I.U.T. Amiens
Département Informatique

Année Universitaire 2003/2004

Unix : Programmation Réseau

C. Drocourt – Màj le 08/12/2003

Sommaire

1 - Introduction et Rappels

1.1 - LE MODELE OSI (Open System Interconnection)

Le modèle ISO-OSI, de loin le plus répandu, décrit des niveaux de transmission mais pas des normes. Il divise l'ensemble des protocoles en sept couches indépendantes entre lesquelles où sont définis deux types de relations ; les relations verticales entre les couches d'un même système (interfaces) et les relations horizontales relatives au dialogue entre deux couches de même niveau.

Messages Segments	7	Application
	6	Présentation
	5	Session
Datagrammes	4	Transport
Paquets	3	Réseau
Trames	2	Liaison de Données
Bits	1	Physique

La couche Liaison de Données se découpe en deux souscouches :

- *Logical Link Control*
- *Medium Access Control*

Description des couches :

1. Physique -> Brochage, Tensions...
2. Liaison de Données -> Structuration, Correction, Accès au medium...
3. Réseau -> Routage, Fragmentation. Communication entre systèmes
4. Transport -> Communication entre processus
5. Session -> Connexion, Déconnexion.
6. Présentation -> Structuration des données typées.
7. Application -> Communication entre "utilisateurs "

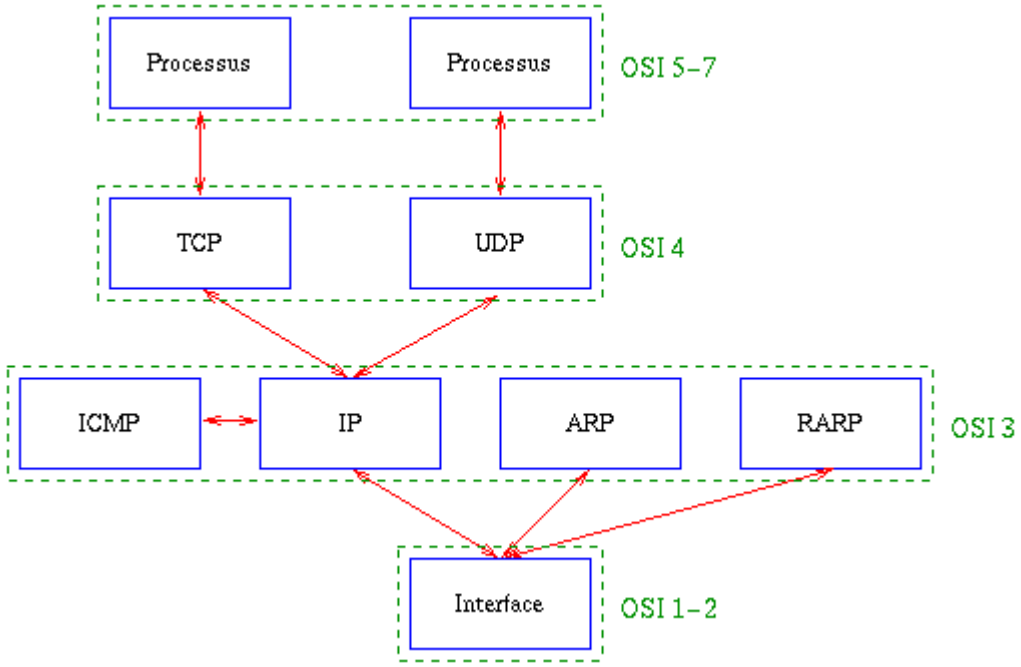
1.2 - TCP/IP

1.2.1 - Description

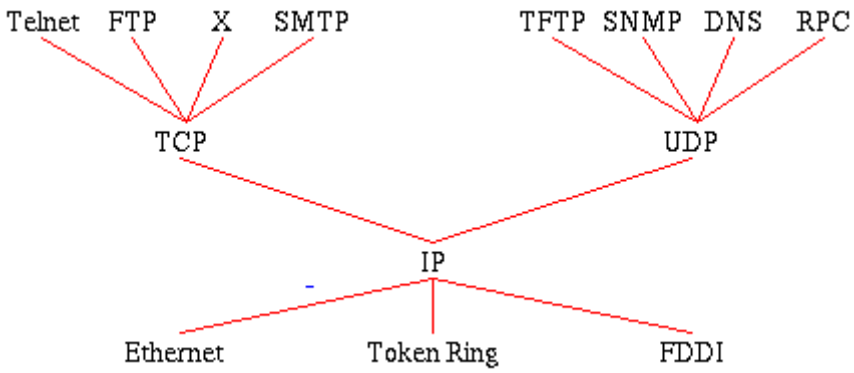
TCP/IP fournit un protocole standard pour résoudre le problème de connexion entre différents réseaux. TCP (Transfert Contrôle Protocole) se charge du transport de bout en bout pour toute application alors que IP (Internet Protocole) est responsable du routage à travers le réseau. D'autres protocoles sont aussi inclus comme ARP (Address Resolution Protocole), FTP (File Transfert Protocole), SMTP (Simple Mail Transfert Protocole),...

TCP/IP est structuré en quatre niveaux :

- L'interface réseau (1 et 2 du modèle OSI)
- Le routage (3 du modèle OSI
- Le transport (4 et 5 du modèle OSI)
- L'application (5, 6 et 7 du modèle OSI).



Il a été mis au point dans le cadre du projet DARPA au début des années 1980. Il est le précurseur des modèles en couches mais sa structure ne lui permet pas de s'intégrer à la norme OSI (Open System Interconnexion) de l'ISO (International Standard Organisation), TCP/IP a été bâti sur l'expérience, avec le souci permanent de la performance.



1.2.2 - Les adresses IPs

L'adresse TCP/IP d'une machine est une adresse de niveau réseau codée sur 32 bits (ie 4 octets) qui est en général notée sous la forme de 4 chiffres séparés par des points. On parle de notation en décimal pointé. Chaque champ, qui représente un octet, peut prendre des valeurs entre 0 et 255, Exemple : 192.93.116.3

L'adresse IP est constituée d'un champ numéro de réseau (1,2 ou 3 octets) et d'un champ numéro de machine dans le réseau (3,2 ou 1 octets)

@ip = #réseau + #machine

#réseau : attribué par un organisme officiel : le NIC aux US (ou ses représentants)

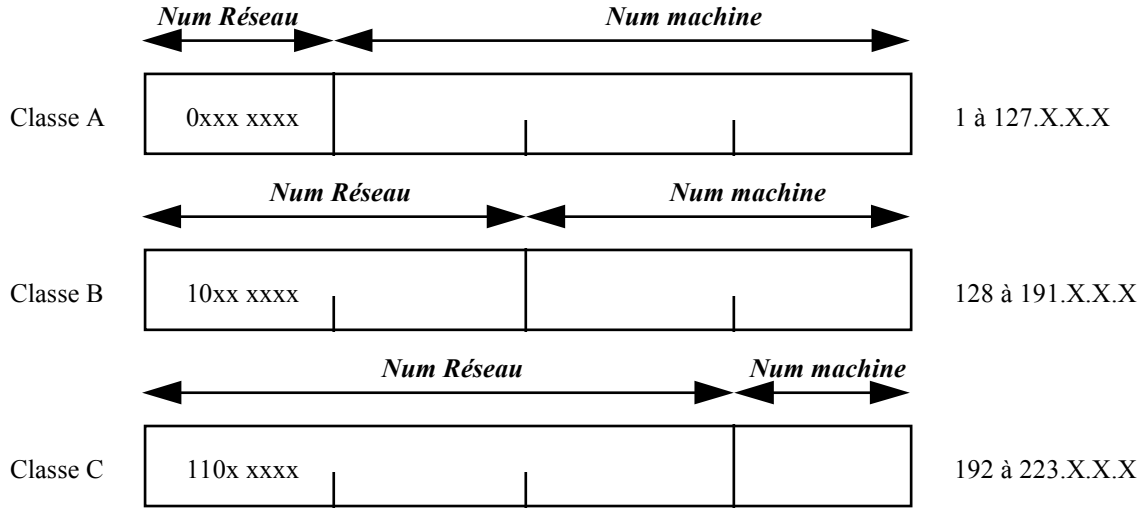
#machine : attribué localement par le gestionnaire du réseau (nota: il est possible de découper le champ de droite en sous-réseau+machine

Les réseaux TCP/IP sont rangés en 3 classes A, B ou C en fonction de la taille du champ numéro de réseau:

classe A : 1 à 127 .X.X.X

classe B : 128 à 191 .X.X.X

classe C : 192 à 223 .X.X.X (les adresses >223 sont réservées à d'autres usages)



Classes d'adresses TCP/IP

Le nombre de machines dans le réseau dépend donc de la classe du réseau. Chaque octet du champ machine peut prendre des valeurs entre 1 et 254.

Les valeurs 0 (tous les bits à 0) et 255 (tous les bits à 1) sont réservées:

- un champ machine tout à 0 sert à désigner le numéro de réseau (notamment pour le routage)
- un champ tout à 1 indique un message de broadcast adressé à toutes les machines IP du réseau.

Sur les fichiers de configuration on a un masque réseau (netmask) qui, associé à l'adresse IP, indique le champ à prendre en compte pour #réseau (bits à 1) , et celui à prendre en compte pour #machine (bits à 0). ex: dans un réseau de classe A sans sous-réseau : netmask=255.0.0.0

1.2.3 - Les adresses réservées

0.0.0.0 est réservée pour la route par défaut. Tous les paquets destinés à un réseau inconnu, seront dirigés vers cette route.

127.0.0.0 est réservée au trafic IP de la machine locale. Une interface locale porte en générale l'adresse 127.0.0.1 appelée adresse de "loopback".

Certaines adresses peuvent également être librement utilisées pour monter un réseau privé:

A	10.0.0.0		
B	172.16.0.0	à	172.31.0.0
C	192.168.0.0	à	192.168.255.0

Aucun paquet provenant de ces réseaux ou à destination de ces réseaux, ne sera routé sur l'internet.

1.2.4 - La commande ifconfig (Interface Configuration)

La commande ifconfig permet de configurer les interfaces réseaux et d'associer à chacune d'elle une adresse IP, à noter qu'il est possible d'associer plusieurs adresses IPs à une seule interface réseau. Elle permet également lorsqu'elle est utilisée sans arguments de consulter les différentes interfaces disponibles.

1.3 - ICMP

1.3.1 - Description

Le besoin : Le protocole ICMP (Internet Control Message Protocol) permet d'envoyer des messages de contrôle ou d'erreur vers d'autres machines ou passerelles. ICMP rapporte les messages d'erreur à l'émetteur initial. Beaucoup d'erreurs sont causées par l'émetteur, mais d'autres sont dûes à des problèmes d'interconnexions rencontrées sur l'Internet : machine destination déconnectée, durée de vie du datagramme expirée, congestion de passerelles intermédiaires. Si une passerelle détecte un problème sur un datagramme IP, elle le détruit et émet un message ICMP pour informer l'émetteur initial. Les messages ICMP sont véhiculés à l'intérieur de datagrammes IP et sont routés comme n'importe quel datagramme IP sur l'internet. Une erreur engendrée par un message ICMP ne peut donner naissance à un autre message ICMP (évite l'effet cummulatif).

1.3.2 - La commande ping

Envoie une suite de paquets de demande d'écho à une machine distante et affiche ensuite des statistiques sur l'opération réalisée. Permet de tester si une machine distante est en vie. Par défaut, la commande s'arrête par un <control-c>.

```
$ping tux
PING tux.iut.fr (172.20.0.23) from 172.20.0.22 : 56(84) bytes of data.
64 bytes from 172.20.0.23: icmp_seq=0 ttl=255 time=0.3 ms
64 bytes from 172.20.0.23: icmp_seq=1 ttl=255 time=0.2 ms
64 bytes from 172.20.0.23: icmp_seq=2 ttl=255 time=0.2 ms
64 bytes from 172.20.0.23: icmp_seq=3 ttl=255 time=0.2 ms

--- tux.iut.fr ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.3 ms
$
```

1.4 - Les protocoles UDP et TCP

1.4.1 - UDP : Livraison non fiable, sans connexion

UDP (User Datagram Protocol) permet à une application d'envoyer des messages à une autre en mode datagramme non connecté. Les paquets UDP peuvent arriver dans le désordre, au même ne pas arriver : les états d'arrivée d'un paquet n'est pas géré par ce protocole.
Les démons écoutent à des numéros de port définis dans le fichier */etc/services* ou à d'autre numéros, auquel cas les services offerts par ces démons sont **privés**.

1.4.2 - TCP : Transport de flot fiable

Le protocole TCP (Transmission Control Protocol) est utilisé pour réaliser une connexion de type circuit virtuel, connue sous le nom de connexion en flot.

- Accusé de réception pour chaque paquet,
- Réémission du paquet s'il n'est pas acquittée au bout d'un certain temps,
- Paquets numérotés qui peuvent donc être remis dans le bon ordre s'ils arrivent dans le désordre.

Les démons écoutent à des numéros de port définis dans le fichier */etc/services* ou à d'autre numéros, auquel cas les services offerts par ces démons sont **privés**.
On remarquera que des services tels que *telnet* ou *rlogin* sont fournis à travers le protocole TCP, contrairement à des services tels que *talk* ou *nfs* qui utilisent UDP.

1.4.3 - Identification du service : les ports

les adresses IP désignent les machines entre lesquelles les communications sont établies. Lorsqu'un processus désire entrer en communication avec un autre processus, il doit adresser le processus s'exécutant cette machine. L'adressage de ce processus est effectué selon un concept abstrait indépendant du système d'exploitation des machines car :

- Les processus sont créés et détruits dynamiquement sur les machines,
- Il faut pouvoir remplacer un processus par un autre (exemple reboot) sans que l'application distante ne s'en aperçoive,
- Il faut identifier les destinations selon les services offerts, sans connaître les processus qui les mettent en oeuvre,
- Un processus doit pouvoir assurer plusieurs services.

Ces destinations abstraites permettant d'adresser un service applicatif s'appellent des ports de protocole. L'émission d'un message se fait sur la base d'un port source et un port destinataire.
Les processus disposent d'une interface système leur permettant de spécifier un port ou d'y accéder (socket). Les accès aux ports sont généralement synchrones, les opérations sur les ports sont tamponnés (files d'attente).

Les protocoles TCP et UDP utilisent des numéros de port séparés ; ainsi TCP/5 est totalement distinct de UDP/5. Les services TCP et UDP disponibles sont listés dans le fichier */etc/services* mais d'autres services peuvent être ajoutés sur une machine : la connaissance de leur numéro de port et de leur protocole permettra alors à une application cliente de s'y connecter.
Certains services sont offerts à la fois par TCP et UDP : par convention, ils ont le même numéro de port.

1.4.4 - La commande netstat

Elle affiche des informations sur les connections locales et réseaux actuellement ouvertes et/ou utilisées (sockets) avec les numéros de ports. Exemple :

```
$ netstat
Connexions Internet actives (sans serveurs)
Proto Recv-Q Send-Q Adresse locale          Adresse distante        Etat
tcp        0    200 etud.iut.fr:ssh         fgw2.iut.fr:61996       ESTABLISHED
Sockets du domaine UNIX actives(sans serveurs)
Proto RefCpt Indicatrs   Type       Etat          I-Node Chemin
unix  9      [ ]         DGRAM                    985    /dev/log
unix  2      [ ]         DGRAM                    21562
unix  2      [ ]         DGRAM                    1729
unix  2      [ ]         DGRAM                    1646
unix  2      [ ]         DGRAM                    1498
unix  2      [ ]         DGRAM                    1256
unix  2      [ ]         DGRAM                    1062
unix  2      [ ]         DGRAM                    994
$
```

1.4.5 - Le fichier /etc/services

Le fichier /etc/services est une simple base de données qui associe des noms compréhensibles par l'homme à des ports service compréhensibles par la machine. Son format est tout à fait simple. Le fichier est un fichier texte dont chaque ligne représente une entrée de la base de données. Chaque entrée comprend trois champs séparés par des caractères espace ou tabulation. Ces champs sont :
nom port/protocole alias # commentaire

nom : Un simple mot qui représente le service décrit.

port/protocole : ce champ est divisé en deux.
- port : Un nombre qui spécifie le numéro de port où le service désigné sera disponible. La plupart des services ont des numéros assignés. Ils sont décrits dans la RFC-1340.
- protocole : C'est soit tcp soit udp.

Il est important de noter qu'une entrée comme 18/tcp est très différente de 18/udp et qu'il n'y a pas de raisons techniques que le même service existe sur les deux. Normalement le bon sens prévaut et c'est vraiment pour un service particulier disponible à la fois sur tcp et udp que vous verrez une entrée pour les deux.

alias : Autre nom qui peut être utilisé pour désigner ce service.

Tout texte apparaissant après le caractère `#' est ignoré et traité comme commentaire.

Exemple : un extrait du fichier */etc/services*

```
ftp             21/tcp
ssh             22/tcp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
```

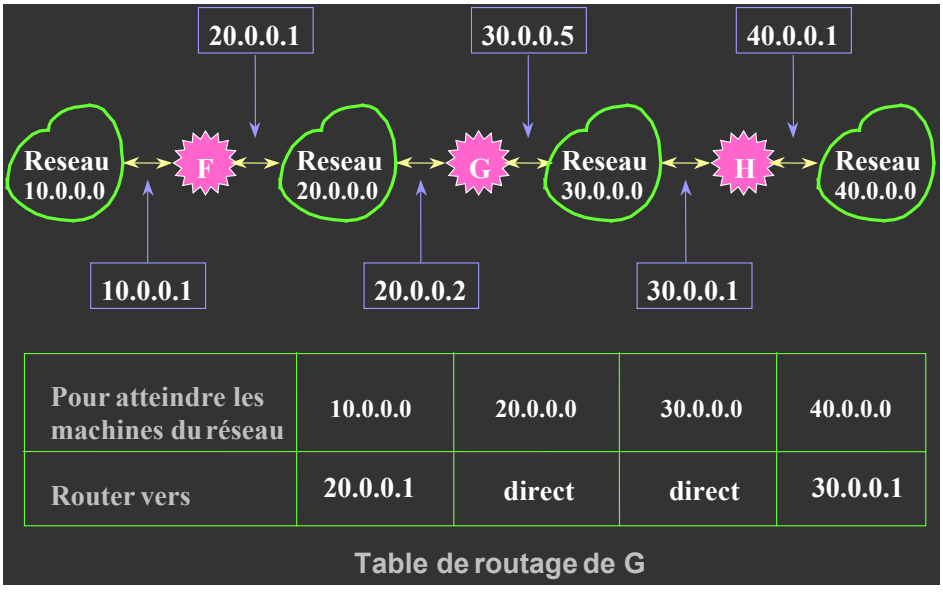
1.5 - Le routage

Le routage est le processus permettant à un datagramme d'être acheminé vers le destinataire lorsque celui-ci n'est pas sur le même réseau physique que l'émetteur. Le chemin parcouru est le résultat du processus de routage qui effectue les choix nécessaires afin d'acheminer le datagramme.
Les routeurs forment une structure coopérative de telle manière qu'un datagramme transite de passerelle en passerelle jusqu'à ce que l'une d'entre elles le délivre à son destinataire. Un routeur possède deux ou plusieurs connexions réseaux tandis qu'une machine possède généralement qu'une seule connexion.
Machines et routeurs participent au routage :

- Les machines doivent déterminer si le datagramme doit être délivré sur le réseau physique sur lequel elles sont connectées grace à leur adresse IP et au masque de réseau (routage direct), ou bien si le datagramme doit être acheminé vers une passerelle; dans ce cas (routage indirect), elle doit identifier la passerelle appropriée.
- Les passerelles effectuent le choix de routage vers d'autres passerelles afin d'acheminer le datagramme vers sa destination finale.

Les tables de routage IP, pour des raisons évidentes d'encombrement, renseignent seulement les adresses réseaux et non pas les adresses machines.

Typiquement, une table de routage contient des couples (R, P) où R est l'adresse IP d'un réseau destination et P est l'adresse IP de la passerelle correspondant au prochain saut dans le cheminement vers le réseau destinataire. La passerelle ne connaît pas le chemin complet pour atteindre la destination. Pour une table de routage contenant des couples (R, P) et appartenant à la machine M, P et M sont connectés sur le même réseau physique dont l'adresse de niveau réseau (partie Netid de l'adresse IP) est R.



Pour atteindre les machines du réseau	10.0.0.0	20.0.0.0	30.0.0.0	40.0.0.0
Router vers	20.0.0.1	direct	direct	30.0.0.1

Table de routage de G

1.5.1 - La commande route

Sous unix pour ajouter ou supprimer une route on utilise la commande route en précisant la destination (machine ou reseau), le masque de reseau, l'adresse de passerelle et l'interface réseau a

utiliser. Une machine pouvant posseder plusieurs adresses IPs ainsi que plusieurs interfaces réseaux (cas par exemple d'un firewall) il peut y avoir plusieurs configuration a ajouter à la table de routage. Pour consulter cette table on utilise la même commande sans arguments. Exemple :

```
$route
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
194.57.105.1    *               255.255.255.255 UH    0      0        0 eth1
172.20.0.1      *               255.255.255.255 UH    0      0        0 eth0
194.57.105.0    *               255.255.255.0   U     0      0        0 eth1
172.20.0.0      *               255.255.0.0     U     0      0        0 eth0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         194.57.105.4    0.0.0.0         UG    0      0        0 eth1
$
```

1.6 - Le DNS

Lorsque les machines communiquent sur un réseau informatique, c'est toujours par l'utilisation d'une adresse (IP ou autre) source ou destination. Mais ces adresses bien que nécessaires, sont difficiles à mémoriser et ne permettent pas de souplesse dans les configurations des stations.
Pour quelqu'un de normalement constitué, il est difficile de se souvenir de 155.124.198.56, alors que www.victim.com sera assez aisé à mémoriser. C'est le but du protocole DNS : fournir une association (adresse IP, nom FQDN) et inversement.
Le service DNS est donc utilisé pour la « résolution de noms ». Cette opération consiste à fournir aux clients DNS qui en font la demande une association adresse IP, un nom symbolique, et vice-versa. C'est à dire qu'on pourra demander à un DNS : « Quelle est ou quelles sont les adresses IP de www.yahoo.com ». Le serveur DNS réalisera alors diverses opérations afin de vous répondre, souvent en contactant différentes entités sur l'Internet pour trouver la réponse.

1.6.1 - La commande hostname

Sous unix pour connaître le nom de la machine locale on utilise la commande hostname :

```
$hostname
tux.iut.fr
$
```

1.6.2 - Le fichier /etc/host.conf

Il indique quels services doivent être utilisés pour la résolution de noms et dans quel ordre.
Dans ce fichier, chaque option doit se trouver sur une ligne séparée. Les champs peuvent être délimités par des espaces ou des tabulations. Le signe dièse (#) introduit un commentaire qui s'étend jusqu'à la fin de la ligne. Les options disponibles sont :

order : Détermine l'ordre dans lequel les services vont être sollicités. Les valeurs valide s sont *bind* pour l'interrogation du serveur de nom, *hosts* pour une recherche dans le fichier /etc/hosts, et *nis* pour utiliser NIS. Tous peuvent être spécifiés éventuellement.

multi : Prend les arguments *on* ou *off*, indiquant si un hôte cité dans le fichier /etc/hosts est autorisé à posséder plusieurs adresses IP ou non. Cette option n'a aucun effet sur les requêtes DNS ou NIS.

nospoof : Les tentatives d'envoi d'un faux nom sont appelées le *spoofing*. Pour se préserver, le resolver peut être configuré pour tester si l'adresse IP originale est vraiment associée avec le nom

obtenu. Si ce n'est pas le cas, le nom est rejeté et une erreur est retournée. Ce comportement est mis en service en mettant *nospoof on*.

alert : Cette option prend les arguments on ou off. Si elle est en service (on), toute tentative de *spoofing* sera enregistrée dans les fichiers de trace du système via syslog.

trim : Prend en argument un nom de domaine qui sera éliminé des noms d'hotes avant la recherche. Très utile pour les entrées hosts où on ne trouve que des noms de machines sans domaine local.

1.6.3 - Fichier /etc/hosts

Contient la correspondance entre **l'adresse IP** et le **nom symbolique** des machines. En effet, pour que deux machines Unix puisse communiquer par leur nom, il est indispensable que :
Soit elle utilise un serveur DNS.
Soit chacune, dans son fichier local */etc/host*, contiennent l'adresse IP et le nom de l'autre machine, et également sa propre adresse IP associée à son nom. On peut aussi définir des alias supplémentaires sur ces machines.

Exemple – un extrait du fichier */etc/hosts* du RISC3

```
127.0.0.1     localhost              localhost.localdomain
172.20.0.11   risc1.iut.fr           risc1 RISC/IUT
172.20.0.12   risc2.iut.fr           risc2
172.20.0.13   risc3.iut.fr           risc3
172.20.0.23   tux.iut.fr             tux
```

1.6.4 - Le fichier /etc/resolv.conf

Le fichier /etc/resolv.conf est le principal fichier de configuration pour le code de résolution de nom. Son format est très simple. C'est un fichier texte avec un mot clé par ligne. Il y a trois mots clés typiquement utilisés, qui sont :

domain : Ce mot-clé indique le nom de domaine local.

search : Ce mot-clé spécifie une liste d'autres noms de domaine pour rechercher un nom d'hôte.

nameserver : ce mot-clé, qui peut être utilisé plusieurs fois, spécifie l'adresse IP d'un serveur de nom de domaine pour la résolution de noms.

Un exemple de /etc/resolv.conf pourrait ressembler à ceci :

```
$cat /etc/resolv.conf
domain iut.fr
search iut.fr iut-amiens.fr
nameserver 172.20.0.5
nameserver 172.20.0.6
nameserver 194.57.109.129
$
```

Cet exemple spécifie que le nom de domaine par défaut à ajouter aux noms non qualifiés (c'est-à-dire sans domaine) est iut.fr, et que si l'hôte n'est pas trouvé dans ce domaine on peut aussi essayer

le domaine iut-amiens.fr directement. Trois entrées de serveurs de noms sont fournies, chacune d'elles pouvant être appelée par le solveur de noms.

1.6.5 - Le protocole DNS

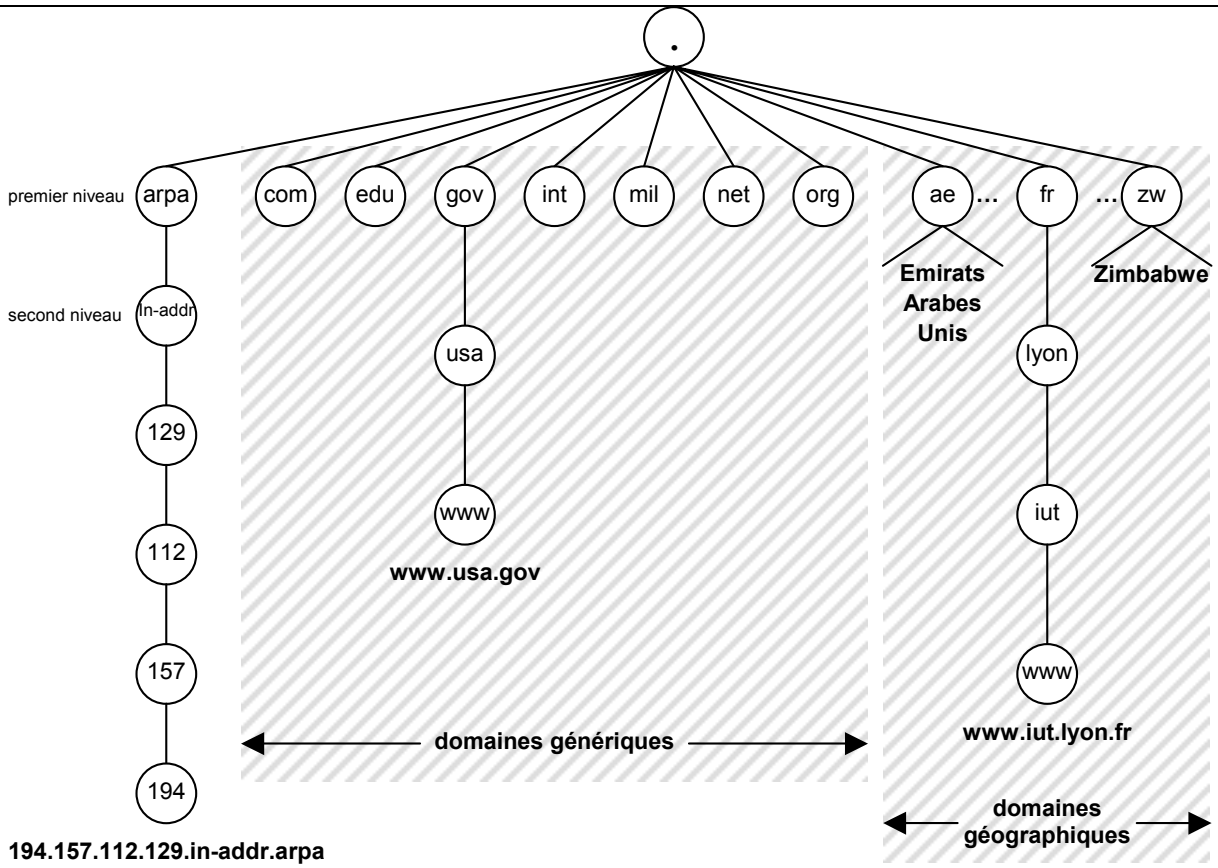
Lorsqu'un client DNS (votre station Windows qui possède une entrée DNS dans sa configuration IP par exemple, un programme quelconque d'un système Unix configuré pour utiliser le DNS, etc.) émet une requête à son serveur DNS, les opérations réalisées sont les suivantes :

t=1 Le client (ST1.FOO.COM) demande au serveur de résoudre un nom de station WWW.AV.COM
t=2 Le serveur DNS détermine (configuration) si ce domaine est géré localement ou par un tiers
t=3 Le serveur DNS ne connaissant la réponse, demande à un serveur (contenu dans la configuration) particulier qui sont les DNS de la zone AV.COM. Ces serveurs connaissant tous les serveurs de noms de chaque domaine sont appelés les ROOT SERVERS
t=4 Le ROOT SERVER répond à notre serveur DNS que les serveurs de noms de la zone AV.COM sont situés sur les adresses IP X.X.X.X, Y.Y.Y.Y, etc...
t=5 Notre serveur DNS contacte alors le premier serveur de noms renvoyé par le ROOT SERVER, et lui pose l'ultime question: « Quelle est l'adresse IP de WWW.AV.COM » ?
t=6 Le serveur DNS de la zone AV.COM répond alors X.X.X.X
t=7 Notre serveur DNS ayant obtenue la réponse escomptée répond alors à la demande initiale du client, en lui fournissant la réponse à sa question. La réponse est également placée dans un cache DNS (côté serveur) pour une utilisation ultérieure (réponse plus rapide)

A la question simple de savoir quelle adresse IP est associée à quel nom (symbolique), on s'aperçoit que les échanges de données nécessaires sont multiples.
Mais les DNS sont également capables de se redistribuer les informations qu'ils contiennent, par le jeu de serveur primaire et secondaires. Pour chaque zone (par exemple domain.com), un serveur DNS est désigné comme primaire (et sera en priorité interrogé par les clients DNS du monde entier), et des serveurs DNS secondaires seront également présents, dans le cas d'une défaillance du primaire. Les serveurs secondaires sont généralement configurés pour prendre une copie de la zone en question sur un autre serveur DNS (par exemple, les secondaires vont chercher une copie de la zone toutes les 8 heures).
Les requêtes DNS des clients (demande de résolution) se font par dessus UDP, alors que les transferts de zone se font via TCP. Dans les deux cas, c'est le port 53 qui est utilisé. On trouvera donc dans les définitions normalisées des ports (/etc/services) les entrées 53/tcp et 53/udp associés tous les deux au service DNS.

1.6.6 - Hiérarchie des DNS

Les zones (généralement des noms de domaine, mais également des plages d'adresses IP) sont hiérarchisées pour permettre une recherche rapide d'une information.



Lorsqu'une requête est soumise par le resolver à un serveur DNS, ce dernier contacte un ROOT SERVER présent dans sa configuration (puis un autre si celui-ci ne répond pas jusqu'à épuisement de la liste) et demande quelle est l'autorité sur la zone demandée. Il contactera ensuite ce serveur pour poser la question à laquelle se dernier doit savoir répondre, et retournera cette information au resolver.

La zone « in-addr.arpa » est une zone quelque peu spéciale, puisqu'elle permet de résoudre les adresses IP en noms symboliques (requêtes inverses). Les zones sont déclarées dans le sens opposé à leur utilisation (reverse), on aura donc une déclaration DNS de la classe IP « 192.168.16.0/24 » qui ressemblera à « 16.168.192.in-addr.arpa. ».

Il est important de noter que le caractère « . » est le délimiteur de niveau hiérarchique. La racine la plus haute de la hiérarchie étant ce même caractère « . ». C'est la racine non nommée. Les extensions « .com. » sont donc une sous-hiérarchie de la zone « . ».

1.6.7 - Bases de données « whois »

Les informations sur les noms de domaines sont déclarées par les prestataires de service lors de l'achat du domaine auprès des organismes de nommage (NIC France, InterNIC, etc.). Ces informations sont généralement les noms des serveurs de noms qui seront autoritaires sur la zone, leurs adresses IP, ainsi que le NIC HANDLE technique du domaine, le NIC HANDLE du responsable légal, et le NIC HANDLE du payeur.

L'accès à cette base de données est libre, et permet donc à chacun de savoir si un nom de domaine est libre, ou par qui il est utilisé. La syntaxe de la commande whois est la suivante :

```
whois <zone>@<server_whois>
```

On peut donc connaître les informations fournies lors de l'enregistrement de la zone « av.com » comme suit :

```
$ whois av.com@whois.internic.net
[whois.internic.net]

Whois Server Version 1.3

Domain names in the .com, .net, and .org domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: AV.COM
   Registrar: NETWORK SOLUTIONS, INC.
   Whois Server: whois.networksolutions.com
   Referral URL: http://www.networksolutions.com
   Name Server: NS2.ALTAVISTA.COM
   Name Server: NS1.ALTAVISTA.COM
   Name Server: NS3.ALTAVISTA.COM
   Updated Date: 26-aug-2002


>>> Last update of whois database: Mon, 2 Dec 2002 16:45:10 EST <<<

The Registry database contains ONLY .COM, .NET, .ORG, .EDU domains and
Registrars.
$
```

Historiquement, l'InterNIC était le seul prestataire habilité à gérer les zones de premier niveau, mais depuis le début de l'année 2000, il existe plusieurs prestataires mondiaux habilités à gérer les domaines. On peut donc se connecter à nouveau sur le serveur whois de l'organisme d'enregistrement (le registrar), ici NETWORK SOLUTIONS, INC.

```
% whois av.com@whois.networksolutions.com
[whois.networksolutions.com]
The Data in the VeriSign Registrar WHOIS database is provided by VeriSign for
information purposes only, and to assist persons in obtaining information about
or related to a domain name registration record.  VeriSign does not guarantee
its accuracy.  Additionally, the data may not reflect updates to billing contact
information.  By submitting a WHOIS query, you agree to use this Data only for
lawful purposes and that under no circumstances will you use this Data to:
(1) allow, enable, or otherwise support the transmission of mass unsolicited,
commercial advertising or solicitations via e-mail, telephone, or facsimile; or
(2) enable high volume, automated, electronic processes that apply to VeriSign
(or its computer systems).  The compilation, repackaging, dissemination or other
use of this Data is expressly prohibited without the prior written consent of
VeriSign.  VeriSign reserves the right to terminate your access to the VeriSign
Registrar WHOIS database in its sole discretion, including without limitation,
for excessive querying of the WHOIS database or for failure to otherwise abide
by this policy.  VeriSign reserves the right to modify these terms at any time.
By submitting this query, you agree to abide by this policy.


Registrant:
Altavista Company (AV22-DOM)
   1070 Arastadero
   Rd.
   Palo Alto
   CA,94304
   US
```

```
   Domain Name: AV.COM

   Administrative Contact:
      DNS Administrator, AltaVista  (ADI396)            dns-admin@AV.COM
      AltaVista Company
      1070 Arastradero Road
      Palo Alto, CA  94304
      US
      650-320-7700 650-320-6433
   Technical Contact:
      AltaVista Company  (AO111-ORG)            dns-technical@AV.COM
      AltaVista Company
      1070 Arastradero Road
      PALO ALTO, CA 94304
      US
      650-320-7700 fax: 650-330-6433

   Record expires on 21-Oct-2003.
   Record created on 21-Oct-1998.
   Database last updated on 3-Dec-2002 05:08:37 EST.

   Domain servers in listed order:

   NS1.ALTAVISTA.COM            209.73.164.76
   NS2.ALTAVISTA.COM            209.73.164.7
   NS3.ALTAVISTA.COM            209.73.176.204
```

Dans la réponse à nos requêtes whois, on voit que les informations principales (notamment sur le serveur de l'InterNIC) sont les adresses des serveurs de noms de cette zone. C'est à eux qu'il faut s'adresser pour toute résolution dans la zone av.com. On contactera par exemple l'un de ces serveurs pour la résolution de www.av.com. Pour la zone .fr on contactera le serveur whois.nic.fr.

```
% whois altavista.fr@whois.nic.fr
[whois.nic.fr]

Tous droits reserves par copyright.
Voir http://www.nic.fr/outils/dbcopyright.html
Rights restricted by copyright.
See http://www.nic.fr/outils/dbcopyright.html

domain:      altavista.fr
descr:       ALTAVISTA
descr:       54-56 Avenue Hoche
descr:       75008 Paris
admin-c:     BC532-FRNIC
tech-c:      IN195-FRNIC
zone-c:      NFC1-FRNIC
nserver:     ns1.altavista.com
nserver:     ns2.altavista.com
nserver:     ns3.altavista.com
mnt-by:      FR-NIC-MNT
mnt-lower:   FR-NIC-MNT
changed:     frnic-dbm-updates@nic.fr 20011130
source:      FRNIC

role:        ISDnet NOC
address:     Paris
e-mail:      noc@isdnet.net
admin-c:     CB58-FRNIC
tech-c:      DP22-FRNIC
```

```
nic-hdl:      IN195-FRNIC
mnt-by:       ISDNET-NOC
changed:      david@isdnet.net 19981110
changed:      migration-dbm@nic.fr 20001015
source:       FRNIC

role:         NIC France Contact
address:      AFNIC
address:      immeuble international
address:      2, rue Stephenson
address:      Montigny-Le-Bretonneux
address:      78181 Saint Quentin en Yvelines Cedex
address:      France
phone:        +33 1 39 30 83 00
fax-no:       +33 1 39 30 83 01
e-mail:       tech@nic.fr
trouble:      Information: http://www.nic.fr/
trouble:      Questions: mailto:nic@nic.fr
trouble:      Spam: mailto:abuse@nic.fr
trouble:      Test: mailto:ping@nic.fr
admin-c:      NFC1-FRNIC
tech-c:       PL12-FRNIC
tech-c:       JP-FRNIC
tech-c:       EM634-FRNIC
tech-c:       MS1887-FRNIC
tech-c:       VL-FRNIC
tech-c:       PR1249-FRNIC
tech-c:       PV827-FRNIC
tech-c:       GO661-FRNIC
tech-c:       FT1632-FRNIC
tech-c:       MS-FRNIC
tech-c:       AI1-FRNIC
tech-c:       SDA2-FRNIC
tech-c:       JJB-FRNIC
nic-hdl:      NFC1-FRNIC
mnt-by:       FR-NIC-MNT
changed:      tech@nic.fr 20011025
changed:      tech@nic.fr 20020711
source:       FRNIC

person:       Beatrice Cuvelier
address:      ALTAVISTA
address:      54-56 Avenue Hoche
address:      75008 Paris
phone:        +33 1 58 44 97 30
e-mail:       beatrice.cuvelier@av.com
nic-hdl:      BC532-FRNIC
changed:      frnic-dbm-updates@nic.fr 20011130
source:       FRNIC
```

1.6.8 - Tester un DNS

Les serveurs DNS sont généralement packagés avec des outils d'interrogation interactif nommés nslookup, host et plus récemment dig. La commande dig peut être utilisée de la manière suivante :

```
$ dig @a.root-servers.net www.altavista.com

; <<>> DiG 9.2.0 <<>> @a.root-servers.net www.altavista.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38241
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 13
```

```
;; QUESTION SECTION:
;www.altavista.com.             IN      A

;; AUTHORITY SECTION:
com.                    172800  IN      NS      A.GTLD-SERVERS.NET.
com.                    172800  IN      NS      G.GTLD-SERVERS.NET.
com.                    172800  IN      NS      H.GTLD-SERVERS.NET.
com.                    172800  IN      NS      C.GTLD-SERVERS.NET.
com.                    172800  IN      NS      I.GTLD-SERVERS.NET.
com.                    172800  IN      NS      B.GTLD-SERVERS.NET.
com.                    172800  IN      NS      D.GTLD-SERVERS.NET.
com.                    172800  IN      NS      L.GTLD-SERVERS.NET.
com.                    172800  IN      NS      F.GTLD-SERVERS.NET.
com.                    172800  IN      NS      J.GTLD-SERVERS.NET.
com.                    172800  IN      NS      K.GTLD-SERVERS.NET.
com.                    172800  IN      NS      E.GTLD-SERVERS.NET.
com.                    172800  IN      NS      M.GTLD-SERVERS.NET.

;; ADDITIONAL SECTION:
A.GTLD-SERVERS.NET.     172800  IN      A       192.5.6.30
G.GTLD-SERVERS.NET.     172800  IN      A       192.42.93.30
H.GTLD-SERVERS.NET.     172800  IN      A       192.54.112.30
C.GTLD-SERVERS.NET.     172800  IN      A       192.26.92.30
I.GTLD-SERVERS.NET.     172800  IN      A       192.43.172.30
B.GTLD-SERVERS.NET.     172800  IN      A       192.33.14.30
D.GTLD-SERVERS.NET.     172800  IN      A       192.31.80.30
L.GTLD-SERVERS.NET.     172800  IN      A       192.41.162.30
F.GTLD-SERVERS.NET.     172800  IN      A       192.35.51.30
J.GTLD-SERVERS.NET.     172800  IN      A       192.48.79.30
K.GTLD-SERVERS.NET.     172800  IN      A       192.52.178.30
E.GTLD-SERVERS.NET.     172800  IN      A       192.12.94.30
M.GTLD-SERVERS.NET.     172800  IN      A       192.55.83.30

;; Query time: 108 msec
;; SERVER: 198.41.0.4#53(a.root-servers.net)
;; WHEN: Tue Dec  3 14:28:52 2002
;; MSG SIZE  rcvd: 467

$ dig @a.gtld-servers.net www.altavista.com

; <<>> DiG 9.2.0 <<>> @a.gtld-servers.net www.altavista.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61779
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;www.altavista.com.             IN      A

;; AUTHORITY SECTION:
altavista.com.          172800  IN      NS      ns2.altavista.com.
altavista.com.          172800  IN      NS      ns1.altavista.com.
altavista.com.          172800  IN      NS      ns3.altavista.com.

;; ADDITIONAL SECTION:
ns2.altavista.com.      172800  IN      A       209.73.164.7
ns1.altavista.com.      172800  IN      A       209.73.164.76
ns3.altavista.com.      172800  IN      A       209.73.176.204

;; Query time: 114 msec
;; SERVER: 192.5.6.30#53(a.gtld-servers.net)
;; WHEN: Tue Dec  3 14:29:41 2002
;; MSG SIZE  rcvd: 137

$ dig @ns1.altavista.com www.altavista.com any
```

```
; <<>> DiG 9.2.0 <<>> @ns1.altavista.com www.altavista.com any
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15051
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;www.altavista.com.             IN      ANY

;; ANSWER SECTION:
www.altavista.com.      600     IN      CNAME   altavista.com.

;; AUTHORITY SECTION:
altavista.com.          600     IN      NS      ns1.altavista.com.
altavista.com.          600     IN      NS      ns2.altavista.com.
altavista.com.          600     IN      NS      ns3.altavista.com.

;; ADDITIONAL SECTION:
ns1.altavista.com.      600     IN      A       209.73.164.76
ns2.altavista.com.      600     IN      A       209.73.164.7
ns3.altavista.com.      600     IN      A       209.73.176.204

;; Query time: 201 msec
;; SERVER: 209.73.164.76#53(ns1.altavista.com)
;; WHEN: Tue Dec  3 14:31:44 2002
;; MSG SIZE  rcvd: 151

$
```

La commande host peut également être utile pour rapidement vérifier le bon fonctionnement du resolver et du DNS.

```
% host www.microsoft.com
www.microsoft.com is a nickname for www.microsoft.akadns.net
www.microsoft.akadns.net has address 207.46.197.113
www.microsoft.akadns.net has address 207.46.230.219
www.microsoft.akadns.net has address 207.46.230.229
www.microsoft.akadns.net has address 207.46.197.101
% host -t ns altavista.com
altavista.com name server ns2.alta-vista.net
altavista.com name server ns3.alta-vista.net
altavista.com name server ns1.alta-vista.net
% host -t mx altavista.fr
altavista.fr mail is handled (pri=10) by
inbound.altavista.fr.criticalpath.net
```

1.6.9 - Les objets DNS

Chaque zone dispose donc (en théorie) de deux DNS (minimum requis pour l'achat d'un nom de domaine auprès des organismes de nommage). Les administrateurs de ces zones vont déclarer différents objets dans cette zone, comme par exemple le numéro de la version actuelle, l'adresse de courrier électronique de l'administrateur de cette zone, les serveurs de courriers à utiliser, etc…

A	Un enregistrement A défini une adresse IP
CNAME	CNAME permet d'associer un nom à un autre nom. C'est un alias
HINFO	Chaîne d'informations sur la station, contenant (par convention d'utilisation) le CPU

de la station suivi du nom du système d'exploitation.

MX Défini le ou les serveurs de courrier pour la zone. Lors d'un envoi d'email à un utilisateur de cette zone, le serveur SMTP émetteur contactera le serveur SMTP contenu ici, en se connectant sur le port standardisé 25/tcp

NS Défini quels sont les serveurs de noms pour cette zone

PTR Enregistrement de pointeur (PTR). Cet enregistrement permet de définir un nom de machine associé à une adresse IP.

1.6.10 - Contrôleurs primaire et secondaires

Nous avons vu que chaque zone dispose d'un minimum de deux serveurs de noms (ceux-là même qui sont enregistrés dans les bases whois). Il existe donc une hiérarchie entre ces serveurs de noms, l'un étant primaire, l'autre secondaire. Ce schéma s'applique avec la règle suivante : « chaque zone dispose d'un serveur DNS primaire, et de serveurs DNS secondaires (entre 1 et 3) ». Chacun de ces serveurs DNS possèdera donc l'ensemble des informations sur la zone concernée.
Puisqu'il est laborieux pour un administrateur DNS (hostmaster) de modifier plusieurs fichiers de configuration de zone sur différents serveurs pour le simple ajout d'un enregistrement DNS, le protocole DNS prévoit une possibilité de transfert des configurations entre DNS autonome : les transferts de zone.
Un serveur DNS secondaire est configuré avec l'adresse du ou des serveurs auxquels il peut demander une copie de la zone en question. Ces connexions des serveurs secondaires se font sur le port 53/TCP du serveur DNS.
Si une configuration est modifiée sur le DNS primaire, son numéro de version (un numéro par zone) est donc incrémenté, et le serveur recharge la configuration de la zone modifiée, et notifie les DNS secondaires de cette zone (déclarés dans la zone) de la modification.
Si une zone est modifiée, sans que le numéro de version ne soit incrémentée, les données conservée en mémoire du serveur DNS seront obsolètes, mais considéré comme actuelles par le serveur DNS disposant (de son point de vue) de la dernière version de la zone.

1.6.11 - Syntaxe d'un fichier de zone DNS

Une définition de zone doit contenir un minimum d'informations, parmi lesquelles :

- Le nom FQDN de la zone
- Le nom FQDN du serveur DNS considéré comme serveur primaire (SOA)
- L'adresse email de l'administrateur de la zone, le caractère '@' sera remplacé par un caractère '.'
- Le numéro de version de la zone
- L'intervalle de temps (en secondes) au bout duquel un serveur secondaire devra se connecter pour obtenir une nouvelle (éventuelle) version de cette zone
- Le délai d'attente avant de retenter un transfert de zone en cas d'échec précédent (serveur primaire non disponible ou non joignable)
- Le temps maximum où une copie de zone sera stocké par le secondaire sans mise à jour du serveur primaire
- Le temps maximum de présence (par défaut) des enregistrements de cette zone dans les caches de DNS extérieurs

```
internet.com. IN SOA mars.internet.com. hostmaster.mars.internet.com. (
1999070301      ; Numéro de série
```

```
10800           ; délai de rafraîchissement des secondaires
3600            ; délai avant une nouvelle tentative si échec
604800          ; délai maximum de validité des données
86400     )     ; TTL  par défaut des enregistrements
```

Les enregistrements ajoutés à chaque entête de zone DNS sont les serveurs de noms de cette zone, les serveurs de courrier, et toutes les associations et alias entre les noms symboliques et les adresses IP.

```
internet.com. IN SOA mars.internet.com. hostmaster.mars.internet.com. (
1999070301      ; Numéro de série
10800           ; délai de rafraîchissement des secondaires
3600            ; délai avant une nouvelle tentative si échec
604800          ; délai maximum de validité des données
86400     )     ; TTL par défaut des enregistrements
; déclaration des serveurs de noms de cette zone
     IN    NS    lune.internet.com.
     IN    NS    remote.backup.com.
     IN    NS    another.network.net.
; déclaration des serveurs de courrier de cette zone
internet.com.    IN    MX    10    mx-01.internet.com.
internet.com.          IN    MX    10    mx-02.internet.com.
internet.com.          IN    MX    100   bk1.mail.net.
internet.com.          IN    MX    200   bk2.mail.net.
internet.com.          IN    MX    200   fail.safe.net.
; déclaration des enregistrements concernant les noms
w1.internet.com.       IN    A           10.0.0.1
www                    IN    CNAME       w1
www                    IN    HINFO       "Linux/1.2.13 AlphaPC 64"
lune                   IN    A           10.0.0.10
mail.in                IN    CNAME       mx-01
mx-01                  IN    A           10.0.0.2
mx-02.internet.com.    IN    A           10.0.0.2
```

1.7 - Rappels divers

1.7.1 - Client

Un **client** est un programme qui utilise une ressource réseau en se connectant dynamiquement aux serveurs appropriés. Il peut être éventuellement sur une autre machine que le serveur.

1.7.2 - Protocoles

Un **protocole** est un ensemble de règles précises définissant l'interaction entre clients et serveurs.

- Commandes reconnues par le serveur.
- Format des données échangées.
- Acquittement.
- Traitement des erreurs.

1.7.3 - serveur

Un **serveur** est un programme qui rend une ressource ou un service disponibles au travers du réseau. Il s'exécute sur la machine qui possède la ressource. Il attend passivement et acceptent les demandes de connexion des clients.

1.7.4 - démon

Un **démon** désigné un serveur qui fonctionnent en permanence. C'est un processus, détaché de tout terminal, qui est lancé en arrière-plan au démarrage du système. Son nom se termine en général par un "d" :

- *lpd* : démon d'impression.
- *ftpd* : démon de transfert de fichiers.
- *inetd* : super démon.

1.7.5 - super démon

Créer un démon par service aurait été trop coûteux en temps machine : on a créé le super démon *inetd* chargé de l'écoute de plusieurs ports et de lancer dynamiquement le serveur réclamé par le client. Le fichier */etc/inetd.conf* contient la liste des serveurs gérés par *inetd*. Ce démon est maintenant souvent remplacé par *xinetd* dont le fichier de configuration est /etc/xinetd.conf et qui permet un paramétrage plus fin.

1.7.6 - Connexion distante

1.7.6.1 - Concept de login distant

La connexion est établie à travers un démon de login distant qui fournit au demandeur, après identification, un shell sur la machine distante. Les commandes sont alors saisies en **local**, exécutées sur la machine **distante** et les affichages ont lieu en **local** via le pseudo-terminal et le réseau.

1.7.6.2 - Connexions distantes avec telnet

- La commande *telnet* tourne sur la machine locale et se connecte au démon *in.telnetd* sur la machine distante.
- L'utilisateur local doit avoir une entrée dans le fichier */etc/passwd* distant et fournir son nom et son mot de passe sur la machine distante.
- Le serveur *in.telnetd* sur la machine distante est créé à la demande d'un client sur la machine locale et disparaît dès qu'il est inutile.

Les commandes reconnues par *telnet* sont :

- *Open* : connexion à un serveur (nom de machine ou adresse IP).
- *Close* : fermeture de la liaison.
- *Quit* : quitter la commande.
- *?* : aide en ligne.

1.7.7 - Transfert de fichiers distants par ftp

La commande *ftp* s'appuie sur le protocole File Transfert Protocol et permet des transferts de fichiers entre machines ayant des systèmes d'exploitation différents. La commande *ftp* doit être invoquée avec le nom ou l'adresse IP de la machine distante ; les contrôles d'accès sont identiques à ceux de *telnet*.

1.7.8 - Communication distante

- La commande **talk** permet une connexion interactive bidirectionnelle avec un utilisateur distant.
- La commande **mail** permet d'envoyer du courrier à un utilisateur distant.

2 - Les sockets : Création

2.1 - Le principe

Une socket est un point de communication par lequel un processus peut émettre ou recevoir des informations.

La commande UNIX **netstat -a** affiche les sockets déjà installés.

Dans un processus, une socket est identifiée par un descripteur, comme un fichier, ce qui permet de lui appliquer les primitives telles que read, write...

⇒ Mise en réseau sans problème des applications standard
⇒ Héritage des sockets par fork
⇒ Redirections possibles

Les différentes constantes nécessaires sont macro-définies dans les fichiers **<sys/types.h>** et **<sys/socket.h>**.

2.2 - Domaine d'un socket

2.2.1 - Domaine Unix

Une socket est **locale** au système où elle est définie et possède une référence dans l'arborescence des fichiers d'Unix. La commande **ls -l** affiche un « s » en début de ligne pour signaler le type de ce fichier.

⇒ **IPC interne**.

La structure d'une adresse de socket est définie dans le fichier à inclure **<sys/un.h>** :

```
struct sockaddr_un {
   short      sun_family ;    /* domaine UNIX: AF_UNIX   */
   char       sun_path[108];  /* référence du fichier         */
};
```

Pour une socket local on peut mettre indifferemment AF_LOCAL, PF_LOCAL, AF_UNIX, PF_UNIX et même AF_FILE et PF_FILE.

2.2.2 - Domaine Internet

Un socket permet un IPC **externe**, c'est à dire une communication entre des processus qui sont lancés sur des machines hôtes différentes.

La structure d'une adresse de socket est définie dans le fichier à inclure **<netinet/in.h>** :

```
struct  in_addr {
    u_long      s_addr;
} ;
struct  sockaddr_in {
    short           sin_family;     /* domaine Internet: AF_INET    */
    u_short         sin_port;       /* le numéro de port            */
    struct in_addr  sin_addr;       /* l'adresse Internet           */
    char            sin_zero[8];    /* un champ de huit zéros       */
} ;
```

Ici encore, le domaine peut être AF_INET ou PF_INET. Pour utiliser de l'IPv6 on utilisera AF_INET6 ou PF_INET6.

Dans ce domaine, l'association entre deux processus communiquant via un socket sera définie par cinq élèments :

- protocole (UDP ou TCP)
- adresse IP de la machine sur laquelle s'exécute le processus A
- port associé à A sur cette machine
- adresse IP de la machine sur laquelle s'exécute le processus B
- port associé à B sur cette machine

Par exemple, un utilisateur connecté sous windows lance une connexion distante sur tux par telnet : un processus « telnet » s'exécute sous windows avec un numéro de port quelconque tandis qu'un processus « telnetd » s'exécute sur etud avec un numéro de port égal à 23, le protocole est TCP.

2.3 - Type d'une socket

2.3.1 - Type datagramme

Correspond aux sockets destinées à la communication en mode non connecté pour l'envoi de datagrammes de taille bornée. Dans le domaine Internet, le protocole sous-jacent est **UDP/IP**.

2.3.2 - Type flot

Correspond aux sockets dédiées à la communication en mode connecté. Dans le domaine Internet, le protocole sous-jacent est **TCP/IP**.

2.4 - Primitives générales de manipulation

2.4.1 - Création

```
int  socket(int  domaine, int  type,int  protocole) ;
```

La primitive **socket** permet de créer une socket en précisant son domaine (AF_UNIX : IPC local ou AF_INET :IPC distant) , son type (SOCK_STREAM en mode connecté ou SOCK_DGRAM en mode datagramme) ainsi que le protocole utilisé. Il est souhaitable de laisser ce dernier paramètre à 0 de façon à laisser le système choisir lui-même le bon protocole.

La valeur de retour est un descripteur qui permet d'accèder à la socket créée.

2.4.2 - Suppression

Une socket est effectivement supprimée après la fermeture par **close** du dernier descripteur permettant d'y accéder : il y a alors libération des ressources utilisées.

2.4.3 - Attachement

Sans ce mécanisme, la socket ne serait connue que par le processus qui l'a créée et par sa descendance : exactement comme pour un tube ordinaire ! ! !

2.4.3.1 - Forme générale de la primitive

```
int  bind(int  desc, struct  sockaddr *ptr_adr , int  lg_adr) ;
```

Un appel à cette primitive réalise l'attachement de la socket de descripteur **desc** à l'adresse ***ptr_adr**, le paramètre **lg_adr** doit être égal à la taille en octets de cette adresse.

La structure **sockaddr** est générique : elle doit être remplacée par la structure adaptée au domaine de la socket.

2.4.3.2 - Attachement d'une socket dans le domaine Unix

L' attachement d'une socket du domaine Unix est réalisé avec un fichier de type « socket » qui apparaîtra dans l'arborescence globale des fichiers Unix. Il est indispensable que la référence correspondante n'existe pas. La suppression de la référence associée à une socket est réalisée par la commande **rm** ou par la primitive **unlink**. Voici un exemple de création et d'attachement :

```
...
char *sock_name= "/tmp/chaussette"
int sd;
struct sockaddr_un socketEcoute;
...
sd = socket(AF_UNIX,SOCK_STREAM,0);
...
unlink(sock_name);    /* Pour etre sur que le fichier n'existe pas */
socketEcoute.sun_family = AF_UNIX;
strcpy(socketEcoute.sun_path,sock_name);
...
bind(sd,&socketEcoute,sizeof(struct sockaddr_un);
...
```

2.4.3.3 - Création d'une paire de sockets associées dans le domaine Unix

```
int socketpair(int dom,int typ,int prot,int *ptr_desc) ;
```

Cette primitive est utilisable pour les types SOCK_STREAM et SOCK_DGRAM mais seulement dans le domaine Unix. Le paramètre **ptr_desc** est un tableau de deux entiers dans lequel on récupère les deux descripteurs permettant l'accès aux deux sockets. Exemple :

```
...
int desc[2] ;
...
socketpair(AF_UNIX,SOCK_STREAM,0,desc) ;
...
```

Ce qui sera écrit sur desc[0] pourra être lu sur desc[1] tandis que ce qui sera écrit sur desc[1] pourra être lu sur desc[0] : il s'agit d'un IPC bidirectionnel ! !

Les sockets n'étant pas « nommées » , seuls le processus créateur et sa descendance pourront utiliser ces sockets.

2.4.3.4 - Le cas particulier du domaine Internet

```
#include <netinet/in.h>
struct  in_addr {
    u_long     s_addr;
} ;
struct  sockaddr_in {
    short           sin_family;     /* domaine Internet: AF_INET     */
    u_short         sin_port;       /* le numéro de port             */
    struct in_addr  sin_addr;       /* l'adresse Internet            */
    char            sin_zero[8];    /* un champ de huit zéros        */
} ;
```

Ecrire la valeur **INADDR_ANY** dans le champ sin_addr.s_addr de l'adresse permet d'associer la socket à toutes les adresses IP possibles de la machine :ceci est particulièrement utile lorsque cette machine sert de passerelle entre réseaux.
Lorsqu'une autre adresse IP doit être recherchée, on utilise une fonction de résolution, comme **gethostbyname**, qui va explorer le fichier **/etc/hosts** et/ou interroger le DNS pour renvoyer une structure **hostent** dans laquelle on accède à l'adresse IP.

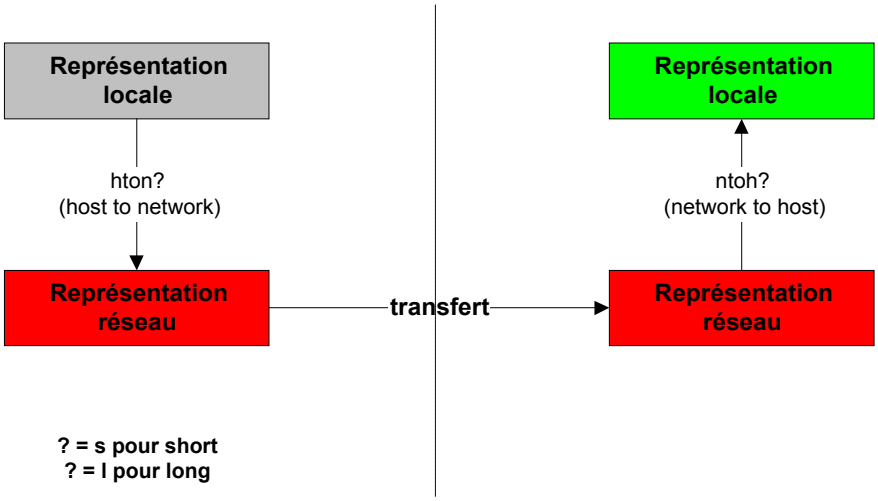
Ecrire la valeur 0 dans le champ sin_port de l'adresse permet de laisser le système choisir lui-même le numéro de port : ceci pourra être réalisé par un processus client mais pas par un processus serveur qui, lui, doit attacher la socket à un numéro de port connu de tous.
Lorsqu'un autre numéro de port doit être recherché, par exemple pour accéder à un service internet spécifique, on utilise une fonction de résolution, comme **getservbyname**, qui va explorer le fichier **/etc/services** et renvoyer une structure **servent** dans laquelle on accède au numéro de port.

```
...
int desc ;
struct  sock_addr_in  adresse ;
int longueur=sizeof(struct  sock_addr_in) ;
...
desc=socket(AF_INET,SOCK_DGRAM,0) ;
adresse.sin_family=AF_INET ;
adresse.sin_addr.s_addr=INADDR_ANY ;
adresse.sin_port=2013 ;
bind(desc,&adresse,longueur) ;
...
```

2.5 - Les problèmes liés au domaine Internet

2.5.1 - La représentation des entiers

Les nombres entiers short comme des numéros de ports ou long comme des adresses IP ne sont pas toujours représentés de la même façon sur les différentes machines d'un réseau :il est donc prudent de les représenter sous leur forme réseau avant de procéder à des échanges entre machines.



Ainsi, dans l'exemple précédent, il aurait été préférable d'écrire :

```
adresse.sin_addr.s_addr=htonl(INADDR_ANY) ;
adresse.sin_port=htons(2013) ;
```

2.5.2 - Les fonctions de résolution et les fichiers administratifs

2.5.2.1 - La fonction gethostbyname

```
#include <netdb.h>
struct hostent *gethostbyname(char *hote) ;
```

Cette fonction prend en paramètre le nom d'une machine et, si ce nom est valide, renvoie l'adresse d'une structure **hostent** créée en zone statique ; cette structure correspond en gros à une ligne du fichier **/etc/hosts**, d'une interogation DNS ou de la map NIS correspondante.

Si le nom de machine n'est pas connu, la fonction retourne le pointeur NULL.

La fonction gethostbyaddr réalise le même genre d'opération à partir de l'adresse IP. D'autres fonctions comme gethostname ou gethostid permettent de récupérer le nom ou l'adresse IP de la machine locale (attention pour les machines qui sont des passerelles).

```
#include <netdb.h>

struct hostent
{
   char  *h_name ;       /* nom officiel de la machine    */
   char  **h_aliases ;   /* liste d'alias                 */
   int   h_addrtype ;    /* type d'adresse (AF_INET)          */
   int   h_length ;      /* longueur de l'adresse en octets    */
   char  **h_addr_list ; /* liste d'adresses              */
} ;

#define  h_addr h_addr_list [0]          /* la première adresse     */
```

2.5.2.2 - La fonction getservbyname

```
#include <netdb.h>
struct servent *getservbyname(char *serv,char *proto) ;
```

Cette fonction prend en paramètre le nom d'un service standard et le protocole correspondant. Si ce couple est valide, la fonction renvoie l'adresse d'une structure **servent** créée en zone statique ; cette structure correspond en gros à une ligne du fichier **/etc/services** ou de la map NIS correspondante. Si le couple (service, protocole) n'est pas connu, retour du pointeur NULL.

La fonction getservbyport réalise le même genre d'opération à partir du numéro de port et du protocole.

```
#include <netdb.h>

struct servent
{
   char  *s_name ;       /* nom officiel du service      */
   char  **s_aliases ;   /* liste d'alias                */
   int   s_port ;        /* numéro de port               */
   char  *proto ;        /* protocole de transport       */
} ;
```

2.5.2.3 - La fonction getsockname

La fonction getsockname permet de retrouver l'adresse d'attachement d'une socket (utile si on a laissé le système choisir le numéro de port) .

```
int getsockname(
   int descripteur,
   struct sockaddr *ptr_adresse,
   int *ptr_lg_adresse,
);
```

A l'appel **`*ptr_lg_adresse`** est la taille de l'espace réservé pour récupérer le résultat, par exemple sizeof(struct sockaddr_in).
Au retour, **`*ptr_lg_adresse`** aura comme valeur la longueur effective de cette adressse.

3 - Les sockets : Communication

3.1 - La communication par datagrammes

3.1.1 - Introduction

Un processus souhaitant communiquer avec le monde extérieur par l'intermédiaire d'une socket du type SOCK_DGRAM doit réaliser, selon les circonstances, un certain nombre des opérations suivantes :

- demander la création d'une socket dans le domaine adapté aux applications avec lesquelles il souhaite communiquer: il s'agira du domaine AF_UNIX pour une communication locale dans un univers BSD ou par exemple AF_INET pour une communication distante (ou locale) en utilisant le protocole UDP
- demander éventuellement l'attachement de cette socket sur un port convenu ou un port quelconque selon qu'il joue un rôle de serveur attendant des requêtes de client ou celui de client prenant l'initiative d'interroger un serveur
- construire l'adresse de son interlocuteur en mémoire: tout client désirant s'adresser à un serveur doit en connaître l'adresse et donc la préparer en mémoire (il pourra consulter la base de données des services et utiliser la fonction standard gethostbyname pour obtenir l'adresse Internet de la machine distante à partir de son nom). S'il s'agit d'un serveur, il recevra avec chaque message l'adresse de son émetteur qu'il pourra donc utiliser pour expédier la réponse
- procéder à des émissions et des réceptions de messages. Les sockets de ce type sont utilisées en mode non connecté : chaque demande d'envoi d'un datagramme s'accompagne de la spécification complète de l'adresse de son destinataire et chaque réception d'un message s'accompagne de celle de son émetteur.

3.1.2 - L'envoi d'un message

```
int sendto(
        int descripteur,                    renvoyé par la primitive socket
        void *message,                      message à envoyer
        int longueur,                       longueur de ce message
        int option,                         0
        struct sockaddr *ptr_adresse,       adresse du destinataire
        int longueur_adresse                longueur de cette adresse
) ;
```

La valeur de retour de le primitive est le nombre de caractères effectivement envoyés.

Remarque : si l'attachement n'est pas effectué avant le premier envoi utilisant cette socket (client) attachement est effectué automatiquement sur un port quelconque de la machine locale.

3.1.3 - La réception d'un message

```
int recvfrom(
        int descripteur,                    renvoyé par la primitive socket
        void *message,                      adresse du buffer de réception
        int longueur,                       taille du buffer de réception
        int option,                         0
        struct sockaddr *ptr_adresse,       adresse de l'émetteur
        int *ptr_longueur_adresse           pointeur sur la longueur de cette adresse
) ;
```

Le message lu dans le tampon de réception correspond exactement au message envoyé par un appel à sendto.

L'adresse de l'émetteur du message sera récupérée à l'adresse ptr_adresse et la taille de cette adresse à l'adresse ptr_longueur_adresse au retour de la primitive recvfrom à condition d'avoir initialisé *ptr_longueur_adresse avant l'appel à la primitive ! ! !

3.1.4 - Mode pseudo-connecté

Il est également possible de réaliser une pseudo-connection en UDP, c'est a dire la possiblité d'utiliser les primitives classiques read et write. En fait la connection n'est pas vraiment réalisée mais l'adresse de destionation est mémoriser pour ne pas avoir à le préciser à chaque envois/réception de message. Ceci se fait grâce à la primitive connect.

3.2 - La communication en mode connecté

3.2.1 - Introduction

C'est le mode de communication associé aux sockets de type SOCK_STREAM. Il permet à des applications réparties de s'échanger des séquences de caractères continues et non structurées en messages. Il est par exemple adapté à l'implantation d'un mécanisme de connexion à distance: les applications telnet ou rlogin l'utilisent. Etant donné que la communication se réalise entre deux points spécifiques et que cette communication s'étale sur une durée qui peut être assez longue et donne lieu à un volume d'échange de caractères susceptible d'être important, une connexion (un circuit virtuel) est établie entre les deux entités. Dans le cas du domaine AF_INET, le protocole sous-jacent utilisé est le protocole TCP : la communication entre les processus est donc fiable.

Avec les sockets de type SOCK_STREAM et la communication en mode connecté, la dissymétrie entre les deux entités est clairement marquée.

3.2.1.1 - Schéma général d'un serveur TCP

Un serveur a un rôle passif dans l'établissement de la connexion : après avoir avisé (par un appel à la primitive **listen**) le système auquel il appartient qu'il est prêt à accepter les demandes de connexion des clients, le serveur se met en attente de demande de connexion. Pour cela il dispose d'une **socket d'écoute** attachée au port correspondant au service et donc supposé connu des clients. Lorsqu'une demande de connexion arrive à l'entité TCP du système où s'exécute le serveur, le

système crée une nouvelle socket dédiée à cette nouvelle connexion et que nous appellerons **socket de service**. C'est par ce moyen qu'il est possible de multiplexer sur la même socket plusieurs connexions. Le processus serveur prend connaissance de l'existence d'une nouvelle connexion par un appel à la primitive **accept** : au retour de cet appel, le processus reçoit un descripteur lui permettant d'accéder à cette socket de service.

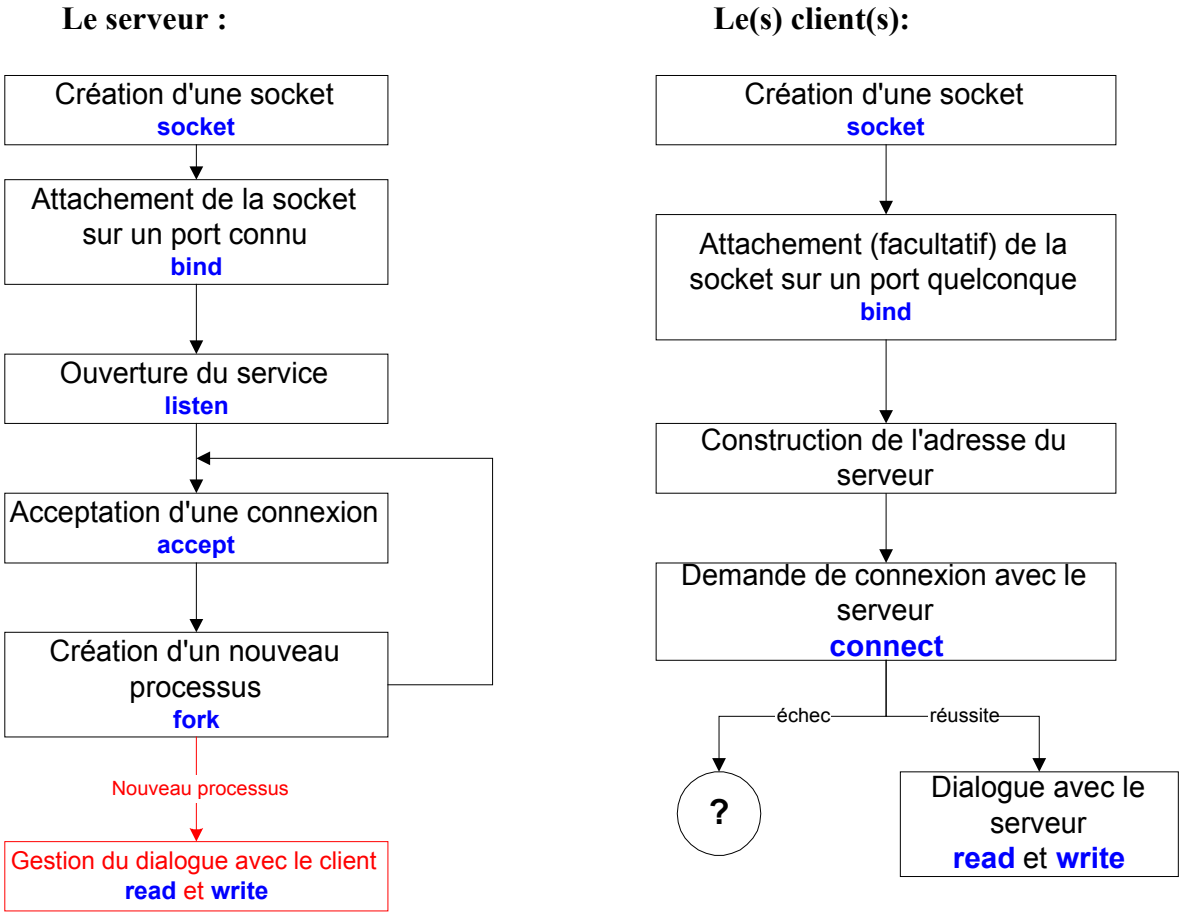
Il existe un certain déphasage entre l'acceptation des connexions au niveau du protocole TCP et la connaissance de celles-ci par le processus lui-même. Les connexions acceptées au niveau TCP mais non encore prises en compte par le processus sont dites pendantes. Une fois prise en compte par le serveur (par un appel à la primitive accept), une connexion devient effective et est enlevée de la liste des connexions pendantes.

Après avoir accepté une connexion, le serveur se décharge en général du dialogue proprement dit sur un processus de service (ou sur une activité thread si le système supporte cette fonctionnalité).

3.2.1.2 - Schéma général d'un client TCP

Un client est l'entité active dans le processus d'établissement d'une connexion avec le serveur : c'est lui qui prend l'initiative. Le client commence par créer la socket, l'attache éventuellement à un numéro de port quelconque puis tente de connecter cette socket à celle du serveur.

3.2.1.3 - Scénario client/serveur TCP



3.2.2 - Les primitives listen et accept

```
int listen(int descripteur, int nb_pendantes) ;
```

La primitive listen permet à un processus de déclarer un service ouvert auprès de son système local :après cet appel l'entité TCP du système local commence à accepter des connexions. Le paramètre nb_pendantes de la primitive correspond au nombre maximum de connexions pendantes acceptables :ce nombre sera automatiquement modifié en tenant compte de l'implémentation du système Unix.

```
int accept(
   int descripteur,
   struct sockaddr *ptr_adresse;
   int *ptr_lg_adresse ;
) ;
```

La primitive accept permet à un processus de prendre connaissance d'une nouvelle connexion :celle-ci est extraite de la liste des connexions pendantes et un descripteur est renvoyé pour une nouvelle socket (de service) dédiée à cette connexion.

L'adresse du client avec lequel on va dialoguer via cette socket de service est récupérée à l'adresse ptr_adresse pourvu que celle-ci ne soit pas NULL tandis que la longueur de cette adresse est obtenue à l'adresse ptr_lg_adresse.

Toujours initialiser *ptr_lg_adresse avant d'appeler la primitive accept ! ! !

3.2.3 - La primitive connect

```
int connect(
   int descripteur,
   struct sockaddr *ptr_adresse;
   int lg_adresse ;
);
```

Toute connexion entre deux sockets de type SOCK_STREAM dans le domaine AF_INET est initiée par cette primitive (pour le domaine AF_UNIX, on peut utiliser également la primitive socketpair pour créer directement une paire de sockets de type SOCK_STREAM connectées).

La primitive connect demande l'établissement d'un circuit virtuel entre la socket (locale) dont on donne le descripteur et la socket (distante) dont on passe l'adresse et la longueur de l'adresse.

3.2.4 - La réception de données

Le primitive :

```
ssize_t read (int desc, void*pt,size_t taille);
```

Permet de lire les données sur un descripteur de socket. Par défaut, cette primitive est bloquante tant qu'il n'y a rien à lire sur le descripteur de la socket.

Paramètres :

- desc : descripteur de la connexion sur laquelle recevoir les données
- pt : pointeur sur une zone mémoire réservée pour recevoir les données lues
- taille : taille de la zone réservée

Retour :

- nombre de caractères lus
- -1 : en cas d'erreur

Remarque :

Le message est tronqué si sa taille est supérieure à taille
La primitive read ne permet pas d'extraire les messages urgents (TCP).

3.2.5 - Envoi de données

Le primitive :

```
ssize_t write (int desc, void*pt,size_t taille);
```

Permet d'envoyer des données via un descripteur de socket.

Paramètres :

- desc : descripteur de la connexion (ou pseudo-connexion) sur laquelle envoyer les données
- pt :pointeur sur zone mémoire contenant les données à envoyer
- taille : taille en octets des données à envoyer

Retour :

- nombre de caractères ecrits
- -1 : en cas d'erreur errno modifiée

Remarques :

La primitive est bloquante si le tampon d'émsion est plein.

La primitive write ne permet pas d'exploiter complètement les possibilités des sockets TCP.

4 - Les démons

4.1 - Les problèmes liés aux serveurs

4.1.1 - Les zombies

Dans le cas d'un serveur UDP, la situation habituelle est plutôt celle d'un serveur « wait », c'est à dire que le processus serveur traite directement les requêtes des clients. Dans le cas d'un serveur TCP, la situation habituelle est celle d'un serveur « no wait », c'est à dire que le serveur va se cloner et faire traiter les requêtes des clients par ses fils.
Attention, dès qu'un serveur se clone, il faut penser à l'élimination des zombies avec l'une des primitives sigaction ou signal.

4.1.2 - Les descripteurs de fichiers

Il faut penser à fermer tous les descripteurs inutiles, en effet on évite en général de garder tous les descripteurs hérités ouverts.

```
#include <sys/param.h>
for (i=0; i<NOFILE ;i++)
    close(i);
```

4.1.3 - Le directory

Il vaut mieux se placer à la racine du système par l'appel à :

```
chdir ("/");
```

qui permet les démontages éventuels du système de fichiers sans arrêter le serveur.

4.2 - Les obligations d'un serveur

4.2.1 - Le détachement ou la naissance d'un démon

Une caractéristique essentielle d'un serveur est qu'il est détaché de tout terminal :ceci est réalisé dans POSIX avec la primitive **setsid** qui crée une nouvelle session dont le processus sera le leader. Pour éviter des interférences avec le contrôle de jobs qui peut être actif avec certains shells, on réalise un premier appel fork : le processus initial se termine immédiatement d'où réveil du shell courant tandis que le processus fils qui est adopté par le processus init, réalise un appel à setsid puis exécute le code du serveur proprement dit.

```
main(...) {
   ...
   if( fork() != 0)
        exit(0) ;
   setsid() ;
```

```
    ...
}
```

4.2.2 - Historique d'un démon

Un serveur qui joue un rôle important a souvent été programmé pour signaler à son administrateur les erreurs rencontrées et les actions effectuées au cours d'un session. Ceci peut être réalisé de trois façons :

4.2.2.1 - Ecrire sur la console

Il suffit d'ouvrir le fichier **/dev/console** et d'envoyer les messages de trace sur ce fichier en se servant éventuellement de la primitive dup pour rediriger stdout et stderr . ⇒ les messages ne sont pas enregistrés

4.2.2.2 - Ecrire dans un fichier spécifique au serveur

```
FILE *f ;
main(...)  {
   f=fopen("trace_serveur","w") ;
   ...
   fprintf(f, "format données",données) ;
   ...
}
```

4.2.2.3 - Envoyer les messages de trace au démon syslogd

Le démon **syslogd** est chargé de gérer l'historique du système en envoyant des messages dans des fichiers, à des groupes d'utilisateurs ou au démon syslogd d'une autre machine. Le PID du démon syslogd local est contenu dans le fichier **/etc/syslogd.pid** tandis que le fichier **/etc/syslogd.conf** permet de configurer ce démon.

Pour utiliser syslogd pour gérer l'historique d'un nouveau démon, il faut d'abord, en tant que root, réaliser les trois opérations suivantes :

1. Créer un fichier vide nommé par exemple /var/log/etud
2. Ajouter une ligne au fichier de configuration :
   local0.info /var/log/etud
   local0 (champ FACILITY) précise l'origine du message
   info (champ LEVEL) précise le niveau d'importance du message
3. Forcer le processus syslogd à relire son fichier de configuration en lui envoyant le signal SIGHUP
   kill -HUP <pid_syslogd>

Les fonctions à utiliser sont :

```
#include <syslog.h>
void openlog(char *ident, int option, int  facility)
```

- Ouvre une connexion a syslog pour le programme. La chaîne pointée par ident est ajoutée a chaque message, par défaut (c'est a dire si aucun appel a la fonction openlog n'a été effectué) elle est positionné sur le nom du programme. L'utilisation d'openlog est optionnelle, elle sera

automatiquement appelée par syslog() si nécessaire. Les valeurs pour option et facility sont données plus loin.

```
#include <syslog.h>
void syslog(int priority, char *format, ...)
```

- Génère un message de log, qui sera distribué par syslogd(8), priority est une combinaison de facility et level données ci-dessous. Le dernier argument est une chaîne formatée comme printf exceptée que la chaîne %m sera remplacée par la chaîne du message d'erreur correspondant à la valeur actuelle de errno.

```
#include <syslog.h>
void closelog(void)
```

- Ferme le descripteur utilisé pour dialoguer avec syslog, l'utilisation de closelog est optionnelle

Option

L'argument option pour openlog() est un OU les différentes options suivantes :

LOG_CONS :	Ecrit directement sur la console système s'il y a une erreur pendant l'envoi d'un message au syslog
LOG_NDELAY :	Ouvre la connexion immédiatement (Normalement , la connexion est ouverte quand le premier message est envoyé a syslog)
LOG_PERROR :	Envoie également le message sur stderr
LOG_PID :	Inclus le PID avec chaque message

Facility et Level

Consulter les pages de man.

Pour que le nouveau démon utilise alors les services proposés par syslogd, on inclura dans son code des lignes du style :

```
char tmp[]="Serveur Personnel";
...
openlog(tmp,LOG_PERROR,LOG_LOCAL0);
...
syslog(  LOG_LOCAL0|LOG_INFO,
              "serveur bidule actif PID=%d UID=%d\n",
              getpid(),getuid());
...
```

4.2.3 - Le contrôle

Un démon peut être programmé pour lire un fichier de configuration lors de son lancement mais il peut être aussi important de l'obliger à relire ce fichier, sans l'arrêter, surtout quant ce serveur a un rôle fondamental dans la bonne marche du système tout entier.
Ceci est réalisé par l'envoi du signal SIGHUP et son déroutement dans le code du serveur par le biais de la primitive sigaction.
En règle générale, le fichier de configuration se termine par l'extension *".conf"*.

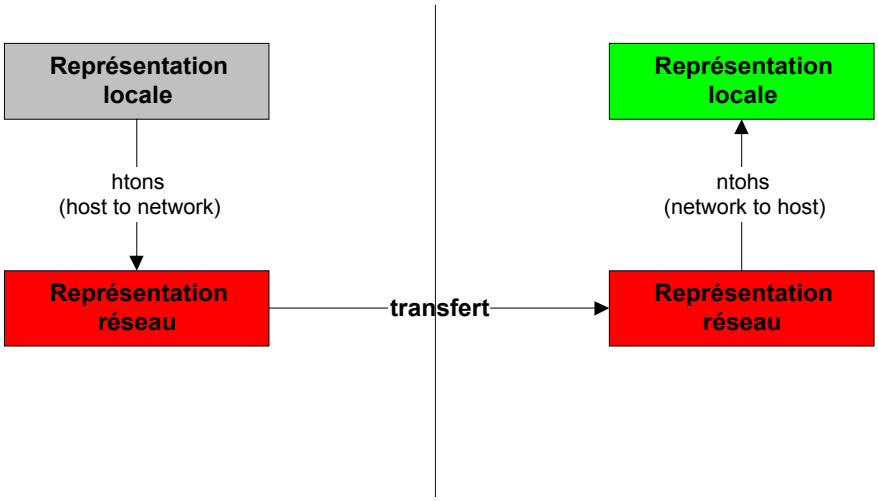
5 - Le protocole XDR

5.1 - Introduction

5.1.1 - Intérêt du protocole

L'échange d'informations typées ou structurées entre différentes machines de heurte au problème de la non-unicité de la représentation interne des objets.

Pour les entiers longs codés sur quatre octets, comme les adresses Internet, nous avons déjà utilisé les fonctions **htonl** et **ntohl** qui utilisent une représentation « réseau » de ces entiers. De même pour les entiers courts codés sur deux octets , comme les numéros de port TCP et UDP, nous avons utilisé les fonctions et .



Le protocole XDR vise à résoudre les problèmes liés à toutes les différences de représentation des objets qui doivent être échangés sur un réseau :

- la taille des objets typés
- l'ordre des octets
- la représentation proprement dite
- les problèmes d'alignement

5.1.2 - Exemple concret

On va tenter ici de faire passer un float et un double d'une machine « junon » à une autre machine « jupiter » à partir d'une machine « pergame » (vous ne pouvez pas effectuer cette manipulation étant donné que la commande rsh est désactivée).

```
$hostname
pergame
$rsh  junon  cat  ecr.c
/* Ecrivain */
main(void)
{
  float f = 12.45;
  double d = 12.45;
  write(1,(char *)&f,sizeof(float));
  write(1,(char *)&d,sizeof(double));
}
$rsh  jupiter  cat  lec.c
/* Lecteur */
main(void)
{
  float f;
  double d;
  read(0,(char *)&f,sizeof(float));
  read(0,(char *)&d,sizeof(double));
  printf("f = %f\nd = %lf\n",f,d);
}
$rsh jupiter ecr | rsh junon lec
...
...
```

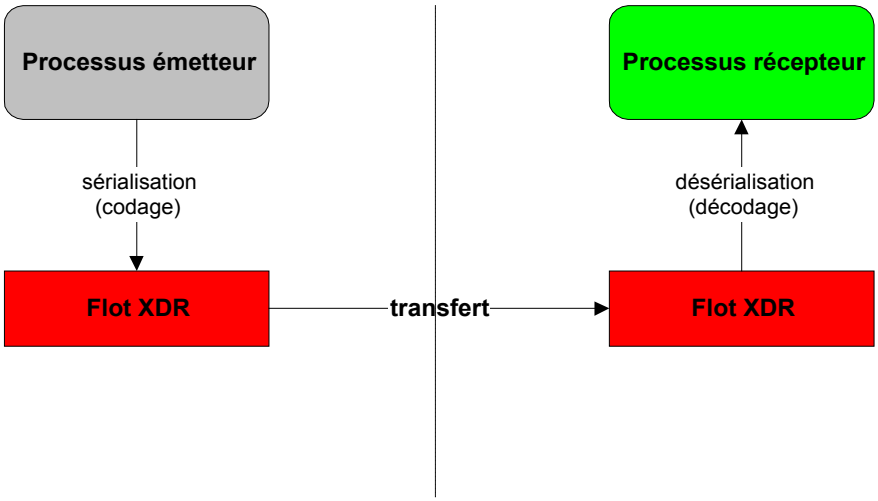
Si les deux machines « junon » et « jupiter » ont la même architecture, il n'y aura aucun problème et on obtiendra :
f = 12.450000
d = 12.450000

Si ce n'est pas le cas, on obtiendra n'importe quoi ! ! !

5.2 - Principe général d'utilisation

Si l'utilisation de la bibliothèque XDR conjointement à celle de la bibliothèque RPC (et plus particulièrement en cas d'utilisation de la commande **rpcgen**) est relativement transparente aux utilisateurs, il est intéressant de connaître les mécanismes simples qu'elle met en jeu si on veut l'utiliser pour des applications utilisant l'interface des sockets.
Un processus souhaitant transmettre une suite de valeurs procède à une sérialisation de ces différentes valeurs. Cette opération consiste à mettre bout à bout les représentations XDR des différents objets, c'est-à-dire à créer un flot d'informations que nous appellerons dans la suite flot XDR .

Le processus émetteur applique donc différentes fonctions (à chaque type d'objet correspond une fonction spécifique) associant aux objets leur représentation standard. A l'autre bout de la chaîne, un processus récepteur pourra alors extraire du flot XDR les représentations XDR. d'objets et leur appliquer des fonctions de conversion afin d'en obtenir la représentation locale.



La principale critique qu'on peut formuler relativement à ce protocole est que si l'échange d'informations a lieu entre deux machines de même type sur lesquelles la représentation des informations est différente de la représentation standard, on devra néanmoins procéder à un encodage sur la machine source et à un décodage sur la machine cible, deux opérations à priori inutiles puisque la représentation sur les deux machines est identique.

5.3 - Opérations de base sur les flots XDR

5.3.1 - Les fichiers standard

Les fichiers **<rpc/types.h>** et **<rpc/xdr.h>** doivent être inclus dans cet ordre dans les programmes utilisant [es fonctions de la biliothèque XDR. Ils contiennent la définition de différents types et constantes de base constituant l'interface des applications avec la bibliothèque :

- le type **bool_t**
  les objets de ce type peuvent prendre les valeurs symboliques
  FALSE et TRUE
- l'énumération **enum xdr_op**
  ensemble des valeurs XDR_ENCODE XDR_DECODE XDR_FREE qui correspondent chacune à un type de flot
- le type **XDR**
  correspond à la structure d'un flot XDR (structure opaque)
  équivalent pour la bibliothèque XDR au type FILE utilisé par la bibliothèque standard des E/S

5.3.2 - Le type XDR

La désignation d'un flot XDR se fait par l'intermédiaire d'un pointeur sur un objet de ce type. Un tel objet de type XDR contient les différentes informations nécessaires à sa gestion, c'est-à-dire le type du flot correspondant, la dernière opération réalisée et différentes adresses en mémoire définissant son correspondant physique (adresse mémoire ou FILE).

Le type d'un flot spécifie si le flot est un flot d'encodage (XDR_ENCODE) ou de décodage (XDR_DECODE). Il est spécifié à la création de celui-ci et il détermine la nature des opérations qui pourront lui être appliquées.
Un troisième type (XDR_FREE) autorise, au cours de certaines opérations de décodage, la libération de l'espace mémoire alloué automatiquement au cours d'opérations d'encodage antérieures.

5.3.3 - Création d'un flot

5.3.3.1 - Généralités

Les différentes primitives disponibles pour initialiser un objet de type XDR en vue de son utilisation ultérieure admettent comme paramètres :

- un pointeur sur un objet de type XDR ;
- la ressource à laquelle le flot XDR est associé et ses caractéristiques éventuelles
- le type du flot XDR

La forme générale d'un appel à l'une des fonctions de création aura l'une des formes suivantes :

```
#include <rpc/types.h>
#include <rpc/xdr.h>
XDR objet;

xdrmem_create(&objet,…);
```

```
#include <rpc/types.h>
#include <rpc/xdr.h>
XDR *ptr_objet;

ptr_objet=(XDR *) (malloc(sizeof (XDR));
xdrmem_create(ptr_objet, .....);
```

5.3.3.2 - Les flots XDR en mémoire

```
#include <rpc/xdr.h>
void xdrmem_create(XDR  *ptr_xdr,
                   const  void  *adresse,
                   const  unsigned  int  taille,
                   const  enum  xdr_op  type
                   ) ;
```

Cette fonction permet d'initialiser l'objet *ptr_xdr : la zone d'adresse donnée de taille octets va ainsi pouvoir être utilisée comme un flot XDR en mémoire. Le type des opérations (soit d'encodage, soit de décodage) est déterminé par le dernier paramètre qui, en plus des valeurs XDR_ENCODE et XDR_DECODE, peut également prendre la valeur XDR_FREE : ce type de flot permet la libération de l'espace alloué au cours d'un décodage antérieur dans un flot de type XDR_DECODE ayant entraîné l'allocation dynamique de mémoire.

La création d'un flot d'encodage associé en mémoire à un tableau de 1024 caractères pourra par exemple être réalisée par la séquence suivante :

```
#include <rpc/types.h>
#include <rpc/xdr.h>
char tab[1024] ;
XDR xdr_memoire;
xdrmem_create(&xdr_memoire, tab, 1024, XDR_ENCODE);
```

Le protocole RPC utilise ce type de flots pour les échanges de messages au-dessus des protocoles UDP /IP. Les messages échangés sont construits en mémoire avant leur transmission via une socket par un appel à la primitive sendto.

5.3.3.3 - Les flots XDR et les fichiers

Un fichier manipulé au niveau de la bibliothèque standard et donc pour lequel un pointeur sur un objet de type FILE est défini (en particulier stdin et stdout), peut être associé à un flot XDR au moyen de la fonction :

```
#include <rpc/xdr.h>
void xdrstdio_create(
XDR  *ptr-xdr,
const  FILE  *ptr-file,
const  enum  xdr_op  type ) ;
```

Par exemple, la séquence suivante réalise l'association de l'entrée standard stdin à un flot XDR de décodage :

```
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
XDR  xdr_file;
xdrstdio_create(&xdr_file, stdin, XDR_DECODE) ;
```

5.4 - Opérations de sérialisation et de désérialisation

5.4.1 - Principe général

Il consiste à ajouter ou extraire des informations dans un flot XDR en en réalisant le codage ou le décodage. A chaque type d'objet correspond une fonction spécifique agissant comme un filtre et qui, selon la nature du flot XDR, réalise un encodage, un décodage ou la libération d'espace alloué au cours d'un décodage précédent.

Pour chaque type d'objet élémentaire xyz une fonction (ou filtre) xdr_xyz est définie ; elle admet deux paramètres:

- un pointeur sur un descripteur de flot de type XDR
- un pointeur sur un objet du type correspondant

Les différentes fonctions de base renvoient la valeur TRUE ou FALSE (correspondant aux valeurs du type prédéfini bool_t)) selon que l'opération demandée s'est ou non bien déroulée.

5.4.2 - Le type void

La fonction xdr_void n'admet aucun paramètre et ne réalise aucune opération sur le flot. Elle permet dans le mécanisme RPC d'appeler des fonctions sans paramètre. Elle renvoie toujours la valeur TRUE.

5.4.3 - Exemple de type scalaire: le type float

La fonction de sérialisation/désérialisation **xdr_float** a pour prototype :

```
#include <rpc/types.h>
#include <rpc/xdr.h>
bool_t  xdr_float(XDR *, float *);
```

Selon le type du flot XDR spécifié, elle va :

- réaliser l'encodage de la valeur contenue à l'adresse spécifiée en second paramètre interprétée selon le type correspondant à la fonction et ajoute au flot XDR le résultat de cet encodage dans le cas où le type du flot est XDR_ENCODE.
- réaliser le décodage dans le flot XDR d'un objet du type donné et écrit le résultat à l'adresse mémoire donné en second paramètre dans le cas où le type du flot est XDR_DECODE.

5.5 - Solution au problème concret

```
$cat  xdr_ecr.c
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
main(void)
{
   float f = 12.45;
   double d = 12.45;
   XDR  xdrflot;
   xdrstdio_create(&xdrflot,stdout,XDR_ENCODE);
   if(!xdr_float(&xdrflot,&f) || !xdr_double(&xdrflot,&d))
   fprintf(stderr,"Erreur d'encodage au format XDR.\n");
}

$cat xdr_lec.c
#include <stdio.h>
#include <rpc/types.h>
#include <rpc/xdr.h>
main(void)
{
   float f;
   double d;
   XDR  xdrflot;
   xdrstdio_create(&xdrflot,stdin,XDR_DECODE);
   if(!xdr_float(&xdrflot,&f) || !xdr_double(&xdrflot,&d))
   fprintf(stderr,"Echec du décodage du format XDR.\n");
else
         printf("f = %f\nd = %lf\n",f,d);
}

$rsh jupiter xdr_ecr | rsh junon xdr_lec
f = 12.450000
d = 12.450000
$
```