INTRODUCTION à la communication IPC Internet

Eric Gressier-Soudan

CNAM-CEDRIC

REFERENCES BIBLIOGRAPHIQUES

RESEAUX Architecture, Protocoles, Applications. Andrew Tanenbaum. InterEditions 1989.

Open Networking with OSI. Adrian Tang, Sophia Scoggins. Prentice Hall 1992.

Unix Network Programming. W. Richard Stevens. Prentice Hall Software Series.1990.

TCP/IP illustrated, Volume 1 : The protocols. W.R. Stevens. Addison Weslay. 1994.

Internetworking with TCP/IP Volume I : Principles, Protocols and Architecture. Douglas E. Comer. Prentice-Hall 1993.

Internetworking with TCP/IP Volume III : Client-Server Programming and Applications, BSD socket version. Douglas E. Comer, David L. Stevens. Prentice-Hall 1993.

The Magic Garden Explained. The Internals of Unix system V release 4. An open systems design. B. Goodheart and J. Cox. Prentice Hall. 1994.

PLAN

1. INTRODUCTION

Internet/Unix - Applications Offertes

--> Transfert de fichiers : "**ftp**", "**tftp**"

--> Terminal virtuel : "**telnet**", "**rlogin**"

--> Exécution à distance :"**rsh**", "**rcp**"

--> Serveur de noms : **Bind**

--> Gestion de Réseaux : **SNMP**

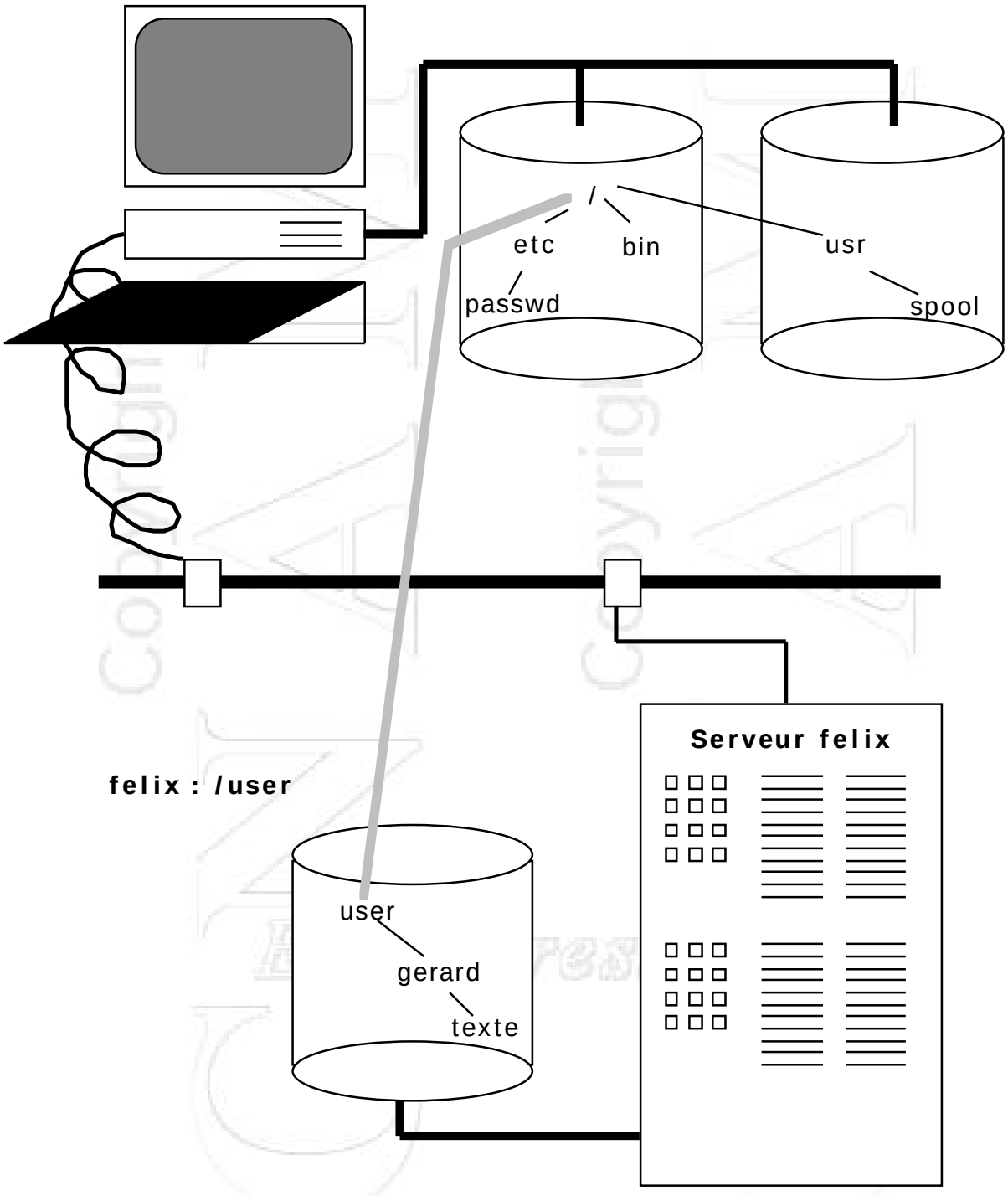
--> Messageries : **SMTP**

--> Systèmes d'Information : **WWW**

--> Partage de fichiers : données, sources, binaires, librairies, utilitaires ...

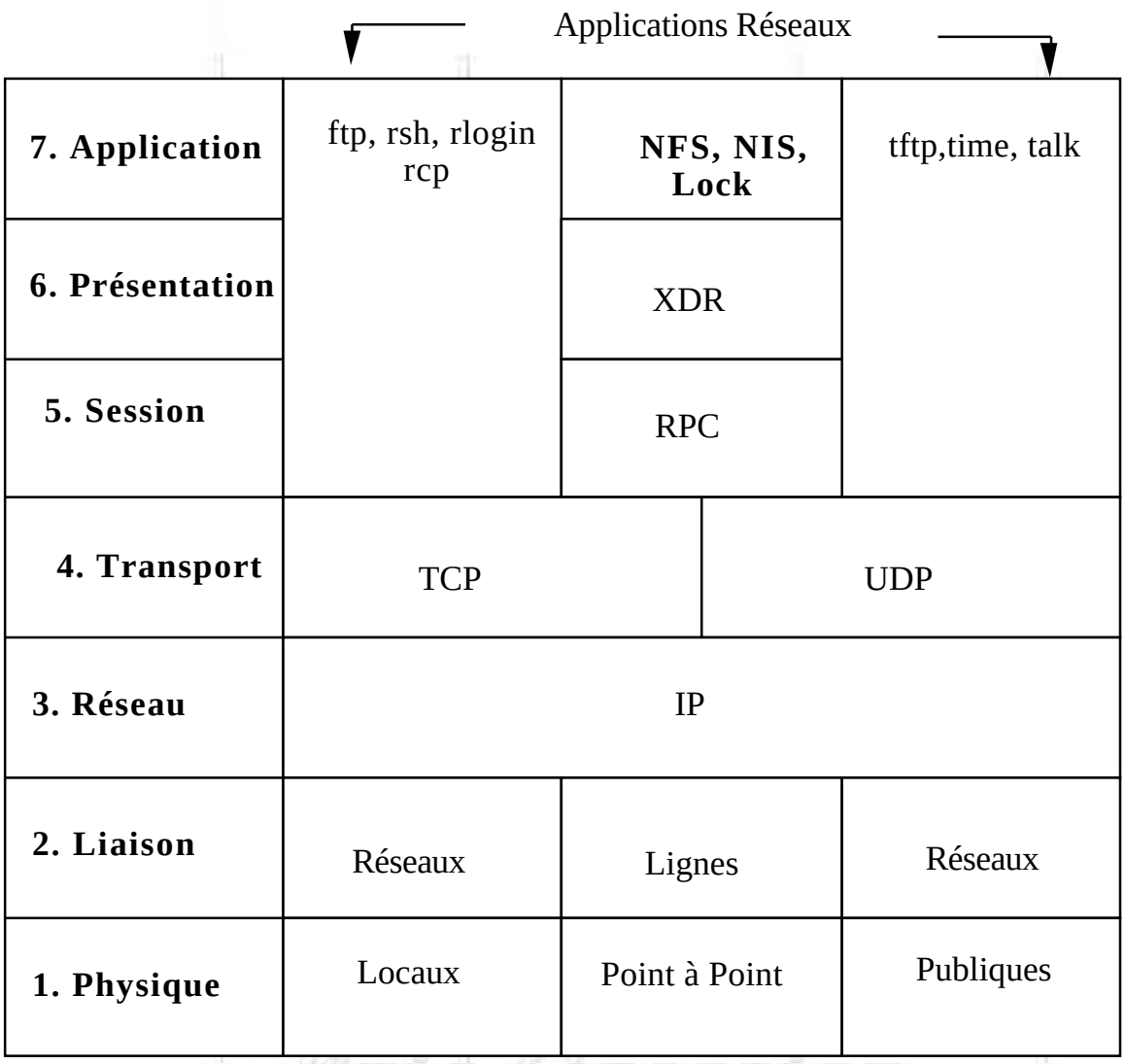
Network File System NFS

NFS


/
etc
bin
usr
passwd
spool
Serveur felix
felix : /user
user
gerard
texte

2. Rappels

Environnement Internet et modèle ISO



TYPOLOGIE DES RESEAUX LOCAUX

Méthodes d'accès :

compétition v.s. **coopération**

! !

V V

Ethernet (802.3) Jeton

Bus (802.4)

Anneau (802.5)

Principale Propriété : Support à diffusion

-> adresses physiques de diffusion

Une adresse = 6 octets

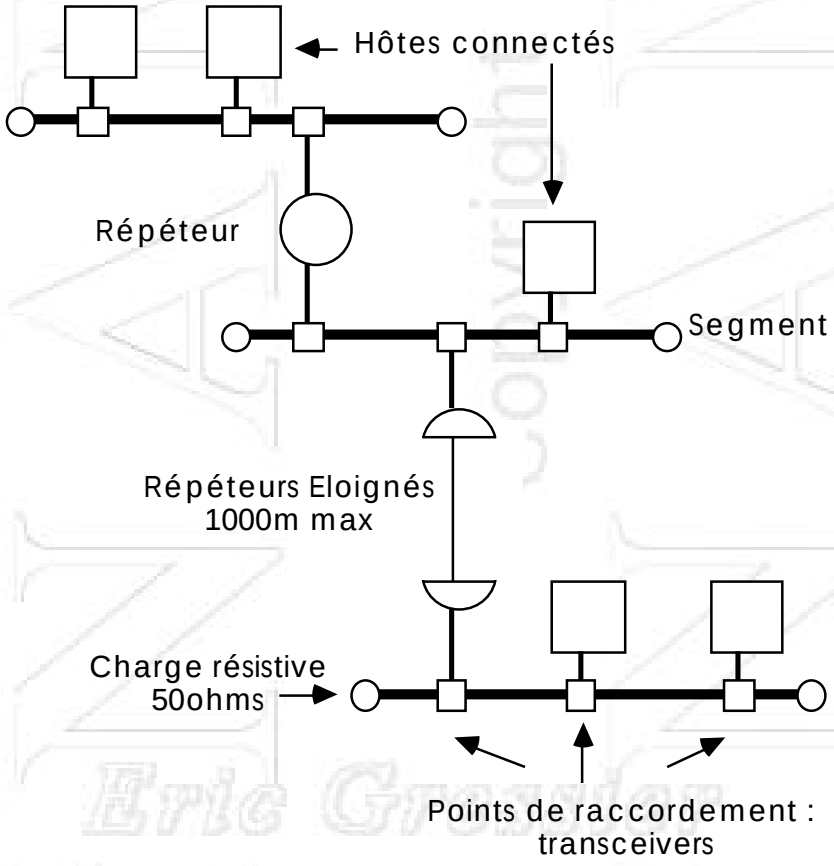
En particulier, **adresse de Destination** :

- **"Broadcast"** : Diffusion à tous les sites
- **"Multicast"** : Groupe de Diffusion

- **"Unicast"** : Adresse d'un seul site

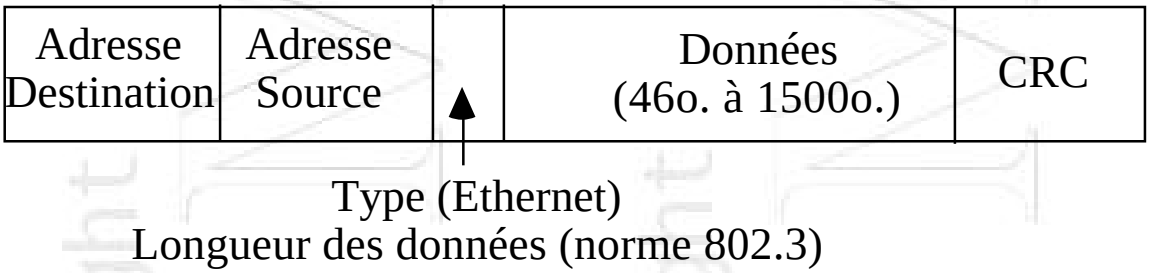
ETHERNET : Caractéristiques Générales

Bus arborescent à **10Mb/s**, avec un protocole fondé sur : écoute de porteuse, détection de collisions, ajournement persistant de la tentative de transmission quand que le canal est occupé (stratégie CSMA/CD 1-persistant)



Ethernet : Structure d'une trame

Structure de la trame Ethernet:



=>Adresse Physique de Destination : 6 octets avec pour l'adresse destinataire des possibilités de :

- "Broadcast" : Diffusion à tous les sites
- "Multicast" : Groupe de Diffusion

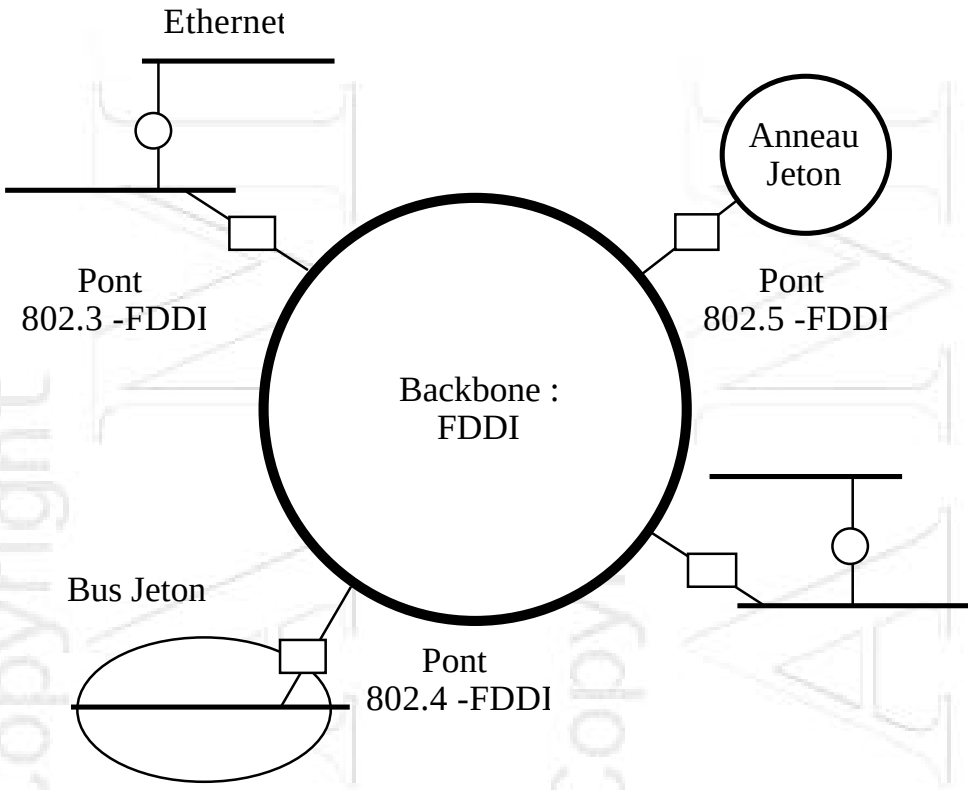
=>Adresse Physique Source : 6 octets

=>Un champ type : 2 octets, qui peut aussi contenir la longueur des données de la trame

=>Taille des Données de 46 (remplissage minimum) à 1500 octets de données

=>Un code CRC de 4 octets pour la détection d'erreurs

Combinaison de Réseaux Locaux :



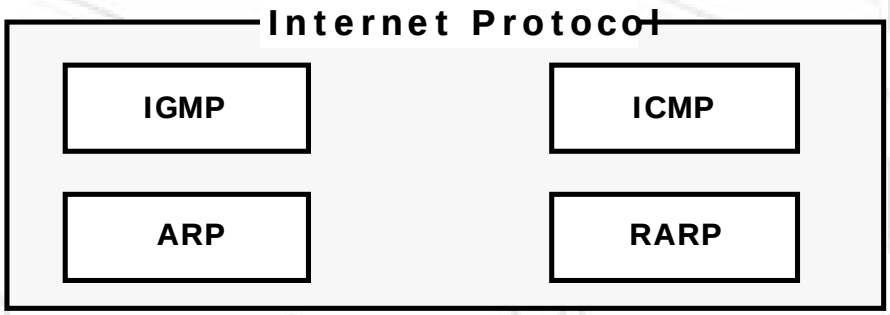
Les Adresses Physiques sur les réseaux locaux (48 bits) sont homogènes :

Les différents types de LAN peuvent être reliés par des Ponts.

Attention : la taille maximale des données suivant les différentes normes n'est pas homogène, le minimum est 1500 octets pour Ethernet.

3. INTERCONNEXION DE RESEAUX : IP

Couche Réseau IP



L' Internet Protocol (IP)

* Adressage Internet

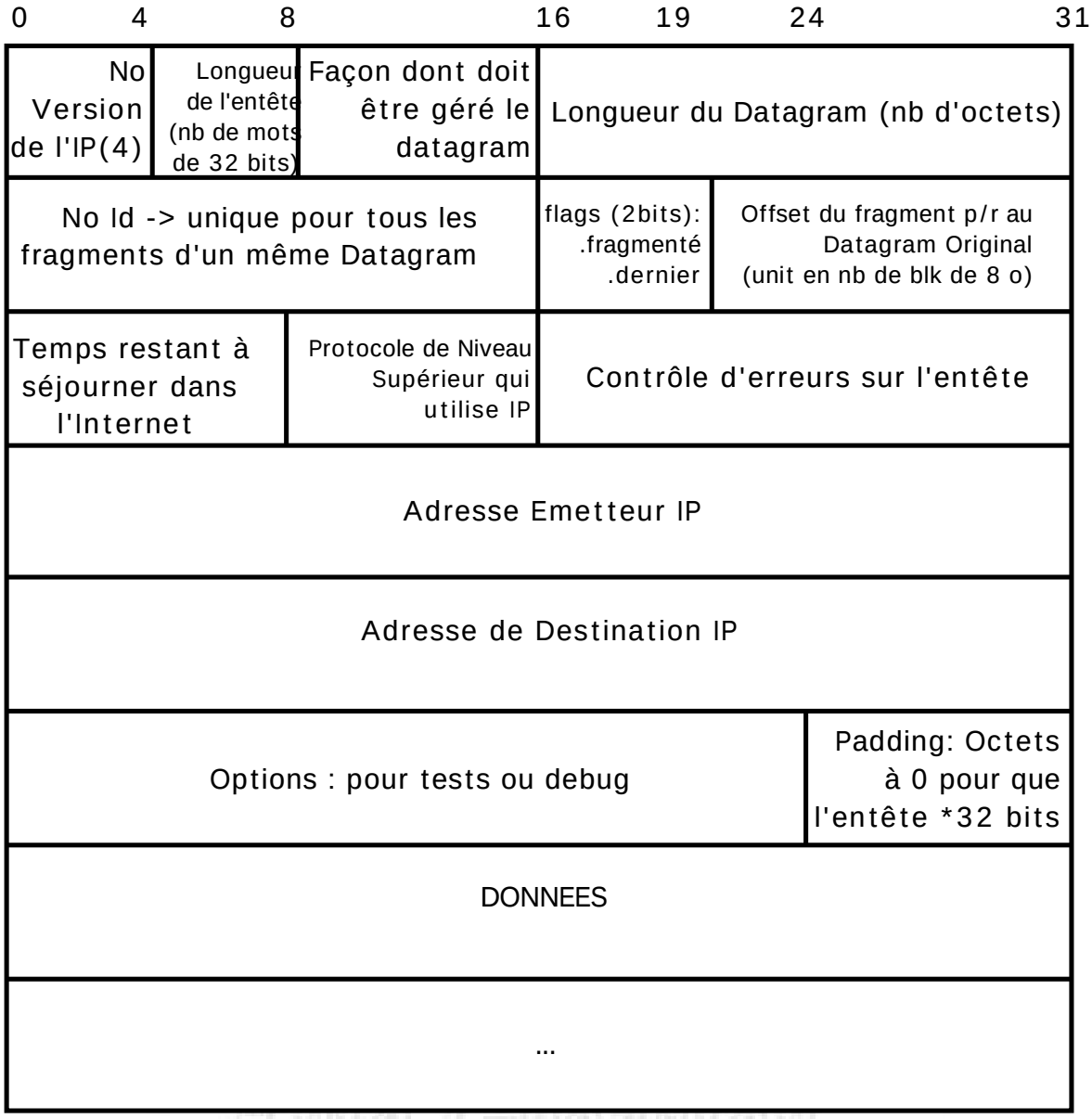
* Conversions d'Adresses(192.200.3.45<->08:00:20:06:4b:8e)

* Routage entre Réseaux

* Fragmentation/Réassemblage, Adaptation de la taille des messages soumis par la couche Transport suivant les possibilités offertes par la couche Liaison.

* Communications dans le mode minimal :
DATAGRAM (mode non connecté), Envois de paquets **sans Acquittements**
=> la détection des messages erronés ou perdus et leurs réémissions sont à la charge de l'émetteur des messages (couche Transport).

* Multiplexage/Démultiplexage par rapport à la couche Transport


0
4
8
16
19
24
31
No Version de l'IP(4)
Longueur de l'entête (nb de mots de 32 bits)
Façon dont doit être géré le datagram
Longueur du Datagram (nb d'octets)
No Id -> unique pour tous les fragments d'un même Datagram
flags (2bits): .fragmenté .dernier
Offset du fragment p/r au Datagram Original (unit en nb de blk de 8 o)
Temps restant à séjourner dans l'Internet
Protocole de Niveau Supérieur qui utilise IP
Contrôle d'erreurs sur l'entête
Adresse Emetteur IP
Adresse de Destination IP
Options : pour tests ou debug
Padding: Octets à 0 pour que l'entête *32 bits
DONNEES
...

Adressage Internet (1)

Adressage uniforme de sous-réseaux

Adresses Physiques de +sieurs types
->Adresses LAN
->Adresses WAN

l'Internet définit des **Adresses Uniques Universelles**

Notation Pointée sur 4 champs:

T . U . V . W

(N°Réseau, N°station)

l'Adresse IP est sur **32 bits- 4 octets**

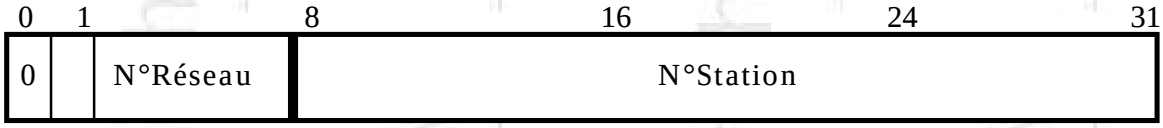
192.200.25.1

Adressage Internet (2)

=> 3 Classes d'Adresses :

Classe A :

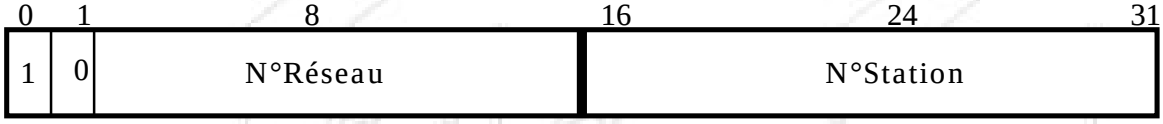
Peu de Réseaux, de nombreuses Stations par Réseau



N°de Réseau : 1 - 126

127 désigne l'adresse locale pour le **rebouclage**

Classe B :



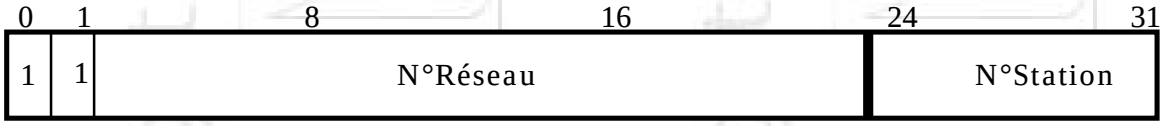
N°de Réseau : 128.1 - 191.254

Adressage Internet (3)

Classe C :

Beaucoup de Réseaux, Peu de Stations par Réseau

La classe la plus répandue



N°de Réseau : 192.0.1 - 223.255.254
N°de Station : 1 - 254
Broadcast : **255** dans le champ **N° de Station**

Pour chaque classe :

L'**adresse du réseau** est désignée avec **0 dans le champ N° de station**.

Adressage Internet (4)

Les Adresses IP sont répertoriées sur un site dans le fichier **/etc/hosts** ou gérées dans la base des **"hosts"** équivalente des Yellow Pages ou par un serveur de nom BIND par nslookup().

Exemple de fichier /etc/hosts :

```
#
128.138.240.1      boulder       boulder.colorado.edu
128.138.243.10     boulder-gw    boulder-gw.colorado.edu
128.138.240.26     tigger        tigger.colorado.edu
#
```

Exemple de fichier /etc/networks :

```
#
loopback-net   127.0.0    Software loopback net
cu-netmask     255.255.255    Netmask
#
colorado  128.138  University of Colorado
cu-engineer     128.138.240   Engineering Center Backbone
cu-boulder      128.138.238   Campus Backbone
#
```

Format des Adresses IP et des entêtes de messages (1)

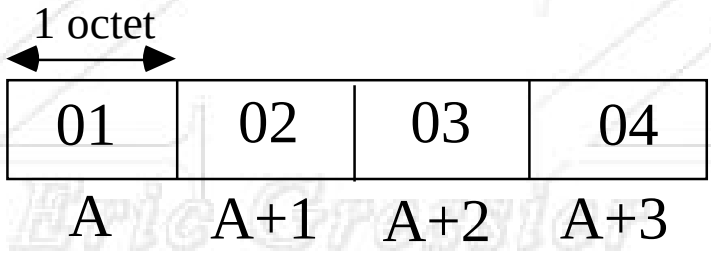
Dans quel ordre doivent être émis les octets dans le réseau pour que toutes les machines puissent comprendre les entêtes de messages IP !!!???

Représentations des données dans les machines :

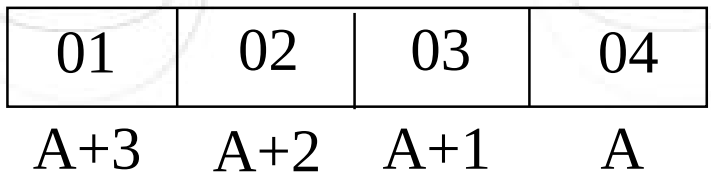
Big Endian vs **Little Endian**

Exemple : nombre 0x01020304

Big Endian : on numérote les octets de gauche à droite



little endian : on numérote les octets de droite à gauche



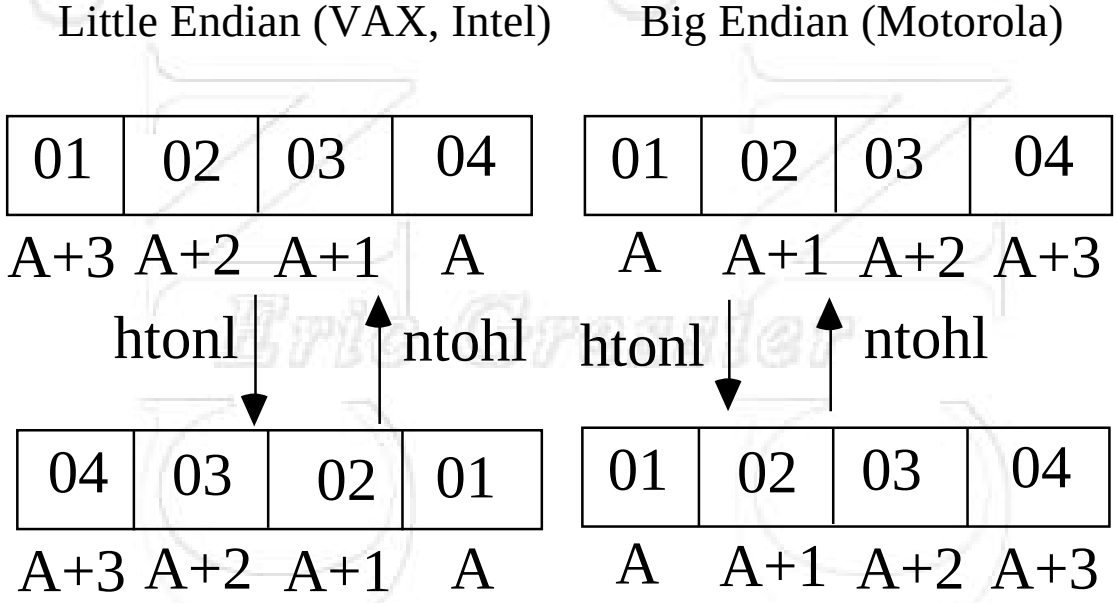
Format des Adresses IP et des entêtes de messages (2)

Solution Internet pour le format des entêtes **:**

Big Endian

Par ailleurs, on envoie l'octet le + significatif d'abord.

On effectue de la conversion au format réseau :



Format Réseau

Format des Adresses IP et des entêtes de messages (3)

- Primitives de conversion d'entiers du format "**host**" au format "**network**" :
  htonl() long - entier 32 bits **,**
  htons() court - entier 16 bits

- Primitives de conversion d'entiers du format "**network**"au format "**host**" :
  ntohl(),
  ntohs()

Ce problème est résolu pour les données applicatives par **XDR** avec ONC, **NDR** avec NCA-OSF/DCE, ou par **ASN1-BER** avec la couche Présentation de l'ISO.

Représentation d'une adresse internet

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

struct in_addr {
   ulong  s_addr
};
```

entier long de 32 bits en représentation réseau

Obtenir l'adresse d'une machine (1)

gethostbyname

nom de la machine à atteindre connu -> **adresse IP ?**

```
struct hostent
        *gethostbyname
                (char *hostname)
```

retourne une structure de données décrite dans <netdb.h>, recherche : locale, par service NIS/Hesiod, ou Serveur BIND

```
#include <netdb.h>

struct hostent {
   char *h_name
   char ** h_aliases
   int  h_addrtype/* AF_INET */
   int  h_length/*long. adresse*/
   char **h_addr_list
};
```

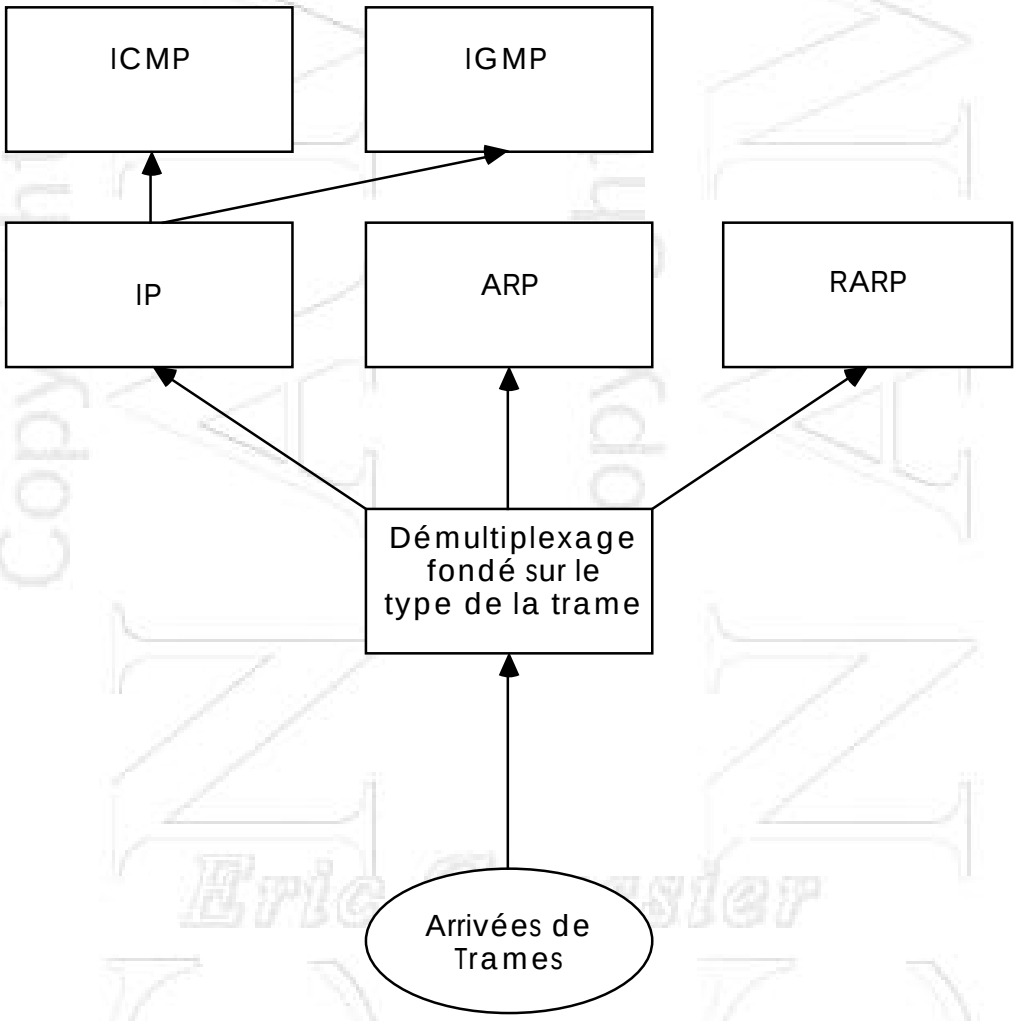
le dernier champ est une liste d'adresses IP au format `in_addr` terminée par NULL :

autre fonction du même type : gethostbyaddress()

Obtenir l'adresse d'une machine (2)

```
/*
 * Print the "hostent" information for every host whose name is
 * specified on the command line.
 */

#include    <stdio.h>
#include    <sys/types.h>
#include    <netdb.h>         /* for struct hostent */
#include    <sys/socket.h>    /* for AF_INET */
#include    <netinet/in.h>    /* for struct in_addr */
#include    <arpa/inet.h>     /* for inet_ntoa() */

main(argc, argv)
int   argc;
char  **argv;
{
      register char               *ptr;
      register struct hostent *hostptr;

      while (--argc > 0) {
            ptr = *++argv;
            if ( (hostptr = gethostbyname(ptr)) == NULL) {
                  printf("gethostbyname error on host: %s %s\n",ptr);
                  continue;
            }
            printf("official host name: %s\n", hostptr->h_name);

            while ( (ptr = *(hostptr->h_aliases)) != NULL) {
                  printf("    alias: %s\n", ptr);
                  hostptr->h_aliases++;
            }
            printf("    addr type = %d, addr length = %d\n",
                        hostptr->h_addrtype, hostptr->h_length);

            switch (hostptr->h_addrtype) {

            case AF_INET:
                  pr_inet(hostptr->h_addr_list, hostptr->h_length);
                  break;
            default:
                  printf("unknown address type");
                  break;
            }
      }
}

/* imprime une liste d'adresses IP */
pr_inet(listptr, length)
char  **listptr;
int   length;
{
      struct in_addr    *ptr;

      while ( (ptr = (struct in_addr *) *listptr++) != NULL)
            printf("    Internet address: %s\n", inet_ntoa(*ptr));
}
```

Enchainement des différentes parties de la Couche IP



Enchainement des différentes parties de la Couche IP - ARP

Address Résolution Protocol : ARP

Savoir traduire des Adresses :

Adresses IP<=> **Adresses de Liaison** (ex : Ethernet)

=> 2 techniques :

* **Statique** : quand on peut modifier les adresses physiques sur les stations raccordées (ex PROTEON).

* **Dynamique** : quand le réseau est souvent reconfiguré (Arrêt / Marche de type Réseau de Stations de Travail).

=> ARP

Fonctionnement :

A ($@I_A$, P_A) doit dialoguer avec B ($@I_B$, P_B)

A doit connaître l'adresse physique P_B (Résolution de l'adresse Internet@ I_B) :

=> Broadcast avec un paquet spécial contenant $@I_B$,

=> B répond seul en envoyant P_B.

Mécanisme de Cache : Il existe une table qui conserve les traductions d'adresses.

Améliorations :

- B mémorise l'adresse physique de A,
- Tous les hôtes mémorisent l'adresse physique de A (à cause du broadcast),
- Dès la mise en marche une station "broadcast" son adresse IP

Exemple du fonctionnement du cache arp sur une station Sun :

Résultat de la commande **arp -a** :

```
argos          (192.200.000.55)   at 8:0:55:6:6e:6a
paris          (192.200.000.56)   at 8:0:59:6:bf:0a
andromaque(192.200.000.50)   at 8:0:a5:6:6e:c0
```

on execute la commande "**ping penelope**", station du réseau mais qui ne figure pas dans le cache arp, on obtient alors :

```
argos          (192.200.000.55)   at 8:0:55:6:6e:6a
paris          (192.200.000.56)   at 8:0:59:6:bf:0a
penelope       (192.200.000.25)   at 8:0:25:4:00:ab
```

Entête de Trame	Message ARP considéré comme des Données

Format des messages ARP/RARP utilisés pour la résolution des Adresses Internet-vers-Ethernet :

0	8	16	31
Type de l'interface Physique pour laquelle l'adresse Liaison est recherchée (ex: 1 pour Ethernet)		Protocole considéré	
Longueur d'une Adresse pour l'interface	Longueur d'une Adresse pour le Protocole	Requête ARP/RARP, ou Réponse ARP/RARP	
Adresse Hardware de l'Emetteur (octets 0-3)			
Adresse Hardware de l'Emetteur (octets 4-5)		Adresse Internet de l'Emetteur (octets 0-1)	
Adresse Internet de l'Emetteur (octets 2-3)		Adresse Hardware de la Cible (octets 0-1)	
Adresse Hardware de la Cible (octets 2-5)			
Adresse Internet de la Cible (octets 0-3)			

Enchainement des différentes parties de la Couche IP - ICMP

L'Internet Control Message Protocol: ICMP

Echange de Messages de Service entre stations qui utilisent l'Internet Protocol :
* Utilise l'Internet Protocol comme le niveau Transport le fait,
* Inclu dans le niveau Réseau.

=> Permet de Tester si une station fonctionne correctement :

ECHO_REQUEST

la station appelée répond :

ECHO_REPLY

Commande : **ping** “adresse IP” ou “nom de station”
Réponse : **"nom de station" is alive** si tout va bien

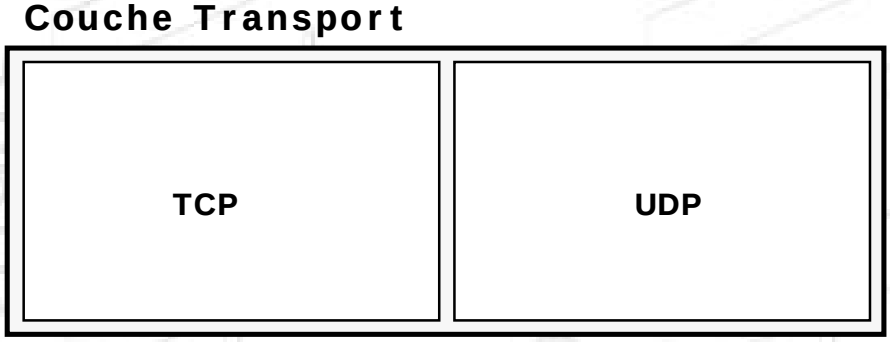
Très utile pour tester si la couche IP d'une station est active et détecter qu'ainsi la station cible est surchargée.

autres fonctions d'ICMP liées aux fonctions de passerelles et mécanismes de routage de l'Internet Protocol.

4. COUCHE TRANSPORT : TCP & UDP

La Couche Transport : TCP-UDP

Couche chargée de l'acheminement des informations de bout en bout dans l'Internet.



Cette couche ne manipule pas des Adresses de sites IP mais des "**N° de Port**" pour atteindre un service distant[1].

Le N° de port distingue un service parmi l'ensemble des services accessibles à travers la couche Transport sur une machine distante.

[1] En fait, la désignation d'un processus unix qui réalise le service distant se fait à l'aide de la notion d'extrêmité. Hors, une extrêmité est un N° de port + une adresse IP.

Obtention des N°ports utilisés par service

getservbyname

nom d'un service sur la machine à atteindre connu ->**N°Port ?**

```
struct servtent
     *getservbyname
          (char *servtname,
            char  *protname)
```

retourne une structure de donnée décrite dans <netdb.h>

```
#include <netdb.h>

struct servent {
   char *s_name
   char ** s_aliases
   int  s_port  /*entier 16 bit*/
   char *s_proto
};
```

voir **/etc/services** pour les n°Ports déjà affectés (fichier 4.xBSD uniquement)

User Datagram Protocol - UDP

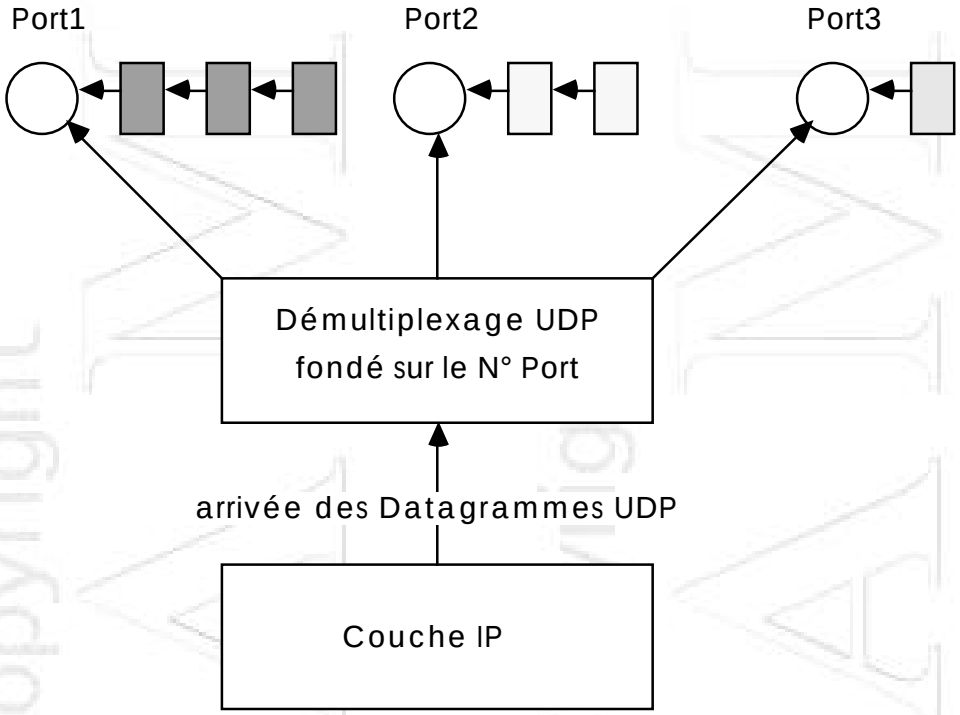
Protocole de Transport :

- sans connexion,
- sans acquits,
- ne conserve pas l'ordre des messages,
- sans contrôle de flux,
-multiplexage/démultiplexage p/r aux ≠ N°Port
- préserve la notion d'enregistrement.

=> possibilités de :
- pertes de messages,
- duplication des messages,
- déséquencement,
- émetteur trop rapide p/r au récepteur

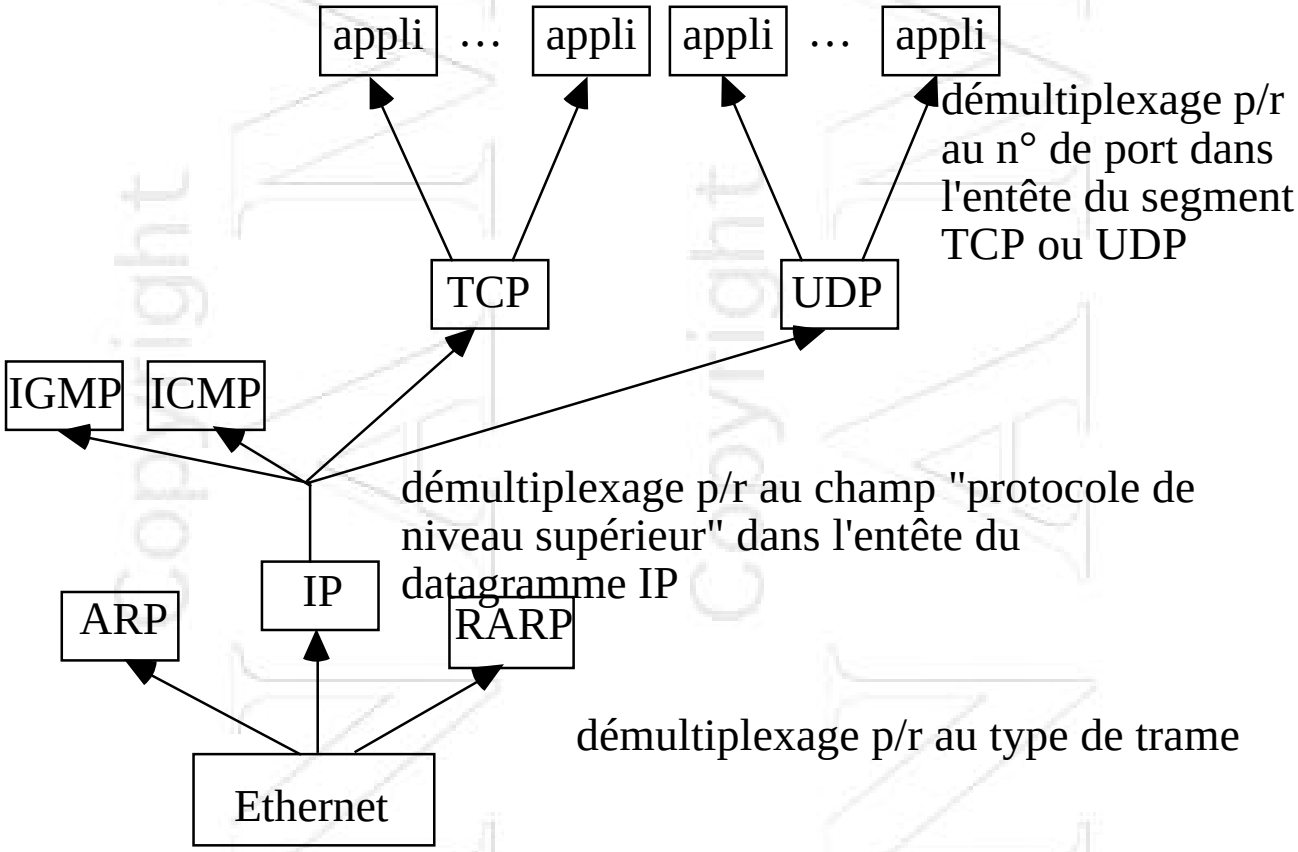
Protocole rapide,
Fiabilité suffisante sur un réseau local
avec des sites
pas trop chargé.

Démultiplexage UDP



Si la **file** des messages en attente derrière le port est **pleine**, le **message** reçu est écarté donc **perdu**.

Démultiplexages en couches dans l'Internet



Transmission Control Protocol - TCP (1)

- Orienté Flot d'octets
  - ne préserve pas la notion d'enregistrement,
  - séquencement des octets garanti.

- Mécanisme de Circuit Virtuel : notion de connexion,en Full-Duplex
  - Acquits Positifs avec Retransmissions en cas d'erreurs,
  - Contrôle de flux
  - Pas de duplications des messages possibles,
  - Données urgentes,
  - Informé des ruptures de connexions.

- Controle d'Erreurs,

- Assemblage/Déassemblage,

- Multiplexage/Démultiplexage.

Transmission Control Protocol - TCP (2)

TCP n'a pas de mécanisme de type pour maintenir l'activité de la connexion quand aucun message n'est échangé entre les deux extrêmités.

pas d'IDLE_MESSAGE comme dans OSI-TP4

on ne peut pas savoir si l'autre extrêmité est toujours présente ...on le sait qu'au moment où on émet

Toutefois, il est possible de le gérer par l'interface socket par configuration à l'aide de la primitive setsocketoption() et l'option SO_KEEPALIVE.

Transmission Control Protocol - TCP (3)

Pour relier deux entités communicantes **TCP** ajoute la notion d'extrêmité :

< adresse IP, N°Port>

Une extêmité peut être vue comme une référence locale d'une connexion.

- une paire d'extrêmités définit une connexion,

- **Attention :** une même extrêmité peut servir deux connexions qui mettent en jeu des machines différentes :

(CMU) <128.2.254.139, 1184> dialogue avec **<128.10.2.3, 53> (Purdue)**

(Purdue-ISI) <128.9.000.32, 1184> dialogue avec **<128.10.2.3, 53> (Purdue)**

En fait, pour des raisons de commodité, la notion d'extrêmité s'étend au protocole UDP même si aucune connexion n'est mise en oeuvre avec ce protocole.

Format d'un paquet TCP

<table>
<tr><td colspan="3">No de Port Source</td><td>No de Port Destinataire</td></tr>
<tr><td colspan="4">No de Séquence (n° d'octet dans le flot)</td></tr>
<tr><td colspan="4">No d'Acquittement (n° d'octet dans le flot)</td></tr>
<tr><td>HLEN (x32b) longueur de l'entête</td><td>Réservé</td><td>Masque pour le type du paquet (g à d) : URG, ACK, PSH, RST, SYN, FIN</td><td>Taille du buffer en Réception du site émetteur</td></tr>
<tr><td colspan="3">Contrôle d'erreurs (calcul idem datagram)</td><td>Position dans le flot d'octet où se terminent les données express (avec bit URG)</td></tr>
<tr><td colspan="3">Options (optionnel -> longueur variable de l'entête d'où le champ HLEN)</td><td>Padding</td></tr>
<tr><td colspan="4">Données</td></tr>
<tr><td colspan="4">...</td></tr>
</table>

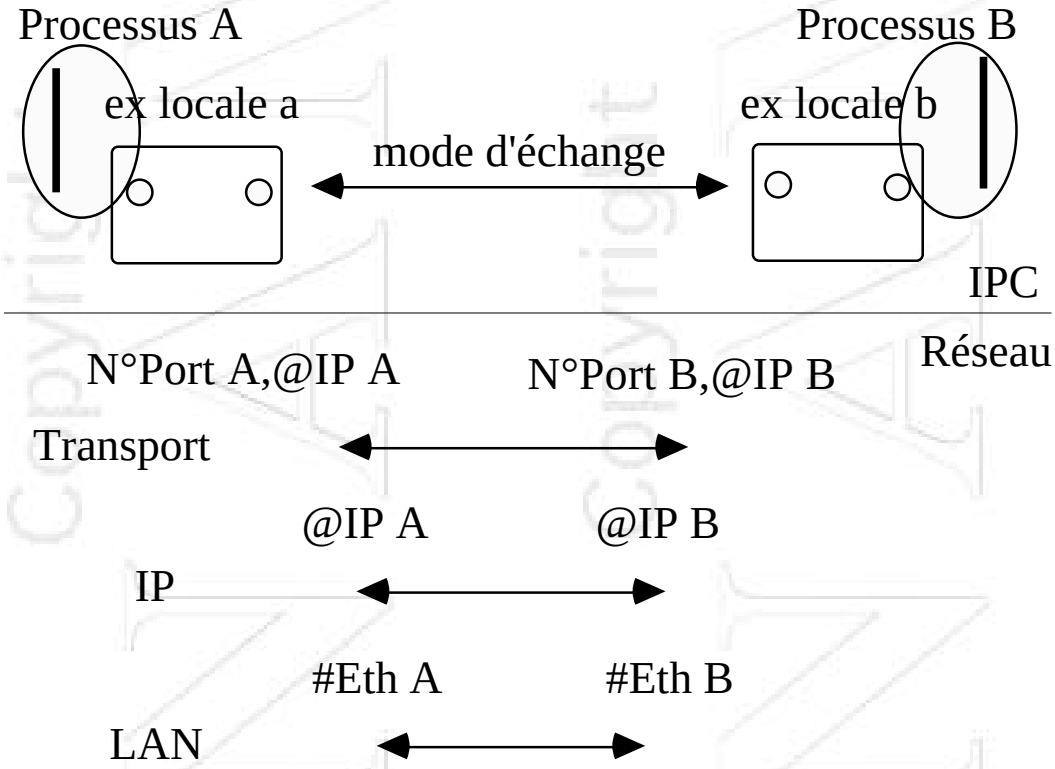
5. API SOCKET

Communication inter-processus (IPC)

Association (Dialogue) IPC:

Local (un Client) Distant (un Serveur)

{processus,extrêmité,mode d'échange,extrêmité,processus}



Pour un participant donné, une association peut n'être que partiellement renseignée, c'est à dire toutes les informations liées à l'association ne sont pas encore définies à un instant donné.

!?! A tout moment, chaque processus a une vue locale de l'association en cours d'utilisation … cette vue peut donc être partielle.

Pas d'état global en univers réparti !

API socket - relation Client/Serveur

API valable pour des échanges de données locaux ou à travers un réseau indépendamment du protocole de communication utilisé.

Client **Serveur**

création de socket :

primitive socket

ouverture de dialogue :

primitive bind

primitive connect ***primitive listen***

primitive accept

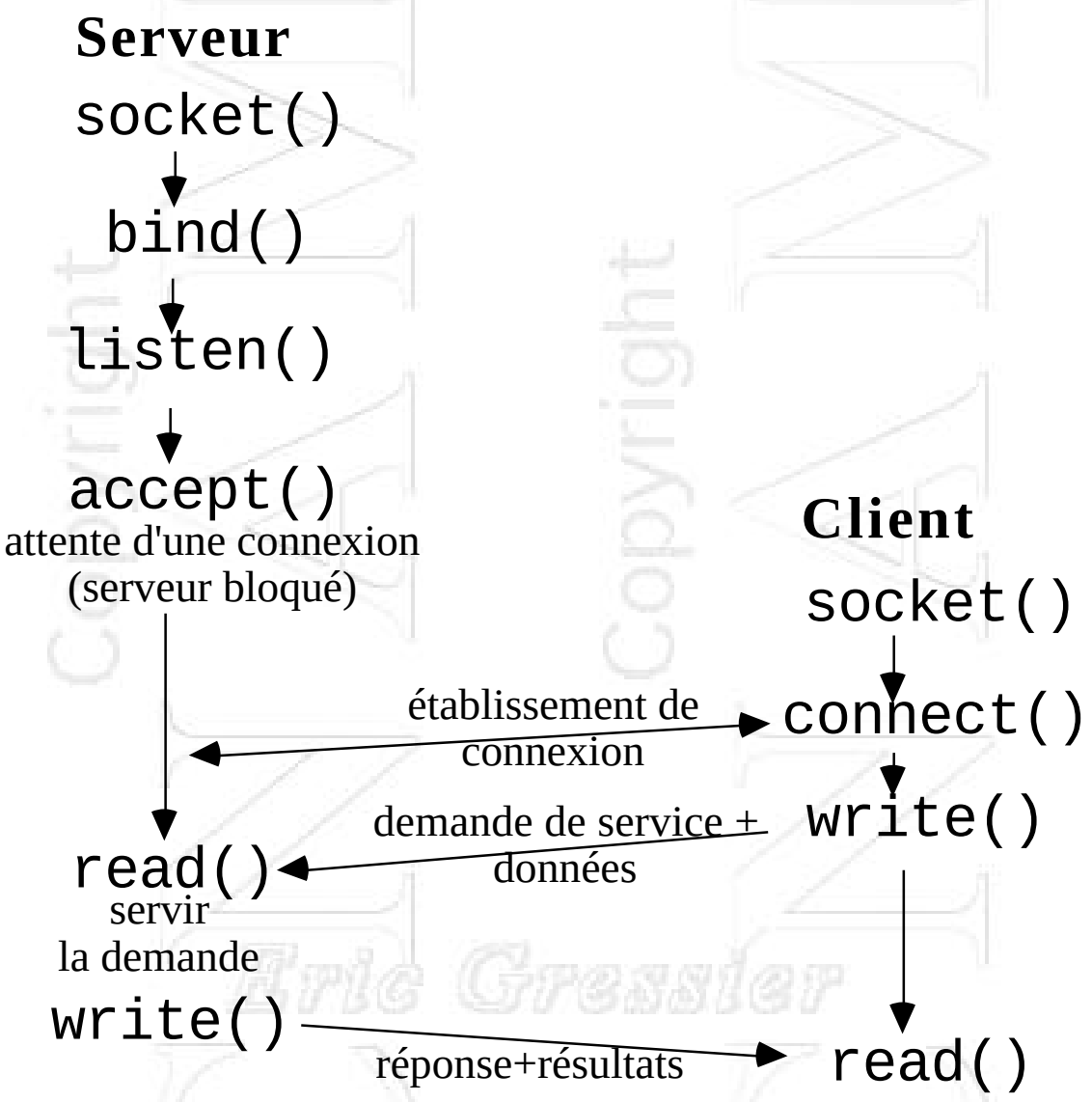
transfert de données :

primitives :
read , write , send , recv , sendto , recvfrom , sendmsg , recvmsg

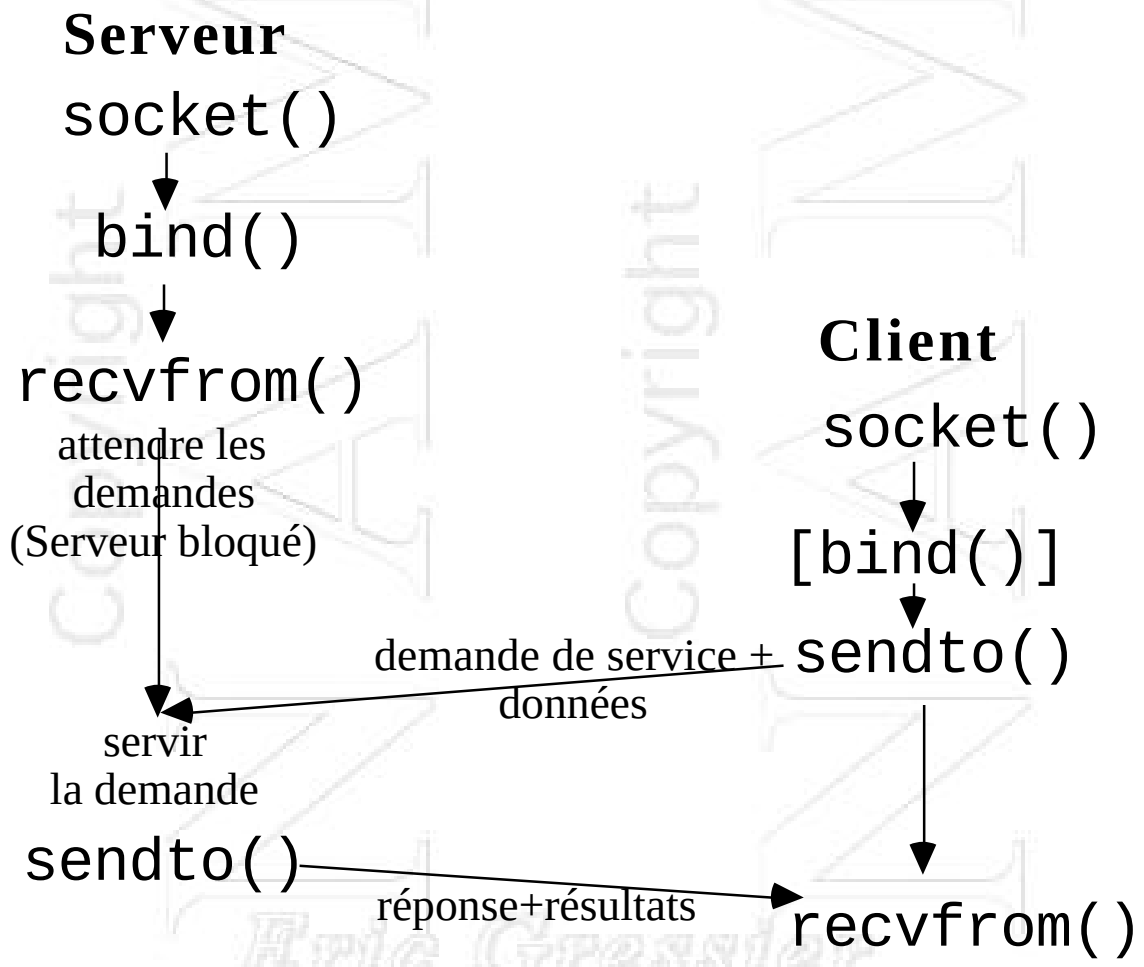
fermeture du dialogue :

primitives : close , shutdown

API- Client/Serveur mode connecté

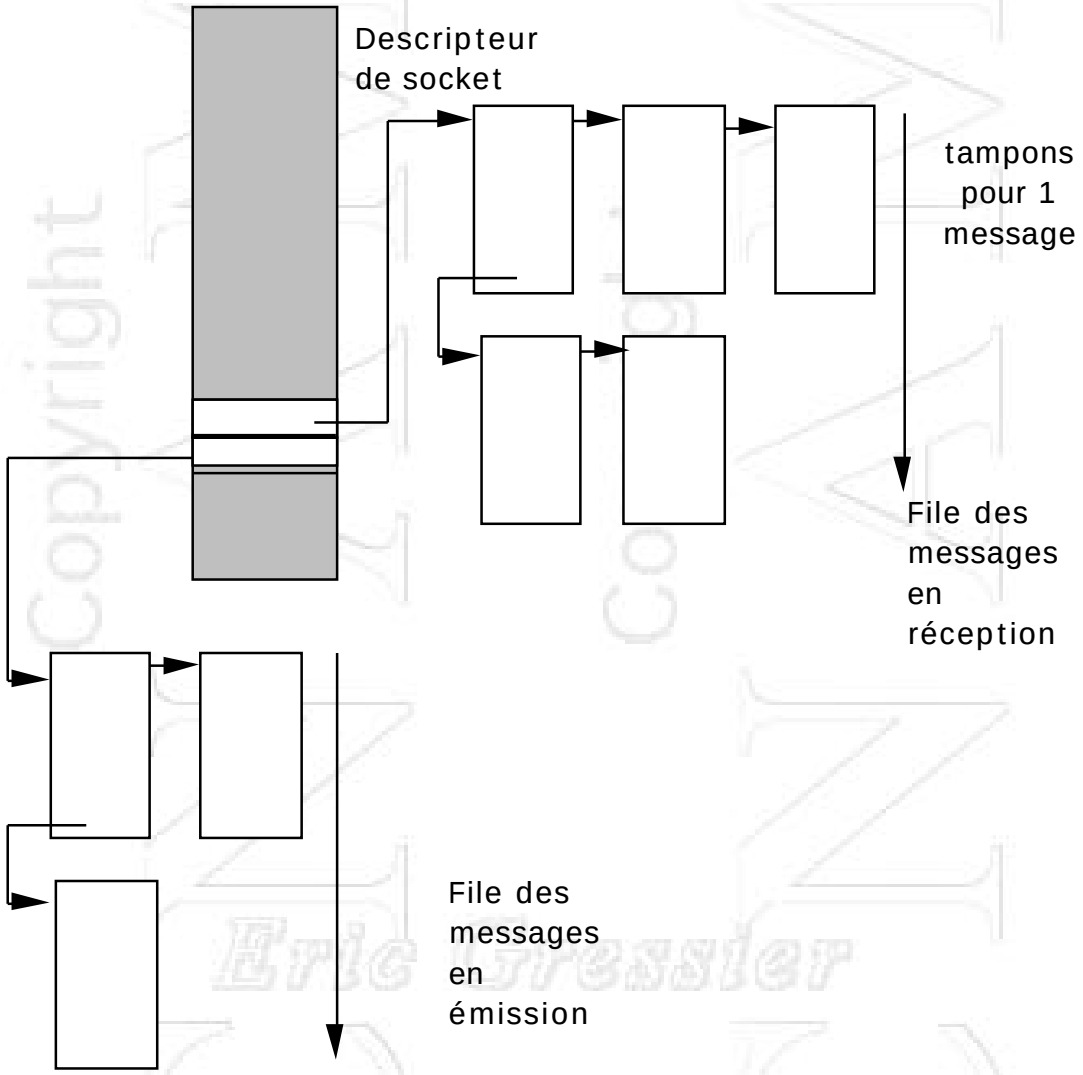


API- Client/serveur mode non-connecté



Les Sockets dans le système

"socket -> une boite aux lettres"



Création de socket : Socket () (1)

```
#include <sys/types.h>
#include <sys/socket.h>
#inclide <netinet/in.h>

int
   socket
     (int family,
        int type,
         int protocol);
```

crée la boite aux lettres, et retourne un n° de descripteur (comparer au résultat de l'appel système `open()` sur fichier)

Famille :
locale
AF_UNIX : communication locale (i-node)
réseau
AF_INET : communication Internet
AF_ISO : communication ISO
....

Création de socket : Socket () (2)

Type ou Mode de Fonctionnement d'une socket:

- **SOCK_STREAM** : "stream" (flot d'octets) mode connecté => acquit, garantit le séquencement des octets, transfert de données sans préserver les bornes du message (pas de notion d'enregistrement), données urgentes,

- **SOCK_DGRAM** : "datagram" (message) mode non connecté => sans acquit, ne garantit pas le séquencement, transfert de données qui préserve les limites de l'enregistrement,

- **SOCK_RAW** : "raw" => accès direct aux protocoles des couches basses.

- **SOCK_SEQPACKET** : format structuré ordonné (pas avec les protocoles Internet)

Création de socket : Socket () (3)

Types de protocoles :

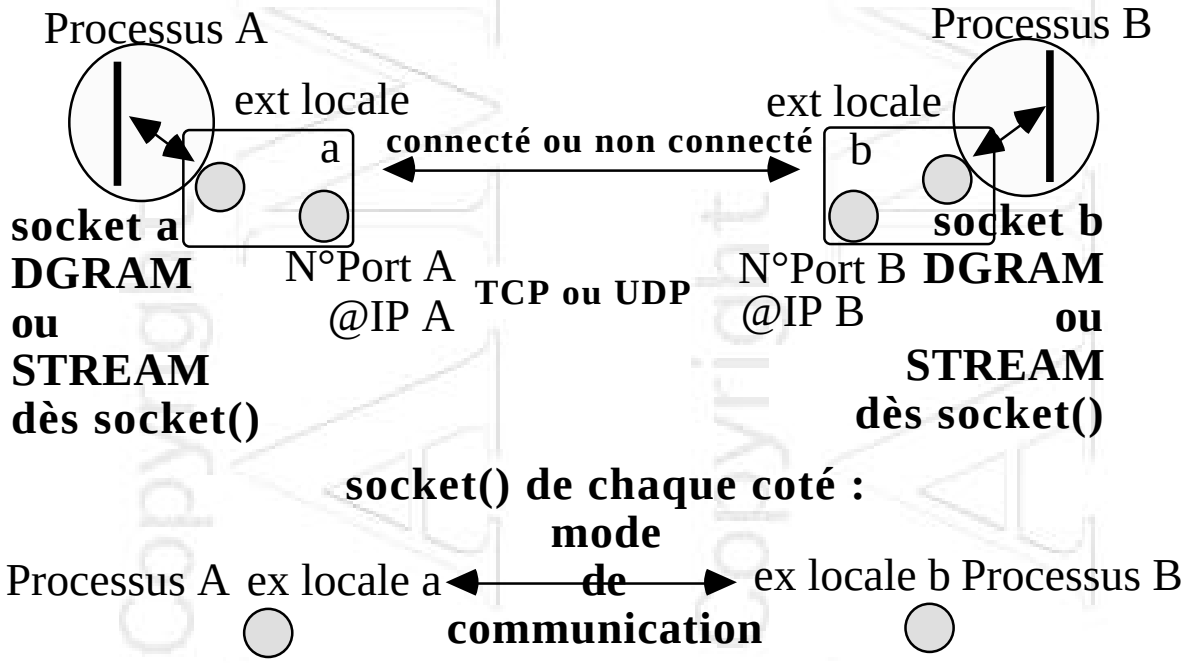
IPPROTO_UDP
IPPROTO_TCP
IPPROTO_ICMP
IPPROTO_RAW

Correspondances pour la famille AF_INET :

Mode	Type protocole	Protocole
SOCK_DGRAM	IPPROTO_UDP	UDP
SOCK_STREAM	IPPROTO_TCP	TCP
SOCK_RAW	IPPROTO_ICMP	ICMP
SOCK_RAW	IPPROTO_RAW	raw

Création de socket : Socket () (4)

Bilan : L'association entre deux processus est partiellement initialisée ...



Primitives qui permettront de compléter l'association après socket() :

	ex locale, process local	ex distant,process distant
serveur mode connecté	bind()	listen(), acccept()
client mode connecté	connect()	
serveur mode non connecté	bind()	recvfrom()
client mode non connecté	bind()	sendto()

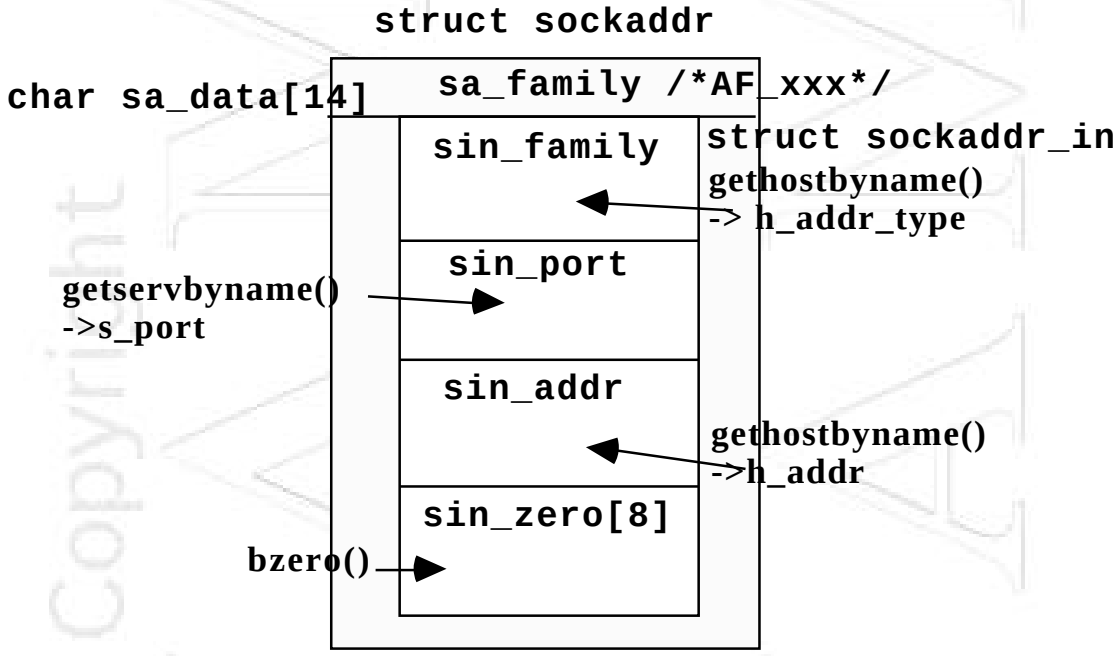
Descripteur associé à une extrêmité

Dans la terminologie Unix, l'identification d'une extrêmité locale ou distante est indiquée dans un **descripteur de socket** ...

En fonction de l'utilisation de la socket, le descripteur associé a un format différent, pour les protocoles Internet on a :

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

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

Remplissage du descripteur associé à une adresse de socket



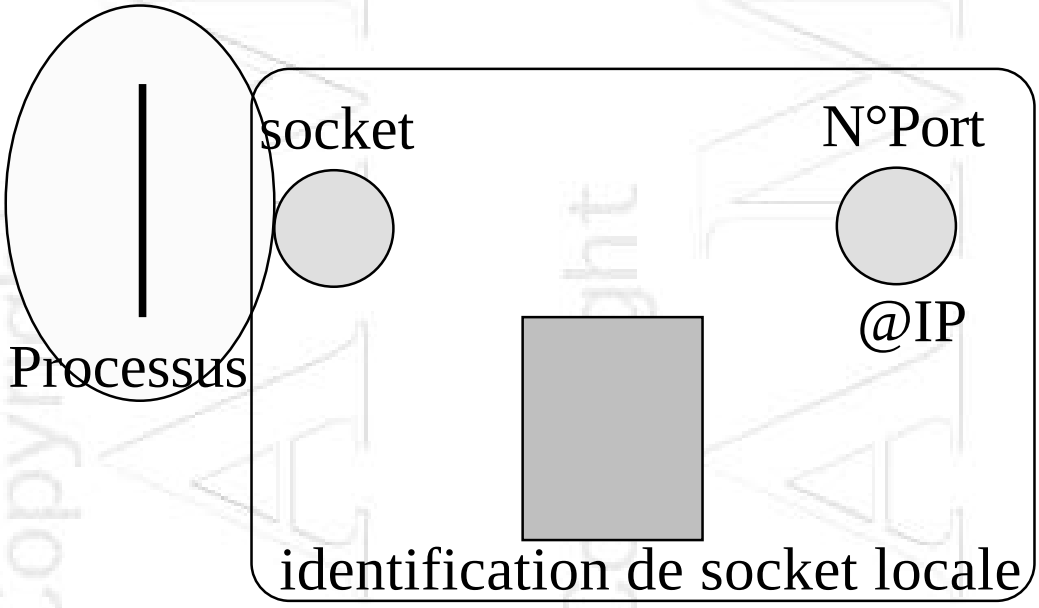
Si la machine a plusieurs adresses IP[2], lors du bind, on pourra utiliser : `INADDR_ANY` pour indiquer que toute adresse de la machine convient.

[2] Si elle est connectée à plusieurs réseaux

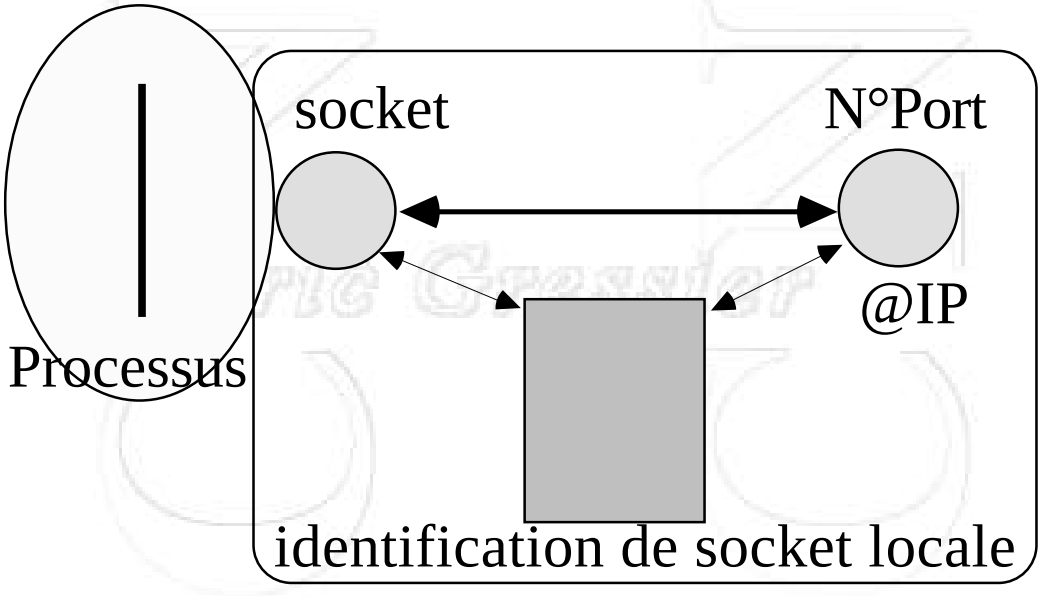
Serveur - bind() (1)

Assigne un descripteur à une socket -> renseigne l'extrêmité de l'association coté Serveur-connecté ou Client-non-connecté

Avant bind :



Après bind :



Serveur - bind() (2)

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

int bind
   (int sd,
     struct sockaddr *myaddr,
        int addrlen)
```

Le 2ème paramètre est spécifique au protocole de communication utilisé.

C'est le seul moyen pour un serveur [3] d'être lié à un numéro de port (connu dans /etc/services) pour pouvoir servir les requêtes qui lui sont adressées.

On peut utiliser bind côté client.

[3] Qu'il soit en mode connecté ou non connecté

Serveur - listen()

seulement en mode connecté

```
int listen (int sd, int backlog)
```

Le serveur se met à l'écoute des demandes de services et indique par *backlog* le nombre maximum de demandes de connexions qui peuvent attendre en même temps dans sa file (5 au maximum habituellement).

Serveur - accept() (1)

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

int accept
   (int sd,
      struct sockaddr *clientaddr,
         int clientaddrlen)
```

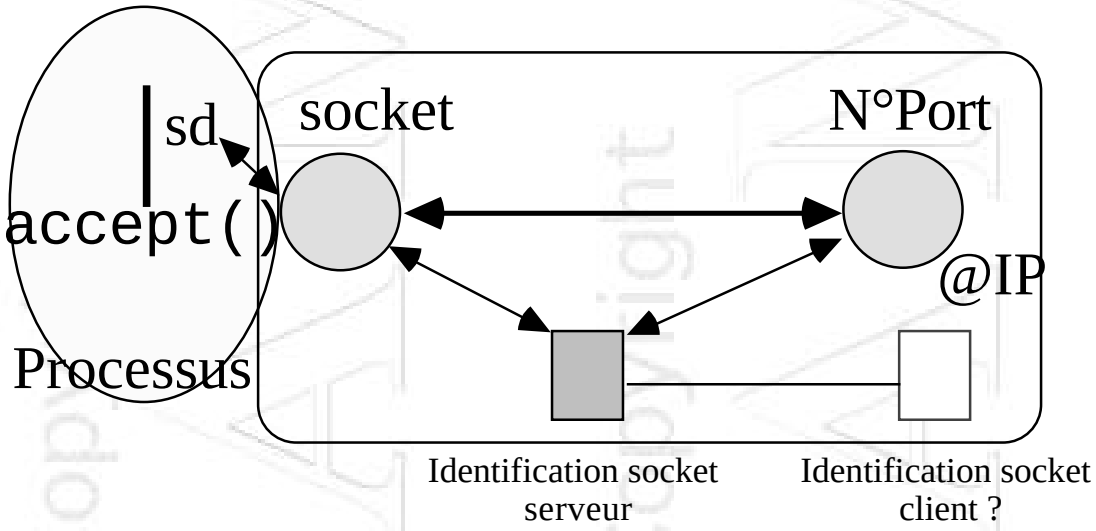
crée une nouvelle socket et retourne le numéro de descripteur de celle-ci (nsd)

retourne l'identification du client demandeur

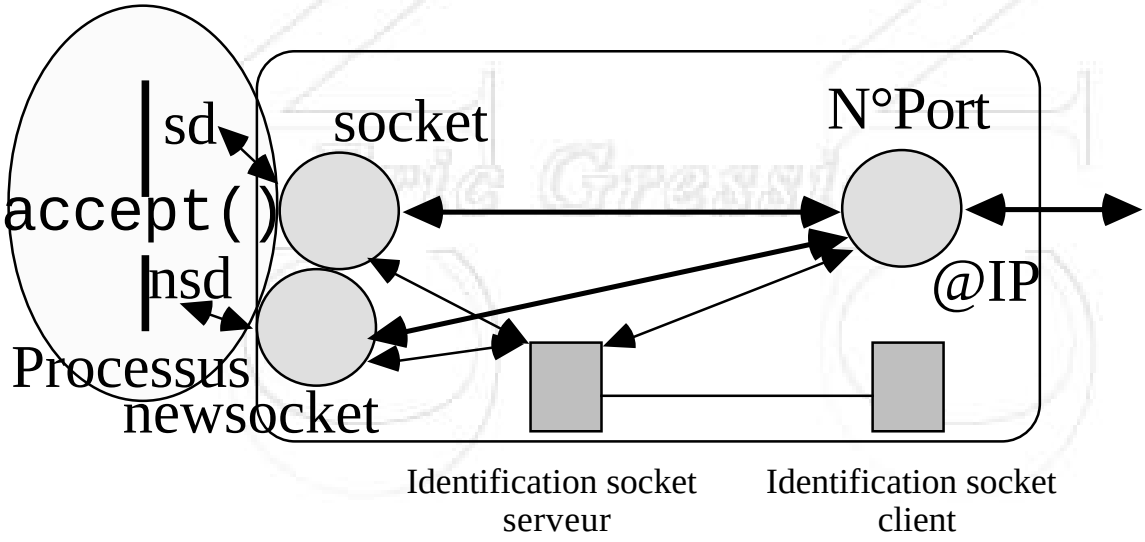
Après l'accept, l'**association est complète** pour l'IPC entre deux processus.

Serveur - accept() (2)

Avant accept :



Après accept :



Client - connect()

Coté client en mode connecté

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

int[4] connect
   (int sd,
      struct sockaddr *servaddr,
   int addrlen)
```

Etablissement de l'association avec le processus serveur visé.

Le connect permet de complèter toutes les informations de l'association aussi bien côté Client que côté Serveur. La partie extrêmité locale relative au client est renseignée automatiquement.

Pour un échange en mode non connecté, le connect permet d'utiliser les primitives read, write, send, recv ... il n'est plus nécessaire de fournir l'adresse du serveur pour le client.

[4]Le code retour indique si l'association s'est bien établie

send(), recv()

appels systèmes proches du read et du write des fichiers qui peuvent aussi être utilisés sur les sockets.

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

int[5] send
    (int sd,char *buff,int nbytes,int flags)

int recv
    (int sd,char *buff,int nbytes,int flags)
```

Les trois premiers paramètres sont identiques à ceux du read ou du write[6].

Le champ flags si non null sert à faire des envois de messages plus perfectionnés, en particulier il permet d'envoyer des données urgentes.

---

[5]Le code retour indique le nombre d'octets écrits ou lus

[6]`int read ( int fd, char *buff, unsigned int nbytes)`

`int write ( int fd, char *buff, unsigned int nbytes)`

close()

```
int close (int sd)
```

identique à la fermeture de fichier, la socket est détruite.

Avant fermeture, le système essaye d'envoyer les messages encore en attente d'émission ou d'acquitter les messages non encore acquittés ... tout dépend du protocole. Appel système qui a un effet assez abrupte sur la connexion.

Pour fermer plus harmonieusement une socket, il faut faire shutdown ().

sendto(), recvfrom()

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

int[7] sendto
    (int sd,char *buff,int nbytes,int flags,
       struct sockaddr *to,int addrlen)

int recvfrom
    (int sd,char *buff,int nbytes,int flags,
       struct sockaddr *from,int *addrlen)
```

idem send et recv, mais on spécifie respectivement le destinataire, ou la source des messages à chaque appel.

avec `from` dans `recvfrom`, le serveur peut connaître qui lui a adressé la demande

[7]Le code retour indique le nombre d'octets écrits ou lus

Exemple Client-Serveur - mode connecté (coté Serveur)

```
/* isockl.c -- set up an INTERNET STREAM socket and listen on it */
/* (C) 1991 Blair P. Houghton, All Rights Reserved, copying and */
/* distribution permitted with copyright intact.                */
/* a stream (a socket opened using SOCK_STREAM) requires the use of */
/* listen() and accept() in a receiver, and connect() in a sender */
/* récupéré par T. Cornilleau sur l'Internet */

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <fcntl.h>


#ifdef __STDC__
extern void     exit( int );
extern void     perror( char * );
extern int      printf( char *, ... );
extern int      bind( int, struct sockaddr *, int );
extern int      socket( int, int, int );
extern int      read( int, char *, unsigned );
extern char  *  strcpy( char *, char *b );
extern int      fcntl( int, int, int );
extern int      accept( int, struct sockaddr *, int * );
extern int      listen( int, int );
extern int      unlink( char * );
extern int      getsockname( int, struct sockaddr *, int * );

void main( int argc, char *argv[] )
#else
main( argc, argv )
int argc; char *argv[];
#endif


{
   int sock;                        /* fd for the listening socket */
   int ear;                         /* fd for the working socket */
   struct sockaddr_in sockaddr;     /* sytem's location of the socket */
   struct sockaddr_in caller;       /* id of foreign calling process */
   int sockaddr_in_length = sizeof(struct sockaddr_in);
   char buf[BUFSIZ];
   int read_ret;
   int fromlen = sizeof(struct sockaddr_in);
   char acknowledgement[BUFSIZ];
```

```
   /*
    *  open a net socket, using stream (file-style i/o)  mode, with protocol irrelevant ( == 0 )
    */
   if ( (sock = socket( AF_INET, SOCK_STREAM, 0 )) < 0 ) {
         char s[BUFSIZ];
         sprintf( s, "%s: can't assign fd for socket", argv[0] );
         perror(s);
         exit(__LINE__);
   }

  /*
   *  register the socket
   */
    sockaddr.sin_family = AF_INET;
    sockaddr.sin_addr.s_addr = INADDR_ANY;      /* not choosy about who calls */
    sockaddr.sin_port = 0;

   if ( bind( sock, (struct sockaddr *) &sockaddr, sizeof sockaddr ) < 0 ) {
         char s[BUFSIZ];
         sprintf( s, "%s: can't bind socket (%d)", argv[0], sock );
         perror(s);
         exit(__LINE__);
   }

  /*
   *  get port number
   */
   if ( getsockname( sock, (struct sockaddr *) &sockaddr, (int *)&sockaddr_in_length ) < 0 ) {
         char s[BUFSIZ];
         sprintf( s, "%s: can't get port number of socket (%d)",
                  argv[0], sock );
         perror(s);
         exit(__LINE__);
   }
   printf("opened socket as fd (%d) on port (%d) for stream i/o\n", sock, ntohs(sockaddr.sin_port));
   printf("struct sockaddr_in {\n\
 sin_family = %d\n    sin_addr.s_addr = %d\n   sin_port = %d\n\
} sockaddr;\n"
, sockaddr.sin_family
, sockaddr.sin_addr.s_addr
, ntohs(sockaddr.sin_port)
   );
```

```
    /* put an ear to the socket, listening for a knock-knock-knocking */
    listen( sock, 1 );                          /* 1: only one queue slot */
    /* ear will be a temporary (non-reusable) socket different from sock */
    if ( (ear = accept( sock, (struct sockaddr *)&caller, &fromlen )) < 0 ) {
            perror(argv[0]);
            exit(__LINE__);
    }

    /* print calling process' identification */
    printf(
"struct sockaddr_in {\n\
    sin_family          = %d\n\
    sin_addr.s_addr     = %s\n\
    sin_port (!!!)      = %d\n\
} caller;\n"
, caller.sin_family
, inet_ntoa(caller.sin_addr)
/* , caller.sin_addr.s_addr -- gives an unsigned long, not a struct in_addr */
, ntohs(caller.sin_port)
    );
```

```
/* optional ack; demonstrates bidirectionality */
   gethostname(buf, sizeof buf);
   sprintf( acknowledgement, "Welcome, from sunny %s (%s.%d)\n",
                    buf,
                    buf,
                    ntohs(sockaddr.sin_port)
                );
   /* write into the ear; the sock is _only_for_rendezvous_ */
   if ( write ( ear, acknowledgement, sizeof acknowledgement ) < 1 )
         perror(argv[0]);

   /* read lines until the stream closes */
   while ( (read_ret = read( ear, buf, sizeof buf )) > 0 )
         printf( "%s: read from socket as follows:\n(%s)\n", argv[0], buf );

   if ( read_ret < 0 ) {
         char s[BUFSIZ];
         sprintf( s, "%s: error reading socket", argv[0] );
         perror(s);
         exit(__LINE__);
   }

   /* loop ended normally:  read() returned NULL */
   exit(0);
}
```

Exemple Client-Serveur - mode connecté (coté Client)

```
/* isockt.c -- open an internet socket and talk into it */
/* (C) 1991 Blair P. Houghton, All Rights Reserved, copying and */
/* distribution permitted with copyright intact.                */
/* récupéré par T. Cornilleau sur l'Internet */

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

#ifdef __STDC__
extern void      perror( char * );
extern int       bind( int, struct sockaddr *, int );
extern int       socket( int, int, int );
extern int       write( int, char *, unsigned );
extern char *    strcpy( char *, char *b );
extern int       strlen( char * );
extern void      exit( int );
extern int       connect( int, struct sockaddr *, int );
extern struct hostent *  gethostbyname( char * );
extern int       fprintf( FILE *, char *, ... );
extern int       atoi( char * );
extern void      bcopy( char *, char *b, int );
#endif

char *line[] = {
   "Mary had a little lamb;\n",
   "Its fleece was white as snow;\n",
   "And everywhere that Mary went,\n",
   "She told everyone that Edison invented\nthe telephone before Bell did.\n"
};
int n_line = 4;
/*
 *  arg 0 is program name; arg 1 is remote host; arg 2 is port number of listener on remote host
 */
#ifdef __STDC__
void main( int argc, char *argv[] )
#else
main(argc,argv)
int argc; char *argv[];
#endif
{
   int plug;                        /* socket to "plug" into the socket */
   struct sockaddr_in socketname;   /* mode, addr, and port data for the socket */
   struct hostent *remote_host;     /* internet numbers, names */
   extern int n_line;
   extern char *line[];
   char buf[BUFSIZ];
   int i;
```

```
   /* make an internet-transmitted, file-i/o-style, protocol-whatever plug */
   if ( (plug = socket( AF_INET, SOCK_STREAM, 0 )) < 0 )
        perror(argv[0]);

   /* plug it into the listening socket */
   socketname.sin_family = AF_INET;
   if ( (remote_host = gethostbyname( argv[1] )) == (struct hostent *)NULL ) {
        fprintf( stderr, "%s: unknown host: %s\n",
                argv[0], argv[1] );
        exit(__LINE__);
   }
   (void) bcopy( (char *)remote_host->h_addr, (char *) &socketname.sin_addr,
                remote_host->h_length );
   socketname.sin_port = htons(atoi(argv[2]));

   if ( connect( plug, (struct sockaddr *) &socketname, sizeof socketname ) < 0 ) {
        perror(argv[0]);
        exit(__LINE__);
   }

   /* wait for ack */
   if ( read( plug, buf, sizeof buf ) > 0 )
        printf(buf);

   /* say something into it; something historic */
   for ( i = 0; i < n_line; i++ ) {
        sleep(1);
        if ( write( plug, line[i], strlen(line[i]) ) < 0 ) {
           perror(argv[0]);
           exit(__LINE__);
        }
   }

   /* all the socket connections are closed automatically */
   exit(0);
}
```

A l'exécution coté serveur :

```
Script started on Wed Feb 23 12:51:53 1994
% isockl &
[1] 29191

% opened socket as fd (3) on port (4192) for stream i/o

struct sockaddr_in {
    sin_family          = 2
    sin_addr.s_addr     = 0
     sin_port           = 4192
} sockaddr;

struct sockaddr_in {
    sin_family          = 2
    sin_addr.s_addr     = 163.173.128.6
    sin_port (!!!)      = 4230
} caller;

isockl: read from socket as follows:
(Mary had a little lamb;
)

isockl: read from socket as follows:
(Its fleece was white as snow;
)

isockl: read from socket as follows:
(And everywhere that Mary went,
)
isockl: read from socket as follows:
(She told everyone that Edison invented
the telephone before Bell did.
)
^C
[1]    Done                 isockl
% ^D
script done on Wed Feb 23 12:53:48 1994
```

A l'exécution coté client :

Script started on Wed Feb 23 12:52:43 1994

% isockt asimov 4192

Welcome, from sunny asimov.cnam.fr (asimov.cnam.fr.4192)

% ^D

script done on Wed Feb 23 12:53:38 1994

Exemple Client-Serveur- mode non connecté- Coté Serveur

```
/* sockl.c -- set up an INTERNET DGRAM socket and listen on it */
/* (C) 1991 Blair P. Houghton, All Rights Reserved, copying and */
/* distribution permitted with copyright intact.                */
/* a DGRAM receiver is thoroughly independent of the sender */
/* récupéré par T. Cornilleau sur l'Internet */

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

#ifdef __STDC__
extern void     exit( int );
extern void     perror( char * );
extern int      printf( char *, ... );
extern int      bind( int, struct sockaddr *, int );
extern int      socket( int, int, int );
extern int      read( int, char *, unsigned );
extern char *   strcpy( char *, char *b );
extern int      fcntl( int, int, int );
extern int      accept( int, struct sockaddr *, int * );
extern int      listen( int, int );
extern int      unlink( char * );
extern int      getsockname( int, struct sockaddr *, int * );
extern int      recvfrom( int, char *, int, int, struct sockaddr *, int );

void main( int argc, char *argv[] )
#else
main( argc, argv )
int argc; char *argv[];
#endif
{
    int sock;                       /* fd for the socket */
    struct sockaddr_in sockaddr;    /* sytem's location of the socket */
    struct sockaddr_in caller;      /* id of foreign calling process */
    int sockaddr_in_length = sizeof(struct sockaddr_in);
    char buf[BUFSIZ];
    int read_ret;
    int fromlen = sizeof(struct sockaddr_in);
    char acknowledgement[BUFSIZ];
```

```
    /*
     *  open a net socket, using dgram (packetized, nonconnected)
     *  mode, with protocol irrelevant ( == 0 )
     */
    if ( (sock = socket( AF_INET, SOCK_DGRAM, 0 )) < 0 ) {
            char s[BUFSIZ];
            sprintf( s, "%s: can't assign fd for socket", argv[0] );
            perror(s);
            exit(__LINE__);
    }

   /*
    *  register the socket
    */
    sockaddr.sin_family = AF_INET;
    sockaddr.sin_addr.s_addr = INADDR_ANY;      /* not choosy about who calls */
    sockaddr.sin_port = 0;                      /* ??? why 0? */
    if ( bind( sock, (struct sockaddr *) &sockaddr, sizeof sockaddr ) < 0 ) {
            char s[BUFSIZ];
            sprintf( s, "%s: can't bind socket (%d)", argv[0], sock );
            perror(s);
            exit(__LINE__);
    }
   /*
    *  get port number
    */
    if ( getsockname(sock, (struct sockaddr *) &sockaddr, (int *)&sockaddr_in_length)
            < 0 ) {
            char s[BUFSIZ];
            sprintf( s, "%s: can't get port number of socket (%d)",
                    argv[0], sock );
            perror(s);
            exit(__LINE__);
    }
    printf("opened socket as fd (%d) on port (%d) for dgram i/o\n", sock, ntohs(sockaddr.sin_port) );
    printf("struct sockaddr_in {\n\
    sin_family = %d\n sin_addr.s_addr = %d\n sin_port = %d\n\
} sockaddr;\n"
, sockaddr.sin_family
, sockaddr.sin_addr.s_addr
, ntohs(sockaddr.sin_port)
    );
    fflush(stdout);
```

```
   /* read and print lines until the cows come home */
   while ( (read_ret = recvfrom( sock, buf, sizeof buf,
                                 0, (struct sockaddr *) &caller,
                                 &fromlen)
            ) > 0 ) {
         printf( "%s: read (from caller (%s, %d)) socket as follows:\n(%s)\n",
                  argv[0],
                  inet_ntoa(caller.sin_addr),
                  ntohs(caller.sin_port),
                  buf );
         fflush(stdout);
   }

   if ( read_ret < 0 ) {
         char s[BUFSIZ];
         sprintf( s, "%s: error reading socket", argv[0] );
         perror(s);
         exit(__LINE__);
   }

   /* loop ended normally:  read() returned NULL */
   exit(0);
}
```

Exemple Client-Serveur- mode non connecté- Coté Client

```
/* disockt.c -- open an internet socket and talk dgrams into it */
/* (C) 1991 Blair P. Houghton, All Rights Reserved, copying and */
/* distribution permitted with copyright intact.                */
/* récupéré par T. Cornilleau sur l'Internet */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <math.h>
#ifdef __STDC__
extern void                 perror( char * );
extern int                  bind( int, struct sockaddr *, int );
extern int                  socket( int, int, int );
extern int                  write( int, char *, unsigned );
extern char *               strcpy( char *, char *b );
extern int                  strlen( char * );
extern void                 exit( int );
extern int                  connect( int, struct sockaddr *, int );
extern struct hostent *  gethostbyname( char * );
extern int                  fprintf( FILE *, char *, ... );
extern int                  atoi( char * );
extern void                 bcopy( char *, char *b, int );
#endif
char *line[] = {
   "Mary had a little lamb;\n",
   "Its fleece was white as snow;\n",
   "And everywhere that Mary went,\n",
   "She told everyone that Edison invented\nthe telephone before Bell did.\n"
};
int n_line = 4;
/*
 *  argv 0 is program name; argv 1 is remote host; argv 2 is port number of listener on remote host
 */
#ifdef __STDC__
void main( int argc, char *argv[] )
#else
main(argc,argv)
int argc; char *argv[];
#endif
{
   int plug;                        /* socket to "plug" into the socket */
   struct sockaddr_in socketname;   /* mode, addr, and port          */
                                    /* data for the socket    */
   struct hostent *remote_host;     /* internet numbers, names */
   extern int n_line;
   extern char *line[];
   char buf[BUFSIZ];
   int i, sendflags;
```

```
    /* make an internet-transmitted, dgram-i/o-style, protocol-whatever plug */
    if ( (plug = socket( AF_INET, SOCK_DGRAM, 0 )) < 0 )
        perror(argv[0]);
    /* plug it into the listening socket */
    socketname.sin_family = AF_INET;
    if ( (remote_host = gethostbyname( argv[1] )) == (struct hostent *)NULL ) {
        fprintf( stderr, "%s: unknown host: %s\n", argv[0], argv[1] );
        exit(__LINE__);
    }
    (void) bcopy( (char *)remote_host->h_addr, (char *) &socketname.sin_addr,
                remote_host->h_length );
    socketname.sin_port = htons(atoi(argv[2]));
    printf("sending %d dgrams to:\n", n_line);
    printf("struct sockaddr_in {\n\
    sin_family = %d\n sin_addr.s_addr = %s\n sin_port  = %d\n\
} socketname;\n"
, socketname.sin_family
, inet_ntoa(socketname.sin_addr)
/* , socketname.sin_addr.s_addr -- gives unsigned long, not struct in_addr */
, ntohs(socketname.sin_port)
    );

    /* say something into it; something historic */
    sendflags = 0;
    for ( i = 0; i < n_line; i++ ) {
        sleep(1);
        if ( sendto( plug,
                line[i], 1+strlen(line[i]),
                sendflags,
                (struct sockaddr *)&socketname, sizeof socketname )
            < 0 ) {
            perror(argv[0]);
            exit(__LINE__);
        }
    }

    /* all the socket connections are closed automatically */
    exit(0);
}
```

A l'exécution coté serveur :

```
Script started on Wed Feb 23 13:13:42 1994
% disockl &
[1] 29423
% opened socket as fd (3) on port (1143) for dgram i/o

struct sockaddr_in {
    sin_family          = 2
    sin_addr.s_addr  = 0
    sin_port            = 1143
} sockaddr;
disockl: read (from caller (163.173.128.6, 1441)) socket as follows:

(Mary had a little lamb;
)

disockl: read (from caller (163.173.128.6, 1441)) socket as follows:
(Its fleece was white as snow;
)

disockl: read (from caller (163.173.128.6, 1441)) socket as follows:

(And everywhere that Mary went,
)

disockl: read (from caller (163.173.128.6, 1441)) socket as follows:

(She told everyone that Edison invented
the telephone before Bell did.
)

^C
% ^D
script done on Wed Feb 23 13:16:56 1994
```

A l'exécution coté client :

Script started on Wed Feb 23 13:15:46 1994

% disockt asimov 1143

sending 4 dgrams to:

struct sockaddr_in {

sin_family = 2

sin_addr.s_addr = 163.173.128.6

sin_port = 1143

} socketname;

% ^D

script done on Wed Feb 23 13:16:43 1994

select() (1)

permet d'attendre sur plusieurs sockets en même temps

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

int select
   (int maxfdl,
      fd_set *readfds,
         fd_set *writefds,
            fd_set *exceptfds,
               struct timeval *timeout)

struct timeval {
   long tv_sec;     /*secondes*/
   long tv_usec;/*microsecondes*/
};
```

macros de manipulation de l'ensemble des descripteurs sur lesquels s'applique le select :

```
FD_ZERO (fd_set *fdset)/*mise à zero de fdset*/
FD_SET(int fd,fd_set *fdset)/*set bit fd dans fdset*/
FD_CLR(int fd,fd_set *fdset)/*clear bit fd dans fdset*/
FD_ISSET (int fd,fd_set *fdset)/*test bit fd dans fdset*/
```

`select()` (2)

Attente sur le select - paramètre *`timeout`* :

- polling des descripteurs : Zero dans chaque champ de *timeout*, l'appelant récupère la main dès qu'il a examiner les descripteurs de sockets

- attendre un délai déterminé : remplir *timeout* à la valeur du délai d'attente

faire un timer plus fin que sleep[8] :

```
select (0,(f_set *)0,(f_set *)0,(f_set *)0, &timeout);
```

- attendre tant qu'il ne se passe rien : l'argument *timeout* doit prendre la valeur NULL

[8] sleep a une granularité de l'ordre de la seconde, ce n'est parfois pas assez fin

select() (3)

choix des descripteurs sur lequel opère le select:

readfds : qqch à lire (le plus interessant)
writefds : descripteur prêt à écrire (l'écriture précédente est terminée)
exceptfds : détection d'une exception (données urgentes ou informations de contrôle liées à l'utilisation de pseudo-terminaux)

Définition de descripteurs dans une variable de type fd_set:

```
fd_set fdvar ;

FD_ZERO(&fdvar);
FD_SET(3, &fdvar); /*descripteur 3*/
FD_SET(5, &fdvar); /*descripteur 5*/
```

L'appelant doit tester les descripteurs au moment du retour par `FD_ISSET`.

Le code de retour de la primitives indique le nombre de descripteurs prêts[9].

`maxfdpl` indique le nombre de descripteurs sur lesquels s'effectue l'attente : il faut prendre le numéro du plus grand descripteur +1.

[9] Zéro si aucun prêt dans le cas d'une fin de timer, -1 en cas d'erreur

Autres primitives à examiner

getpeername (),
getsockname ()

getsockopt (), setsockopt() avec ioctl() et fcntl()

pour setsockopt, il faut regarder les options TCP_NODELAY, SO_BROADCAST

fflush() pour vider une socket ... à confirmer ?

Fonctions diverses

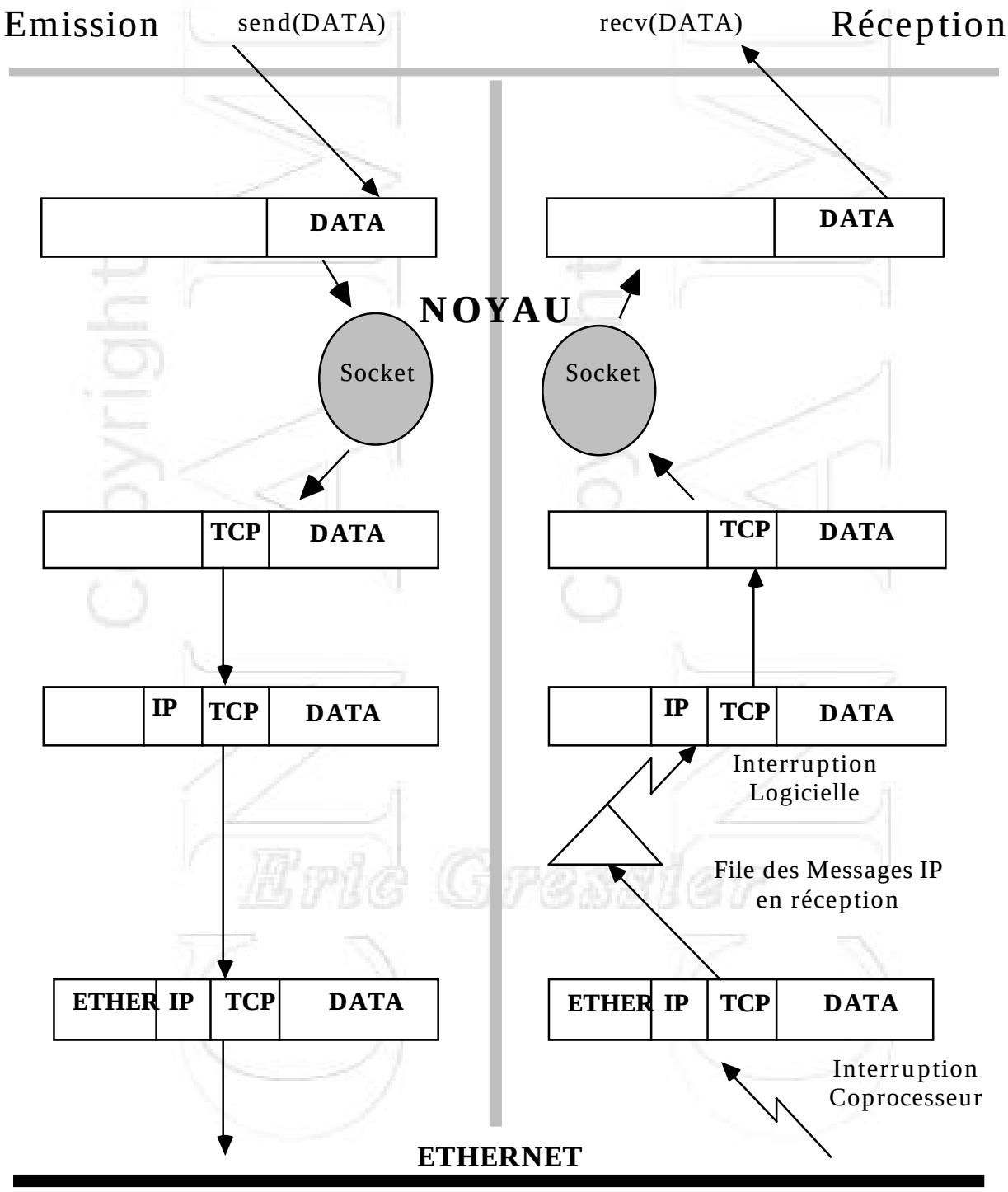
Manipulations de suites d'octets :

```
bcopy (char *src, char *dest, int nbytes)

bzero (char *dest, int nbytes)

int bcmp(char *ptr1,char *ptr2,int nbytes)
```

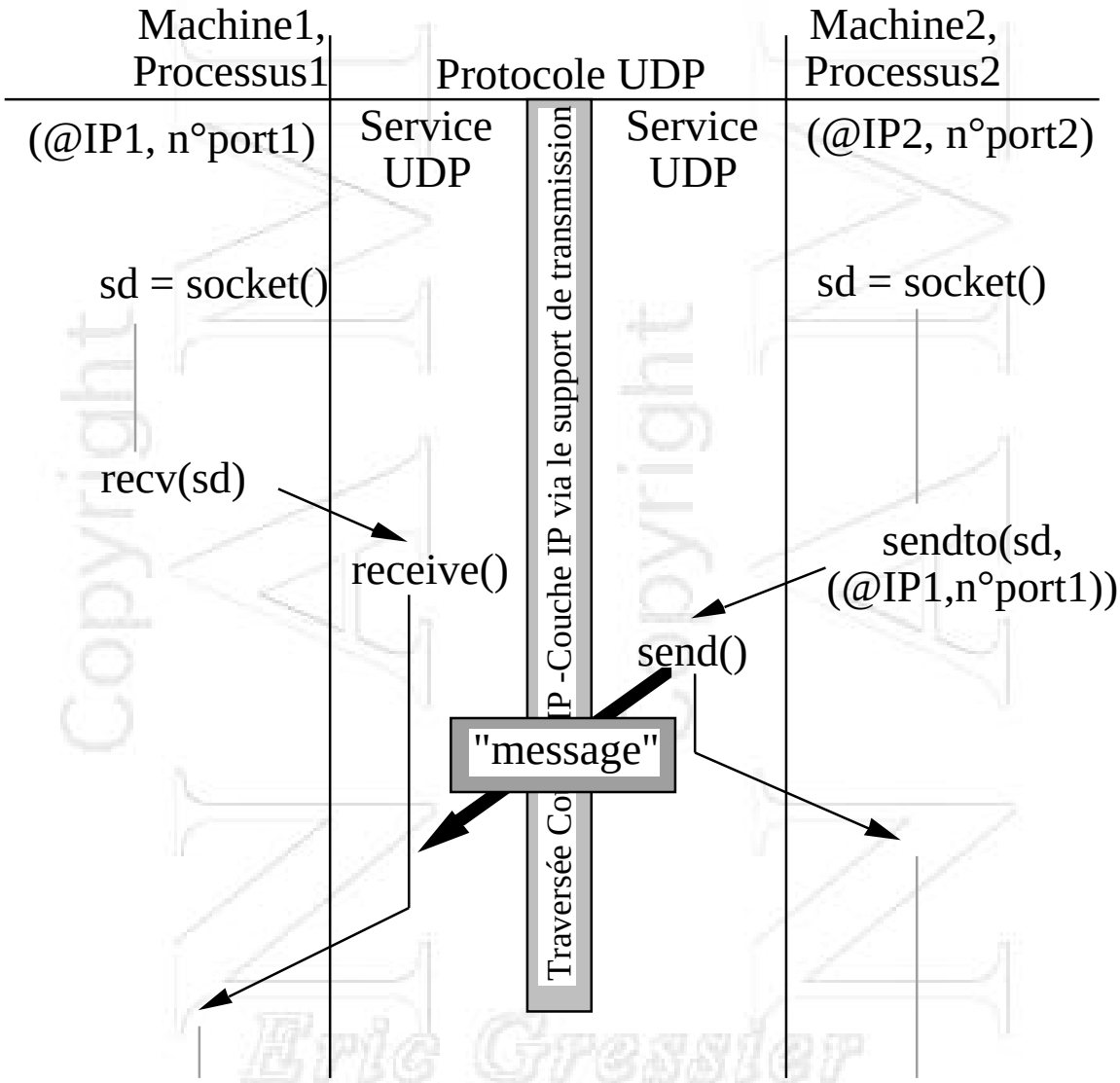
Conversions d'adresses :

```
unsigned long inet_addr (char *ptr)

char *inet_ntoa (struct in_addr ipaddr)
```

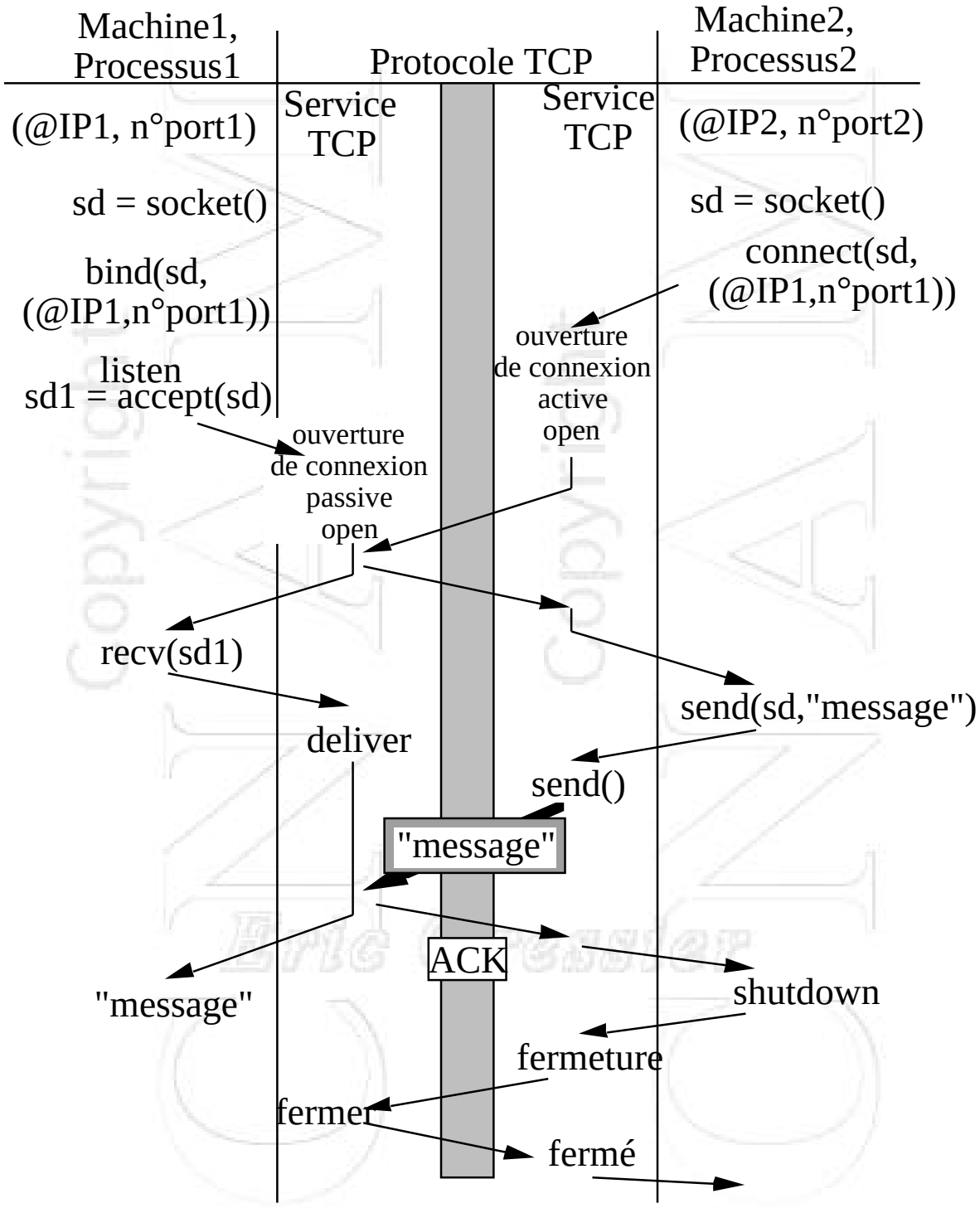
Domaine AF_INET => TCP-UDP/IP



Principes Généraux API socket et UDP

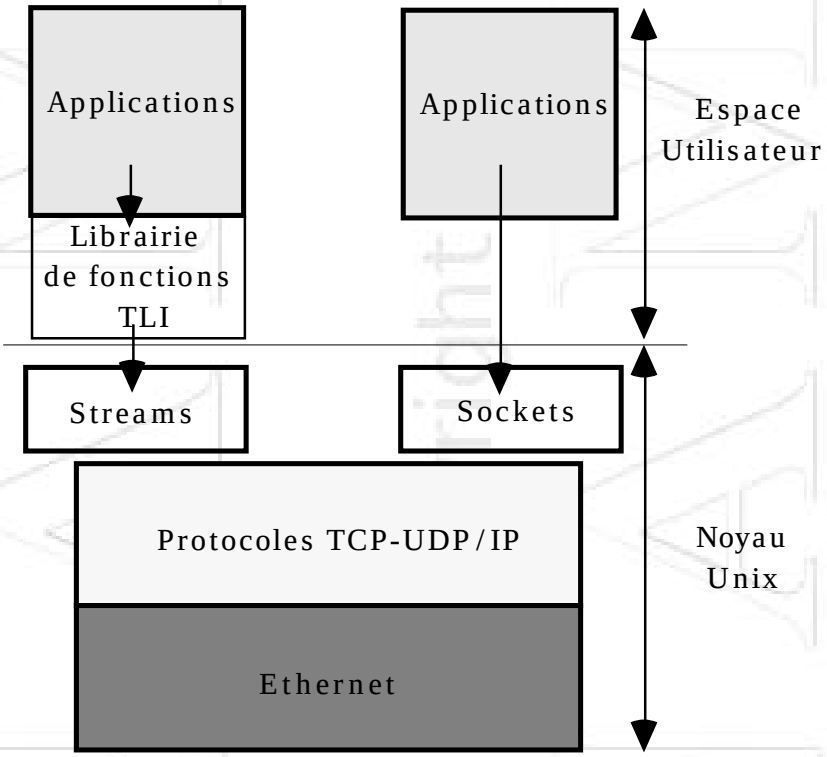


Principes Généraux API socket et TCP



6. API TLI

Les Sockets : Interface avec les protocoles de communication



TLI et les Streams

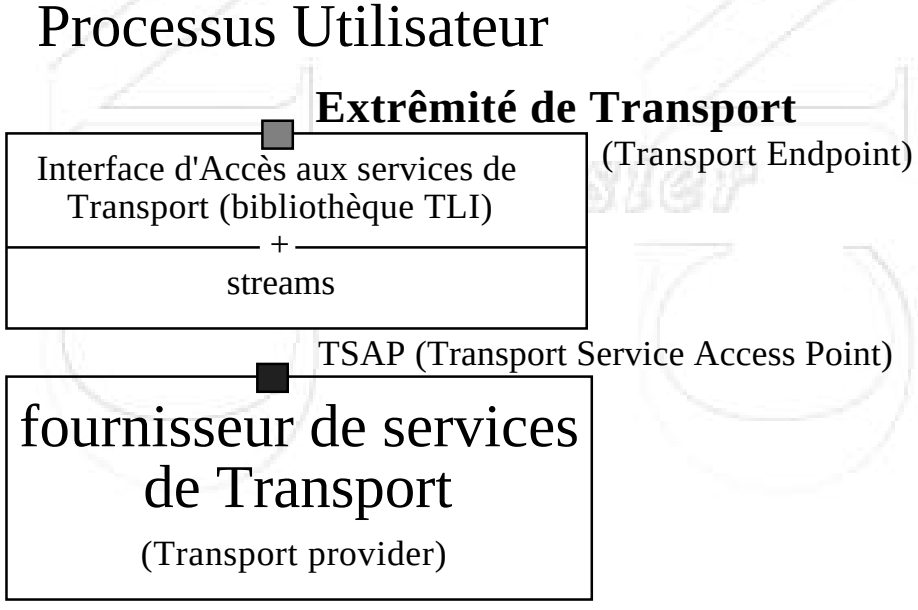
TLI pour Transport Layer Interface, c'est une bibliothèque de fonctions qui sert à accéder des protocoles de communication … attention, ce ne sont pas les protocoles eux-mêmes !!!!

Pour les utiliser, il faut appeler une bibliothèque à l'édition de liens :

```
cc prog.c -lnsl_s
```

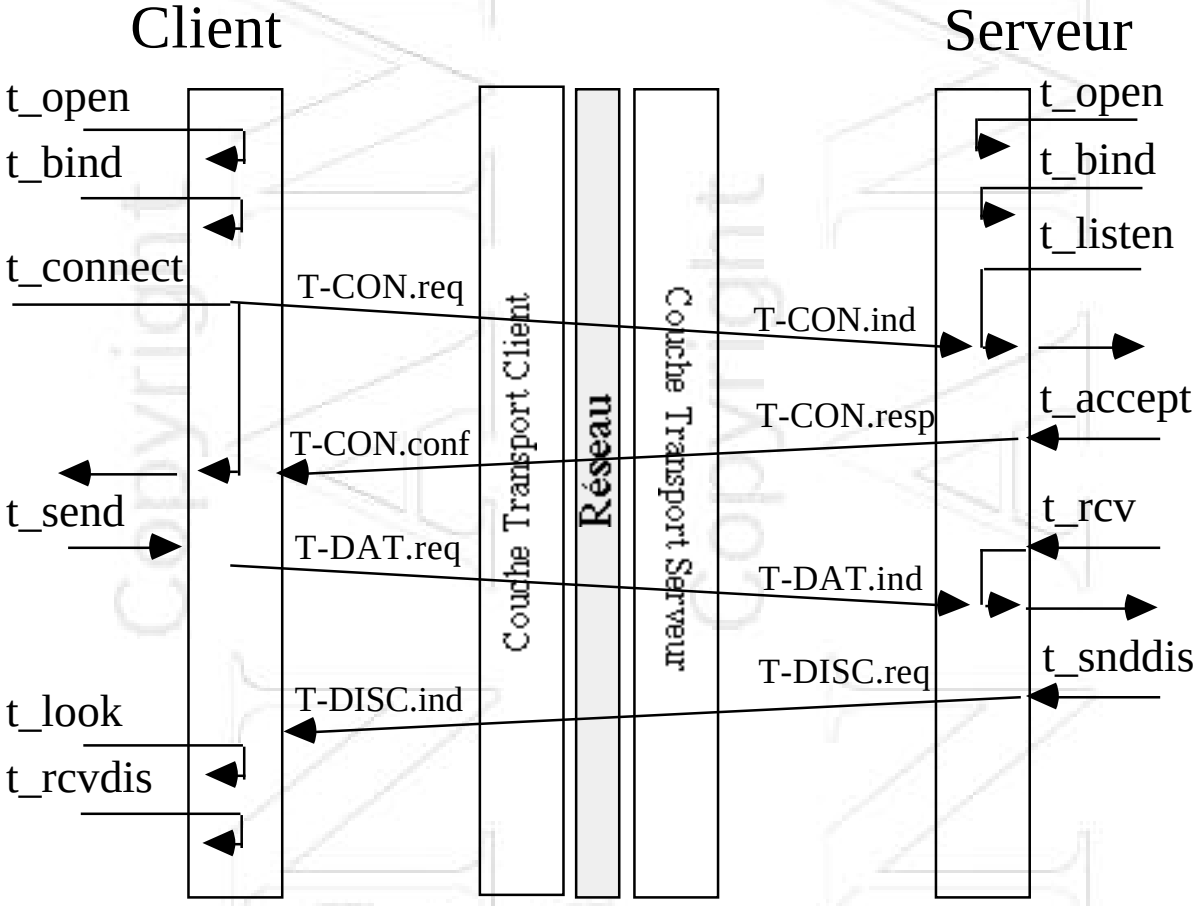
TLI est apparu avec Unix System V release 3.0. (ATT) et doit être vu comme une **API** pour accéder à la couche Transport d'un réseau de communication.

La version normalisée par le groupe X/Open est : XTI. TLI est donc très marqué par la logique ISO :



Deux Extrêmités de Transport sont mises en relation lors d'une connexion

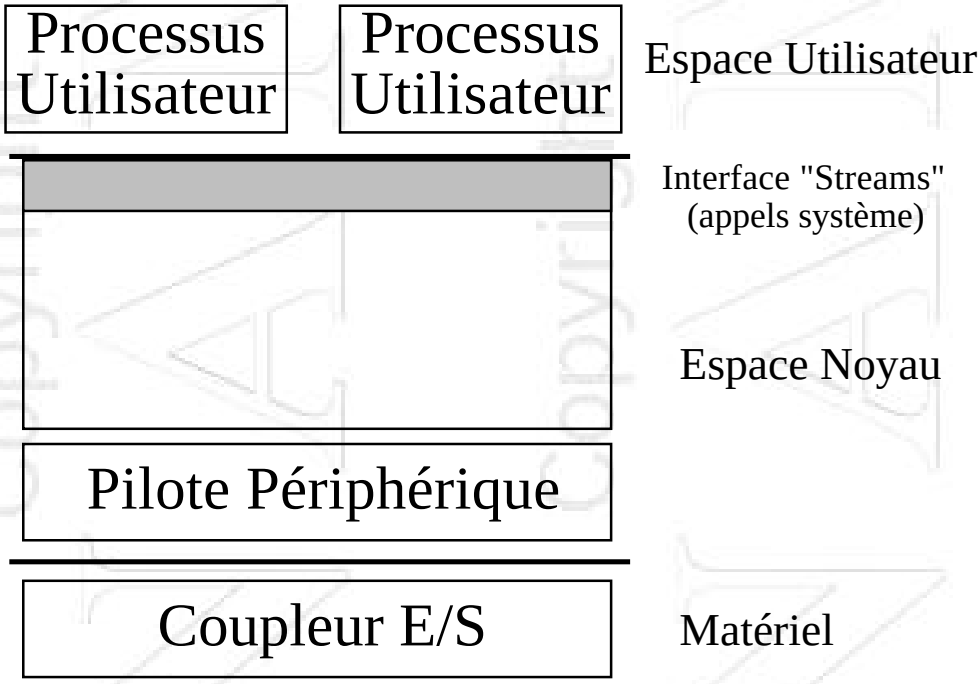
Utilisation de TLI dans l'univers ISO



Ca marche aussi avec TCP/IP !!!

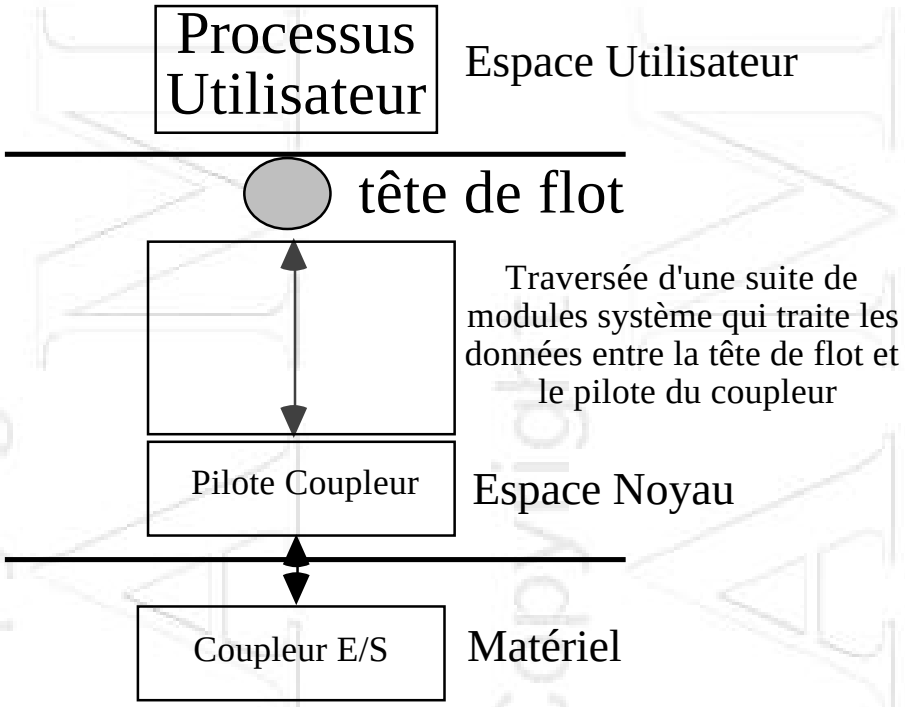
STREAMS

Modèle abstrait :



Streams : image du flot de données établi entre l'utilisateur et un périphérique d'E/S via son "driver"

Streams : vue système



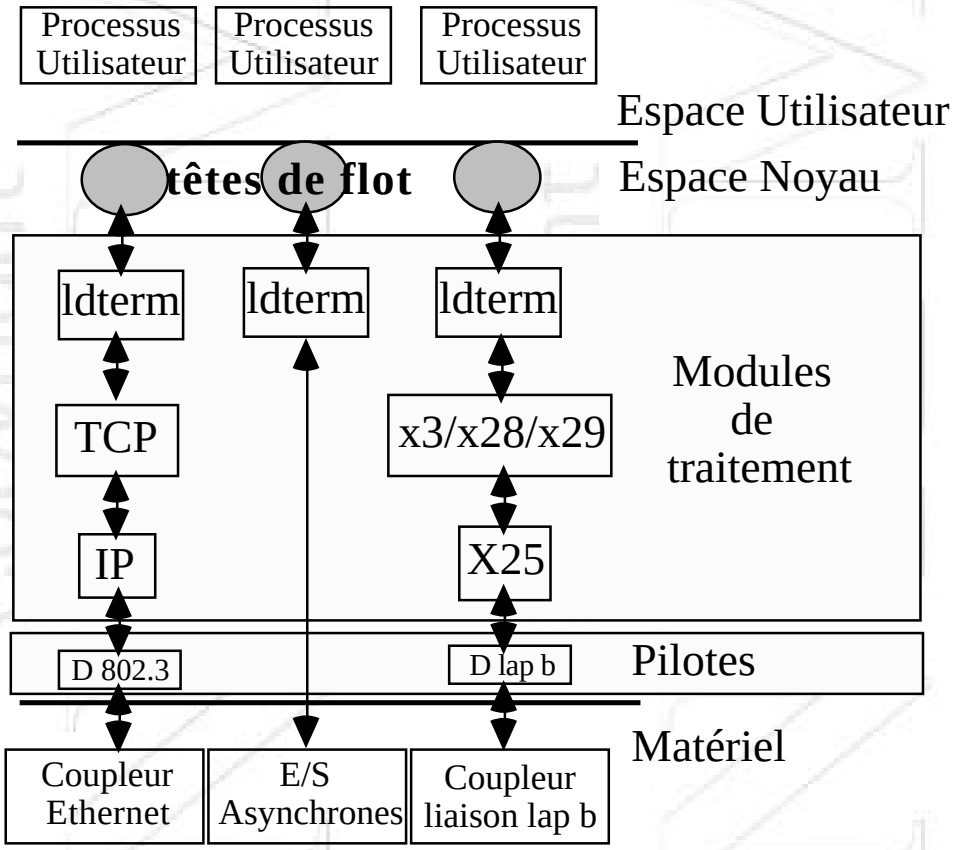
Les flots SVR4[10] représentent un **mécanisme système générique** pour implanter des politiques de gestion de périphériques : réseau, terminal...

L'objectif est de pouvoir développer du code de façon modulaire et réutilisable. Pour cela on peut insérer des "modules de traitement" entre la tête de flot et le pilote de périphérique.

[10] On parlera de flots SVR4 pour Streams system V R4.0, même si le mécanisme de streams existe dans des versions anterieures de System V.

Streams : Vue système - Exemple (1)

Streams et Contrôle d'E/S Terminal : Chaque processus utilisateur pense être connecté à une ligne de terminal

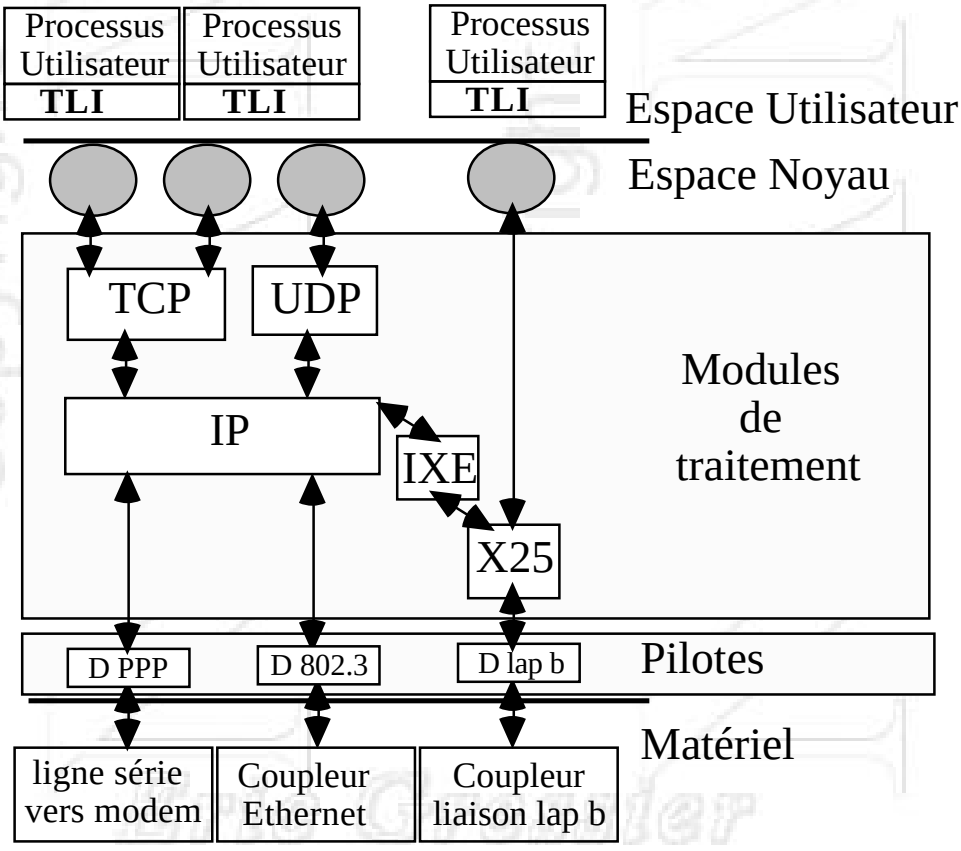


- Une liaison terminal emprunte TCP/IP sur un réseau local Ethernet.
- Une liaison terminal emprunte une ligne série asynchrone.
- Une liaison terminal emprunte une connexion X25.

Exemple de réutilisation du module ldterm

Streams : Vue système - Exemple (2)

Streams et Communication TCP/IP



Avec ce type d'architecture, on peut imaginer que chaque module peut être fourni par un constructeur différent.

TLI : bibliothèque de fonctions

t_open : création d'une extrêmité de transport qui identifie le protocole de Transport utilisé et la qualité de service demandée, l'extrêmité de Transport est vue comme un fichier
t_bind : association d'une adresse (TSAP en OSI) à l'extrêmité de transport
t_unbind : contraire de t_bind

t_look : permet de connaîre l'état courant associé à l'extrêmité de connexion
t_getstate : permet de connaîre l'évènement courant associé à l'extrêmité de connexion
t_getinfo : permet de connaître les paramètres de l'extrêmité de transport qui ont pu changer après l'acceptation d'une ouverture de connexion
t_optmgmt : modification des paramètres d'un protocole

t_listen : demande d'attente d'une requête de connexion
t_accept : acceptation d'une demande d'ouverture de connexion, une nouvelle extrêmité de transport peut être fournie par l'utilisateur, au retour de l'appel système cette extrêmité servira aux échanges de données ultérieurs, on peut spécifier la même extrêmité de transport que celle qui a servi à accepter la connexion
t_snd/t_rcv : émettre/recevoir en mode connecté
t_sndrel : demande de fermeture de connexion sans perte de données
t_rcvrel : indication de fermeture de connexion sans perte de données
t_sndis : demande de fermeture de connexion abrupte ou refus d'ouverture de connexion
t_rcvdis : indication de fermeture de connexion abrupte ou attente de rejet de connexion

t_sndudata/t_rcvudata : émettre/recevoir en mode non connecté
t_rcvuerr : réception d'erreur pour l'émission/réception de données en mode non connecté

t_close : fermeture de connexion avec liberation des ressources associées à l'extrêmité de transport

Etats d'une extrêmité de transport

T_UNINIT : avant création

T_UNBND : créé mais pas d'adresse associée

T_IDLE : en attente de données ou de connexion

T_INCON : ouverture de connexion passive (serveur)

T_OUTCON : ouverture de connexion active (client)

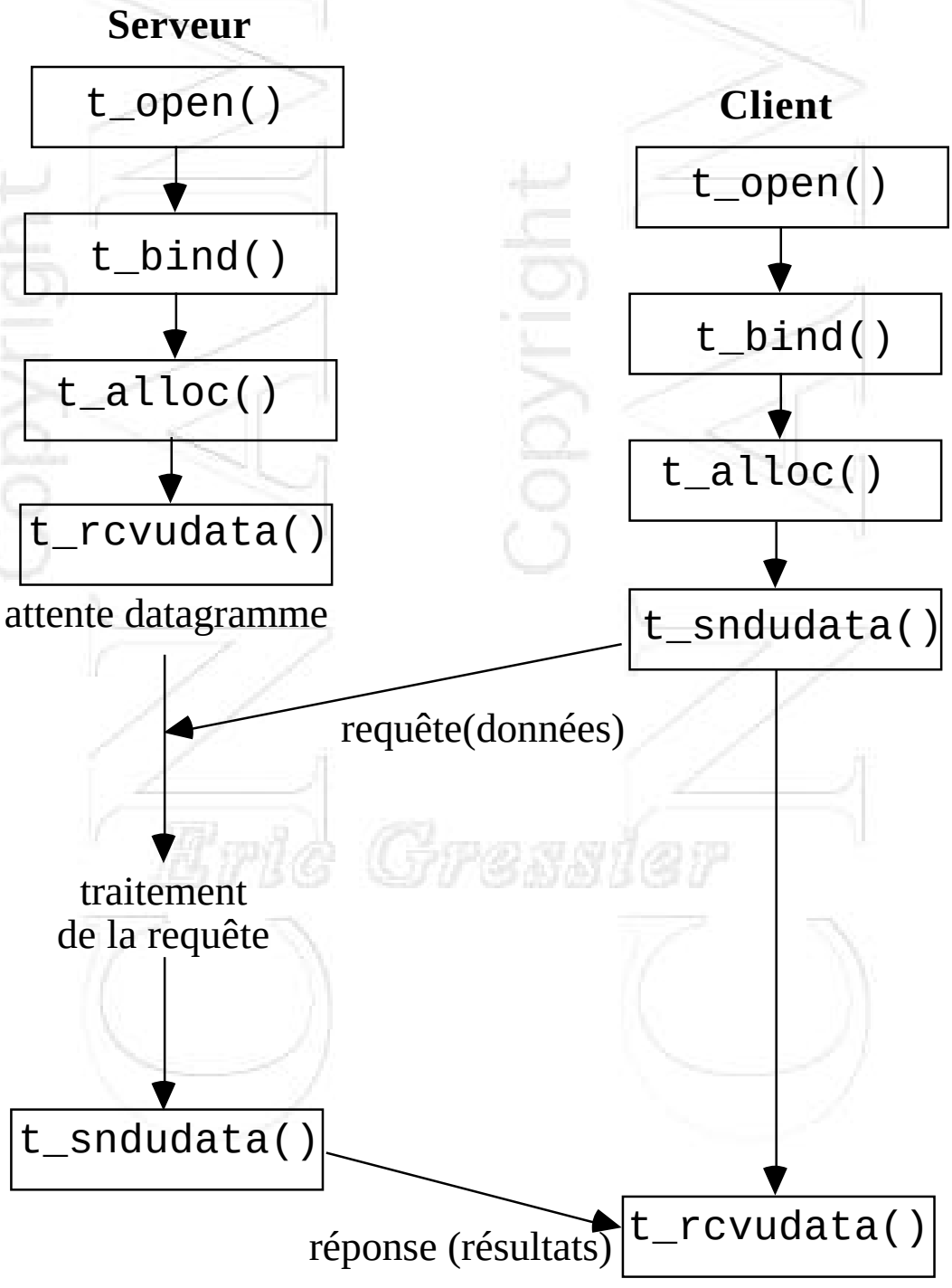
T_DATAXFER : transfert de données possible sur une extrêmité de transport avec connexion établie

T_INREL : fermeture de connexion ordonnée reçue

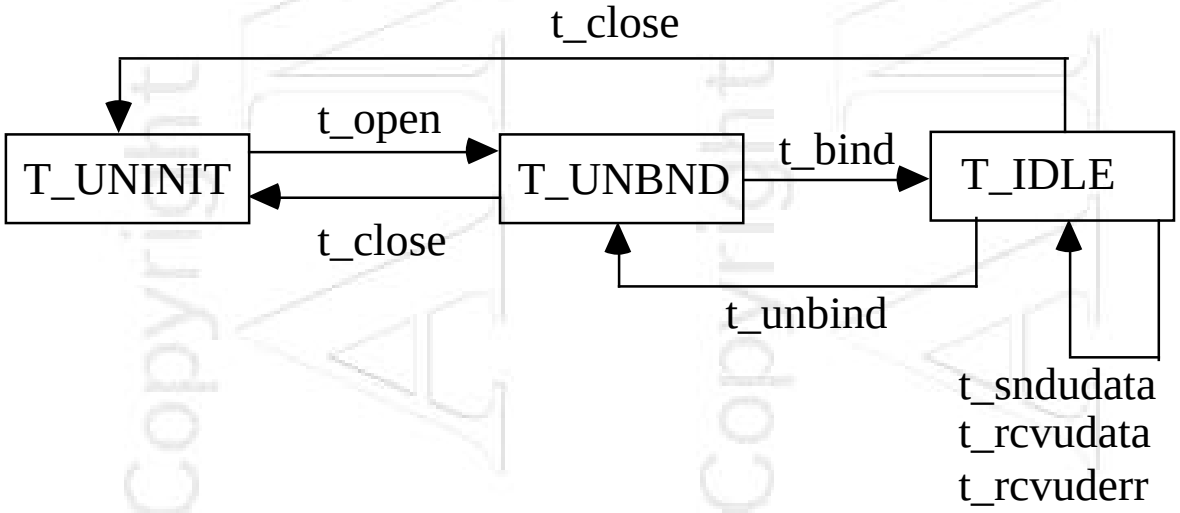
T_OUTREL : fermeture de connexion ordonnée demandée

L'état d'une extrêmité de transport autorise l'utilisateur à se servir de certaines fonctions de la bibliothèque TLI. Ces fonctions en retour modifient l'état de l'extrêmité de transport.

TLI : Client-Serveur en mode non connecté

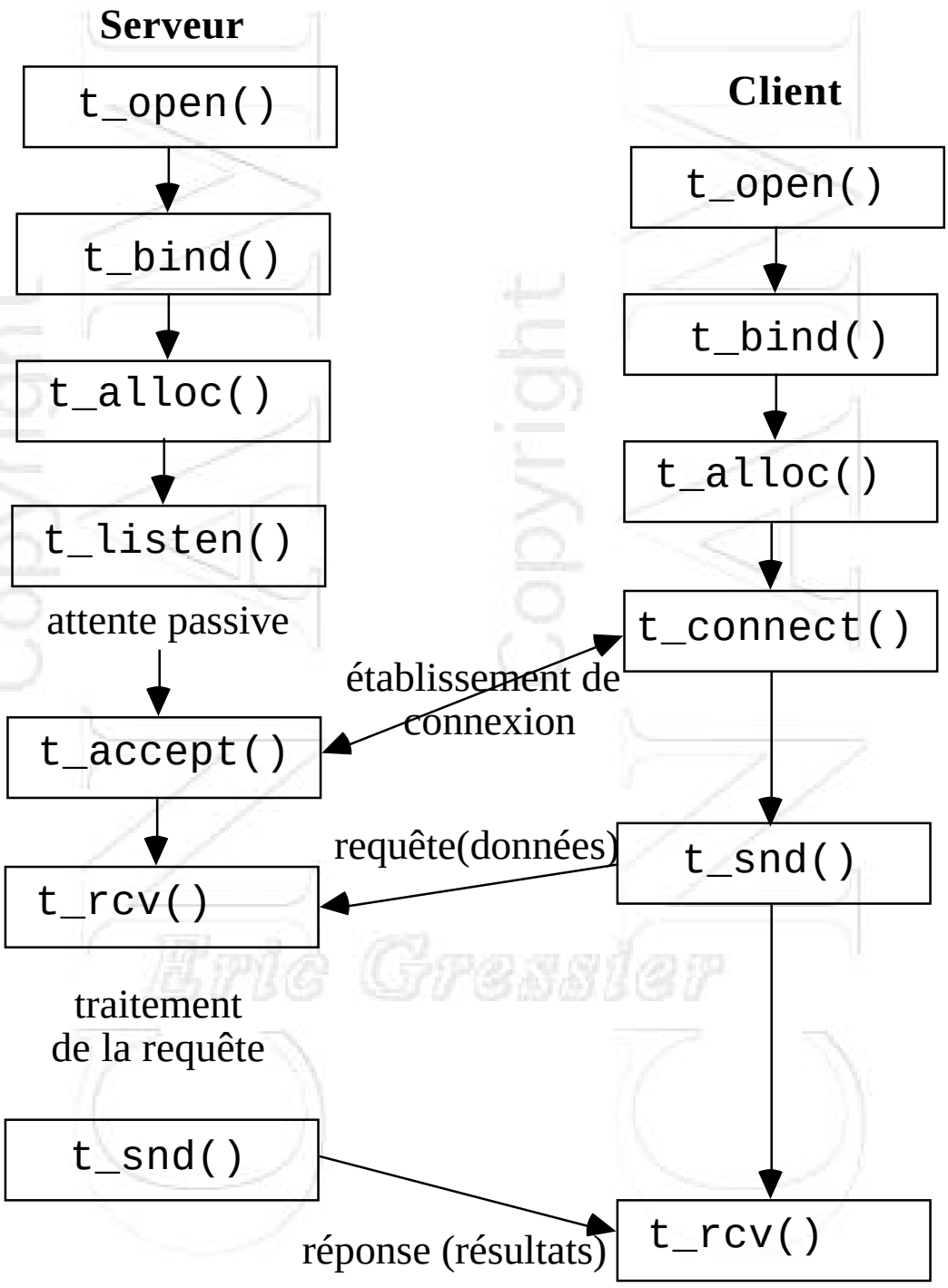


Automate de changement d'états d'une extrêmité de transport pour le mode non connecté[11]



[11] source : cours IIE-CNAM sur l'interface TLI de Laurence Duchien. 1995.

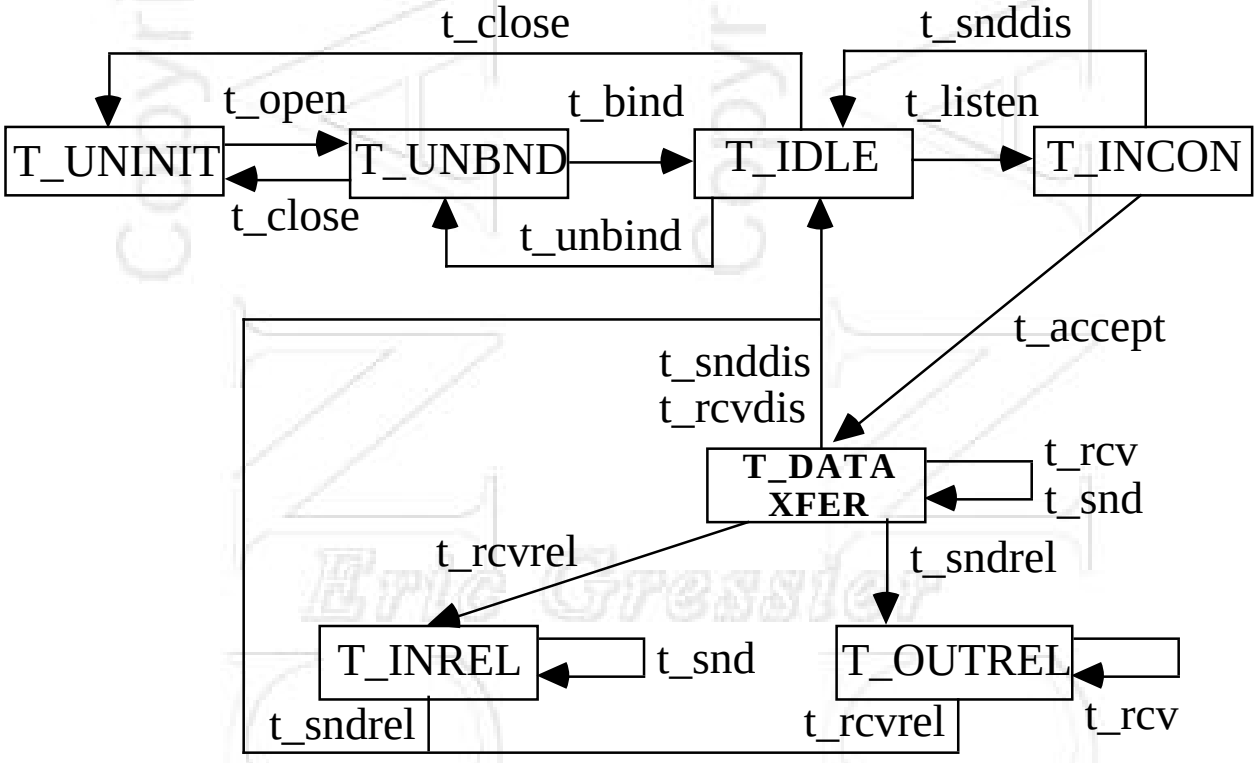
TLI : Client-Serveur en mode connecté



Automate de changement d'états d'une extrêmité de transport pour le mode connecté, côté serveur [12] (1)

Le `t_connect()` ne se comporte pas comme le `connect()` de l'interface socket même s'il a une fonction identique, il y a deux possilités :

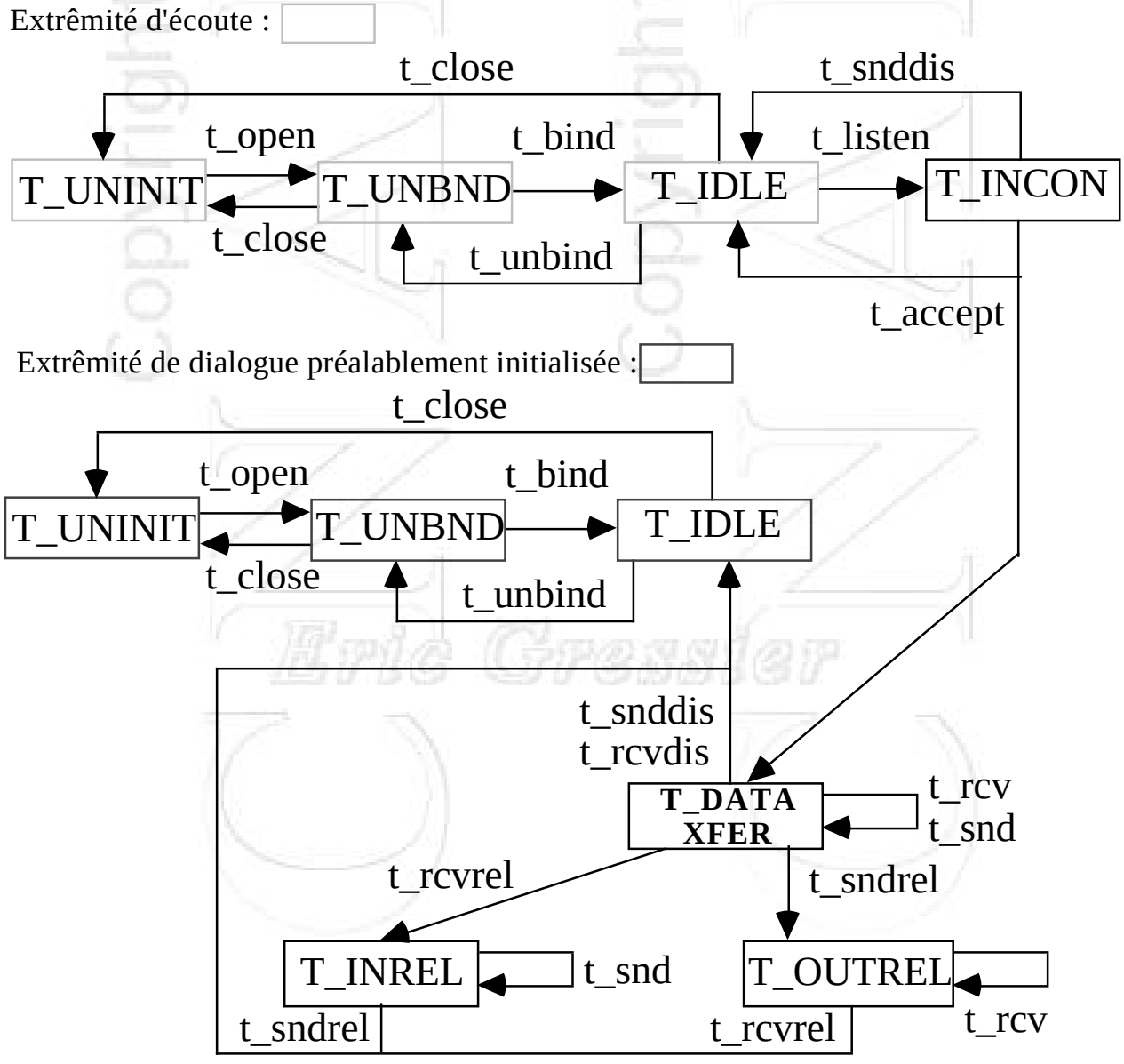
a) On continue le dialogue sur l'extrêmité de transport qui a servi à écouter



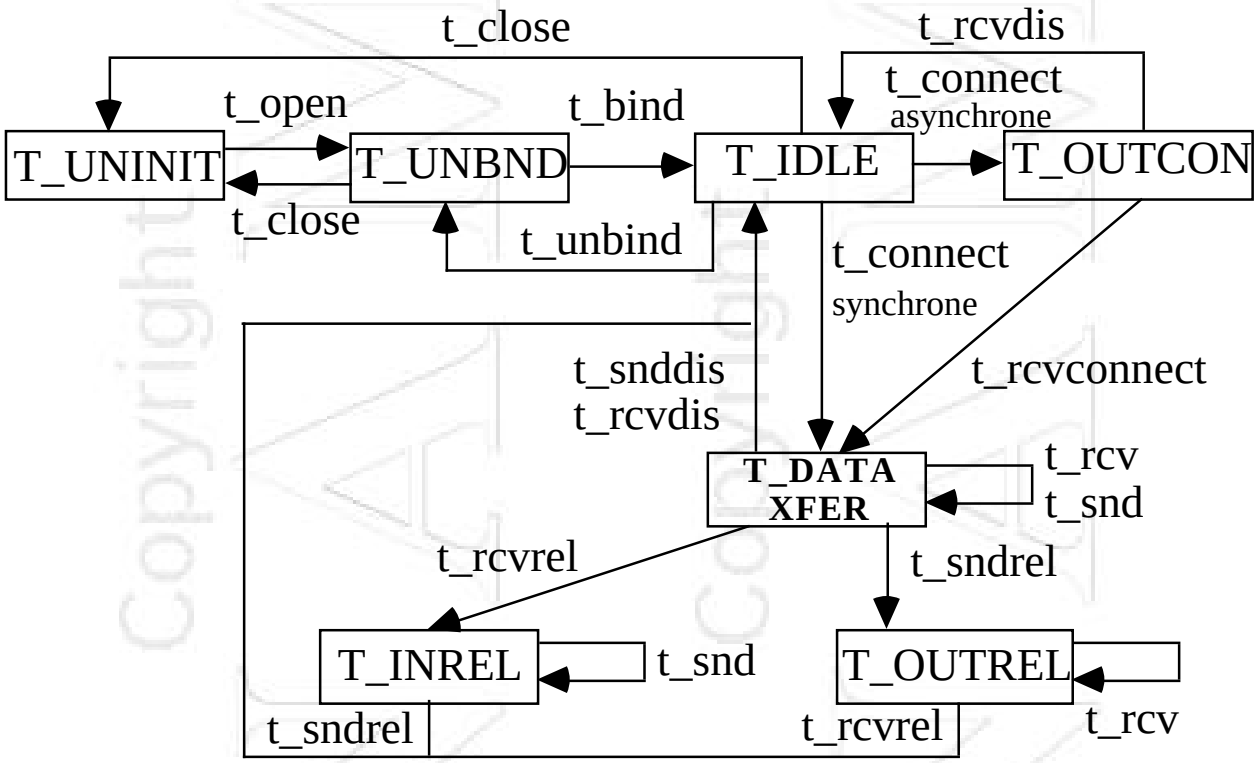
[12] source : cours IIE-CNAM sur l'interface TLI de Laurence Duchien. 1995.

Automate de changement d'états d'une extrêmité de transport pour le mode connecté, côté serveur (2)

b) On continue le dialogue sur une autre extrêmité préalablement créée et intialisée avec une adresse de transport, la première extrêmité est l'extrêmité d'écoute, la seconde est l'extrêmité de dialogue



Automate de changement d'états d'une extrêmité de transport pour le mode connecté, côté client [13]



[13] source : cours IIE-CNAM sur l'interface TLI de Laurence Duchien. 1995.

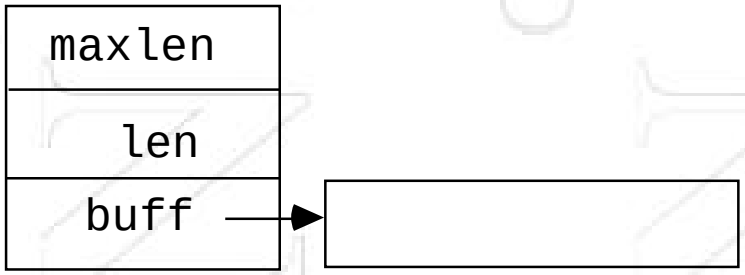
TLI - Primitives d'allocation de ressources

Deux primitives ont un rôle particulier : allocation/désallocation de ressources mémoire pour l'échange de données entre le processus utilisateur et la tête de flot via la bibliothèque TLI (penser à la notion d'IDU du modèle ISO-OSI)

`t_alloc() et t_free()`

Les buffers qui traversent l'interface ont un format spécifique à chaque protocole, mais aussi à chaque fonction de la bibliothèque, ils sont composés d'une ou plusieurs structures de type `netbuf` :

`struct netbuf :`



zone mémoire de format spécifique
au paramètre et à la fonction TLI
impliqués dans l'appel `t_alloc`

t_alloc() (1)

```
#include <tiuser.h>
char *t_alloc(int fd, int structtype, int fields);
```

- `fd` est le résultat retourné par l'appel à `t_open()`

- `structtype` peut être :

`T_BIND` pour la fonction `t_bind()`
`T_DIS` pour les fonctions `t_rcvdis()`
`T_INFO` pour la fonction `t_getinfo()`
`T_OPTMGMT` pour la fonction `t_optmgmt()`
`T_UNITDATA` pour les fonctions `t_*udata()`
`T_UDERROR` pour la fonction `t_rcvuderr()`
`T_CALL` pour les autres fonctions qui nécessite une structure de données de type t_call

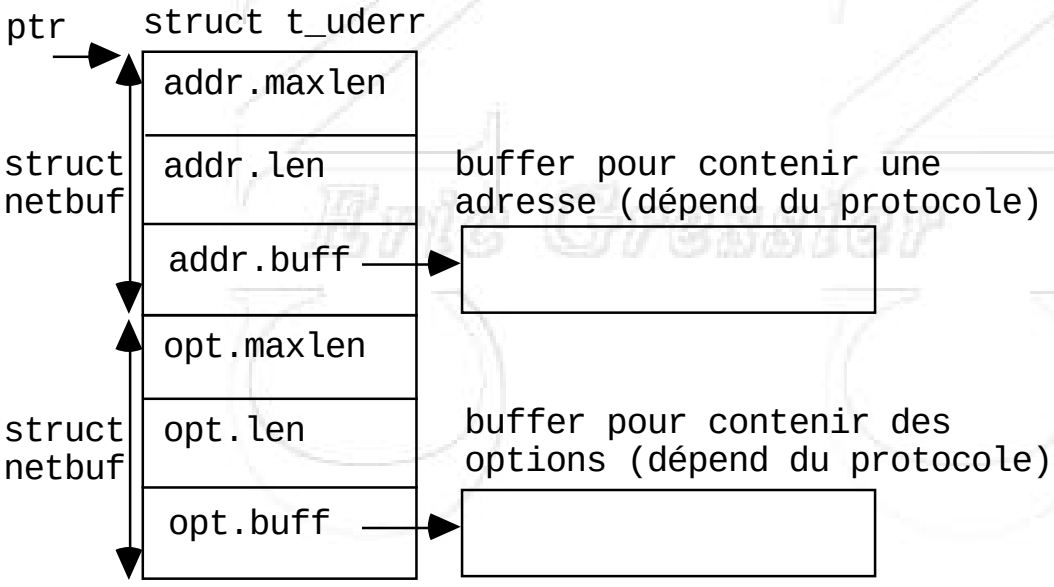
- `fields` permet le contrôle des buffers alloués, plus facile `T_ALL`

t_alloc() (2)

Exemple : allocation d'une structure t_uderr pour la fonction t_rcvuderr

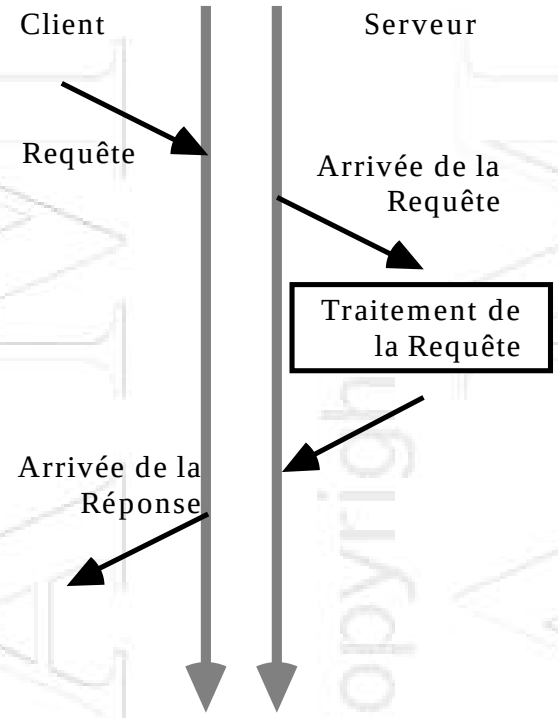
```
int           fd;
struct t_err *ptr;

if ((fd = t_open("/dev/tcp",O_RDWR, (struct t_info*) 0)) < 0)
    err_sys(...);
if ((ptr = (struct t_uderr *)t_alloc (fd, T_UDERR,T_ALL))==NULL)
    err_sys(...);
```

résultat :



7. MODELE Client/Serveur

Principe du Client/Serveur

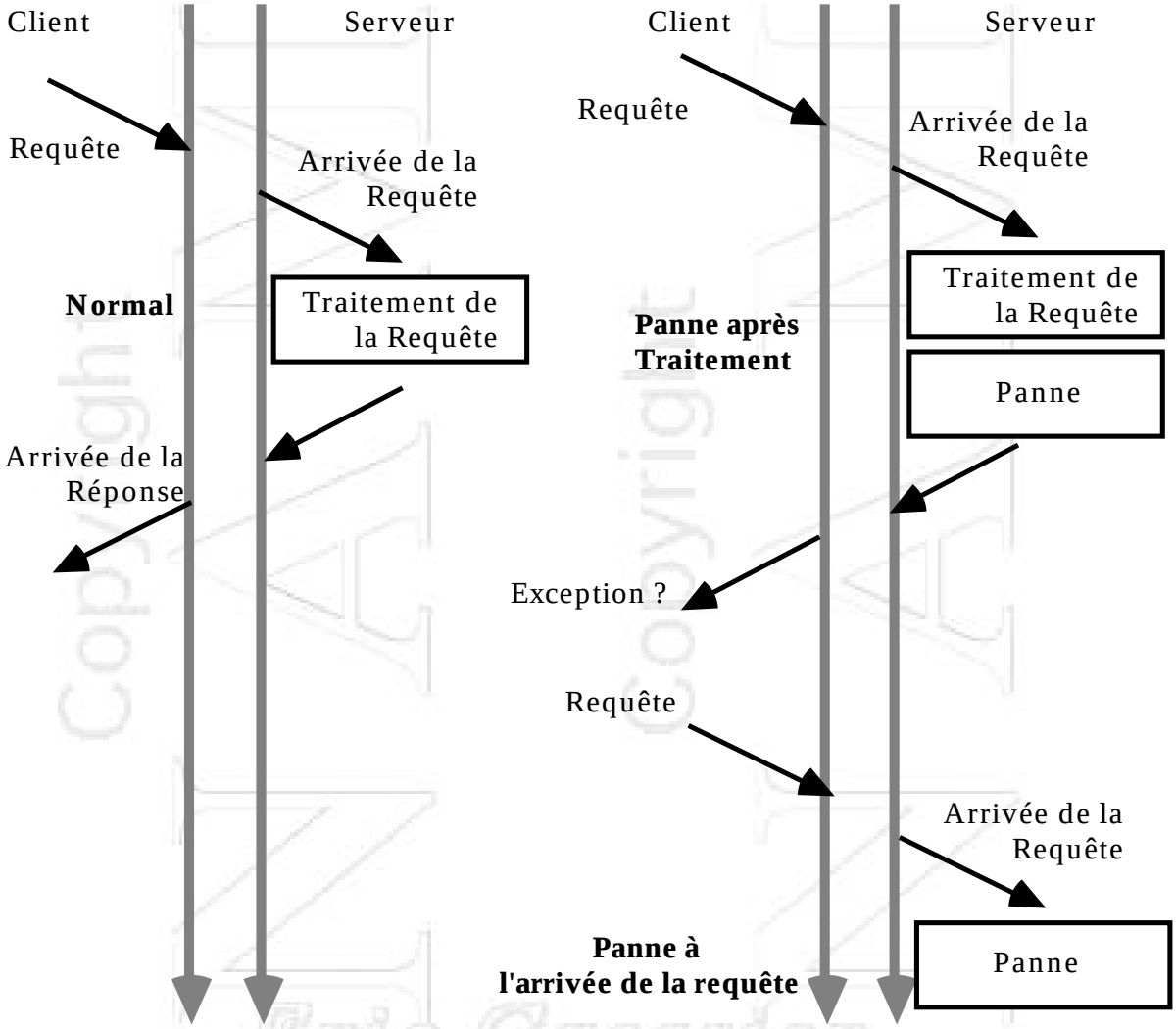


=> Problème de **panne d'un serveur**.

=> **Panne du réseau**

=> Problème de **panne d'un client** qui crée des traitements sans utilité chez le serveur : "orphelins".

Panne de Serveur



Serveur avec ou sans état

Pour remédier à l'effet des pannes de clients ou d'autre serveurs, un serveur peut **maintenir des informations sur** les opérations en cours avec **ses clients** et leur état .... dans ce cas on dit que c'est un **serveur avec état**.

Ces informations permettent :
- d'éviter de traiter deux fois la même requête[14]
  => sémantique d'exécution du service :

au moins une fois,
au plus une fois,
exactement une fois,
peut-être

- de reprendre un traitement à l'endroit où il a été arrêté ou presque (avec des points de reprise)

Quand le serveur ne maintient aucune information sur ses clients, il est dit **serveur sans état**.
=> autonomie de chaque demande de service
(ex ouvrir et fermer un fichier à chaque demande de lecture)

[14] Une requête peut être dupliquée soit à cause des réémissions du client

Panne du Réseau

Mode connecté ou mode datagramme

Pertes de Messages :

- **Perte de la demande :** Armer un délai de garde

- **Perte de la réponse :** Armer un délai de garde, et Problème de la sémantique d'exécution du service

idempotence des requêtes ($f^n = f$)

Le client n'a aucun moyen de savoir ce qu'a fait le serveur ...

même problème à résoudre que celui de la fermeture de connexion

Panne de Client (1)

Eliminer les traitements qui ne correspondent plus à un client demandeur (traitements orphelins)

Extermination :

Avant appel, le client **enregistre dans un journal sur disque** le nom du serveur et l'action qu'il demande.
Après la panne, il reprend le journal pour demander la destruction des traitements en cours.
Inefficace : ne résoud pas le problème des orphelins d'orphelins, ne résiste pas au partitionnement de réseau.

Réincarnation :

On divise le temps en **époques numérotées**, quand une machine redémarre, elle diffuse le fait qu'elle débute une nouvelle époque, ce qui permet aux serveurs de détruire les traitements inutiles à la réception du message diffusé.
Si le réseau est partionné, il reste des orphelins, mais ils sont détectés lors de la réponse par un numéro d'époque invalide.

Panne de Client (2)

Reincarnation douce :

Réincarnation, où le traitement n'est détruit que si le programme (et non la machine) qui a lancé le traitement a disparu.

Expiration :

Une requête dispose d'un certain délai pour s'exécuter. Si ce délai est insuffisant, le serveur doit redemander un nouveau quantum.
Trouver la valeur du délai qui est très différente en fonction des requêtes.

Détruire un orphelin n'est pas simple :

- il détient des **ressources systèmes** (exemple verrous de fichiers) qui peuvent rester indéfiniment **mobilisées**
- ils peuvent avoir lancer d'autres traitements

Edition de liens

Localiser la procédure à exécuter
=
Trouver où est le serveur ?

1. **Edition de liens statique :** L'adresse du serveur est écrite "en dur" dans le code du client.
Difficile en cas de changement : si le serveur migre, s'il change d'adresse, ...

2. **Edition de liens dynamique :** L'adresse n'est connue qu'au moment de l'invocation de la procédure
a. Le serveur **exporte son interface** en envoyant la spécification de ses procédures à un serveur dédié à cette fonction (éditeur de liens) : serveur centralisé ou service réparti. Il est enregistré et connu.
Le serveur donne une référence sur lui-même (adresse réseau, nom unique ...)
b. Le Client s'adresse à l'**éditeur de liens**, pour connaître le serveur qui sert la procédure dont il demande l'exécution. L'éditeur de lien lui donne l'adresse d'un serveur disponible (ils peuvent être plusieurs à pouvoir fournir le même service -> tolérance aux pannes)
c. Le client s'adresse au serveur pour exécuter la procédure souhaitée

Comment servir les clients ?

Pb : un serveur de fichiers

- 1er client : demande un fichier de 200Mo

- 2ème client : demande un autre fichier de 20Ko

Si on sert au fur et à mesure et un par un … le 2ème est pénalisé alors qu'il pourrait être servi plus vite !!!

Quelle performance ?

Solution : Parallèlisme / Concurrence

Concurrence Serveur mode connecté (1)

Serveur non concurrent ou itératif

```
sd = socket ( ...)

bind (sd, ...)

listen (sd,5)

while (1) {
   nsd = accept (sd, ...);
   serv_request (nsd);
   close(nsd);
}
```

Une seule demande de service est traitée à la fois. Les nouvelles demandes sont en attente sur la première socket pendant qu'il sert.

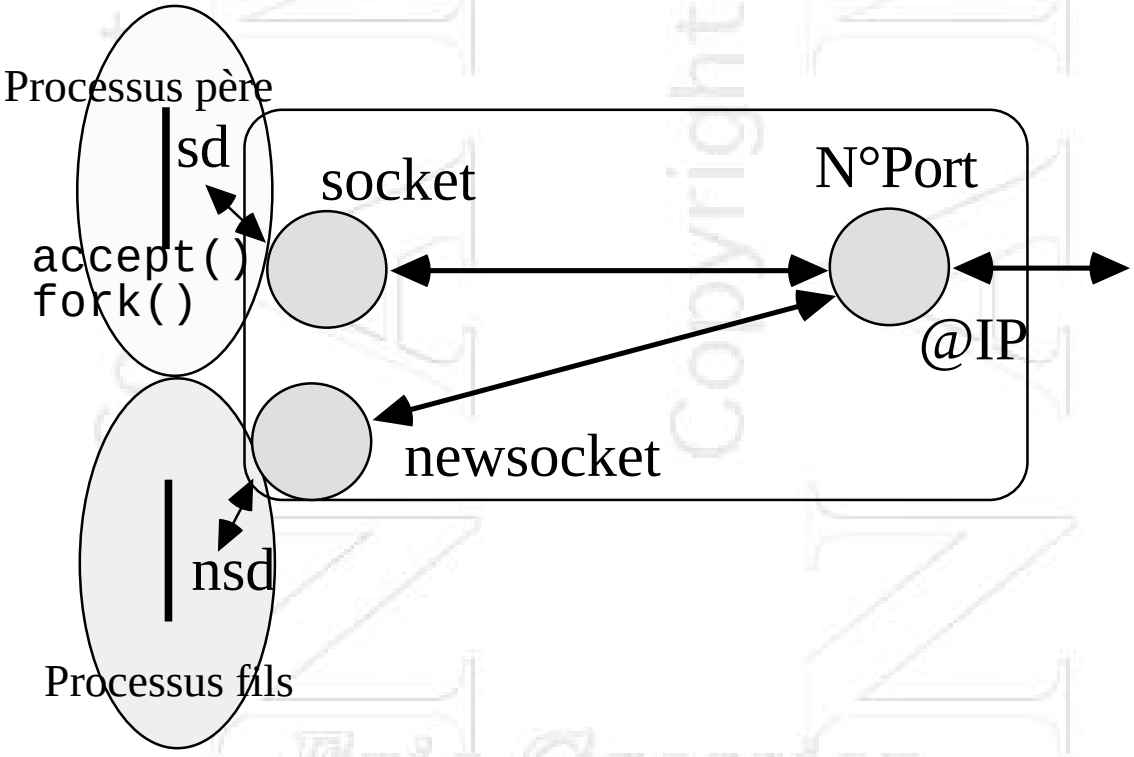
Concurrence Serveur mode connecté (2)

Serveur concurrent

```
sd = socket ( ...)

bind (sd, ...)

listen (sd,5)

while (1) {
   nsd = accept (sd, ...);
   if fork() == 0 {
      close (sd);/*processus fils*/
      serv_request(nsd);
      exit(0);
      }
   close(nsd); /*processus pere*/
}
```

Plusieurs demandes sont traitées en même temps.

Concurrence Serveur mode connecté (3)



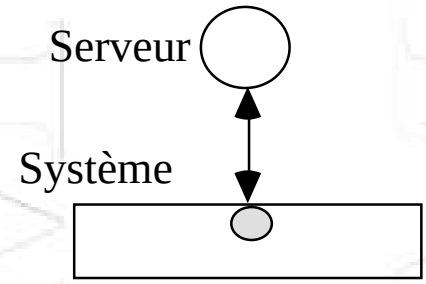
Serveur concurrent en mode non-connecté

```
sd = socket ( ...)

bind (sd, ...)

while (1) {
   recvfrom (sd, ...);
   if fork() == 0 {
      /*processus fils*/
      serv_request(sd);
      sendto (sd, ...)
      exit(0);
      }
}
```

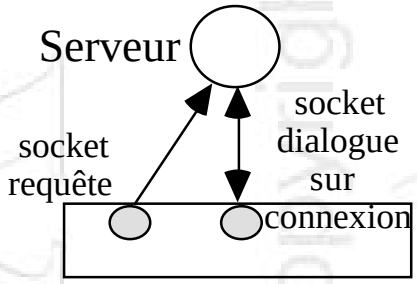
Plusieurs demandes sont traitées en même temps, chaque fils traite une requête et se termine.

Configurations possibles

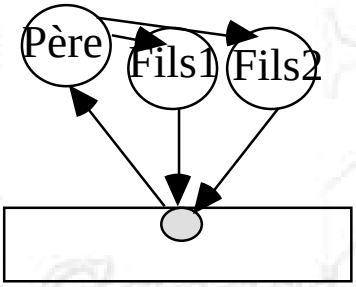
- Serveur itératif mode non connecté



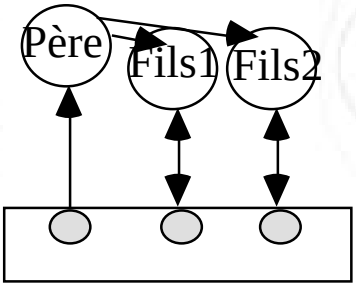
- Serveur itératif mode connecté



- Serveur concurrent mode non connecté



- Serveur concurrent mode connecté



Serveur en mode connecté

Avantages :

- Les **problèmes de communication sont gérés automatiquement** par le protocole grace à la gestion de la connexion

- Primitives d'émission et de réception simples

Désavantages :

- Le **mode connecté implique plus d'opérations**, par exemple la fermeture de connexion s'effectue en 3 messages (three way handshake).

- **Blocage du serveur quand le client tombe en panne ...** la connexion utilise des ressources, si le client tombe en panne après la dernière réponse du serveur, le serveur ne peut pas le savoir (il n'y a pas de messages échangés sur une connexion inutilisée), il va maintenir les ressources d'une connexion inutile, lorsque le client va redémarré, il ouvrira une nouvelle connexion, **encore plus grave si le serveur est itératif ...**

- **Consommateur de ressources système** : une socket par connexion

- **Mode flot d'octets**, il faut délimiter les messages dans le flot

Lenteur et Ressources consommées

Serveur en mode non connecté

Avantages :

- Moins de ressources système consommées

- Permet la **diffusion**

Désavantage :

- Gérer les problèmes de communication : protocole de récupération d'erreur entre applications :
  . Timer + retransmission
  . N° de messages pour détecter les duplications

On refait la couche Transport !!!

Se débarrasser des Zombies

Les fils lorsqu'ils se terminent (exit(0)), attendent un signal du père avant de disparaître ... ils deviennent **processus zombie**.

On effectue dans la partie initialisation du serveur l'opération suivante :

```
signal(SIGCHLD, achever);
```

avec pour achever :

```
int achever()
{
union wait status;

while(wait3(&status,WNOHANG,(struct rusage *)0) >= 0)
       /* vide */;
}
```

C'est une incantation magique … qui se transmet de hackers en hackers .... elle n'est pas garantie, la tradition orale se déforme parfois :-). Toutefois, Richard Stevens dans Unix Network Programming donne cette incantation p82 pour 4.3BSD (parent d'Ultrix).

INETD un exemple de Serveur (1)

Le processus **inetd** est un superserveur qui contrôle le lancement d'autres serveurs : ftp, telnet, rlogin, …

Il peut être utilisé pour lancer des processus serveur qui communiquent aussi bien par TCP que par UDP.

1. **inetd** crée toutes les sockets pour chaque service qu'il offre à travers le réseau (décrit dans /etc/inetd.conf)

2. Il effectue le bind pour ces sockets, puis le listen() seulement pour les sockets en mode connecté.

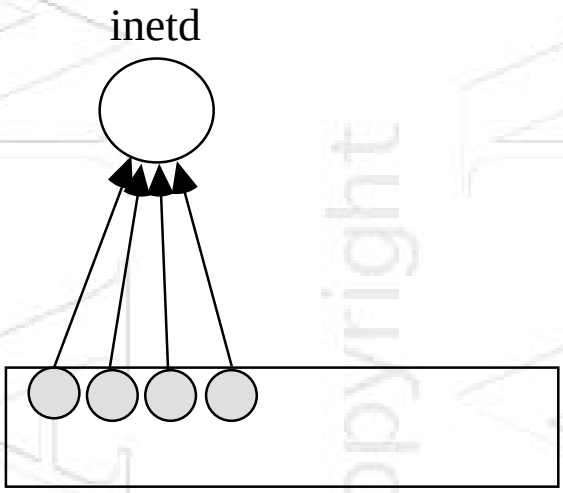
3. Il effectue un select() pour se mettre en attente des demandes de service.

4. Il crée un processus fils pour gérer chaque demande. Un serveur concurrent n'utilise habituellement que le fork(), inetd utilise le fork() suivi d'un exec() pour lancer le serveur approprié.

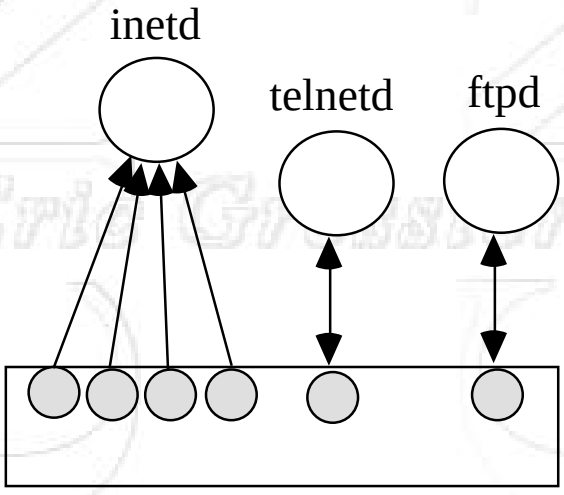
5. inetd retourne sur le select() pour attendre une nouvelle requête.

INETD un exemple de Serveur (2)

Avant la demande de service :



Après demande de service :



Maîtrise de la concurrence

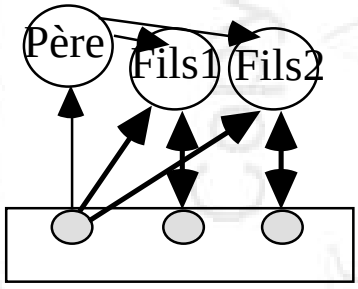
- Allocation de processus à la demande

Surcharge et retard du service

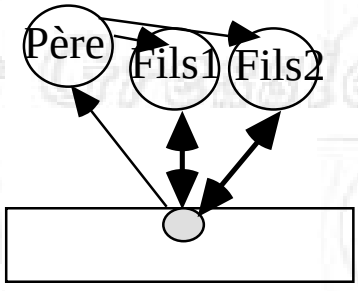
- Pré-allocation : **Pool de Processus**

Le processus père crée toutes les ressources pour le pool de processus, puis il crée les processus fils. Le père n'a plus aucun rôle spécifique après.

en mode connecté : les fils se bloquent sur un **accept()**, un seul fils à la fois est débloqué lorsqu'arrive une connexion entrante



en mode non-connecté : les fils sont tous en attente sur une socket, un parmi ceux qui sont prêts sert la demande et répond



gain de temps : il n'y a plus de création de processus

Concurrence pour les Clients

On se pose le problème de concurrence chez les serveurs, il se pose aussi pour les clients.

Il permet d'améliorer le parallèlisme !!!

->

. Un processus pour chaque fonction en cours.
mise en commun de données :
mémoire partagée, sémaphores, files de messages, ...
difficile

. Un seul processus qui utilise le **mécanisme du select** pour gérer les différents évènements :