Université de Provence
Marseille

Année 2000/2001
Maîtrise d'Informatique

TD/TP Réseaux

Enoncés des TD/TP + quelques notes d'aide

Sommaire

• TD/TP Réseaux n°1 - Installation/administration réseau sous Linux Mandrake 6.1
Configuration carte réseau, tables de routage, NFS, NIS,…

• TD/TP Réseaux n°2 - Interconnexion de réseaux, subnetting
Différentes classes de réseaux, création de sous-réseaux, détections de problèmes réseaux

• TD/TP Réseaux n°3 - Applications client/serveur TCP/IP en C
Programmation d'une petite application : envoi de données du client au serveur

• TD/TP Réseaux n°4 - Applications client/serveur TCP/IP en C
Reprise d'un exercice fait en TP de système : échanges client ↔ serveur

• TD/TP Réseaux n°5 - Client/serveur TCP/IP en C
Devoir à rendre : programmation d'un petit jeu du pendu en réseau

• TD Réseaux n°6 - Applications client/serveur TCP/IP en JAVA
Reprise du programme du TD/TP n°3 en JAVA

Laurence Pierre
http://www.cmi.univ-mrs.fr/~laurence

Université de Provence
Marseille

Année 2000/2001
Maîtrise d'Informatique

TD/TP Réseaux n°1

Installation/administration réseau sous Linux Mandrake 6.1

Préambule. Les diverses étapes de configuration que nous allons voir ici sont générales à tout réseau local de machines UNIX. La plupart des noms de commandes et de fichiers sont également assez classiques. Cependant, certains identificateurs (en particulier pour les fichiers) sont spécifiques à Linux Mandrake 6.1.

Dans ces travaux pratiques, nous allons :

1. Installer toutes les machines en leur attribuant un *nom*, une *adresse IP*, et un *nom de domaine*.

2. Nous assurer que les machines sont en mesure d'atteindre toute autre machine, sur le même réseau ou sur un autre. C'est ici la *configuration des tables de routage* qui est concernée.

3. Configurer les machines pour avoir accès à un ou plusieurs *serveurs de noms*.
Interroger manuellement ces serveurs.

4. Vérifier de visu ce qui a été dit en cours concernant les fichiers */etc/inetd.conf* et */etc/services*.
Activer/désactiver un service.

5. Réaliser, grâce a *NFS*, des montages de partitions distantes.

6. Installer un domaine *NIS* avec un serveur, et toutes les autres machines en clients.

I. Configuration de base : nom de machine, adresse IP, nom de domaine

1. La machine doit avoir une adresse IP, associée à un nom logique. Cette association adresse/nom est donnée dans le fichier **/etc/hosts**. D'autres machines peuvent aussi y être déclarées, par exemple :

```
# Fichier /etc/hosts
127.0.0.1     localhost

193.50.131.100  pc1.univ-mrs.fr pc1 loghost
193.50.131.101  pc2
193.50.131.102  pc3
193.50.131.103  pc4
```

2. La configuration réseau de base, utile en particulier pour la configuration de la carte Ethernet, se trouve dans les fichiers **/etc/sysconfig/network** et **/etc/sysconfig/network-scripts/ifcfg-nom** (où **nom** est le nom de device associé à la carte éthernet, a priori "eth0").

```
# Fichier /etc/sysconfig/network (Linux Mandrake 6.1)

NETWORKING=yes
FORWARD_IPV4=false
HOSTNAME=pc1.univ-mrs.fr
DOMAINNAME=univ-mrs.fr
GATEWAY=193.50.131.1
GATEWAYDEV=eth0
```

```
# Fichier /etc/sysconfig/network-scripts/ifcfg-eth0 (Linux Mandrake 6.1)

DEVICE=eth0
IPADDR=193.50.131.100
NETMASK=255.255.255.0
NETWORK=193.50.131.0
BROADCAST=193.50.131.255
ONBOOT=yes
```

Remarques: `/etc/sysconfig/network` est exécuté au boot par `/etc/rc.d/rc.sysinit`, et `/etc/sysconfig/network-scripts/ifcfg-eth0` est utilisé par la commande *`ifconfig`* (configuration de la carte réseau), également exécutée au boot (voir `/etc/sysconfig/network-scripts/ifup`).

II. Configuration pour routage IP

Les "tables de routage" sont indispensables pour savoir comment atteindre d'autres machines, sur le même réseau ou sur un autre réseau. Ces tables sont mises à jour par la commande

```
route
```

Cette commande permet d'insérer dans les tables de routage, ou d'en enlever, diverses informations. En particulier, pour ajouter une entrée (c'est à dire un couple réseau/passerelle pour atteindre ce réseau, ou "gateway") dans la table de routage, il faut exécuter la commande

```
route add <réseau> gw <passerelle>
```

Par exemple, avec les informations données dans les deux fichiers /etc/sysconfig/network et /etc/sysconfig/network-scripts/ifcfg-eth0 ci-dessus, les commandes suivantes sont exécutées au boot :

```
route add 193.50.131.0 gw 193.50.131.100
route add default gw 193.50.131.1
```

Autrement dit, pour atteindre le réseau local `193.50.131.0` (rappel : ceci est un réseau de classe C, ce qui explique pourquoi la partie "réseau" est sur 3 octets), la machine est sa propre passerelle, et pour atteindre tout autre réseau (`default`), la passerelle est la machine de numéro `193.50.131.1`.

Il est a noter que la commande pour visualiser les tables de routage est

```
netstat -r
```

III. Configuration pour accès au DNS

A partir de maintenant, on sait comment atteindre des machines distantes, connaissant leur numéro IP (rappel : ceci grâce au protocole ARP "Address Resolution Protocol" qui permet d'associer une adresse physique, i.e. adresse Ethernet, à une adresse logique, i.e. adresse IP).

Cependant, en général, l'utilisateur ne connait que les noms des machines et pas leurs numéros IP ! Alors :

- soit l'association nom/adresse est donnée localement dans le fichier /etc/hosts (et on n'a besoin de rien de plus !)
- soit elle ne l'est pas, et il faut avoir recours à un "serveur de noms", ou serveur DNS ("Domain Name Service")

Remarque : en fait, nous reviendrons sur ces affirmations plus loin, au sujet des NIS.

Dans le deuxième cas, la machine doit connaître un ou plusieurs serveurs de noms, ceux-ci sont déclarés dans le fichier **/etc/resolv.conf**. En fait ce fichier contient le nom de domaine et les adresses IP d'un ou plusieurs serveurs, par exemple :

```
# Fichier /etc/resolv.conf (Linux Mandrake 6.1)

search univ-mrs.fr
nameserver 193.51.105.2
nameserver 139.124.1.2
```

Le fichier **/etc/nsswitch.conf** permet, entre autres, d'indiquer comment doit se faire la résolution d'adresse (sur la ligne "hosts"). Par exemple, la suite

```
files dns nis
```

signifie qu'on cherche d'abord dans les fichiers locaux (i.e. fichier /etc/hosts), puis qu'on interroge le serveur DNS, et enfin qu'on interroge le serveur NIS (voir plus loin).

Il est a noter que la commande

```
nslookup
```

permet d'interroger des serveurs de noms "manuellement" : en donnant un nom de machine connue du serveur, on obtient son numéro IP, avec `server` on peut changer de serveur, et avec `ls nom-de-domaine` on peut voir la liste de toutes les machines connues du serveur sur ce domaine.

Il est bon de connaître quelques commandes très utiles, en particulier

`ping <machine>` (pour voir si la machine est joignable)

et

`traceroute <machine>` (pour connaître en outre la route utilisée pour atteindre la machine)

IV. Les fichiers /etc/inetd.conf et /etc/services

Rappel : le démon `inetd` doit être lancé au boot, il est chargé de recevoir les requêtes de services particuliers, et d'activer les serveurs appropriés à la demande. Autrement dit, il reçoit les requêtes formulées au travers des commandes `ftp`, `telnet`, … et permet l'accès au mécanisme RPC (Remote Procedure Call) d'appel de fonctions distantes.

Le fichier **/etc/inetd.conf** donne la liste des services disponibles. Plus exactement chaque ligne donne :
- le nom du service
- le type de "socket" utilisé ("stream" -> TCP, et "dgram" -> UDP)
- le protocole utilisé ("tcp" ou "udp")
- une option, "wait", ou "nowait" (obligatoire pour TCP)
- le nom du propriétaire du démon (root en général)
- le chemin absolu du démon à lancer et une liste de paramètres si nécessaire (ou "internal" si le service est réalisé par `inetd` lui-même).

Le fichier **/etc/services** associe à chaque service le numéro de port qui lui correspond et le protocole sur lequel il doit s'appuyer ("tcp" ou "udp").

Remarque : le démon `portmap` (ou `rpcbind` en System V) permet de répondre aux demandes d'exécution de fonctions en provenance de clients. Il permet notamment à un client de connaître le numéro de port associé à un service RPC donné (voir la fonction C "pmap-getport").

V. Configuration de serveur et clients NFS (Network File System)

Il s'agit d'un service d'accès à des fichiers distants : la machine sur laquelle les fichiers sont localisés est dite serveur, et les machines pouvant formuler des demandes d'accès à ces fichiers sont des clients. Les clients vont avoir le droit de "monter" (commande `mount`) des (parties de) partitions du serveur.

1. Côté serveur.

Le fichier **/etc/exports** donne la liste des (parties de) partitions autorisées à l'exportation, ainsi que d'éventuelles restrictions quant à l'exportation (seulement certaines machines sont autorisées, leurs droits d'accès sont restreints,…). Par exemple :

```
# Exemple de fichier /etc/exports (Linux Mandrake 6.1)

/usr            capucine(ro) camelia
/home           camelia(no_root_squash)
```

L'option `ro` signifie que la partition sera "read only", et l'option `no_root_squash` signifie que l'utilisateur root de la machine client aura les mêmes droits sur la partition que l'utilisateur root du serveur.

Lors d'une modification de ce fichier, il faut ré-exécuter la commande

```
exportfs
```

Remarque: dans d'autres systèmes, Solaris par exemple, on utilise le fichier `/etc/dfs/dfstab` qui fait appel à la commande `share`.

2. Côté client.

Le fichier **/etc/fstab** contient la liste des partitions locales à monter et leurs points de montage :

```
device         mount point          type                 mount options
                                    ("ext2": local,      ("defaults" pour le défaut)
                                     "msdos": DOS,
                                     "nfs": remote)
```

Il suffit de rajouter dans ce fichier les partitions du serveur qu'on souhaite monter, avec la syntaxe `nom-de-serveur:partition`

Remarque : la commande `mount` prend ses informations dans ce fichier, mais peut aussi être utilisée directement, avec des paramètres.

VI. Configuration de serveur et clients NIS

NIS = Network Information Service (anciennement Yellow Pages).
C'est un système qui permet le partage de fichiers d'identification sur un ensemble de machines : les machines sont regroupées à l'intérieur d'un "domaine NIS" qui contient un serveur maître (et éventuellement des esclaves) qui maintient des bases d'information auxquelles les clients ont accès. Grâce à ce mécanisme, l'utilisateur dispose d'un environnement identique (mot de passe, répertoire de login,…) sur toutes les machines du domaine.

Le serveur maintient des "bases de données NIS" (qui sont au format DBM, voir la commande `makedbm`). Lorsqu'il y a des changements dans ces bases, les esclaves en sont avisés par l'intermédiaire de la commande `yppush`. Les clients interrogent le serveur (ou les esclaves si le serveur est indisponible, par exemple à cause d'une panne). Pour ce faire, ils doivent faire tourner le démon `ypbind`.

Remarque : dans les anciennes versions de `ypbind` sous Linux (et encore dans certains Unix), le client n'a pas besoin de connaître le nom de son serveur, il fait un broadcast sur le réseau pour le trouver. Dans les versions de `ypbind` sous Linux au delà de la 3.3, le nom du serveur est donné dans un fichier de configuration (pour des raisons de sécurité).

1. Côté serveur.

L'installation requiert les étapes suivantes :

- s'assurer que `ypserv` et `makedbm` sont installés
- configurer comme on le souhaite le fichier `/var/yp/Makefile`. En particulier, s'assurer que `makedbm` est lancé avec l'option `-c` si on a une version de `ypserv` au delà de la 1.2.

  Il faut aussi surtout enlever, sous l'étiquette `all`, les "maps" qu'on ne souhaite pas construire.
- configurer les fichiers `/var/yp/securenets` et `/etc/ypserv.conf`
- positionner le nom de domaine NIS avec la commande

  `/bin/domainname nom-de-domaine-NIS`
- s'assurer que `portmap` a bien été lancé au boot, et lancer `ypserv`
- générer la base de données NIS par la commande `/usr/lib/yp/ypinit -m`
- Lancer `ypbind` qui permettra en particulier d'utiliser les "yp tools" (par exemple `ypwhich`)

Remarque 1 : Si le démon rpc.yppasswd est intégré à ypserv (ce qui est le cas dans les dernières versions de ypserv pour Linux), il n'y a pas besoin de le lancer séparément. Sinon, il faut également lancer ce démon.

Remarque 2 : lorsque l'on voudra prendre en compte des changements dans la base, on exécutera la commande make dans le répertoire /var/yp

Sur une machine esclave, il faut lancer la commande

```
/usr/lib/yp/ypinit -s nom-du-maître
```

à la place de /usr/lib/yp/ypinit -m, et sur la machine maître il ne faut pas oublier de déclarer la machine esclave dans /var/yp/ypservers

2. Côté client.

L'installation requiert les étapes suivantes :

- donner les informations nécessaires dans le fichier /etc/yp.conf, essentiellement sur la ligne
  ```
  domain nom-de-domaine-NIS server adresse-du-serveur
  ```
- positionner le nom de domaine NIS avec la commande
  ```
  /bin/domainname nom-de-domaine-NIS
  ```
  (il faudra aussi faire en sorte que cette commande soit exécutée au boot avant le lancement de ypbind)
- s'il n'existe pas, créer le répertoire /var/yp
- s'assurer que portmap a bien été lancé au boot, et lancer /usr/sbin/ypbind
- configurer le fichier /etc/nsswitch.conf, ainsi que le fichier /etc/passwd (de même que /etc/shadow et /etc/group si applicable). Une configuration assez répandue du fichier /etc/passwd consiste à ajouter à la fin +::::::

Remarque : la commande ypwhich permet de vérifier le nom de son serveur, la commande ypcat permet de visualiser l'état courant de la base (i.e. des "maps").

Sources.

* Jean-Marie Rifflet, "La communication sous Unix", Ediscience, 1996.

* Document URL http://www.redhat.com/mirrors/LDP/HOWTO/

Université de Provence
Marseille

Année 2000/2001
Maîtrise d'Informatique

TD/TP Réseaux n°2

Interconnexion de réseaux - Subnetting

Exercice 1.

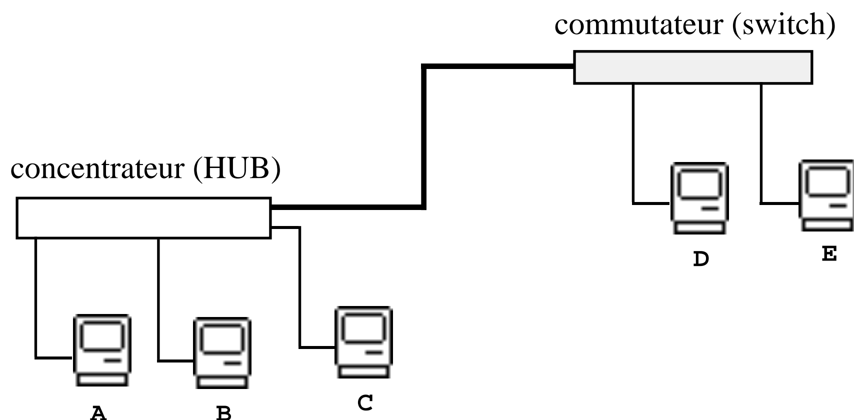
Soit l'adresse de réseau 102.0.0.0

1. Quelle est la classe de ce réseau, et quel est son masque de réseau ?

2. Comment peut-on découper ce réseau en 64 sous-réseaux ? Donner le masque de sous-réseau correspondant.
Quelles sont les plages d'adresses de machines sur ces sous-réseaux ?

Exercice 2.

Soit le réseau suivant :



1. Si un paquet de broadcast ARP est émis par la machine A, quelles machines recevront ce paquet ?

2. Si un paquet est émis par la machine A en direction de la machine C, quelles machines recevront ce paquet ?

3. Si un paquet est émis par la machine A en direction de la machine E, quelles machines recevront ce paquet ?

Exercice 3.

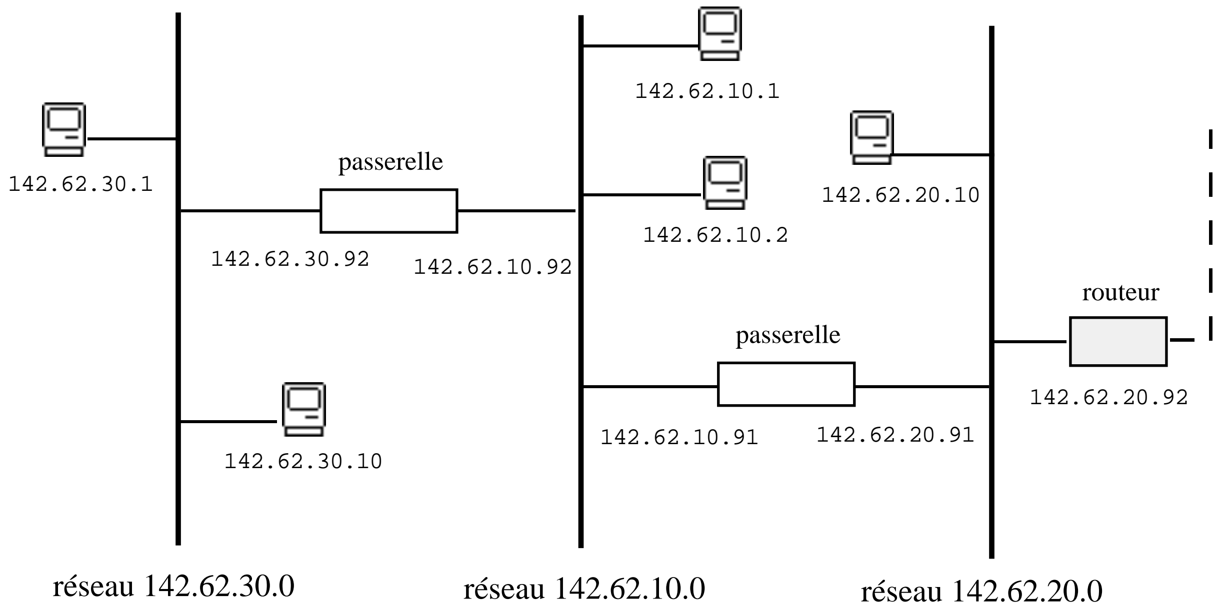
Soit le réseau donné par le dessin ci-dessous.

1. Quel est le protocole utilisé ?

2. Dans quelle classe de réseau nous trouvons-nous ?
Quelle est le masque de réseau correspondant ?

3. Pourrait-on affiner la valeur de ce masque ? Expliquer.

4. L'utilisateur sur la machine 142.62.20.10 signale que la machine 142.62.30.1 ne lui est plus accessible alors que la machine 142.62.10.1 peut être jointe sans problème.
Quelles commandes pouvez-vous utiliser sur les machines 142.62.20.10 et 142.62.30.1 pour tenter de d'identifier le problème ?



Exercice 4.

Dans tout l'exercice, on suppose que les machines considérées sont des PCs sous Linux Mandrake 6.1.

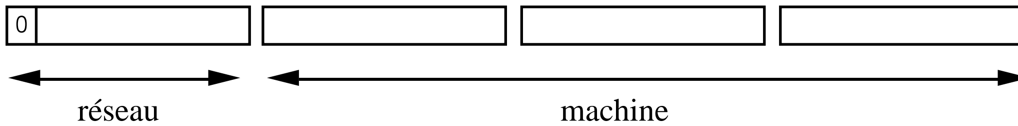
1. Expliquer les différentes phases de la procédure qu'une machine M1 doit effectuer pour atteindre une machine M2 se trouvant sur un réseau distant, lorsque l'utilisateur se sert d'une commande (par exemple `telnet`) en lui passant en paramètre le nom logique de M2 (par exemple `capucine.univ-mrs.fr`). On supposera que le service requis par l'utilisateur est bien disponible sur les machines.

2. Expliquez de quelle facon vous devez configurer une machine `tulipe` et une machine `begonia`, se trouvant sur le même réseau local, de telle façon que les comptes utilisateurs de `tulipe` (qui se trouvent dans le répertoire `/home/tulipe`) puissent être accessibles sur la machine `begonia`, avec le même chemin d'accès.
Quel sera sur `begonia` le nom du répertoire de login d'un utilisateur `dupont`, également déclaré sur `tulipe` avec le répertoire de login `/home/tulipe/etudiants/dupont` ?

3. Quelles sont les deux commandes que vous utiliseriez pour vous assurer qu'une machine, locale ou distante, est physiquement atteignable ? Quelles informations simples pouvez-vous obtenir en cas de succès de chacune de ces commandes ?

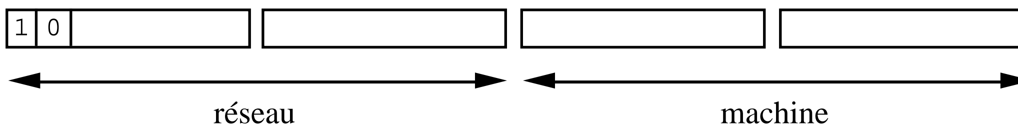
I. Rappels. Classes de réseaux

Classe A :



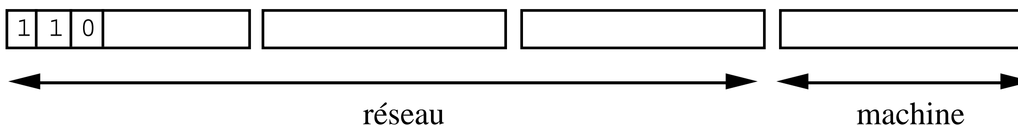
Octet de poids fort compris entre 0 et 2^7-1 = 127

Classe B :



Octet de poids fort compris entre 2^7 = 128 et $2^7+(2^6-1)$ = 191

Classe C :



Octet de poids fort compris entre 2^7+2^6 = 192 et $2^7+2^6+(2^5-1)$ = 223

Adresse de réseau → on positionne tous les bits de la partie adresse de machine à 0.
Adresse de broadcast → on positionne tous les bits de la partie adresse de machine à 1.

II. Subnetting

On peut découper n'importe quel réseau d'une de ces classes en sous-réseaux indépendants. Il suffit d'employer un masque de sous-réseau adapté; celui-ci peut contenir une *suite ininterrompue de bits à 1* hors de la partie adresse de réseau.

Exemple : Réseau 130.150.0.0 (classe B) → masque de réseau 255.255.0.0

En prenant un masque de sous-réseau égal à 11111111.11111111.11100000.00000000 = 255.255.224.0, on construit 8 sous-réseaux.

Sous-réseaux numéro 1: 10000010.10010110.**000**00000.00000000 = 130.150.0.0
numéros de machines de 10000010.10010110.**000**00000.00000001 à 10000010.10010110.**000**11111.11111111 = 130.150.0.1 à 130.150.31.254

Sous-réseaux numéro 2: 10000010.10010110.**001**00000.00000000 = 130.150.32.0
numéros de machines de 10000010.10010110.**001**00000.00000001 à 10000010.10010110.**001**11111.11111111 = 130.150.32.1 à 130.150.63.254
...

Sous-réseaux numéro 8: 10000010.10010110.**111**00000.00000000 = 130.150.224.0
numéros de machines de 10000010.10010110.**111**00000.00000001 à 10000010.10010110.**111**11111.11111111 = 130.150.224.1 à 130.150.255.254

III. Dispositifs d'interconnexion

On utilise généralement le terme de *passerelle* pour décrire les équipements d'interconnexion entre réseaux. En fait, ces dispositifs sont plus ou moins complexes :

1. Le *répéteur* (concentrateur, HUB) : permet d'augmenter la distance séparant des stations (intervient au niveau physique, régénère les données).

2. Le *pont* : permet de relier 2 segments de réseau ou 2 réseaux locaux de même type. Sert de répéteur, assure le filtrage (laisse passer vers un segment de réseau les trames qui lui sont destinées; utilise une table de routage qu'il peut gérer dynamiquement), et détecte les erreurs.

3. Le *commutateur* (switch) : permet de mettre en relation point à point 2 machines distantes (établit une relation privilégiée entre 2 noeuds, et évite l'envoi de trames vers les autres).
Commutation "par segments" → interconnexion de réseaux plutôt que de stations.
Gère une table de correspondances entre les adresses IP, les adresses MAC, et le numéro de port auquel chaque machine est reliée.

4. Le *routeur* : permet de relier des stations situées sur des réseaux qui peuvent être différents (ex : Ethernet et Token Ring).
Routage = fonction consistant à trouver un chemin optimal entre émetteur et destinataire (le protocole doit être *routable*, par exemple IP), elle utilise des tables de routage. Un routeur peut être logiciel.

5. La *passerelle* : couvre toutes les fonctionnalités. Peut être un ordinateur avec 2 cartes réseaux et les logiciels nécessaires.

Sources.

* Pierre-Alain Goupille, "Technologie des ordinateurs et des réseaux", Dunod, 2000.

* Document URL `http://www.teamvall.demon.co.uk/IPAApagebusiness.htm`

Université de Provence
Marseille

Année 2000/2001
Maîtrise d'Informatique

TD/TP Réseaux n°3

Applications client/serveur TCP/IP

L'objet de ces travaux pratiques est la programmation en C d'une petite application client/serveur TCP/IP.

1. Ecrire un programme client qui, après avoir lu une ligne de caractères au clavier, fait parvenir son nom de machine ainsi que la chaîne de caractères saisie à un serveur afin que celui-ci procède à l'affichage de cette chaîne à l'écran. Le nom de la machine serveur sera passé en paramètre à la commande invoquée par le client.

2. Ecrire le programme serveur qui, à chaque fois qu'il reçoit une chaîne de caractères en provenance d'un client, affiche cette chaîne à l'écran (ainsi que le nom de machine du client).

On pourra par exemple avoir le résultat suivant pour une exécution avec la machine "tulipe" en client et la machine "paquerette" en serveur :

```
tulipe> clienttcp paquerette
Mon nom est tulipe,
entrez une ligne au clavier
envoi de la premiere ligne
OK, je la fais parvenir au serveur paquerette...

tulipe> clienttcp paquerette
Mon nom est tulipe,
entrez une ligne au clavier
et j'essaye une deuxieme fois
OK, je la fais parvenir au serveur paquerette...

tulipe> clienttcp paquerette
Mon nom est tulipe,
entrez une ligne au clavier
et meme une troisieme
OK, je la fais parvenir au serveur paquerette...
```

```
paquerette> servtcp &
[1] 2178
paquerette>
**Voila ce que le serveur a recu en provenance de tulipe :
envoi de la premiere ligne
Fin du processus fils !!
**Voila ce que le serveur a recu en provenance de tulipe :
et j'essaye une deuxieme fois
Fin du processus fils !!
paquerette>
**Voila ce que le serveur a recu en provenance de tulipe :
et meme une troisieme
Fin du processus fils !!
```

I. Création et attachement des "sockets"

La communication entre machines se fait au moyen de "sockets". La première opération à réaliser est donc la création de sockets sur chaque machine, puis l'attachement de chaque socket à une adresse de machine.

La **création** d'une socket se fait par appel à la primitive `socket` :

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

En cas d'erreur, la fonction retourne -1, sinon elle retourne le "descripteur" de la socket (comparable à un descripteur de fichier).

Paramètres :

- `domaine` : définit le format des adresses possibles et l'ensemble des sockets avec lesquelles la socket pourra communiquer (AF_INET pour internet)
- `type` : détermine la sémantique des communications que la socket permet de réaliser (SOCK_STREAM pour un échange de séquences continues en mode connecté)
- `protocole` : généralement, il suffit d'utiliser la valeur 0

Remarque : la suppression d'une socket se fait par appel à la primitive `close` (comme pour la fermeture d'un fichier).

L'**attachement** d'une socket à une adresse se fait par appel à la primitive `bind` :

```
int bind(int descripteur, struct sockaddr *p, int len);
```

En cas d'erreur, la fonction retourne -1 sinon elle retourne 0.

Paramètres :

- `descripteur` : descripteur de la socket (résultat de "socket")
- `p` : pointeur sur la structure qui correspond à l'adresse à laquelle la socket est attachée. Dans le cas du domaine AF_INET, cette variable sera effectivement de type `struct sockaddr_in *`
- `len` : taille de l'objet pointé par `p`

Dans le cas du domaine AF_INET, la structure utilisée est :

```
struct sockaddr_in
   { short sin_family;   /* lui donner la valeur AF_INET */
     u_short sin_port;   /* numéro de port */
     struct in_addr sin_addr;   /* adresse de la machine */
     char sin_zero[8];  /* champ rempli de 0 */
   } ;
```

avec

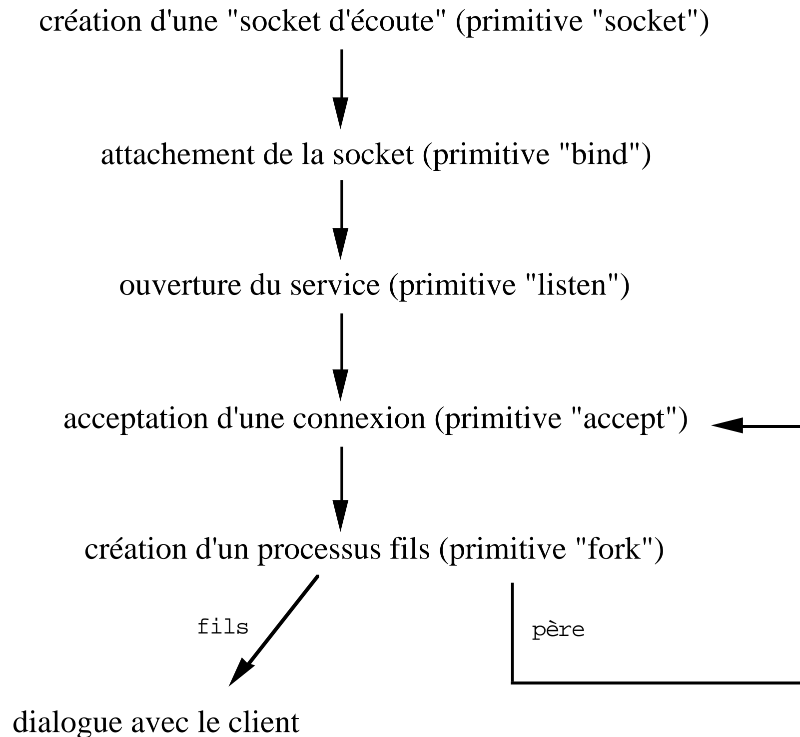
```
struct in_addr { u_long s_addr; } ;
```

Il faudra donc, préalablement à l'appel de "bind", remplir les champs `sin_family`, `sin_port`, et `sin_addr.s_addr` de la variable de type `struct sockaddr_in` utilisée :

- le champ `sin_addr.s_addr` pourra prendre la valeur `htonl(INADDR_ANY)`, où `htonl` est la fonction qui permet de convertir un entier long en notation "Big Endian"
- sur une machine client, le champ `sin_port` pourra simplement prendre la valeur 0; sur la machine serveur, il recevra la valeur `htons(noport)` où `htons` est la fonction qui permet de convertir un entier court en notation "Big Endian" et `noport` sera le numéro de port choisi.

II. Schéma d'un serveur TCP

A l'origine, le serveur crée et attache une *socket d'écoute* pour recevoir des demandes de connexion, et ouvre un service de connexion par la primitive "listen" (l'entité TCP du système commence alors à accepter des connexions). Il est passif jusqu'à la réception d'une demande de connexion. La primitive "accept" lui permet d'accepter une telle demande, par l'intermédiaire d'une *socket de service* nouvellement créée. En général, il délègue alors le traitement de la connexion (dialogue avec le client) à un processus fils, et se replace en position d'accepter une nouvelle demande de connexion. Ce qui peut être schématisé de la façon suivante :



L'**ouverture de service** se fait par appel à la primitive `listen` :

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

En cas d'erreur, la fonction retourne -1 sinon elle retourne 0.

Paramètres :

- `descripteur` : descripteur de la socket d'écoute (résultat de "socket")
- `nb` : nombre maximum de connexions en attente

L'**acceptation de connexion** se fait par appel à la primitive `accept` :

```
int accept(int descripteur, struct sockaddr *p, int *len);
```

En cas d'erreur, la fonction retourne -1, sinon elle retourne le descripteur de la "socket de service" créée.

Paramètres :

- `descripteur` : descripteur de la socket d'écoute
- `p` : pointeur sur la structure qui correspond à la socket du client. Dans le cas du domaine AF_INET, variable de type `struct sockaddr_in *`
- `len` : adresse d'un entier qui recevra la taille de l'objet pointé par `p`

A priori, l'appel à la fonction `accept` est bloquant (le processus est bloqué tant qu'il n'y a pas de demande de connexion).

<u>Remarque</u> : Si on délègue le traitement à un processus fils, il faut pouvoir prendre en compte la "mort" de ce processus pour éviter l'apparition de "zombis" (rappel : tout processus se terminant passe dans l'état zombi, dans lequel il reste tant que son processus père n'a pas pris connaissance de sa "mort"). On prévoira donc, lors de la réception du signal SIGCHLD, un déroutement sur une fonction (un "handler") effectuant un appel à la primitive `wait`, et ceci grâce à la primitive `sigaction`. Pour ce faire, on procédera de la façon suivante :

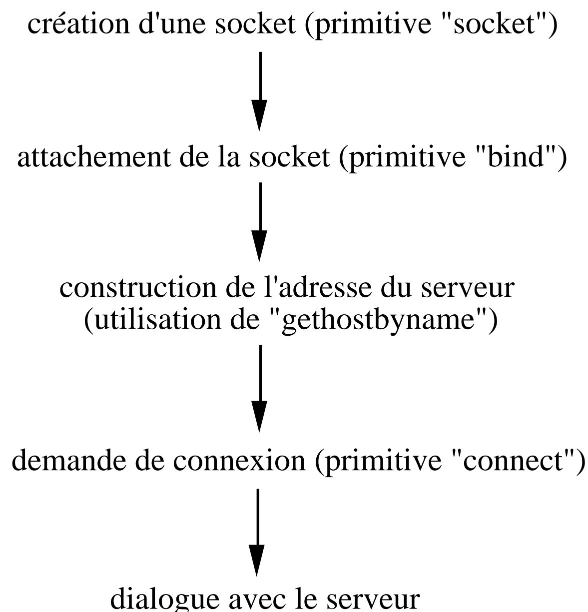
```
struct sigaction a;       /* déclaration d'une variable de type struct sigaction */
a.sa_handler = finfils;   /* le champ sa_handler de cette variable reçoit (le nom de)
                             la fonction sur laquelle le déroutement devra se faire */
sigaction(SIGCHLD, &a, NULL);
```

Et il reste à prévoir un appel à `wait` dans la fonction `finfils`.

Par ailleurs, `accept` retourne avec la valeur -1 et `errno` positionné à `EINTR` lorsqu'il subit une réception du SIGCHLD. On prévoira donc, lorsque ce cas se produit, un appel à la fonction `continue` pour reprendre la boucle à l'appel de `accept`.

III. Schéma d'un client TCP

C'est le client qui prend l'initiative pour l'établissement de la connexion. La demande est réalisée par la primitive "connect" (ce qui donne lieu à un échange de messages avec l'entité TCP du système du serveur). On a donc pour le client le schéma suivant (l'attachement de la socket étant optionnel) :



La **demande de connexion** se fait par appel à la primitive `connect` :

```
int connect(int descripteur, struct sockaddr *p, int len);
```

En cas d'erreur, la fonction retourne -1 sinon elle retourne 0.

Paramètres :

- `descripteur` : descripteur de la socket du client (résultat de "socket")
- `p` : pointeur sur la structure qui correspond à la socket du serveur. Dans le cas du domaine AF_INET, variable de type `struct sockaddr_in *`

- `len` : taille de l'objet pointé par `p`

A priori, l'appel à la fonction `connect` est bloquant. Si la file des connexions en attente sur le serveur est pleine, la demande sera réitérée pendant un certain temps. Si la connexion ne peut finalement pas être établie, la demande sera abandonnée (valeur de retour -1).

Connaissant le nom du serveur, la **récupération des informations** relatives à cette machine se fait par appel à la primitive `gethostbyname` :

```
struct hostent *gethostbyname(char *nom);
```

La structure `struct hostent` est définie de la façon suivante :

```
struct hostent
   { char *h_name;        /* nom de la machine */
     char **h_aliases;    /* liste d'aliases */
     int h_addrtype;      /* AF_INET */
     int h_length;        /* longueur de l'adresse (4 octets) */
     char **h_addr_list;  /* liste d'adresses */
   } ;
```

avec la macro `h_addr` qui correspond à `h_addr_list[0]`.

Le pointeur ramené en résultat par `gethostbyname` pointe en zone statique de la mémoire, il vaut donc mieux faire des recopies physiques (fonction `memcpy` pour recopier le champ `h_addr` dans `&adserv.sin_addr.s_addr`, où `adserv` est la structure qui correspond à la socket du serveur).

Remarque : on utilisera la fonction `gethostname(char *nom, int longueur)` pour que le client connaisse son propre nom de machine.

IV. Dialogue client/serveur

Les sockets permettent des communications bidirectionnelles. Deux tampons sont associés à chaque socket extrémité de la connexion :

- un tampon de lecture dans lequel l'entité TCP écrit les caractères en provenance de l'autre extrémité
- un tampon d'écriture dans lequel l'entité TCP lit les caractères qu'elle doit transmettre à l'autre extrémité.

Hormis les fonctions spécialisées `send` et `recv`, la communication entre le serveur et le client peut se faire simplement par les fonctions `write` et `read`. L'envoi de caractères peut se faire au moyen de :

```
ssize_t write(int descripteur, void *tampon, size_t nbcar);
```

et la réception par :

```
ssize_t read(int descripteur, void *tampon, size_t nbcar);
```

Sources.

* Jean-Marie Rifflet, "La communication sous Unix", Ediscience, 1996.

* Jean-Marie Rifflet, "La programmation sous Unix", Ediscience, 1995.

Université de Provence
Marseille

Année 2000/2001
Maîtrise d'Informatique

TD/TP Réseaux n°4

Applications client/serveur TCP/IP

Reprendre l'application programmée en TP de Systèmes au moyen de threads (récupération nom d'hôte et date) *sur le modèle de client/serveur TCP/IP étudié au TD/TP n°3*.

1. Ecrire un programme client qui saisit au clavier un entier (1 pour demander le nom d'hôte, 0 pour demander la date) et fait parvenir la requête correspondante au serveur, soit sous la forme "`GET /HOST`", soit sous la forme "`GET /DATE`". Il récupère ensuite la réponse du serveur et l'affiche à l'écran.
Le nom de la machine serveur sera passé en paramètre à la commande invoquée par le client.

2. Ecrire le programme serveur qui crée un processus fils pour répondre à chaque nouvelle requête d'un client. Il devra afficher l'adresse IP du client dont il reçoit la requête.

Le résultat d'exécution devra se présenter sous un format similaire à celui ci-dessous (avec la machine "hibiscus" en client et la machine "paquerette" en serveur) :

```
hibiscus> clienttcp paquerette
Requete au serveur : Host ou Date (1/0) ?
0
Le serveur paquerette est contacte, on lui demande la date...
Reponse recue du serveur : Thu Dec  7 18:18:53 2000
Et c'est fini...
hibiscus> clienttcp paquerette
Requete au serveur : Host ou Date (1/0) ?
1
Le serveur paquerette est contacte, on lui demande l'hote...
Reponse recue du serveur : paquerette
Et c'est fini...
hibiscus>
```

```
paquerette> servtcp &
[1] 3158
Recu demande de date de 193.50.131.48
Fils fini !!
Recu demande de nom d'hote de 193.50.131.48
Fils fini !!
```

Université de Provence
Marseille

Année 2000/2001
Maîtrise d'Informatique

TD/TP Réseaux n°5

Client/serveur TCP/IP - Petit jeu du pendu

A rendre le 19 Janvier 2001 au plus tard
(tout programme non correctement documenté ne sera pas corrigé)

Principe du jeu du pendu : un joueur J détient un mot et les autres joueurs doivent le deviner. Pour ce faire, ils soumettent des lettres (une par une) à J qui place la lettre dans le mot si possible et leur indique le résultat de ce placement. Le premier joueur qui a trouvé le mot a gagné. Ils jouent à tour de rôle, et disposent tous d'un maximum M de coups à jouer.

1. Ecrire le programme client (joueur qui cherche le mot) qui, au maximum M fois, a la possibilité d'envoyer au serveur (joueur J) soit une lettre pour tenter de la placer dans le mot, soit une proposition de mot s'il pense l'avoir trouvé. Le serveur lui répond soit en lui envoyant le squelette en cours du mot avec sa lettre placée éventuellement, soit par 1 ou 0 pour indiquer que le mot soumis est bon ou mauvais.
Le nom de la machine serveur sera passé en paramètre à la commande invoquée par le client.

2. Ecrire le programme serveur (joueur J) qui "enregistre" N joueurs pour le jeu, c'est à dire accepte les requêtes d'inscription de N clients, puis lance le jeu. A chaque tour, il reçoit de chaque joueur soit une lettre soit une proposition de mot et répond en conséquence (réponse fournie à tous les joueurs). Dès qu'un joueur a trouvé le mot, il est déclaré gagnant et le jeu s'arrête. Le jeu s'arrête sans gagnant si personne n'a trouvé le mot après M tours.
Il ne sera pas nécessaire d'avoir recours à des processus fils ou à des sous-threads.

Université de Provence
Marseille

Année 2000/2001
Maîtrise d'Informatique

TD Réseaux n°6

Applications client/serveur TCP/IP en JAVA

L'objet de ce TD est la programmation en JAVA de la petite application client/serveur TCP/IP écrite en C au TD n°3. On rappelle l'énoncé :

1. Ecrire un programme client qui, après avoir lu une ligne de caractères au clavier, fait parvenir son nom de machine ainsi que la chaîne de caractères saisie à un serveur afin que celui-ci procède à l'affichage de cette chaîne à l'écran. Le nom de la machine serveur sera passé en paramètre à la commande invoquée par le client.

2. Ecrire le programme serveur qui, à chaque fois qu'il reçoit une chaîne de caractères en provenance d'un client, affiche cette chaîne à l'écran (ainsi que le nom de machine du client).

On pourra par exemple avoir le résultat suivant pour une exécution avec la machine "tulipe" en client et la machine "azalée" en serveur :

```
tulipe% java clienttcp azalee
Mon nom est tulipe, entrez une chaine
salut le serveur
OK, je la fais parvenir au serveur azalee

tulipe% java clienttcp azalee
Mon nom est tulipe, entrez une chaine
deuxieme essai
OK, je la fais parvenir au serveur azalee

-----------------------------------------------------------------------

azalee% java servtcp
Voila ce que le serveur a recu en provenance de tulipe : salut le serveur
Voila ce que le serveur a recu en provenance de tulipe : deuxieme essai
```

I. Généralités

JAVA est un langage objet (langage de classes). Une application en JAVA n'est donc construite qu'à base de classes (pas de variables ni de fonctions globales), qui manipulent des instances de classes. Pour chaque application, exactement une des classes doit contenir une fonction :

```
public static void main(String args[])
  { ... }
```

C'est la fonction qui sera exécutée lors de la demande d'exécution de l'application.

II. Schéma du serveur TCP

Le fonctionnement du serveur TCP est quasiment identique à ce que nous avons vu au TD/TP numéro 3. Cependant, l'utilisation d'une instance de la classe `ServerSocket` permet de rendre la programmation plus légère. Le schéma du serveur devient :

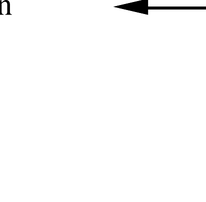
création d'une instance de ServerSocket

ss = new ServerSocket(port);

acceptation d'une connexion

ss.accept();

traitement de la requête



Ce qui donne le squelette de programme suivant :

```
import java.net.*;
import java.io.*;

public class servtcp
{
    final int port = 3000;
    ServerSocket ss;

    void fctserveur (Socket s)
    {
       // Travail du serveur...
    }

    void lance()
    {
        try
        {
           ss = new ServerSocket(port);  // création instance de ServerSocket
        }
        catch (IOException e)
        { System.out.println("exception ServerSocket " + e); }

        while(true)
        {
            try
            {
               Socket s = ss.accept();  // acceptation d'une connexion
               fctserveur(s);
```

```
            s.close();  // fermeture de la socket
        }
        catch (IOException e)
        { System.out.println("exception Socket " + e); }
    }
}

public static void main(String [] args)
{
    new servtcp().lance();
}
}
```

III. Schéma du client TCP

C'est le client qui prend l'initiative pour l'établissement de la connexion. Ici aussi, l'utilisation d'une instance de la classe `Socket` permet de rendre la programmation plus légère.
Il suffit de créer une instance de `Socket` avec l'adresse IP du serveur et le numéro de port en paramètres. Si la création réussit, la connexion est établie. Ce qui donne le squelette de programme suivant :

```
import java.net.*;
import java.io.*;

public class clienttcp
{
    final int port = 3000;
    Socket s;

    void fctclient (String nomserv)
    {
       // Travail du client...
    }

    void lance(String nomserv)
    {
        try
        {
            // création adresse IP du serveur (variable ipserv) :
            // instruction à mettre ici...

            s = new Socket(ipserv,port);  // création et connexion socket
            fctclient(nomserv);
            s.close();  // fermeture de la socket
        }
        catch (UnknownHostException e)
        { System.out.println("exception Socket ou getByName " + e); }
        catch (IOException e)
        { System.out.println("exception Socket " + e); }
    }

    public static void main(String [] args)
    {
        new clienttcp().lance(args[0]);
    }
}
```

Connaissant le nom du serveur, la **récupération de l'adresse IP** de cette machine se fait par appel à la fonction statique `getByName` de la classe `InetAddress`. L'objet retourné est de type `InetAddress` :

```
InetAddress ip;
ip = InetAddress.getByName(nomserv);
```

Remarque : pour que le client connaisse **son propre nom de machine**, on pourra utiliser l'instruction suivante :

```
String nom = InetAddress.getLocalHost().getHostName();
```

c'est à dire un appel à la fonction statique `getLocalHost` de la classe `InetAddress` (qui ramène un objet de type `InetAddress`), suivi d'un envoi du message `getHostName` à l'objet retourné.

IV. Dialogue client/serveur

Afin de pouvoir lire et écrire dans une socket `s`, un flux d'entrée et un flux de sortie lui seront associés :

```
// déclaration et création flux d'entrée :
DataInputStream i = new DataInputStream(s.getInputStream());

// déclaration et création flux de sortie :
DataOutputStream o = new DataOutputStream(s.getOutputStream());
```

La lecture et l'écriture d'une chaîne de caractères pourront se faire par :

```
// Lecture d'une chaîne terminée par un RC
String st = i.readLine();

// Ecriture d'une chaîne
String st;
...
o.writeBytes(st);
```

Remarque : la saisie d'une chaîne au clavier se fera par

```
BufferedReader d =
       new BufferedReader(new InputStreamReader(System.in));
String chaine = d.readLine();
```

Sources.

Documentation JAVA de SUN