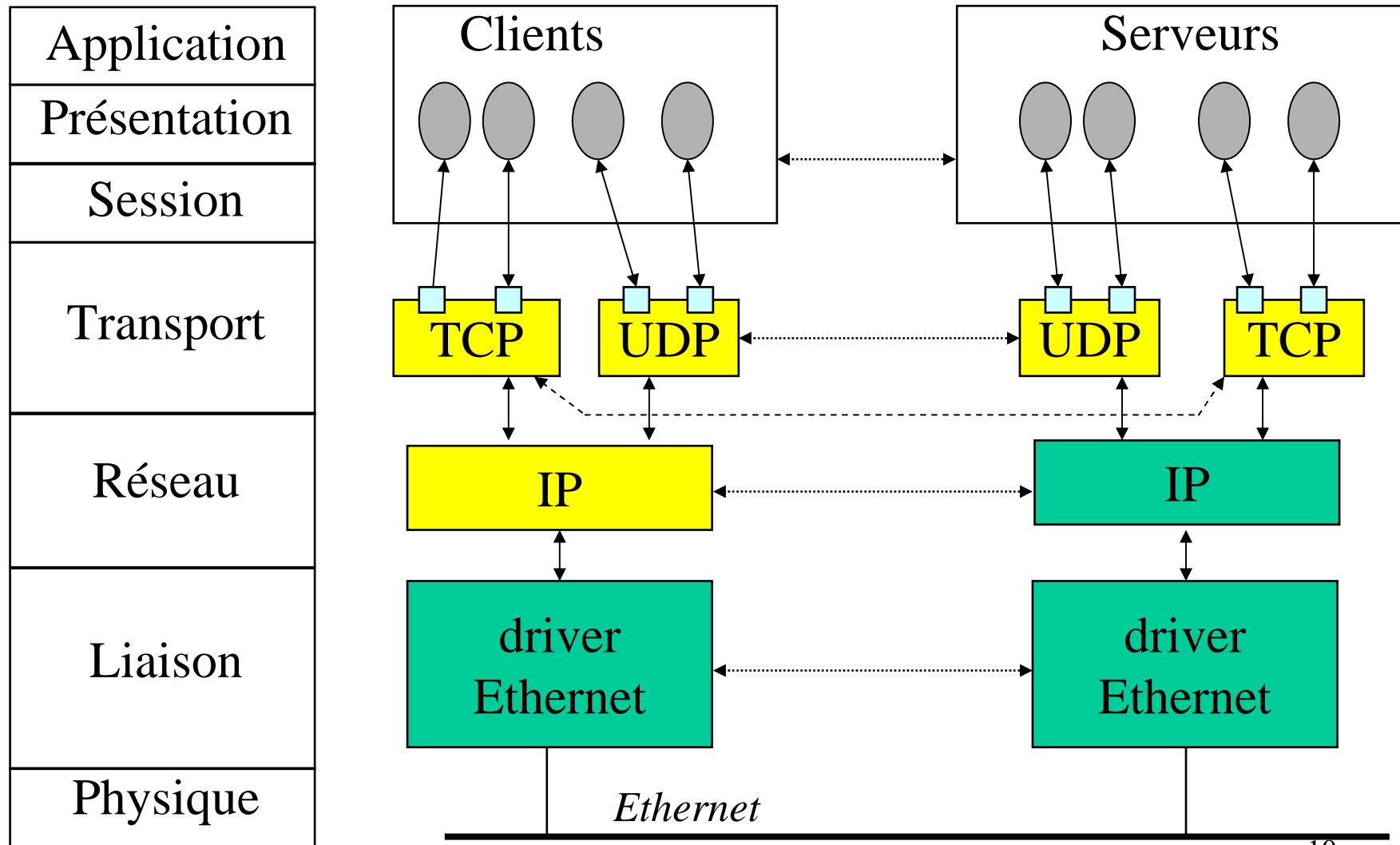
AF_INET:



Rappel : modèle OSI



Rappel : couches de communication



Rappel : services IP

Services assurés par IP :

- Transport des datagrammes de bout en bout
- Mode sans connexion
 - chaque datagramme traité indépendamment des autres
- Pas de garantie de remise des datagra. (non fiable)
- Assure le routage
- Peut fragmenter les messages

Services non offert par IP :

- La vérification du séquençement
- La détection de pertes
- La retransmission en cas d'erreur,
- Le contrôle de flux

Rappel : protocoles de comm.

- UDP : User Datagram Protocol
 - Mode sans connexion
 - Pas de contrôle d'erreur (sans garantie)
- TCP : Transmission Control Protocol
 - Mode connecté : ouverture, fermeture, gestion de connexion
 - Contrôle de flux, ordonné (préservation du séquençement)
 - Sans erreur : contrôle et retransmission si nécessaire
 - Sans perte : « numérotation » et retransmission
 - Système d'acquittement
 - Contrôle de flux (fenêtre d'émission) en full-duplex
 - Identification du service par numéro de port

Création d'une socket

```
#include <netinet/in.h> /* pour protocole ds cas AF_INET */  
#include <sys/types.h>  
#include <sys/socket.h>
```

```
int socket (int Domaine, int Type, int Protocole) /* retourne  
un descripteur sur la socket créée ou -1 en cas d'erreur */
```

Domaine	Type	Protocole (choix auto. si =0)
AF_UNIX	SOCK_DGRAM SOCK_STREAM	
AF_INET (IPv4) AF_INET6 (IPv6)	SOCK_DGRAM SOCK_STREAM SOCK_RAW	IPPROTO_UDP IPPROTO_TCP

Attachement d'une socket

```
int bind (  
    int descripteur,                /* socket à attacher */  
    struct sockaddr *ptr_address, /* descrip. de l'adresse */  
    int longueur_adresse           /* long. struct. précédente */  
)
```

retourne 0 en cas de succès, -1 sinon

2 Cas :

Domaine	Structure adresse	Fichier include
AF_UNIX	sockaddr_un	sys/un.h
AF_INET	sockaddr_in	netinet/in.h

Format d'adresse AF_UNIX

- Désignation similaire à fichier :
socket \Leftrightarrow noeud de type socket (S_IFSOCK)

```
#include <sys/un.h>
struct sockaddr_un {
    short sun_family;    /* AF_UNIX */
    char sun_path[108]; /* référence */
}
```

Particularité AF_UNIX

- Association de sockets non nommées :

```
int socketpair (  
    int domaine,                /*AF_UNIX*/  
    in type,                    /*SOCK_DGRAM STREAM*/  
    int protocole,              /* O : proto. par défaut */  
    int *ptr_descripteur        /* tab. à 2 entiers */  
)
```

- Proche des tubes POSIX mais bidirectionnelles
- Choix mode de communication (STREAM != tube)

Format d'adresse AF_INET

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

struct sockaddr_in {
    short sin_family ;                /* AF_INET */
    u_short sin_port;                /* numéro port */
    struct in_addr sin_addr;         /* @ internet machine*/
    char sin_zero[8];                /* champs de 8 carac. nuls */
}

struct in_addr {
    u_long s_addr;
}
```

Particularités AF_INET

- Si `sin_addr.s_addr = INADDR_ANY`
⇒ la socket est associée à toutes les @ de la machine
(évite utilisation de `gethostname()`)
- Si `sin_port = NULL` ⇒ bind sur port quelconque
- Si processus envoie infos sur socket sans l'attacher
⇒ attachement sur port quelconque

```
int getsockname (  
    int descripteur,  
    struct sockaddr *ptr_adresse, ← obtention port  
    int *ptr_longueur_adresse  
)
```

Compléments : codage @ et port (1)

- Adresse et ports codés par entiers
 - u_short sin_port; (16 bit)
 - in_addr sin_addr; (32 bit)
- Problème :
 - Différentes machines / différents OS
 - little-endian: poids faible 1er
 - big-endian: poids fort 1er
 - Ces machines doivent potentiellement pouvoir communiquer sur les réseaux

```
struct sockaddr_in {  
    short sin_family;    /* AF_INET */  
    u_short sin_port;    /* numéro port */  
    struct in_addr sin_addr;    /* @IP*/  
    char sin_zero[8];    /* 8 caract. nuls */  
}  
  
struct in_addr {  
    u_long s_addr;  
}
```

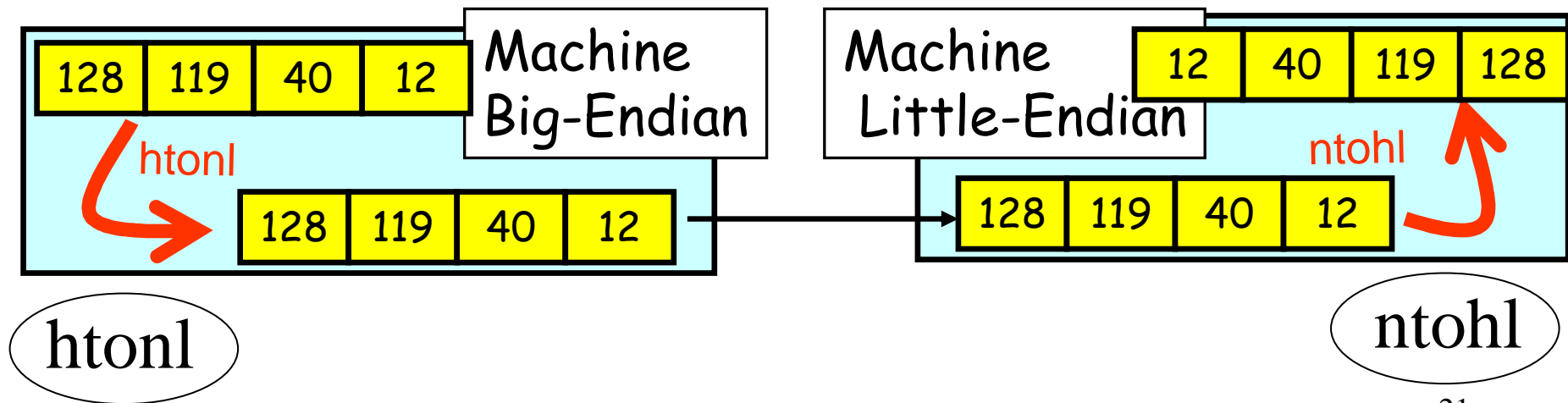
Compléments : codage @ et port (2)

- Définitions:
 - Host Byte-Ordering : byte ordering de l'hôte
 - Network Byte-Ordering: byte ordering réseaux :
 - Toujours big-endian
- Avant transmission sur le réseaux
 - Toutes données envoyées doivent être converties en Network Byte-Order (et reconverties ensuite dans le Host Byte-Order après réception)

Compléments : codage @ et port (3)

- `u_long htonl(u_long x);`
- `u_short htons(u_short x);`
- `u_long ntohl(u_long x);`
- `u_short ntohs(u_short x);`

➤ Sur machine big-indian : ne font rien



Obtention d'une adresse IP

```
#include <netdb.h>
```

```
struct hostent *gethostbyname(const char *name);
```

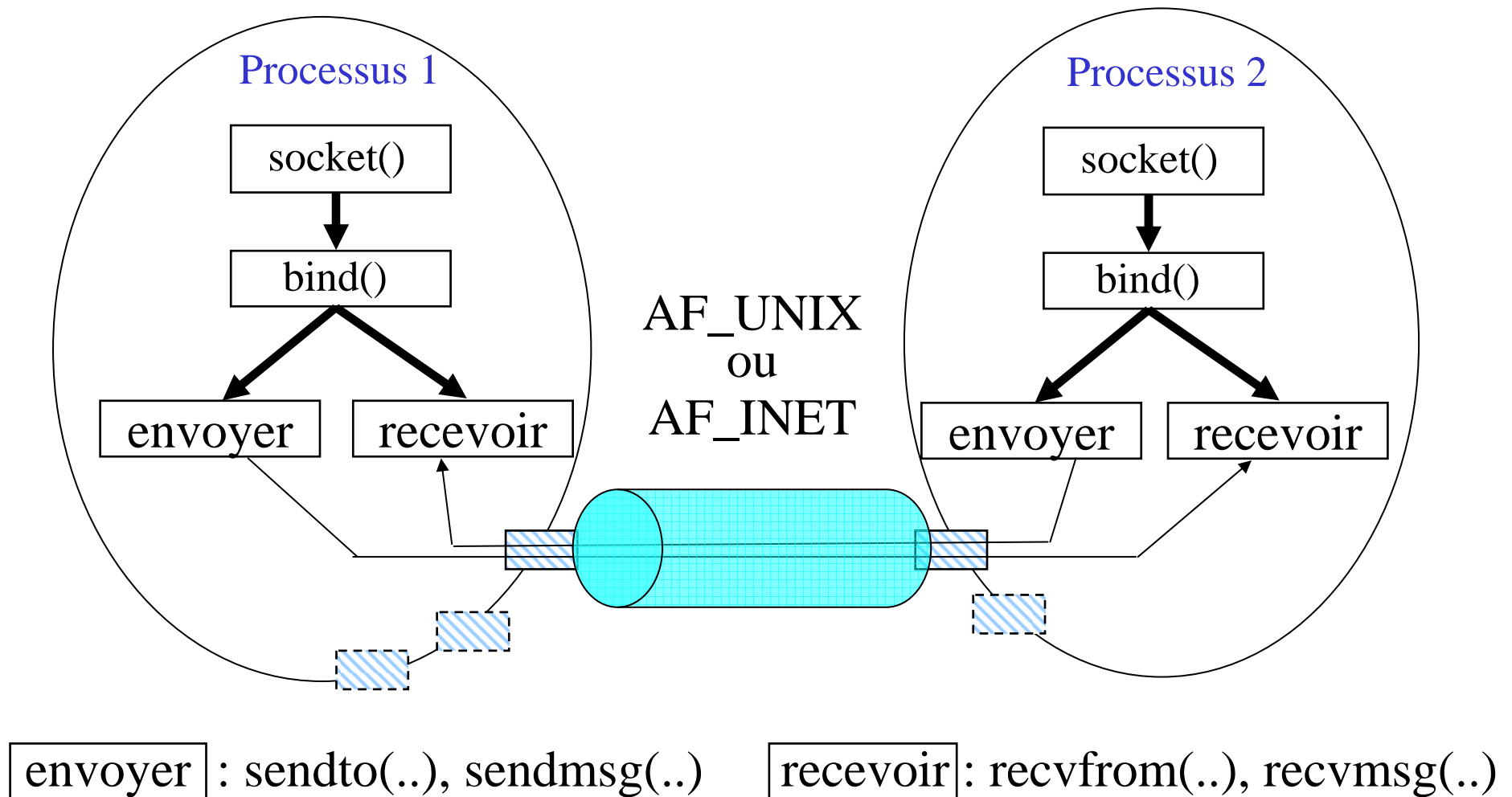
```
struct hostent {  
    char    *h_name;      /* official name of host */  
    char    **h_aliases;  /* alias list */  
    int     h_addrtype;   /* host address type (AF_INET)*/  
    int     h_length;     /* length of address */  
    char    **h_addr_list; /* list of addresses */  
}  
  
#define h_addr h_addr_list[0] /* for backward compatibility */
```

Obtention d'une adresse IP

Exemple :

```
struct hostent *hp;  
struct sock_addr_in adresse_dest;  
  
hp=gethostbyname(...)  
memcpy(&adresse_dest.sin_addr.s_addr, hp->addr, hp->h_length);  
  
adresse_dest.sin_family = AF_INET;  
adresse_dest.sin_port = htons(atoi(..)) ;
```

Communications par datagrammes



DGRAM : envoi d'un message

- Chaque msg. émit : @ dest. (non connecté)
- AF_INET + DGRAM=UDP, pas verif. @dest

```
int sendto (  
    int descripteur,  
    void *message,                /* msg. à envoyer */  
    int longueur,                 /* long. msg., <9000 AF_INET */  
    int option,                   /* 0 */  
    struct sockaddr *ptr_adresse, /* @ destinataire */  
    int longueur_adresse          /* long. de la structure */  
) ⇒ ret. nb carac. envoyés + erreurs locales uniquement
```


DGRAM : réception message

```
int recvfrom (  
    int descripteur,  
    void *message          /* @ de réception msg.*/  
    int longueur,          /*taille @ réservée */  
    int option,            /* 0 ou MSG_PEEK*/  
    struct sockaddr *ptr_adresse /* long. @ */  
    int *ptr_long_adresse  
)
```

- Extrait un msg. complet, un seul msg \Leftrightarrow 1 sendto
- Tronqué si longueur message reçu > taille zone réservée
- Par défaut réception bloquante si tampon récept. vide
- ret. nb carac. reçus ou -1 en cas d'échec


DGRAM : envoi fragments ds msg.

```
int sendmsg (  
    int descripteur,  
    struct msghdr *msghdr,  
    int option  
)
```



```
<sys/uio.h>  
struct iovec {  
    caddr_t iov_base /* @ frag. */  
    int iov_len /* longueur frag. */  
}
```

```
struct msghdr {  
    caddr_t msg_name,          /* @ dest., optionnelle */  
    int  msg_namelen,          /* taille @ */  
    struct iovec *msg_iov, /* tableau de fragments */  
    int msg_iovlen,           /* nb. élément tableau */  
    caddr_t msg_accright, /* pointeur sur descrip. (AF_UNIX) */  
    int msg_accrightrightlen /* longueur descrip. */  
}
```



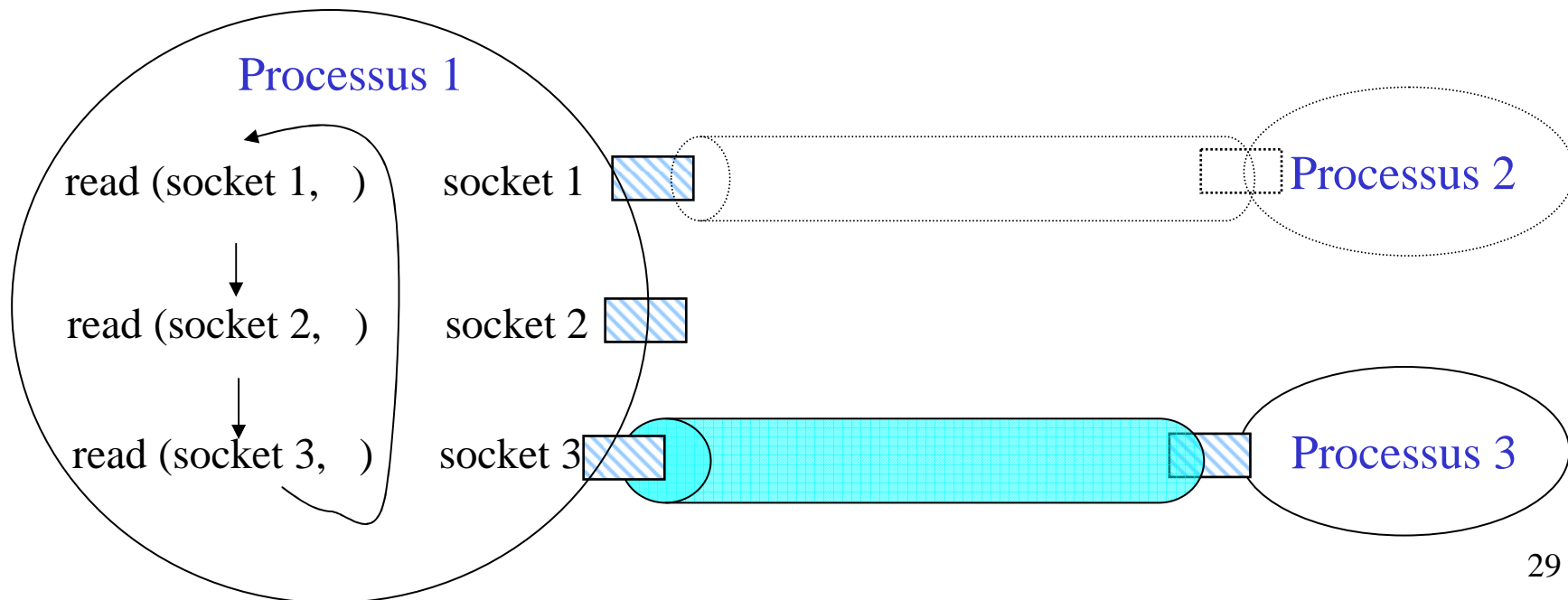
DGRAM : réception fragments

```
int recvmsg (  
    int descripteur,  
    struct msghdr *msghdr,  
    int option  
)
```

- retourne taille msg. reçu : somme taille des fragments
- Par défaut réception bloquante

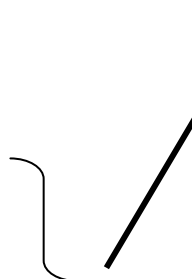
Scrutation de plusieurs descripteurs

- Processus lisant données de plusieurs sockets :
 - Réception bloquante donc risque blocage processus quand ordre réception (émission) aléatoire



Scrutation : fonction « select »

```
#include <sys/type.h>
#include <sys/time.h>
int select (
    size_t nb_desc,
    fd_set *ptr_lecture,
    fd_set *ptr_ecriture,
    fd_set *ptr_exception,
    const struct timeval *ptr_tempo
)
```



Ensembles de descripteurs
manipulés avec :

```
FD_ZERO(fd_set *ptr_set)
FD_CLR(int desc, fd_set *ptr_set)
FD_SET(int desc, fd_set *ptr_set)
FD_ISSET(int desc, fd_set *ptr_set)
```

- Bloquante tant que : un des événements attendu non arrivé
ou temps d'attente écoulé (ret. 0)
ou réception d'un signal (ret. -1)
- En cas d'erreur (desc. incorrect, pointeur incorr.) retourne -1

Pseudo-connexions DGRAM: création

- Éviter de re-spécifier @ destinataire dans msg.

```
int connect (  
    int descripteur, /* sock locale */  
    struct sockaddr *ptr_adresse,  
    int lg_adresse  
)
```

```
Récupération socket paire :  
int getpeername (  
    int desc,  
    struct sockaddr *ptr_adr  
    int *ptr_long_adr  
)
```

➤ Pas de négociation de la connexion (UDP)

- Exemple : destinataire non existant
- Appel non bloquant (**AF_INET/SOCK_DGRAM**)

Pseudo-connexions : envoie msg.

- sendto/recvfrom :
 - génèrent erreurs si @ dest/emet. \neq @ connexion
 - nécessitent @ à chaque utilisation
- send et write (utilisables en mode connecté) :

```
ssize_t send (  
    int descripteur,  
    void *ptr,  
    size_t nb_caracteres,  
    int option  
)
```

(pas d'option en DGRAM)

```
ssize_write (  
    int descripteur,  
    void *ptr,  
    size_t nb_caracteres  
)
```

➤ ret. nb carac. envoyés ou -1 en cas d'échec

Pseudo-connexions : réception msg.

- Option MSG_PEEK : lire sans extraire

```
ssize_t recv (  
    int descripteur,  
    void *ptr,  
    size_t nb_caracteres,  
    int option  
)
```

```
ssize_t read (  
    int descripteur,  
    void *ptr,  
    size_t nb_caracteres  
)
```

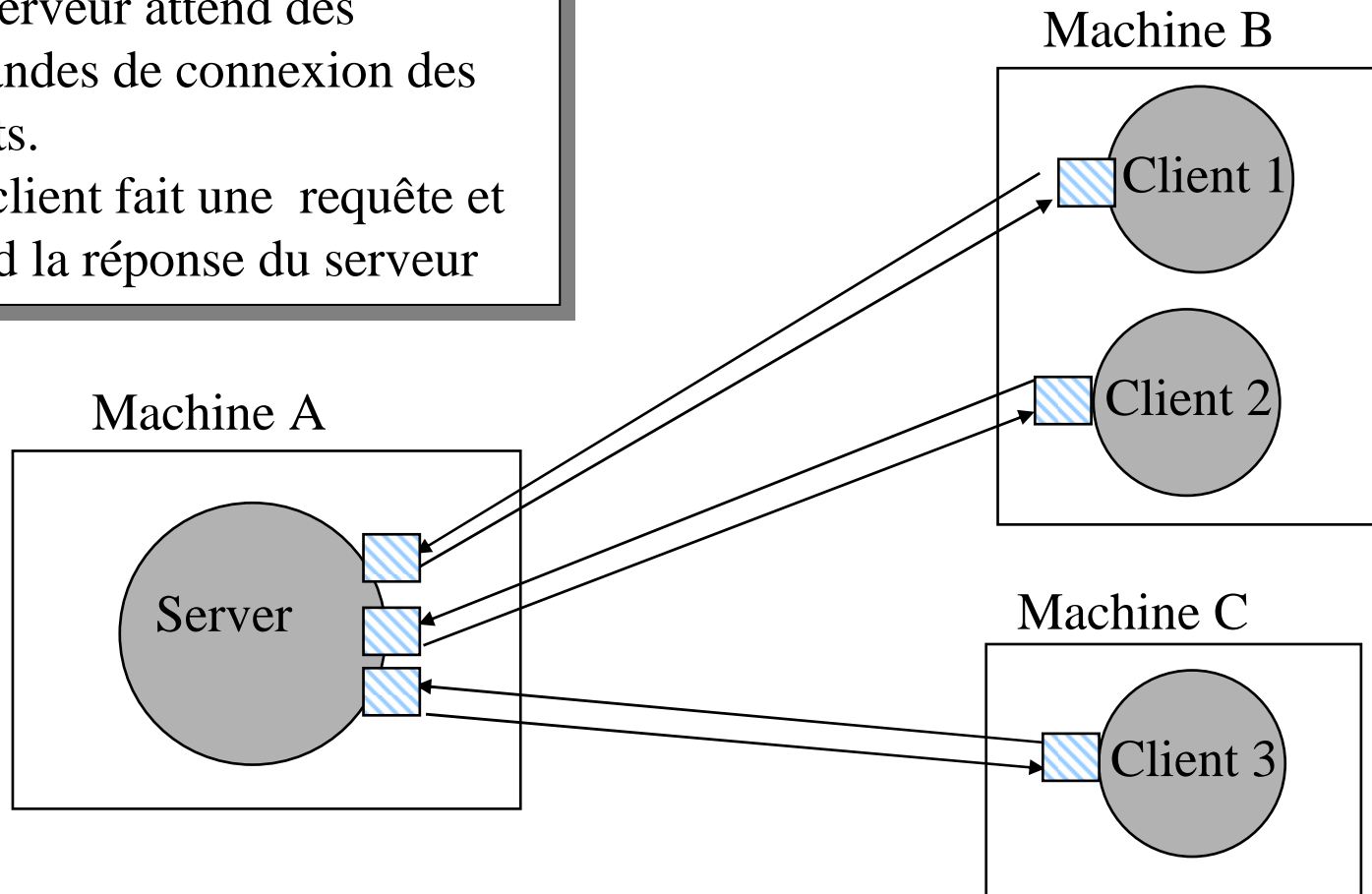
➤ ret. nb carac. envoyés ou -1 en cas d'échec

Communications en mode connecté

- AF_INET, SOCK_STREAM : TCP
 - Sécurité, contrôle
 - Adapté pour connexion distante entre 2 entités
- Flot continu non structuré : gros volumes d'info.
- Dissymétrie de la communication :
 - Serveur : attente passive de connexion
 - Client : initiative de la connexion

Mode connecté : modèle client/serveur

- Le serveur attend des demandes de connexion des clients.
- Un client fait une requête et attend la réponse du serveur



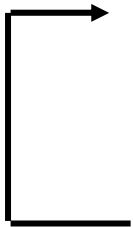
Connexion TCP = (@ serveur, port serveur, @ client, port client)

Mode connecté : principe serveur

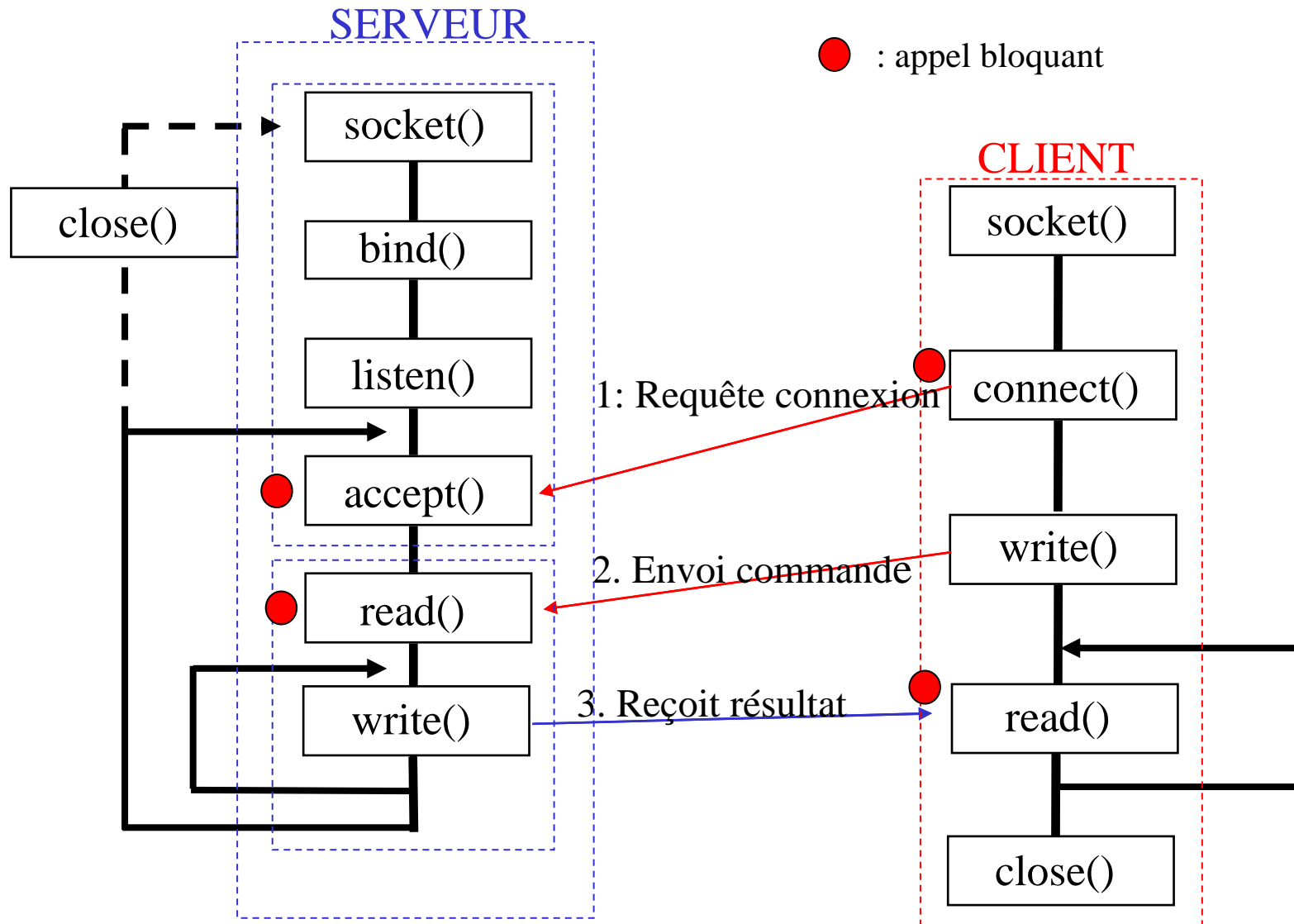
- Crée une socket d'écoute : *socket*
- Associe une @ connue à cette socket : *bind*
- Attend requête connexion d'un client : *listen*
 - Accepte la connexion : *accept*
 - Cela crée une socket de service
 - Dialogue bidirect. client/serveur (proc. service)
 - Fermeture de la socket de communication : *close*

Mode connecté : principe client

- Crée une socket : *socket*
- Connexion au serveur : *connect*
- Dialogue avec serveur
 - écriture/lecture dans la socket : *read/write*
- Fermeture de la socket : *close*



Mode connecté : client/serveur



Mode connecté : serveur listen/accept

```
int listen (  
    int descripteur,  
    int nb_pendantes /*max cnx. pendantes ≤ SOMAXCONN  
                     definit dans <sys/socket.h> */  
)
```

➤ retourne -1 en cas d'échec

```
int accept (  
    int descripteur,  
    struct sockaddr *ptr_adresse, /* ret. @ socket client demandeur */  
    int *ptr_lg_adresse  
)
```

- Retourne un descripteur sur une socket de service connecté à celle du client ou -1 en cas d'échec
- Fonction bloquante par défaut

Mode connecté : client connect

```
int connect (  
    int descripteur, /* sock locale */  
    struct sockaddr *ptr_adresse,  
    int lg_adresse  
)
```

- Appel bloquant tant que connexion non négociée (accept)
- Retourne -1 en cas d'échec ou 0 si
 - Paramètres locaux corrects
 - Existence socket SOCK_STREAM à l'état=listen à ptr_adresse
 - Les 2 sockets sont non connectées (file attente sock dest. non pleine)

Mode connecté : dialogue

```
ssize_t send (  
    int descripteur,  
    void *ptr,  
    size_t nb_caracteres,  
    int option /*MSG_OOB*/  
)
```

```
ssize_t write (  
    int descripteur,  
    void *ptr,  
    size_t nb_caracteres  
)
```

```
ssize_t recv (  
    int descripteur,  
    void *ptr,  
    size_t nb_caracteres,  
    int option /*MSG_PEEK  
                MSG_OOB*/  
)
```

```
ssize_t read (  
    int descripteur,  
    void *ptr,  
    size_t nb_caracteres  
)
```

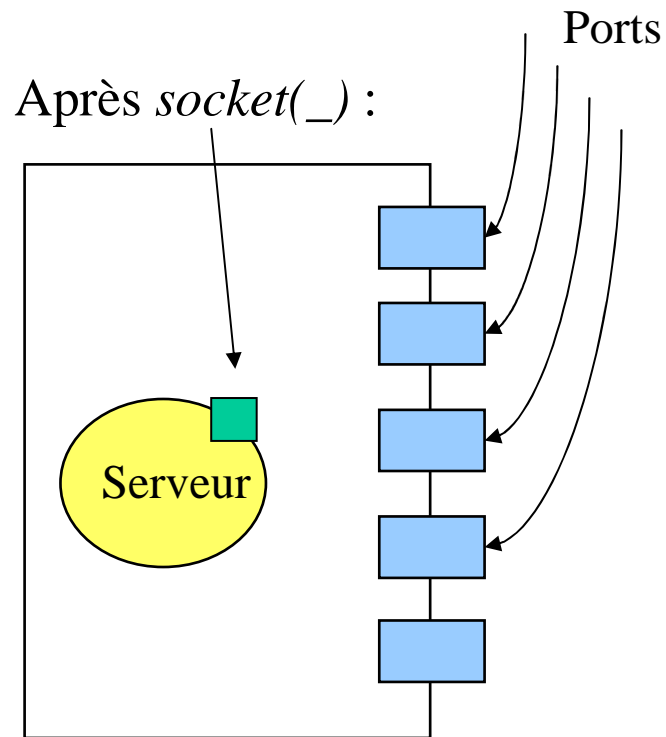
Mode connecté : close/shutdown

- `int close (int descripteur)` : non bloquant par défaut
 - Si tampon non vide : tente d'acheminer les données
 - Possibilité de rendre l'appel bloquant
- ```
#include <sys/ioctl.h>

int shutdown (
 int descripteur,
 int sens /* 0 : pas de lecture, 1: pas d'envoi, 2 : les 2 */
)
```

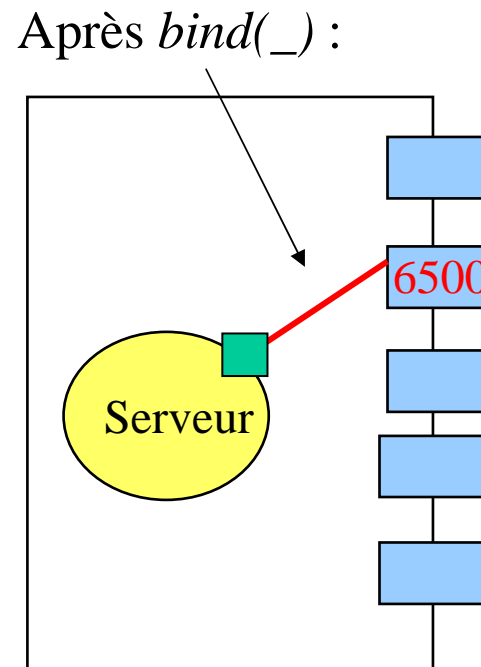
# Mode connecté : exemple (1)

Étape 1: *socket(\_)*



Socket déclarée

Étape 2: *bind(\_)*



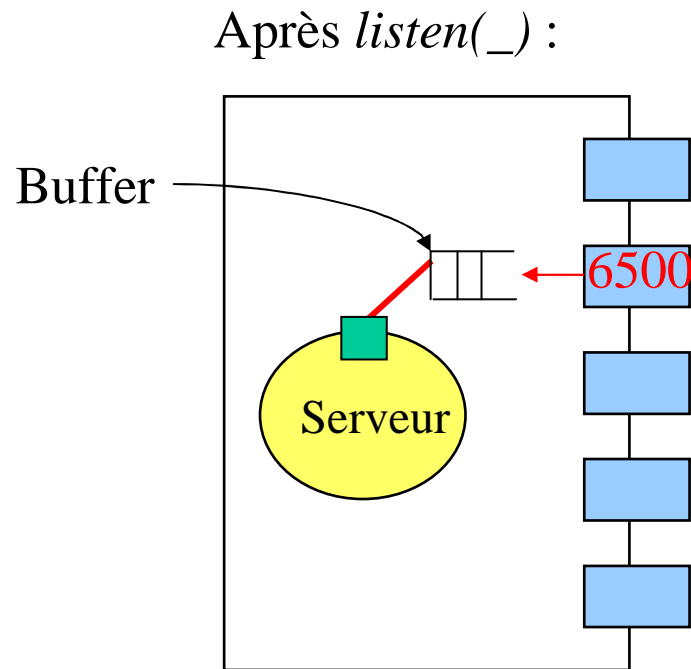
Connecte le processus à  
un port spécifique

Ports libres et réservés

| Application | Port # |
|-------------|--------|
| FTP         | 21     |
| Telnet      | 23     |
| HTTP        | 80     |
| USER        | 5000+  |

# Mode connecté : exemple (2)

Etape 3: *listen*(\_)

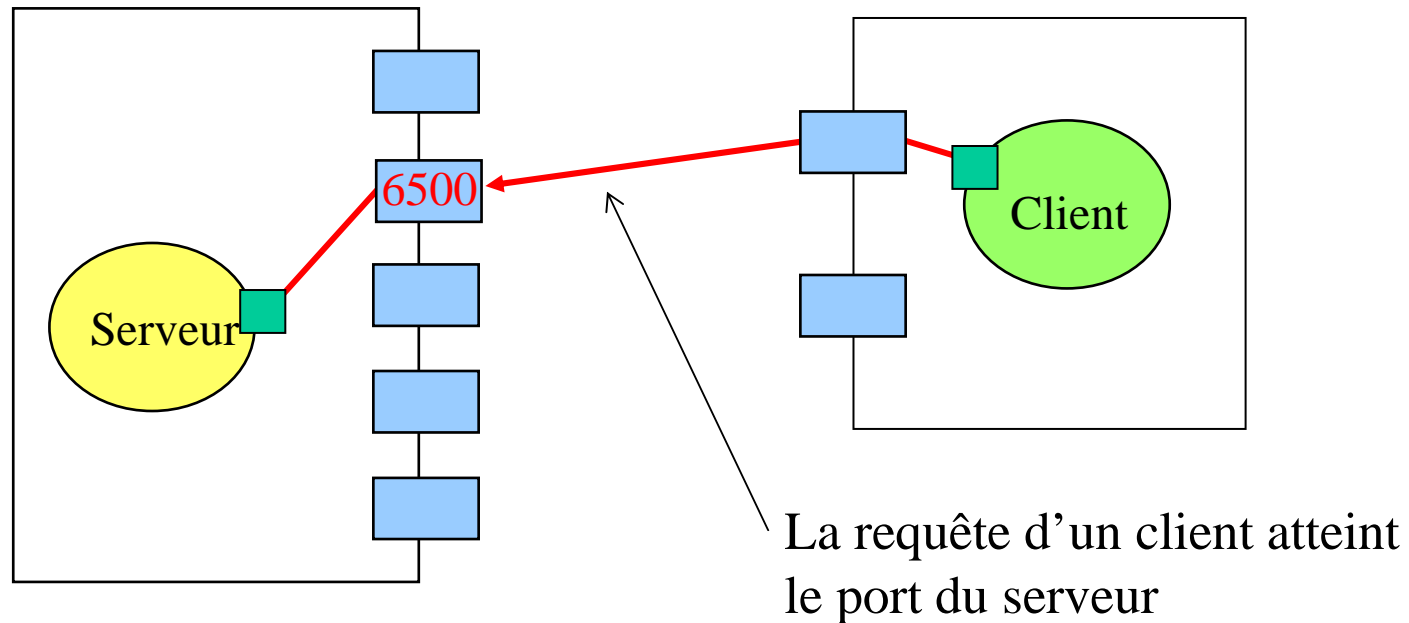


Crée un tampon de réception et attend des connexions

# Mode connecté : exemple (3)

Étape 4 - 1: *accept*(\_)

Après *accept*(\_) :

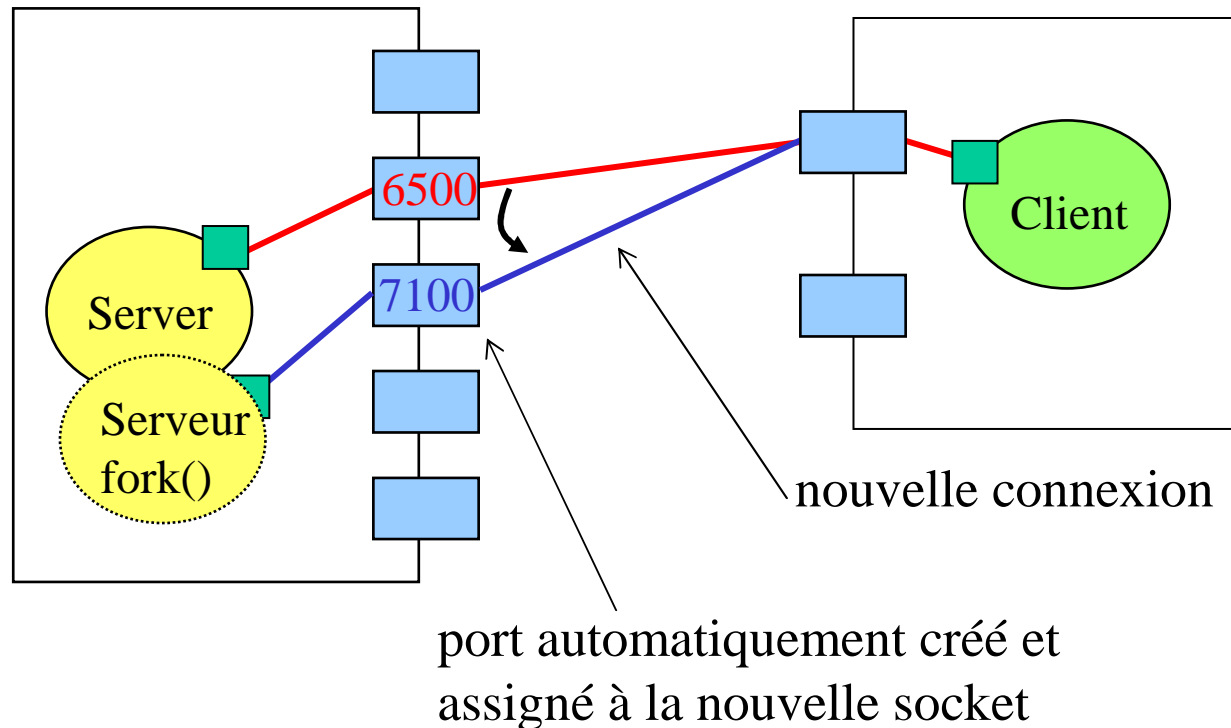


Acceptation d'une connexion par le processus serveur

# Mode connecté : exemple (4)

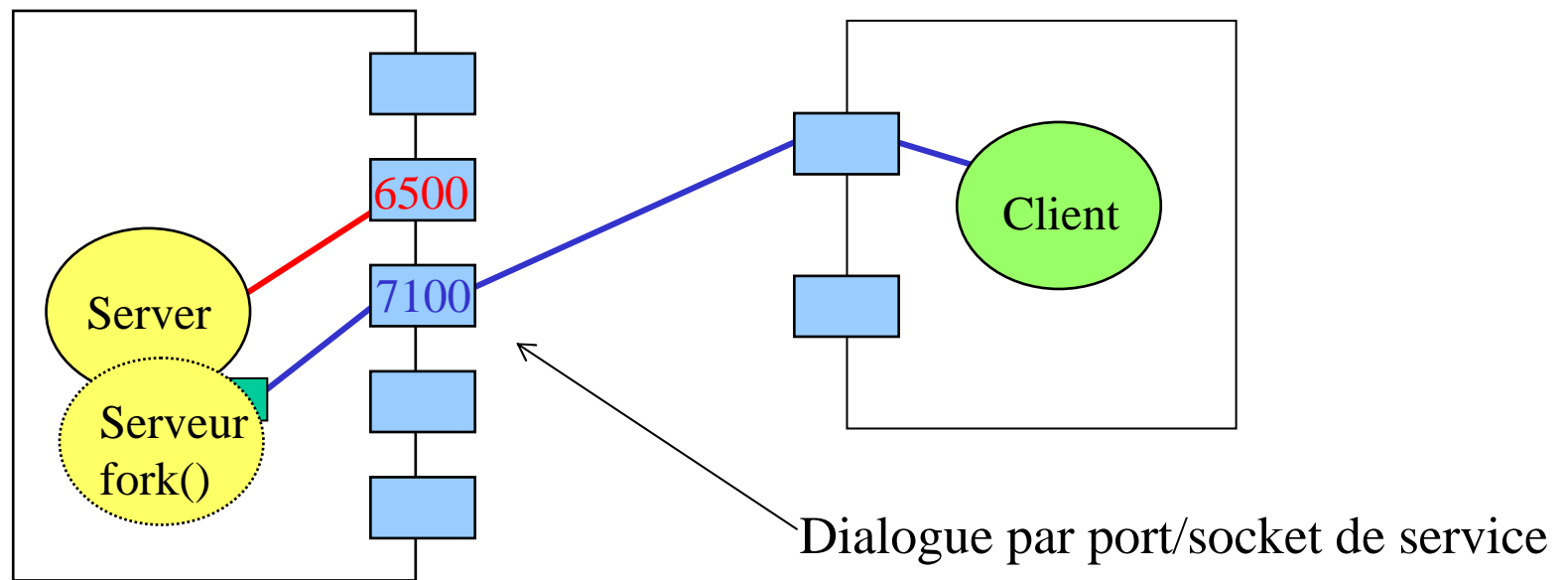
## Etape 4 - 2: *accept( )*

*accept( )* créer une nouvelle socket et l'assigne à un autre port : socket de service



# Mode connecté : exemple (5)

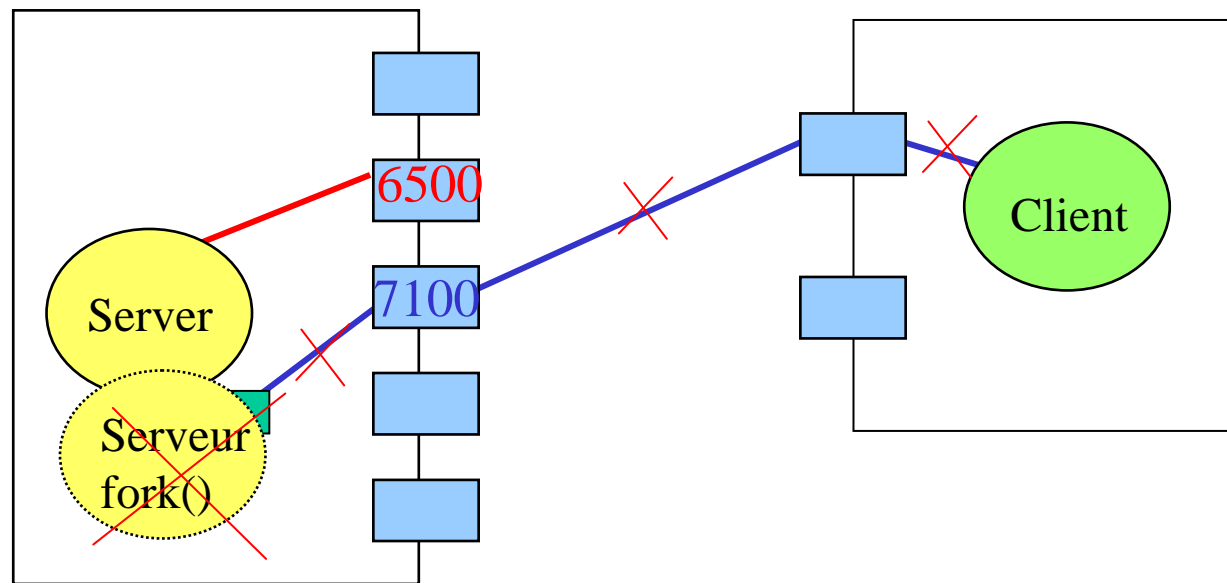
Étape 5: *read( )* et *write( )*



Client et serveur (éventuellement forké) dialogues par la socket créée

# Mode connecté : exemple (6)

Etape 6: *close* ( \_ )



Ferme la socket de service et laisse la socket d'écoute pour un prochain client



# Mode connecté : messages urgents (1)

- Dans certain domaine : « données urgentes »
  - Intérêt : lire ces données sans lire les précédentes
  - AF\_INET : limité à un seul caractère
- Émission : send ( , , option=**MSG\_OOB**)
  - si long. msg. > 1 : seul dernier caractère est urgent
  - Transmis en séquence : si tampon récept. plein...
- Réception : recv ( , , option = **MSG\_OOB**)
  - Carac. non intégré dans tampon : position relative
  - Si nouveau arrive et ancien pas lu : perdu
  - La lecture « bute » sur ces caractères (« séparateurs »)

# Mode connecté : messages urgents (2)

- Possibilité de forcer écriture dans tampon :  
`setsockopt(desc, SOL_SOCKET, SO_OOBINLINE, &un, sizeof(un))`
  - Lecture sans option, mais toujours butée
  - Pour le trouver : lire et tester les caract. un à un :

```
int reponse ;
ioctl(desc, SIOCATMARK, &reponse);
⇒ reponse = 1 si urgent
```
  - Si nouveau carac. urgent écrit : précédent  $\Leftrightarrow$  normal
- Émission du signal SIGURG par demande send
  - Si lecture avant sa transmission : `errno=EWOULDBLOCK`

# Résumé : non connecté

« CLIENT » ou « SERVEUR » :

- Création une socket
- Association d'une adresse socket au service
- Lecture/écriture sur la socket
- Fermeture la socket

# Résumé mode connecté

## CLIENT

- Création socket
- Connexion serveur avec adresse socket distante (@IP et port). Cette connexion attribue automatiquement un nouv. port au client
- Lecture/écriture sur la socket
- Fermeture socket

## SERVEUR

- Création socket « d 'écoute »
- Association @ IP et port
- Écoute des connexions entrantes
- Pour chaque connexion entrante :
  - » accepte la connexion (création d 'une nouvelle socket)
  - » lit/écrit sur la nouvelle socket
  - » ferme la nouvelle socket

# Complément : gestion des erreurs

- Tester et afficher le retour des fonctions avec  
void perror(const char \*s)
- Récupérer les numéros d'erreur :  
variable errno ( #include <errno.h> )

# Complément : gestion des erreurs

- Adresse machine distante (nslookup, /etc/hosts) :

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
struct hostent *gethostbyname(char *nom)
```

```
struct hostent {
 char *h_name; /* nom machine */
 char **h_aliases;
 int h_addrtype; /* AF_INET */
 int h_length; /* long. @ (4) */
 char **h_addr_list; /* les @ */
}
#define h_addr h_addr_list[0]
```

→ @ à pour le bind ou le sendto...

utiliser memcpy(void \*dest, void \*src, size\_t long)

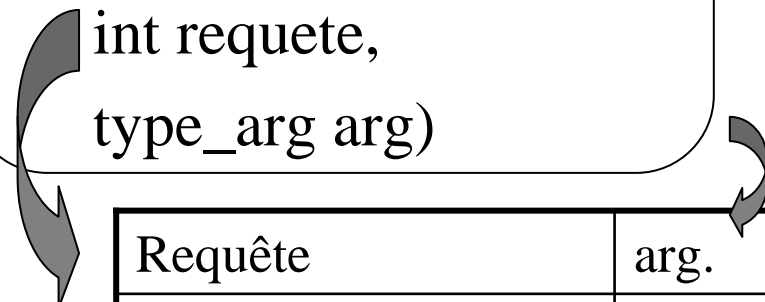
# Compléments : contrôle socket (1)

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

```
int ioctl (
 int desc,
 int requete,
 type_arg arg)
```

Exemple :

```
int on=1;
ioctl(sock, FIOSNBIO, &on);
```



| Requête    | arg.  | effet / valeur retour                                    |
|------------|-------|----------------------------------------------------------|
| FIOSNBIO   | int * | si *arg !=null mode non bloquant                         |
| FIONREAD   |       | nb de caract. lisible                                    |
| FIOASYNC   | int * | si *arg!=null : mode asynchrone (SIGIO)                  |
| FIOSETOWN  | int * | *arg = pid voulu prop. socket                            |
| FIOGETOWN  | int * | *arg = pid propriétaire socket                           |
| SIOCATMARK |       | ret. 1 si pointeur lecture = carac. urgent <sup>55</sup> |

# Compléments : contrôle socket (2)

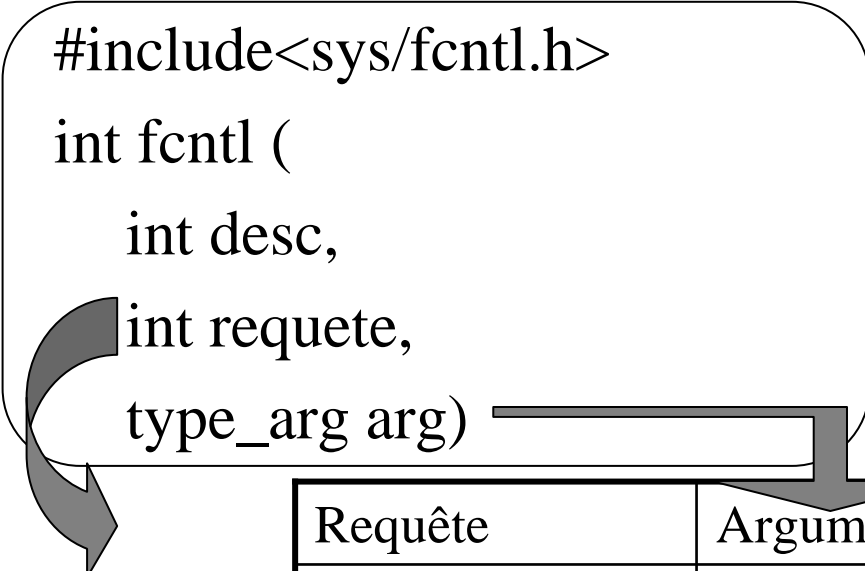
```
#include<sys/fcntl.h>
```

```
int fcntl (
```

```
 int desc,
```

```
 int requete,
```

```
 type_arg arg)
```



| Requête  | Argument   | Effet                       |    |
|----------|------------|-----------------------------|----|
| F_SETFL  | FASYNC     | mode asynchrone (SIGIO)     |    |
| F_SETFL  | FNDELAY    | mode non bloquant (BSD)     |    |
| F_SETFL  | O_NDELAY   | mode non bloquant (Sys V)   |    |
| F_SETFL  | O_NONBLOCK | mode non bloquant Posix     |    |
| F_SETOWN | int        | fixe processus propriétaire |    |
| F_GETOWN |            | retourne proc. propriétaire | 56 |



# Compléments : contrôle socket (3)

- Mode non bloquant : effet sur lecture/écriture
  - Émission, si taille tampon restant < nb. carac :
    - Envoie ce qui est possible, reste = perdu
    - Si aucun caractère écrit :
      - FIOSNBIO : retourne -1 et errno = EWOULDBLOCK
      - O\_NONBLOCK : retourne -1 et errno = EAGAIN
      - O\_NDELAY : retourne 0
  - Lecture, si rien à lire :
    - FIOSNBIO : retourne -1 et errno = EWOULDBLOCK
    - O\_NONBLOCK : retourne -1 et errno = EAGAIN
    - O\_NDELAY : retourne 0

# Compléments : contrôle socket (4)

- Mode non bloquant effets sur *accept*, *connect*
  - Acceptation des connexion *accept* :
    - FIONBIO/O\_NDELAY : ret. -1 et errno= EWOULDBLOCK
    - O\_NONBLOCK : retourne -1 et errno = EAGAIN
  - Demande de connexion *connect* :
    - Retour immédiat mais demande non abandonnée
    - Effet identique dans tous les modes non bloquants :
      - 1er appel : retourne -1 et errno=EINPROGRESS
      - Autres appels avec meme @ dest. : retourne 0 si connexion, ou -1 et errno=EALREADY si en cours ou errno = ETIMEOUT si échec

# Compléments : paramétrage (1)

- Plusieurs niveaux de paramétrage (AF\_INET)
  - Niveau socket : SOL\_SOCKET
  - Niveau protocole : IPPROTO\_IP, IPPROTO\_TCP...
- Deux types d'options
  - booléennes : autorise / interdit une fonction
  - non booléennes : paramétrage taille tampons, type de la socket...

# Compléments : paramétrage (2)


```
#include <sys/socket.h>
int getsockopt(
 int desc,
 int niveau,
 int option,
 void *p_val_option,
 int *p_long_option
)
```

```
#include <sys/socket.h>
int setsockopt(
 int desc,
 int niveau,
 int option,
 void *p_val_option,
 int *p_long_option
)
```

Pour option bool : retourne 0 si option positionnée  
retourne -1 et errno = ENOPRTOOPT  
errno= EINVAL si problème de niveau.

# Compléments : paramétrage (3)

- Options booléennes, niveau **SOL\_SOCKET**
  - **SO\_BROADCAST** : diffusion (si possible physiquement)
  - **SO\_DONTROUTE** : pour **SOCK\_STREAM**, cc. routage (démons)
  - **SO\_KEEPALIVE** : **SOCK\_STREAM** tjrs act. (paire non crashée)
  - **SO\_OOBINLINE** : **SOCK\_STREAM** données OOB placées ds tampon
  - **SO\_REUSEADDR** : autorise bind sur sock déjà « bindée »
  - **TCP\_DELAY** (niveau **IPPROTO\_TCP**) : pas d'attente pour délivrer données
  - **SO\_LINGER** : **SOCK\_STREAM** durée d'attente ack. pour close (mais pas d'info. sur read)



```
struct linger {
 int l_onoff; /* 0 ou 1 */
 int l_linger; /*durée sec. */
}
```

# Compléments : paramétrage (4)

- Options non booléennes, niveau SOL\_SOCKET
  - SO\_TYPE : type (int) de la socket
  - SO\_RCVBUF : pour la lecture, modification taille
  - SO\_SNDBUF des buffers Receive/Send  
(TCP: fenêtre de contrôle de flux, déf. avant cnx.)
- Niveau **IPPROTO\_TCP** :
  - TCP\_MAXSEG : taille max. des segments TCP
  - TCP\_KEEPALIVE : durée quand SO\_KEEPALIVE

# Compléments : signaux

- Indiquer au Kernel de générer signal lié au changements « d'état » d'un descripteur
- Signal SIGIO pris en compte par un handler
  - Déterminer l'évènement d'origine du signal
    - UDP : arrivée de datagramme, erreur
    - TCP : établissement cnx., début discnx., discnx. effective, 1/2 cnx. HS, données envoyées (place dans buffer), réception message, erreur
  - Effectuer le traitement approprié
- Signal SIGURG dans TCP quand données urgentes (OOB)

# Compléments : inetd

- Idée : éviter multiplication des démons
  - Ressources gaspillées
- $\Rightarrow$  serveur unique : inetd
  - Reçoit les requêtes des clients
  - Scrute chacun des ports clients « select »
    - Crée le processus démon serveur correspondant
    - ou Traite lui même la requête (« internal »)
    - socket de l'accept transmise par entrée standard
- Liste des services (démons) dans /etc/inetd.conf



# inetd : /etc/inetd.conf

|         |        |     |        |      |                   |         |
|---------|--------|-----|--------|------|-------------------|---------|
| ftp     | stream | tcp | nowait | root | /usr/lbin/ftpd    | ftpd -l |
| telnet  | stream | tcp | nowait | root | /usr/lbin/telnetd | telnetd |
| #tftp   | dgram  | udp | wait   | root | /usr/lbin/tftpd   | tftpd   |
| login   | stream | tcp | nowait | root | /usr/lbin/rlogind | rlogind |
| shell   | stream | tcp | nowait | root | /usr/lbin/remshd  | remshd  |
| ntalk   | dgram  | udp | wait   | root | /usr/lbin/ntalkd  | ntalkd  |
| daytime | stream | tcp | nowait | root | internal          |         |
| daytime | dgram  | udp | nowait | root | internal          |         |
| time    | stream | tcp | nowait | root | internal          |         |
| echo    | stream | tcp | nowait | root | internal          |         |

|     |        |       |              |        |      |            |
|-----|--------|-------|--------------|--------|------|------------|
| ↑   | ↑      | ↑     | ↑            | ↑      | ↑    | ↑          |
| NOM | TYPE   | PROTO |              | PROPRI |      | PARAMETRES |
|     | SOCKET |       | wait = un    |        | PATH |            |
|     |        |       | seul serveur |        |      |            |