Cours d'administration Unix

F. Butelle M. Mayero

2015

Table des matières

1 Rappels d'unix

Unix est un système d'exploitation. C'est à dire qu'il est une interface entre des êtres humains utilisateurs ou administrateurs et une machine. Il gère ainsi les disques durs, la mémoire et le Système de Gestion de Fichiers (SGF). Pour Unix tout est fichier ou organisation de fichiers, c'est pourquoi on ne considérera ici que la partie SGF d'Unix.

Unix est multitâches et multi-utilisateurs. Il dispose donc de mécanismes pour gérer plusieurs processus lancés par plusieurs utilisateurs (locaux ou distants) et des accès partagés ou non etc.

L'objectif de ce cours est de donner une connaissance relativement approfondie de l'utilisation usuelle d'Unix.

1.1 Bref historique et quelques versions

— 1965-70 : Premiers systèmes d'exploitation (CTSS, TENEX, MULTICS). MULTICS : Multiplexed Information and Computing System.
— 1969 : Ken Thomson et Denis Ritchie (Laboratoires BELL) développent un système mono-utilisateur, interactif. Ce système est écrit en langage machine sur PDP-7 (DEC) puis sur PDP-9 (Ce système est issu des remarques faites sur les défauts de MULTICS, c'est une version réduite de Multics appelée UNICS (UNiplexed Information and Computing Service), rapidement contractée en Unix).
La date du 1er janvier 1970 (dite « epoch ») est considérée comme étant la date de naissance du système Unix, ce qui explique pourquoi toutes les horloges système des systèmes d'exploitation Unix démarrent à partir de cette... époque.
— 1971 : Version temps partagé sur PDP-11
— 1971-73 : D. Ritchie : Développement du langage C à partir du langage BCPL. Depuis, Unix est indissociable du langage C, car une partie de plus en plus croissante du système est écrite en C plutôt qu'en langage machine ou en assembleur.
— 1975 : UNIX version 6 commence à être proposé par les laboratoires BELL. (toujours sur PDP-11).
— 1978 : UNIX V7, première version réellement commercialisée (Unix est une marque déposée des laboratoires BELL). Début des efforts de portabilité en isolant la partie liée à la machine.
— 1981 : Premier DOS
— 1982 : UNIX System III : Ajoute au précédent une idée nouvelle de communication entre processus : les fichiers FIFO. Ajoutons à cela des outils de comptabilité et d'atelier logiciel.
— 1983 : UNIX System V ; Apport de l'éditeur pleine page VI.
— 1985 : premier Windows
— 1985 : UNIX V8. Création de Minix par Andrew S. Tanenbaum pour ses étudiants.
— 1989 : UNIX System V R4.
— 1991 : Linus Torvalds, décida de concevoir Freax, sur le modèle de Minix. Tanenbaum refuse d'intégrer les améliorations proposées dans Minix. Freax est devenu Linux à cause du nom du répertoire du serveur ftp dans lequel il était hébergé.
— 1993 : Création de la distribution debian.

En parallèle de ce développement, l'université de Berkeley a développé sa vision d'Unix. Ce sont les UNIX BSD (Berkeley Software Distribution). La première version UNIX BSD est sortie en 1977.

Il existe aussi des systèmes « UNIX-like »c'est à dire des réécritures d'Unix sur diverses plates-formes (citons SunOs, Solaris, HPUnix, IDRIS, MINIX, Linux, XENIX, ULTRIX, SPIX,...).
L'avenir s'oriente peut-être vers une réunification de ces versions sous une norme : POSIX (malheureusement vendue très cher par l'IEEE).

1.2 Interpréteurs de commandes

Un interpréteur de commandes dit aussi « Shell » (coquillage en Français) est un langage de commande dans le monde UNIX. On l'appelle ainsi car ce n'est que l'enveloppe externe du système et c'est pourtant la seule chose que voit l'utilisateur courant.

Il n'y a pas un interpréteur de commandes sur UNIX, il y en a une longue liste, mais on cite souvent les deux versions les plus importantes :

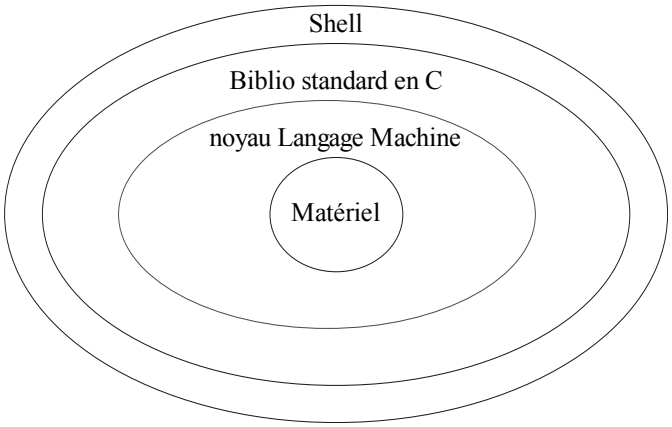
— L'interpréteur de commandes de base fournit par ATT : le Bourne Shell qui est lancé par `/bin/sh`. L'invite (ou « prompt ») par défaut est le $.
— Le C-shell `/bin/csh` de l'université de Berkeley disponible à peu près sur toutes les plates-formes UNIX. Il est compatible avec le Bourne Shell mais il présente des fonctions plus riches, tel un historique des commandes, des alias, une gestion des processus. L'invite par défaut est le %.

A ces interpréteurs de commandes s'ajoutent de nombreuses variantes dont les plus connues sont : `/bin/bash`, `/bin/ksh`, `/bin/tcsh`,...

A noter que le Bourne Shell du début n'a plus grand chose à voir avec la version normalisée : le POSIX-shell, bien plus riche que sont prédécesseur et qui a repris à son compte les avancés du C-shell.

On annonçait aussi la généralisation du Korn-shell, à 99% compatible avec le Bourne Shell et reprenant à son compte les avancées du C-shell. Mais ce qui semble se détacher à l'heure actuelle, en particulier sur Linux, c'est le BASH (Bourne-Again Shell), version très riche se basant sur le Bourne shell et incorporant à la fois les idées du C-shell et du Korn-shell.

C'est un langage (interprété) permettant à un utilisateur de lancer depuis un terminal l'exécution de commandes. Mais c'est aussi un langage de programmation permettant de créer des fichiers de commandes. Ces fichiers de commandes sont paramétrables, ils utilisent des variables et des structures de contrôle pour les commandes répétitives (voir section 3.9).



1.3 Session (en mode texte)

Lorsque le terminal est sous tension ou lorsque la session telnet (ou ssh) est ouverte, le terminal affiche `login:` A ce niveau il faut fournir le nom par lequel le système nous connaît, à savoir un nom de login. A l'origine, ce nom ne pouvait comporter plus de 8 caractères. Ensuite le système demande un mot de passe : `password:`

Une fois le mot de passe saisi, la session est ouverte et un message est affiché comportant diverses informations comme la date, le numéro de version du système, les éventuels problèmes du site, l'existence de courrier reçu, etc.

La fermeture d'une session UNIX : logout ou exit. Dans certains cas, le système peut refuser la fermeture de la session en cas de processus subsistant en tâche de fond. Il faut alors commencer par terminer ces processus (voir la gestion des processus) puis fermer la session.

1.4 Ligne de commande

Une ligne de commande UNIX peut être aussi simple que :

`<commande> <paramètres><RC>`

(<RC> = Retour Chariot ou *Carriage Return*) ou bien elle peut être composée d'appels à plusieurs programmes reliés ou non entre eux.

Par exemple :

`<commande1> <paramètres> | <commande2> ; <commande4> && <commande 5>`

ce qui n'est toujours qu'une seule ligne de commande...

1.5 Modes d'exécution

On distingue plusieurs façons d'enchaîner les commandes.

— Les commandes simples, dites « synchrones » : chaque commande est complètement achevée avant le lancement de la suivante. Elles sont terminées par un ; ou par un <RC> (Retour Chariot) ou par un opérateur logique (**&&** ou **||**) enchaînant le lancement de la commande suivante suivant le résultat (code d'erreur) généré par la commande précédente.

— Les commandes simples en tâche de fond ou « asynchrones »(« background ») : si une commande est terminée par & (et commercial ou « ampersand »), on n'attend pas sa terminaison pour lancer la suivante.

— Les commandes groupées : on peut mettre une suite de commandes entre parenthèses pour les grouper. Le système permet alors de considérer cette commande groupée comme une commande simple.

— Fichiers de commandes (ou « shell-scripts ») : ce sont des listes de commandes enregistrées dans un fichier texte dont on lance l'exécution simplement en tapant le nom du fichier (voir fichiers batch ou .bat en MsDos/Windows).

1.6 Commandes de base

Des détails et d'autres commandes seront données en travaux pratiques. Les commandes qui suivent sont utilisables dans tous les Shells.

— date : donne la date et l'heure
— passwd : permet de changer son mot de passe
— tty : donne le numéro logique du terminal auquel est attaché l'utilisateur
— who : donne la liste des utilisateurs connectés
— whoami : donne l'utilisateur courant

Exercice (tour de table) : que font les commandes suivantes ?

```
ls -a -F ; ls -l
cat > toto ; less toto
ln toto titi ; ln -s toto tata
ps auxww
ps -ef |grep xemacs
man ps
top
cd ~/ens
cd ..
cd
cd ens/2009/admunix/cours1
file ../CoursUnix.tex
mkdir truc ; cd truc
touch a b c ; rm a
rm -f a
alias
cd ..; rm truc; rmdir truc
rm -rf truc
for i in `seq 1 10`; do echo $i; done
export var=salut; echo var; echo $var
bash; echo $var; export var=bye
exit; echo $var
df
du ../../ ; du -s ../../
less /etc/fstab
find ../.. -name coursUnix.tex -print
find ../.. -name coursUnix.tex \
        -exec grep IUT {} \;
```

2 Système de Gestion de Fichier

Sous Unix un fichier :

— est toujours désigné par un nom.
— possède un unique i-node (certaines informations concernant le fichier).
— possède les fonctionnalités suivantes : ouverture, fermeture, lecture (consultation), écriture (modification)

2.1 Notion d'i-node

Définition (i-node) : (ou « nœud d'index »)
Structure crée au même moment qu'un fichier afin de contenir ses informations fondamentales. Tous les i-nodes sont conservés dans une table, et sont identifiés par un numéro d'index.

Un SGF Unix est usuellement (voir ext2fs) muni d'une table des i-nodes.

Tout fichier physique possède un descriptif unique appelé i-node. L'i-node contient la totalité des informations sur le fichier, sauf le nom. Les i-nodes sont tous de même taille. Les informations sont les suivantes :

— Type de fichier : **-**, **d**, **l**, **c**, **p**, **b**, **s** (voir plus loin)
— Permissions/Droits d'accès (idem)
— Nombre de liens pointant sur cet i-node (physiques) : correspond au nombre de références c'est à dire au nombre de noms.
— UID : identité du propriétaire (créateur ou affecté par `chown`)
— GID : identité du groupe propriétaire (affecté par chgrp ou hérité du répertoire ou de l'identité de groupe du processus créateur).
— Taille du fichier.
— atime : date de la dernière lecture (`access`) (voir `ls -ltu`).
— mtime : date de la dernière modification du fichier physique (voir `ls -l`).

— ctime : date de la dernière modification de l'i-node lui-même (voir `ls -ltc` et la commande `stat` pour toutes ces dates).
— Adresse du fichier (adresses des blocs occupés).

2.2 Types de fichier

Du point de vue de l'utilisateur, un fichier est finalement comparable à un tableau de caractères. La structure interne est choisie suivant l'application qui l'utilise.

— Les fichiers simples ou « réguliers »(codés `-`) : En général se sont des fichiers textes : des suites de lignes, chacune terminée par le caractère <LF> (line feed, et non <CR><LF> comme sous MsDos). Les autres fichiers peuvent contenir tout caractère ou code.
— En plus de ces fichiers simples, il existe des fichiers de gestion de périphérique. En Unix, tout périphérique (`device`) est associé à un fichier unique spécial. On distingue deux types de fichiers de gestion de périphérique :
  — Les périphériques à accès en mode bloc (codés **b**) : associés à des périphériques accessibles par blocs entiers de donnés : par exemple les disques durs.
  — Les périphériques à accès en mode caractère (codés **c**) : associés à des périphériques dont l'accès se fait en mode caractère uniquement : la mémoire, l'écran, le clavier, etc.

  Les fichiers de gestion de périphériques sont très généralement regroupés dans le répertoire `/dev`.
— Les fichiers FIFO ou « tube nommés »(ou pipes, codés **p** ou |) forment un 4e type : ils sont des files d'attente de communication entre processus.
— Les répertoires (codés **d**) : voir le paragraphe suivant.
— Les liens symboliques (codés **l**) idem.

Parfois on rencontre d'autres codages pour d'autres fichiers spéciaux : n (network), s (socket)... (dépendants de la variante d'UNIX utilisée).

2.3 Répertoires

Un répertoire Unix contient en fait un catalogue de n-uplets dont les champs (numéro de i-node, nomfichier). Ceci permet en particulier d'avoir plusieurs noms pour un même i-node donc un même fichier (voir la commande `ln`).

Vu par l'utilisateur, un répertoire (ou directory) est un fichier contenant un ensemble d'enregistrements (numéro de i-node, nomfichier) (Unix System V) ou plus sophistiqué avec d'autres informations permettant d'autoriser des tailles de nom de fichiers variables (jusqu'à 255 caractères) (Unix BSD). La création d'un répertoire produit ce « fichier »avec deux enregistrements particuliers avec les noms de fichiers « . » (le répertoire lui-même) et « .. »(son répertoire père).

L'information contenue dans un répertoire n'est manipulée que par le système (sur commandes de l'utilisateur).

2.4 Permissions/Droits de base : protections (attributs) d'un fichier et d'un répertoire

Les mécanismes sont tous basés sur les attributs d'un fichier : Type, Droits d'accès, Nombre de liens, Propriétaire, Groupe, Dates (accès, modification).

Dans les droits d'accès on distingue le propriétaire (« user »), les utilisateurs du même groupe que le groupe auquel appartient le fichier (« group ») et les autres utilisateurs (« others »).

Pour chacune des trois classes d'utilisateurs qui précède, on attribue au fichier 3 droits distincts : `r` : lecture, `w` : écriture, `x` : exécution.

Exemple : un fichier peut avoir les droits suivants : `rwxr-xr--`

C'est à dire que le propriétaire de ce fichier a tous les droits. Les utilisateurs appartenant au même groupe que le groupe auquel appartient ce fichier ont le droit de lecture et exécution. Enfin, les autres utilisateurs n'ont que le droit de lecture.

Remarque : pour un répertoire, l'interprétation des droits d'accès est le suivant :

- `r` : permet de lire le contenu du répertoire
- `w` : indique la possibilité de créer ou de détruire des fichiers dans ce répertoire
- `x` : indique le droit de « traverser »ce répertoire.

Les droits sont codés en binaires 1 pour actif, 0 pour inactif. Ensuite les codes binaires sont groupés en 3 paquets de 3 (par exemple `rwxr-xr--` devient `111 101 100`) puis codés en octal (l'exemple devient 754).

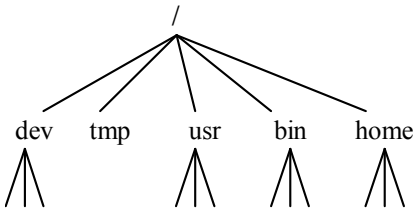
Quelques commandes de modification des droits d'accès :

`chmod {u,g,o,a}{+,-,=}{r,w,x}` <fichier ou répertoire>
changer les droits (u : user, g : group, o : others, a : all), on peut aussi donner une valeur octale :
`chmod` <valeur> <fichier>.
`chgrp <group> <fichier>` : changer le groupe.
`chown <nom> <fichier>` : changer le propriétaire.

Attention il existe d'autres droits matérialisés par des drapeaux autres que `rwx`, voir la section Sécurité.

2.5 Arborescence

Il y a un et un seul répertoire racine notée / (et non \ comme sous Dos/Windows).

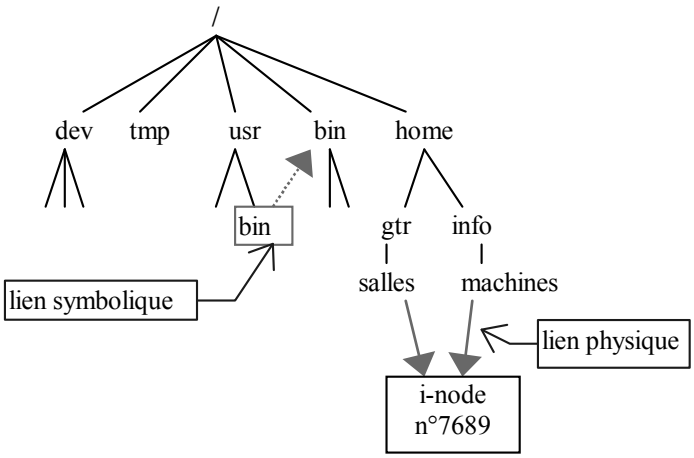


2.6 Liens

La relation entre un nom de fichier et un fichier physique (*i-node*) s'appelle un lien. C'est le lien physique que l'on distingue du lien symbolique qui relie un nom de fichier avec un autre nom de fichier.

Dans l'exemple de la figure suivante, `/usr/bin` est un lien symbolique sur `/bin` ;

`/home/gtr/salles` et `/home/info/machines` sont un seul et même i-node comptant donc deux références. Si on détruit `/home/gtr/salles`, le fichier `/home/info/machines` existera toujours mais l'i-node ne comptera plus qu'une seule référence.



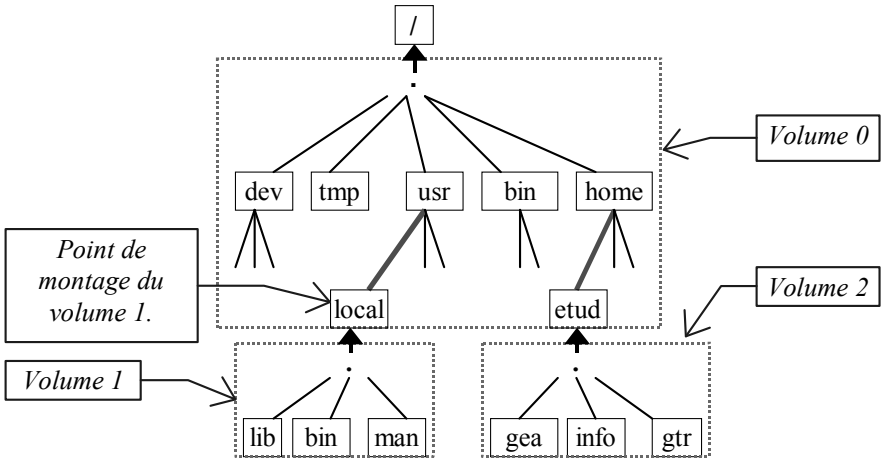
Restrictions en matières de liens :

- Aucune en matière de lien symbolique. En particulier on peut créer un lien symbolique sur un fichier qui n'existe pas.
- Un répertoire ne comporte que deux liens physiques (lui-même et son père).
- On ne peut établir de lien physique entre fichiers ou répertoires appartenant à des SGF différents (voir les volumes dans la section suivante.

2.7 Montage et partitions

Principes de « l'arborescence globale » :
Tout support physique organisable en système de fichiers (disques, disquettes, CDs, clés USB,...) peut contenir un sous-arbre de l'arborescence globale. L'unité support de la racine joue un rôle particulier (il doit être toujours accessible).

Les autres partitions sont « montés »(mount) selon les besoins par une commande particulière en un point de l'arborescence.



2.7.1 Les types de systèmes de fichiers

Unix utilise principalement deux types de systèmes de fichiers :
- Swap qui sert à stocker la mémoire virtuelle, qui est utilisée quand la mémoire vive est pleine
- Ext2/3 qui sert à stocker les fichiers et les répertoires (il existe de nombreuses alternatives à Ext2, à savoir Ext3, Ext 4, ReiserFS, XFS, JFS, etc...).

2.7.2 Découpage et taille

Traditionnellement, on crée une partition avec un système de fichiers de type Swap de taille égale à la taille de la mémoire vive quand celle-ci est supérieure ou égale à 128 Mo. Mais si l'on sait que l'on va utiliser des applications très gourmandes en mémoire on peut choisir de plus grandes tailles.

Cette partition est appelée partition de Swap ou partition d'échange. Pour stocker les fichiers et les répertoires, on crée souvent plusieurs partitions avec un système de fichiers de type Ext3 (ou une de ses alternatives).

Pour les serveurs, les administrateurs Linux ont souvent pour habitude de sectionner le système de fichiers en de nombreuses partitions pour assurer une meilleure résistance du système aux pannes de disque dur, aux failles de sécurité et aux attaques de tout type. Par exemple, il ne faudrait pas qu'un simple utilisateur puisse saturer la partition sur laquelle se trouve la racine du système de fichiers juste en remplissant son répertoire personnel (`/home/son_login/`), car ceci pourrait rendre le système instable. Il ne faudrait pas non plus que les journaux système (ou logs) qui se trouvent dans le répertoire `/var/log/` remplissent la partition sur laquelle se trouve la racine suite à une attaque réseau ce qui aurait la même conséquence. Il est également bon de mettre sur une partition à part le répertoire `/tmp/` car il est accessible en écriture à tous les utilisateurs et à tous les programmes.

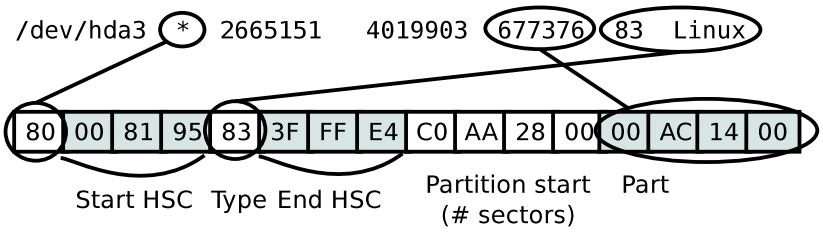
Sur une machine personnelle, de telles précautions ne sont pas forcément nécessaires et imposent des contraintes inutiles sur la taille des répertoires.

Exemple de choix de répartition des partitions :

Partition	**Taille**
Swap	Égale ou double de la mémoire vive
/tmp	150 Mo
/var	300 Mo
/	2 Go
/home	200 Mo par utilisateur

2.7.3 La table de partition

Un enregistrement représentant la table des partitions contient 16 octets divisés en 6 champs :



La partition représentée ci-dessus est `/dev/hda3` (la 3ème partition) donnée par la commande `fdisk /dev/hda` :

```
Disk /dev/hda: 64 heads, 63 sectors, 1023 cylinders
Units = sectors of 1 * 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1		63	1641023	820480+	c	Win 95 FAT32 (LBA)
/dev/hda2		1641024	2665151	512064	7	HTFS/NTFS
/dev/hda3	*	2665152	4019903	677376	83	Linux
/dev/hda4		4019904	4124735	52416	82	Linux Swap

Détaillons les champs un par un.

- Le premier champ est sur un octet et vaut soit 80h (partition "amorçable") soit 00h. Si le champ est à 00h, fdisk affichera un espace, sinon ce sera une étoile (*).
- Les champs "start HSC" et "end HSC" sont la trace d'une représentation ancienne en "Head/Sector/Cylinder" un octet pour le numéro de tête (Head), 6 bits pour le numéro de secteur et 10 bits pour le numéro de cylindre. Usuellement on note plutôt dans l'ordre CHS et 0/0/1 (cylindre 0, tête 0, secteur 1) est le premier secteur d'un disque, le suivant sera 0/0/2 et ainsi de suite jusqu'à $0/0/N_S$ où N_S est le nombre de secteurs angulaires pour une rotation complète du disque. Ensuite vient 1/0/1, 1/0/2 etc. Le dernier est $N_C - 1/N_H - 1/N_S$ ce qui donne $N_C * N_H * N_S$ secteurs adressables. Cette technique est insuffisante (si on a juste un octet pour le nombre de cylindres) pour les gros disques durs actuels (voir le Linear Bloc Addressing ou LBA, dans lequel les valeurs H=254 S=63 C=1023 sont utilisées pour dire qu'elles ne sont pas utilisables (en hexa FEFFFF) ! D'ailleurs 254 ou 255 têtes n'a aucun rapport avec la réalité physique).
- Le champ "type" donne le type de SGF sur cette partition, ici on a 83h qui est le code correspondant à Linux (Ext2). La liste des types de partitions connues est obtenue par la commande l dans fdisk.
- L'avant dernier champ permet de trouver l'adresse de début de la partition, en nombre de secteurs (attention les octets sont dans l'ordre de poids faible en premier). 28h = 40, AAh = 170, C0h= 192, $40*256^2+170*256+192 = 2665152$ ce qui correspond bien à l'adresse annoncée par `fdisk`.
- Dans le dernier champ on a la taille de la partition (de même, les octets de poids faibles en premier) ACh = 172, 14h = 20, donc $20 * 256^2 + 172 * 256 = 1354752$ secteurs. `fdisk` nous dit que la taille de la partition /dev/hda3 est de 677 376 blocs. Un secteur est de 512 octets, donc 1 354 752 secteurs correspond à 677 376 kio. Un bloc comporte donc 1024 octets.

2.7.4 Le MBR (Master Boot Record)

Le MBR est un secteur (l'unité de stockage de base de 512 octets) qui se trouve obligatoirement au tout début du disque dur. Sa structure est la suivante (ici l'unité est l'octet) :

Début	**Fin**	**Contenu**
0	445	Programme (ex. GRUB) vérifiant la table des partitions et lançant secteur d'amorce de la partition active
446	509	Table des partitions
510	511	Signature "magic number" (=AA55h)

On déduit du tableau précédent que seulement 64 octets sont réservés pour la table des partitions donc il n'y a de la place que pour 4 partitions dites partitions primaires (puisque d'après la description précédente, il faut 16 octets par partition). Pour contourner cette limite, il a été défini des partitions étendues, contenant elles-mêmes des tables de partitions. Ces tables sont stockées dans des Extended Boot Record, en liste chaînée, dont chaque élément à la même structure qu'un MBR.

2.7.5 Quelques commandes utiles

La vieille commande `dd` est très pratique pour accéder aux disques à bas niveau.

Notons, pour sauver le MBR dans le fichier dumpMBR (if="input file", of= "output file", bs= "bloc size", count= nbre de blocs) :

```
dd if=/dev/hda of=dumpMBR bs=512 count=1
```

Pour restaurer le MBR :

```
dd if=dumpMBR of=/dev/hda bs=512 count=1
```

pour "voir" les octets en hexa concernant uniquement la table des partitions (en sautant les 446 premiers octets utilisés pour le programme "loader") :

```
od --format=x1 -j 446 dumpMBR
```

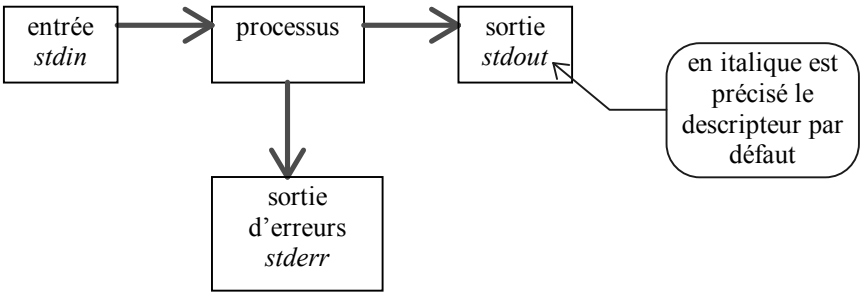
3 Processus et interpréteur de commandes

3.1 Définition

« Un processus est fondamentalement un programme qui s'exécute. »Andrew S. Tanenbaum (Minix, 1987).

En fait il consiste en un programme exécutable, ses données et sa pile d'exécution, son compteur ordinal, son pointeur de pile et autres registres ainsi que toutes informations nécessaires à l'exécution du programme.

En particulier, on parle d'environnement d'un processus : l'ensemble des variables nécessaires à son exécution et qui peuvent être propagées à ses fils ainsi que les descripteurs de fichier d'entrée/sortie du processus :



Principaux états d'un processus :

- « Created » état intermédiaire en général de courte durée (sauf peut être pour les systèmes temps-réel) avant la prise en charge par l'ordonnanceur.
- « Waiting » éligible (dit prêt ou en attente).
- « Running » ou actif : le processus est en cours d'exécution par un processeur.
- « Blocked » : attente d'évènement : signal ou entrée/sortie.
- « Terminated » : terminaison soit par fin normale (exit) soit par réception d'un signal d'arrêt. L'état évolue en « Zombie » lorsque le signal de terminaison lancé par ce processus est ignoré par son processus père (wait), il ne peut alors être supprimé de la table des processus (d'où ce nom : à la fois mort et vivant).

3.2 Détournement des entrées/sorties

Lors du lancement d'un processus, on peut, pour la durée de l'exécution de ce processus, changer l'assignation par défaut des entrées/sorties.

- `commande < fichier` : substitue le fichier donné à l'entrée standard
- `commande > fichier` : substitue la sortie standard au fichier donné — En C-shell et en Bash, il existe une option pour interdire l'écrasement du fichier s'il existe déjà (option positionnée par « set noclobber »).
- `commande >! fichier` : force l'écrasement de fichier s'il existe déjà
- `commande >> fichier` : idem `>` mais en ajoutant à la fin du fichier donné (de même on peut faire `>>!` fichier)
- `commande1 | commande2` : la sortie standard est utilisée comme entrée standard de la commande2.
- `commande >& fichier` ou `commande &> fichier` : redirection de la sortie standard et la sortie d'erreurs dans le fichier donné (de même il existe `>>&`, `>&!`, `>>&!` et `|&`). Attention, en BASH la syntaxe est différente.

Exemples :

`make >& make.out`

Redirection de sortie et sortie d'erreurs dans des fichiers distincts : `(make > make.out) >& make.err`

Remarque : pas d'espace entre `>` et `&` et `!`

En BASH : `make 2> make.err > make.out` (la sortie d'erreur est identifiée par le descripteur 2, la sortie normale par 1 ou rien). En BASH, redirection de la sortie d'erreur et std dans le même fichier : `make >resultat 2>&1` (note : le descripteur 0 est stdin).

Il existe aussi un mécanisme de redirection dit en « T »par le biais de la commande `tee`. Cette commande permet de recopier son entrée standard en sortie standard et également dans un fichier donné en argument.

Exemple : `who | tee liste | wc -l > nombre`

Fournit d'une part la liste des utilisateurs dans le fichier liste et leur nombre dans le fichier nombre.

3.3 Quelques commandes de gestion de processus

- `ps [opt]` liste des processus actifs. On obtient, par exemple :

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 10:00 ?        00:00:00 init [5]
...
```

où :

- UID est le nom de l'utilisateur qui a lancé le processus
- PID correspond au numéro du processus
- PPID correspond au numéro du processus parent
- C est le facteur de priorité : plus la valeur est grande, plus le processus est prioritaire
- STIME correspond à l'heure de lancement du processus
- TTY correspond au nom du terminal
- TIME correspond à la durée de traitement du processus
- COMMAND correspond au nom du processus.

– `nice +/-n <commande>` exécute la commande avec une priorité diminuée ou augmentée (la valeur de priorité va de -20 à +19 avec une valeur par défaut à 10, la plus faible est la moins « gentille/nice »donc la plus prioritaire).
– `sleep n` suspend l'exécution pendant n secondes
– `at <date et/ou heure>` exécution retardée à date et heure fixée
– `wait <num processus>` attente de la fin d'exécution d'un processus défini par son numéro
– `exec <commande>` la commande donnée est exécutée sans créer de nouveau processus. A la fin de la commande la session est donc terminée.
– `exit [n]` termine une suite de commandes avec un code de retour `n`. Si `n` est omis, la valeur de retour de `exit` est celui de la dernière commande exécutée (0 est la valeur par défaut si aucune erreur n'a été rencontrée).
– `top` donne les processus actifs en temps réel
– `kill -[n] pid` envoie le signal de numéro n au processus d'identité pid (par défaut c'est le signal 15 `SIGTERM`).
– `killall chaine` envoie par défaut le signal SIGTERM à tous les processus dont le nom correspond à la chaîne donnée en argument.

Remarque : La gestion de processus se fait par l'envoi de signaux ; la liste des signaux utilisables est fournie par exemple par `man 7 signal`.

3.4 Fork : la création de processus

La fonction fork fait partie des appels système standards d'UNIX (man fork : `pid_t fork(void);`). Cette fonction permet à un processus de se dupliquer, par exemple en vue de réaliser un second traitement, parallèlement au premier.

Il existe une filiation dans les processus : le créateur d'un nouveau processus est appelé le père et le nouveau processus, le fils. Tous les attributs systèmes du père (par exemple les droits sur le système de fichier) sont transmis au fils.

Il est souvent avantageux de remplacer les forks, coûteux en ressources système (car un fork implique la création d'un nouveau processus), par des processus légers (thread, qui, eux, partagent la même zone mémoire ou espace d'adressage).

La fonction fork est beaucoup utilisée dans les applications client-serveur avec plusieurs clients simultanés. Voir la section 3.11 « Mécanisme de fonctionnement d'un Shell » pour un exemple.

3.5 Gestion de processus en C

Voici quelques appels systèmes à utiliser pour créer de nouveaux processus.

3.5.1 La primitive fork()

```
#include <unistd.h>
int fork();
```

L'appel à `fork()` duplique le processus. L'exécution continue dans les deux processus après l'appel à `fork()`. La différence (outre le PID et le PPID) est la valeur retournée par `fork()` : dans le processus père (celui qui l'avait appelé), `fork()` retourne le PID du processus fils créé et dans le processus fils, `fork()` retourne 0.

```
#include <stdio.h>    // pour le prototype de printf()
#include <stdlib.h>   // pour le prototype de exit()
#include <unistd.h>   // pour le prototype de fork()
#include <sys/wait.h> // pour le prototype de wait()
```

```
int main(void) {
    int pid;      /* PID du processus fils */
    int statut=0; /* code de retour de terminaison du fils */

    pid = fork();
    switch (pid) {
    case -1:
        printf("Erreur: echec du fork()\n");
        exit(1);
        break;
    case 0:       /* PROCESSUS FILS */
        printf("Processus fils : pid = %d\n" , getpid() );
        exit(3);  /* fin du processus fils avec une valeur diff. de zero pour test */
        break;
    default:      /* PROCESSUS PERE */
        printf("Ici le pere: le fils a un pid=%d\n" , pid );
        wait(&statut); /* attente de la fin du fils et récupération de son statut*/
        printf("Fin du pere le fils a renvoye %d\n", WEXITSTATUS(statut));
    }
    return(0);  // fin normale du processus pere
}
```

3.5.2 Les primitives getpid() et getppid()

L'appel système `getpid()` retourne le PID du processus appelant. `getppid()` retourne le PID du père du processus.

3.5.3 La primitive exit()

`exit()` est une fonction qui ne retourne jamais rien, puisqu'elle termine le processus qui l'appelle.

```
#include <stdlib.h>
void exit(int status);
```

L'argument status est un entier qui permet d'indiquer au shell (ou au père de façon générale) qu'une erreur s'est produite. On le laisse à zéro pour indiquer une fin normale.

3.5.4 La primitive wait()

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait( int *status);
```

L'appel `wait()` permet à un processus d'attendre la fin de l'un de ses fils. Si le processus n'a pas de fils ou si une erreur se produit, `wait()` retourne -1. Sinon, `wait()` bloque jusqu'à la fin de l'un des fils, et elle retourne son PID. L'argument status doit être initialisé à 0.

3.5.5 La primitive sleep()

```
#include <unistd.h>
unsigned int sleep(unsigned int seconds);
```

L'appel `sleep()` est similaire à la commande shell sleep. Le processus qui appelle sleep est bloqué durant le nombre de secondes spécifié, sauf s'il reçoit entre temps un signal.

L'effet de `sleep()` est très différent de celui de `wait()` : wait bloque jusqu'à ce qu'une condition précise soit vérifiée (la mort d'un fils), alors que sleep attend pendant une durée fixée. sleep ne doit jamais être utilisé pour tenter de synchroniser deux processus.

Remarque : En python on utilise le module `signal`.

3.6 Les tubes

Les tubes permettent l'échange bidirectionnel de données entre deux processus s'exécutant sur la même machine.

L'appel système `pipe()` permet la création d'un nouveau tube. Le tube est alors représenté par un couple de descripteurs, l'un pour écrire, l'autre pour lire. Le tube doit être créé avant l'appel à `fork()`.

```
#include <unistd.h>
int pipe(int filedes[2]);
```

L'appel système `popen(commande,mode)` engendre un processus en créant un tube (pipe), exécutant un `fork()`, et en invoquant le shell. La valeur de retour de cet appel est un descripteur de flux permettant des lectures/écritures.

3.7 Les tubes nommés

`mkfifo(chemin,mode)` permet de créer un fichier spécial de type FIFO utilisable comme un tube par tout processus (contrairement à pipe() qui n'est utilisable qu'à condition de faire un `fork()` après).

3.8 Les sockets

Les sockets sont une généralisation des tubes qui pallie leurs faiblesses avec une API qui permet de voir les communications distantes ou non comme de simples écritures dans un fichier. Voir les cours de programmation système.

3.9 Fonctionnement d'un Shell (C-shell ou Bash)

Au login le C-shell (`/bin/csh`) ou le bash (`/bin/bash`) exécute les commandes contenues dans les fichiers suivants (dans cet ordre mais ils n'ont pas toujours les mêmes noms dans tous les Unix !) :

1. `/etc/csh.cshrc`
2. `/etc/csh.login`
3. `/etc/profile`
4. `/etc/bash.bashrc`
5. `~/.cshrc`
6. `~/.login`
7. `~/.profile`
8. `~/.bashrc`

Chaque fois qu'un C-shell est exécuté en dehors du login il exécute au préalable les commandes contenues dans les fichiers `/etc/csh.cshrc` et `~/.cshrc` ou `/etc/bash.bashrc` et `~/.bashrc`. On peut à tout moment (généralement après une modification) faire ré-exécuter ces fichiers par le shell de login au moyen de la commande `source`.

Exemple : `source .login`

Le fichier `.cshrc` contient en général la définition de paramètres de fonctionnement (set), l'initialisation de variables, la définition des alias.

Le fichier `.login` contient éventuellement la définition des variables d'environnement, et la modification de certains paramètres de la connexion (stty), éventuellement l'initialisation de la variable TERM.

Le fichier `/etc/profile` contient — s'il existe — le profil par défaut utilisé par bash au login de l'utilisateur.

La fichier `.bashrc` permet à chaque utilisateur de personnaliser son shell. Ce fichier est cependant réservé aux non-login shells (lancement d'une xterm par exemple). C'est le fichier `.bash_profile` qui est pris en compte lors du login.

Voici un résumé comparatif de différents shells (les différences entre le Korn Shell et le Bash sont suffisamment faibles pour envisager des scripts commun) : (un comparatif plus complet est disponible sur `http://www.cee.odu.edu/uclhd/uclhd_unix_different_shell.php`)

	Bourne Shell	C-shell	K-shell	Bash
Type caractère	X	X	X	X
Type entier		Opérateur @	X	X
Type tableau		X	X	X
Variables locales	X		X	X
Fonctions	X		X	X
Traitement des chaînes	expr, cut	expr, cut	X	X
Arithmétique entière	Expr	X	X	X
Alias de commandes	X	X	X	X
Traçage à l'exécution	X	X	X	X
Contrôle des jobs		X	X	X
Historique des commandes		X	X	X
Complétion de l'historique				X

3.10 Exécution de commande

Pour l'exécution d'une commande, Unix utilise un ou plusieurs processus. Entres autres, un processus comporte une zone de données : son environnement (qui contient le contexte : fichiers ouverts, répertoire courant,...) et une zone de code binaire exécutable.

Les processus sont gérés par 4 opérations primitives (fonctions de la bibliothèque standard C) offertes par le noyau : fork, exec, exit, wait (voir section 3.5).

3.11 Mécanisme de fonctionnement d'un Shell

On peut découper le comportement de tout shell en quelques phases :

- Phase 1 : Lecture des lignes de commande et décomposition en mots des commandes (séparateurs : espace, tabulation, fin de ligne, point-virgule)
- Phase 2 : Vérification syntaxique du format de la commande et mise en place des redirections.
- Phase 3 : Application des mécanismes de substitution de variables, commandes et génération des noms de fichiers.
- Phase 4 : Mise en place des structures de processus nécessaires à l'exécution de la commande.

Remarque : les redirections sont effectuées avant de savoir si la commande peut être exécutée.

Exemple : `commande > fichier` qui entraîne par exemple « command not found ».

Aucun processus n'a été lancé mais le fichier « fichier » est créé s'il n'existait pas déjà.

L'interpréteur de commande (le shell) est un programme comme un autre : il est lancé à l'ouverture de la session (login). Les entrées/sorties standard du processus sont affectés au terminal de l'utilisateur.

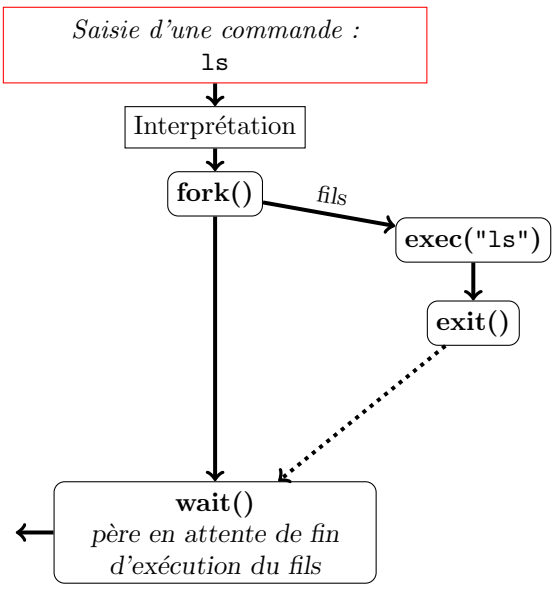


FIGURE 1 – Exécution immédiate

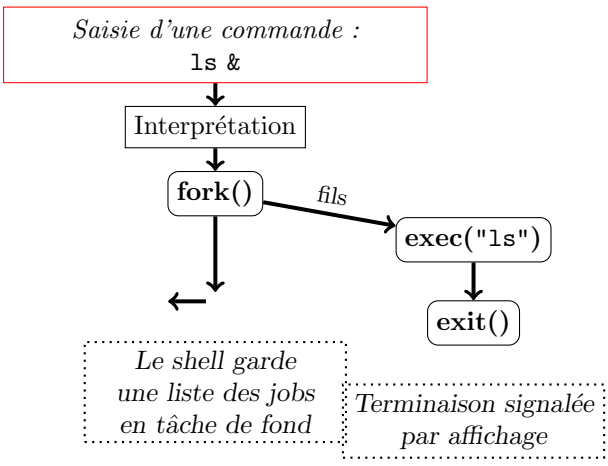


FIGURE 2 – Exécution en arrière plan

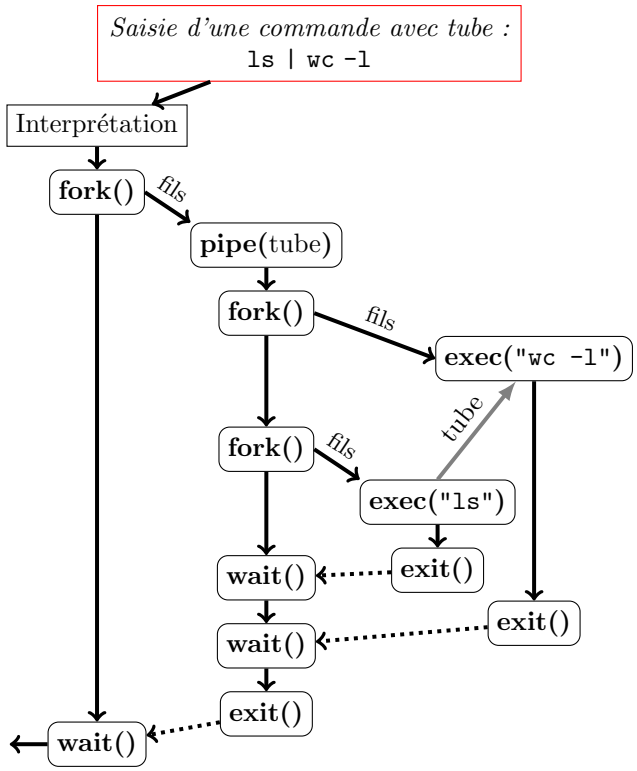


FIGURE 3 – Commandes avec tube

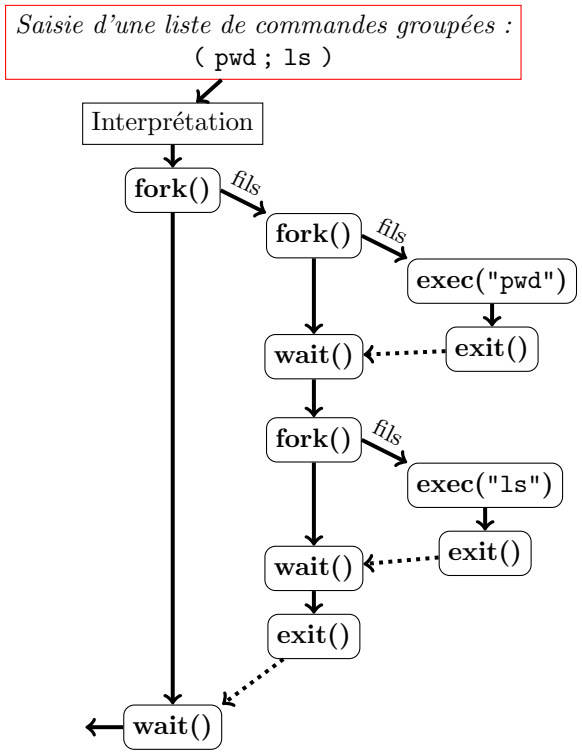


FIGURE 4 – Commandes en série groupées

Lorsqu'une commande est saisie au clavier (ou par fichier) le shell exécute un fork et crée un fils responsable de l'exécution de cette commande. Le fils connaît alors les mêmes fichiers et répertoires que son père. Les arguments sont passés en paramètres. Il exécute les redirections au besoin - ces redirections n'affectent que ce processus et pas son père. Il recherche le binaire correspondant à la commande (sauf si c'est une commande « built-in », c'est-à-dire une commande interne du shell lui-même donc déjà résidente dans le processus) et lance l'exécution par un exec ou par un fork s'il s'agit d'une commande en tâche de fond (voir figures 1 à 4).

3.12 Bash

1. Quelques variables prédéfinies :
   Les variables suivantes sont prédéfinies :
   - `$nombre` tableau des chaînes de caractères données en paramètre
   - `$*` tous les paramètres en une seule chaîne
   - `$$` numéro du processus exécutant la commande actuelle (ou le fichier de commandes)
   - `${#nom-de-variable}` nombre de caractères dans une variable
   - `$?` code de retour après exécution d'une procédure

2. Variables d'environnement (en majuscules par l'usage) :
   - `$HOME` Répertoire par défaut de l'utilisateur. Sa valeur est fixée au login par la lecture du fichier `/etc/passwd`.
   - `$PATH` Répertoires dans lesquels le shell peut va chercher les commandes. La liste des répertoires doit être exhaustive, un programme se trouvant dans un répertoire n'appartenant pas à cette liste n'est pas directement exécutable, sauf au moyen du chemin absolu.
   - `$TERM` Modèle du terminal utilisateur. A la connexion, le shell tente une reconnaissance du type de terminal utilisé. Il est parfois nécessaire de positionner cette variable soi-même.

3. Historique des commandes :
   La commande history donne la liste des dernières commandes tapées.
   - `!!` la dernière commande.
   - `!<nn>` la commande n° `<nn>` (`<nn>` est un nombre).
   - `!<ch>` la plus récente commande commençant par `<ch>` (`<ch>` est une chaîne de caractères).
   - `:n` le n^{e} argument de la référence.

- `:$` le dernier argument (`!!:$` s'abrège `!$`)
- `:a-b` tous les arguments du n° `a` au n° `b`
- `:s/aaa/bbb[/]` substitue la chaîne `bbb` à la première occurrence de `aaa`. `^aaa^bbb` est la forme abrégée de `!!:s/aaa/bbb`
- `:gs/aaa/bbb[/]` substitue `bbb` à toutes les occurrences de `aaa`.
- `:&` répète la substitution précédente (la plus récente substitution effectuée par le mécanisme d'historique).
- `:r` supprime le suffixe (c'est à dire la partie terminale du nom de fichier, de la forme `.xxx`)
- `:p` écrit la commande sans l'exécuter.

Exemple (on suppose que la 30e commande a été `cp toto tata/truc`) :

`!30:gs/t/l` donne la commande : `cp lolo lala/lruc`

4. Substitutions :

   N'importe quelle commande shell peut être placée entre « back-quotes »(anti-apostrophes). Cela a pour effet de déclencher l'interprétation de cette commande avant de réinterpréter toute la ligne de commandes.

   Exemple : `mavar=‘who | wc -l‘`

   A contrario, les apostrophes empêchent les substitutions.

   La commande alias permet de remplacer une chaîne par une autre. Cet outil est souvent utilisé pour abréger des commandes dont on se sert souvent.

   Exemple : `alias h history`

   `unalias h`

   L'ordre de substitution est le suivant :
   - Substitution à partir de l'historique des commandes
   - Substitution des alias
   - Substitution des variables par leur contenu (ou par un code numérique)
   - Exécution des parties comprises entre back-quotes
   - Substitution de fichiers (utilisation des jokers ou « wilcards »)

5. Portée des identificateurs :

   Une variable de l'environnement d'un processus est propagée à ses fils. Une variable normale d'un processus n'est pas exportée.

```
$ var='une variable'
$ echo $var
une variable
$ echo 'bonjour, '$var
bonjour, une variable
$ age=2
$ echo toto a ${age}2 ans
$ echo a$b
a
$ echo "toto a ${age}2 ans"
toto a 22 ans
$ echo 'toto a ${age}2 ans'
toto a ${age}2 ans
$ echo ‘toto a ${age}2 ans‘
bash: toto: command not found
$ Fichier=$(ls /usr/include/*.h |grep std)
$ echo $Fichier
/usr/include/stdint.h /usr/include/stdio_ext.h
/usr/include/stdio.h /usr/include/stdlib.h
/usr/include/unistd.h
```

On peut utiliser les ", les ' ou les ‘ pour modifier l'évaluation des variables :
- avec les ", on ne supprime pas le remplacement des variables
- les ' suppriment toute évaluation
- les ‘ ou les () évaluent la commande

Découpage de chaînes : la notation `##` permet d'éliminer la plus longue chaîne (à partir du caractère le plus à gauche) en correspondance avec le motif ; La notation `#` élimine la plus courte. La notation `%` élimine la plus courte correspondance avec le motif mais en partant de la droite. De même `%%` élimine la plus longue à partir de la droite. Ainsi, on peut éliminer la fin d'une chaîne :

```
$ F='rep/toto.tata.tex'
$ echo ${F##*.}
tex
$ echo ${F#*.}
tata.tex
$ echo ${F%.*}
rep/toto.tata
$ A='toto.tar.gz'
$ echo ${A%.*}
toto.tar
$ echo ${A%%.*}
toto
```

4 Shell et expressions régulières

4.1 Paramètres d'une procédure de commande

- `$n` représente le n^{e} paramètre. Ils sont donc limités à 9 (n=1 à 9).
- `$0` est le nom du fichier de commandes.
- `shift` permet de décaler les paramètres vers la gauche : le premier est perdu, le 2^{e} devient le premier, le 3^{e} devient le 2^{e} etc.. Décaler une liste vide provoque une erreur.

4.2 Conditions logiques de succession de commandes

Basée sur les codes de retour des commandes (si tout se passe bien : 0 sinon différent de 0) on peut enchaîner des commandes suivant leurs résultats :

- `commande1 && commande2`
  La deuxième commande n'est activée que si la première s'est correctement terminée. Le code de retour est alors celui de la deuxième commande.
- `commande1 || commande2`
  La deuxième commande n'est activée que si la première se termine anormalement

4.3 Structures de contrôle en BASH

Exemples d'usages de "if" :

```
if rm $1 2>/dev/null
then echo $1 a ete supprime
else echo "$1 n'a pas ete supprime"
fi
```

D'habitude on utilise les crochets pour tester une condition logique, voici comment tester si un fichier peut être lu *ou* écrit.

```
if [[ ! ( -w /etc/hosts.deny || -r /etc/hosts.deny) ]]
  then
     echo OUI
  else
     echo NON
fi
```

La syntaxe avec un simple crochet est possible par souci de compatibilité mais est à éviter.

4.4 Structures de boucle en BASH

La boucle for sur une liste d'éléments (d'ou le "in") :

```
for i in 1 2 3 ; do
  echo "i =" $i
done
```

La boucle for peut se rapprocher de la syntaxe du C (ici exemple de décrémentation) :

```
for ((i = 10; i >= 0; i--)) do
  echo $i
done
```

Exemple d'utilisation de la boucle while et des variables considérées numériquement :

```
i=0
while [[ $i -le 10 ]] ; do # peut s'ecrire ((i <= 10))
  echo $i
  let i=1+$i # peut aussi s'ecrire (( i=1+$i )) ou (( i++ ))
done
```

4.5 Syntaxe des scripts shell Bash

Un script shell Bash est un fichier exécutable commençant par `#!/bin/bash` (`#!` s'appelle un shebang : un marqueur indiquant quel exécutable doit être utilisé pour interpréter ce script).

Les droits d'exécution ne sont pas nécessaires si on exécute en faisant `source nomfichier`. Voici un exemple de script :

```
#!/bin/bash
for i in ‘seq 1 5‘; do
   if [ $i = 3 ];  then echo;
   else echo $i;
   fi;
done
```

4.6 Expressions régulières

Une expression régulière (pattern, motif) est un mécanisme permettant de rechercher, modifier ou supprimer un motif particulier dans un fichier de type texte (ASCII). L'expression régulière est finalement la généralisation de la notion de chaîne de caractères.

1. Expressions régulières utilisées pour les listes de fichiers :

   Attention, elles sont un peu différentes (« * » au lieu de « .* » par exemple) des expressions des paragraphes suivants.
   - `*` remplace n'importe quelle chaîne
   - `?` remplace un caractère et un seul, quelconque
   - `[ ]` remplace l'un des caractères entre crochets. `[^ ]` remplace tout caractère qui **n'est pas** entre `^` et `]`. Le crochet fermant `]` ne termine pas l'ensemble lorsqu'il apparaît tout de suite après le crochet ouvrant. Le tiret `-` permet de définir un intervalle, ainsi, l'expression `[A-Za-z0-9]` indique un caractère qui peut être une lettre minuscule ou majuscule ou un chiffre. L'expression `[^A-Z]` représente n'importe quel caractère sauf une lettre majuscule. L'expression `[]. *\-]` représente un des caractères `]`, `.` , `\`, `-` ou `*` (le `-` doit apparaître en fin ou en début de liste pour éviter qu'il ne soit utilisé pour un intervalle de caractères). L'expression `[!-@]` est équivalente à l'ensemble des caractères ASCII dont le code est inclus dans l'intervalle spécifié.
   - `{param1,param2,...}` remplace l'une des chaînes param1 ou param2...

   Exemple : `*ce?[0-9]{.tmp,.bak}` désigne tous les fichiers comportant dans leur terminaison la chaîne `ce` suivie d'un caractère quelconque puis un chiffre de 0 à 9 puis un suffixe `.tmp` ou `.bak`

2. Expressions régulières de base :

   Elles sont utilisées par la commande `grep` par exemple. Un caractère normal est associé de façon unique à tout caractère ayant la même codification. Les caractères spéciaux sont les suivants :
   - `\` lorsqu'il précède un caractère normal ou spécial, il lui fait perdre sa signification (sauf les chiffres de 1 à 9 et les parenthèses ouvrantes et fermantes ainsi que les accolades dans certaines constructions complexes).
   - `^` indique le début d'une ligne à condition qu'il soit le premier caractère, sinon, s'il est derrière un crochet ouvrant, il signifie la négation – voir plus loin. Exemple : `^Les` représente une ligne commençant par `Les`.
   - `$` indique la fin de la ligne à condition qu'il soit le dernier caractère. `^$` représente la ligne vide. `\.$` désigne une ligne se terminant par `.`
   - `.` indique n'importe quel caractère sauf le changement de ligne (newline)
   - `*` indique la répétition de 0 à n fois de l'expression régulière composée d'un seul caractère qui précède. `.*` désigne une chaîne quelconque.
   - `[]` Les crochets permettent de représenter un ensemble de caractères comme dans les expression régulières précédentes.

3. Classes de caractères :

   La notation `[:nom:]` représente une classe de caractères définie par une « locale »(dépendante de la langue vivante d'exploitation du système). En standard les classes suivantes sont toujours définies :
   - `[:alpha:]` lettres
   - `[:upper:]` lettres majuscules
   - `[:lower:]` lettres minuscules
   - `[:digit:]` chiffres décimaux
   - `[:xdigit:]` chiffres hexadécimaux
   - `[:alnum:]` caractères alphanumériques

- `[:space:]` caractères produisant un affichage de blancs
- `[:print:]` codes pour imprimante
- `[:punct:]` caractères de ponctuation
- `[:graph:]` caractères affichables
- `[:cntrl:]` caractères de contrôle (codes ASCII en général de 1 à 26).
- `[:blank:]` caractères blancs (espace ou tabulation)

4. Composition d'expressions régulières :
   - Concaténation : il suffit de mettre les expressions régulières les unes derrières les autres (attention pas d'espace, sinon le caractère espace serait pris en compte comme étant lui-même une expression régulière).
   - Reproduction du motif : mettre une étoile après l'expression régulière permet de répéter le motif 0 ou n fois.
   - `\(<expression régulière>\)` : a priori le comportement est identique à la même expression régulière sans les `\( \)` mais mémorise le résultat de la substitution.
   - `\<chiffre n>` : rappelle le résultat de la nième substitution qui a été mémorisée par `\ (` Par exemple, `^\(.*\ ) \1$` est équivalent à toutes les lignes composées de deux chaînes de caractères identiques juxtaposées. Si `\n` est suivi d'une étoile, la reproduction de motif est appliquée.) De plus, les `\( \)` peuvent être encapsulés les uns dans les autres. Par exemple, `\(ab\(cd\)ef\)Z\2*Z\1` permettra de filtrer la chaîne : `abcdefZcdcdZabcdef`.
   - `<expression régulière>\{m\}` : permet de répéter exactement m fois le motif précédent.
   - `<expression régulière>\{m,\}` : permet de répéter au moins m fois le motif précédent.
   - `<expression régulière>\{m,n\}` : permet de répéter au moins m fois et au plus n fois le motif précédent.
   - Règles de recherche de motif : la première chaîne trouvée qui correspond au motif et s'il y a des répétitions possibles alors on recherche à faire le maximum de répétitions possible.
5. Expressions régulières étendues :

   Certaines commandes peuvent interpréter des expressions régulières étendues (`egrep` par exemple). Les règles précédentes sont conservées et on ajoute des fonctions supplémentaires :
   - `<expression régulière>+` : correspond à au moins 1 fois le motif précédent
   - `<expression régulière>?` : correspond à 0 ou 1 fois le motif précédent
   - `<expression régulière>|<expression régulière>` : l'une ou l'autre.

5 Gestion de fichiers texte

5.1 Début et fin d'un fichier

- `head -n fichier` : affiche (sortie standard) les n premières lignes d'un fichier
- `tail +n fichier` : affiche les n dernières lignes du fichier en comptant à partir du début
- `tail -n fichier` : affiche les n dernières lignes du fichier en comptant à partir de la fin
- `tail -f fichier` : boucle indéfiniment, en essayant de lire de plus en plus de caractères à la fin du fichier, celui-ci devant grandir.

5.2 Tri d'un fichier

```
sort [OPTIONS]... [FILES]...
```

Exemples :

```
%cat TEL1
Toto Jean 01.10.20.30.40
Tata Paul 03.10.15.30.25
Toto Anne 02.20.01.17.12

# tri à partir du premier champ, ordre dico.
# idem sort -k 1d TEL1
%sort TEL1
Tata Paul 03.10.15.30.25
Toto Anne 02.20.01.17.12
Toto Jean 01.10.20.30.40

# Tri ordre alphab. sur 1er champ
```

```
# ordre inverse(=r) 2e champ
%sort -k 1,1d -k 2,2dr TEL1
Tata Paul 03.10.15.30.25
Toto Jean 01.10.20.30.40
Toto Anne 02.20.01.17.12

# tri numérique(=n) sur le 3e champ
%sort -k 3n TEL1
Toto Jean 01.10.20.30.40
Toto Anne 02.20.01.17.12
Tata Paul 03.10.15.30.25

# tri d'un fichier dont le séparateur de champ
```

```
# est ":", le critère du tri est l'ordre
# dictionnaire, appliqué au 1er champ.
# 2e clé: le 2e et 3e champ ordre numérique.
%sort -t: -k 1,1d -k 2,3n /etc/passwd
```

5.3 Jointure de fichiers

L'opérateur `join` des bases de données relationnelles peut être appliqué directement sur deux fichiers triés et possédant une clé commune.

Exemple :

```
%cat F1
tata M. Tata B.
toto M. Toto Gerard
truc Mme Truc S.

%cat F2
tata 03.10.15.30.25
toto 01.10.20.30.40
truc 02.20.01.17.12

%join F1 F2
tata M. Tata B. 03.10.15.30.25
toto M. Toto Gerard 01.10.20.30.40
truc Mme Truc S. 02.20.01.17.12
```

5.4 Recherche de doublons

La commande `comm` permet d'afficher les lignes répétées d'un fichier trié. La commande `uniq` est un filtre qui agit sur les lignes d'un fichier trié pour éliminer les doublons (voir option `-u` de `sort`).

5.5 Sélection de champs ou de colonnes : le couper-coller

Voir la commande `cut`. Options utiles : `-d` séparateur pour spécifier que le séparateur de champs est un autre caractère que l'espace par défaut et l'option `-f <liste>` qui spécifie quels sont les champs à prendre.

La commande `paste` permet de fusionner des fichiers (en colonnes) avec un caractère de délimitation précisé.

```
Exemple :
%cut -d \. -f 2 FIC1 > resu1
%cut -d \. -f 3 FIC1 > resu2
%paste -d : resu1 resu2
10:15
10:20
20:01
```

5.6 Substitution des tabulations par des espaces et inversement

`expand` : tabulation -> 8 espaces (ou -n donne n espaces)
`unexpand` : espaces -> tabulation

5.7 Quelques autres commandes de manipulation des fichiers textes

`split` : permet de découper un fichier en fichiers plus petits, que l'on pourra réassembler plus tard par la commande cat.

Exemple : `%split -300 essai essai.morceau.`

le fichier essai va être découpé en fichiers d'au plus 300 lignes dont les noms seront respectivement essai.morceau.aa essai.morceau.ab etc.

`tr` : filtre de transcodage des caractères donnés en entrée standard vers la sortie standard.

Exemples :

```
%echo BonJour | tr a-z A-Z
BONJOUR
%tr "\012" "\015" < alpha > beta
```

La première chaîne désigne la table de transcodage en entrée, la deuxième la table en sortie. Le premier exemple transforme donc les minuscules en majuscules, le second transforme les caractères `<LF>` du fichier alpha par des caractères `<CR>`, le résultat est stocké dans le fichier beta.

5.8 La commande sed

C'est un filtre syntaxique : `sed <commandes> [fichier]`
Voici un extrait de man sed :

```
sed [-n] [-e commande] [-f fichier de commandes] [fichier]
   -n  écrit seulement les lignes spécifiées (par l'option /p) sur la sortie standard
   -e  permet de spécifier les commandes à appliquer sur le fichier. Cette option est
               utile lorsque vous appliquez plusieurs commandes. Afin d'éviter que le shell
               interprète certains caractères, il faut mieux encadrer la commande avec des
               ' ou des " .
   -f  les commandes sont lues à partir d'un fichier.
```

Les commandes peuvent être très complexes et très longues : on a alors la possibilité d'utiliser un fichier :
`%sed -f fichier.sed < truc > toto`
Le nom de la commande vient du fait qu'elle basée sur une version non-interactive de l'éditeur ed. Cet éditeur est un pionnier des éditeurs de textes : il ne peut éditer qu'une ligne à la fois. Les commandes de sed sont reprises de ed et sont proches de celles que peut utiliser vi : elles se basent sur une utilisation des expressions régulières.
Exemples (la commande **s**/expression/expression/options substitue un motif par un autre) :

```
%sed "s/f(\([.0-9]*\),\([.0-9]*\))/fonc(\2,\1)/"  << #fin
f(10.134,1)
f(1456,9)
f(17,23.206)
f(10,20,30)
#fin
fonc(1,10.134)
fonc(9,1456)
fonc(23.206,17)
f(10,20,30) # ligne non modifiée car ne correspond pas au motif recherché

%sed "s/f(\([^,]*\),\([^)]*\))/fonc(\2,\1)/"  << #fin
# produit le même résultat pour les 3 premières lignes, sauf la dernière:
fonc(20,30,10)
```

5.9 Extractions formatées : awk

`awk` est en fait un véritable langage de programmation interprété, inspiré du `C`. `awk` permet de traiter un fichier ligne par ligne. Chaque ligne est analysée par le programme `awk`. La syntaxe de `awk` est basée sur des lignes de texte de la forme : `BEGIN{action} motif{action} END{action}` où tous les éléments sont optionnels.
Extrait de man awk :

```
awk [-Fs] [-v variable] [-f fichier de commandes] 'program' fichier
   -F  Spécifie les séparateurs de champs
   -v  Définie une variable utilisée à l'intérieur du programme.
   -f  Les commandes sont lues à partir d'un fichier.
```

S'il n'y a pas d'action prévue pour la ligne en cours, elle est reproduite intégralement en sortie standard. L'absence de l'expression régulière (motif) dans la syntaxe précédente signifie que l'on exécute l'action systématiquement.
Une action est une séquence d'instructions. Les instructions respectent (sauf next et exit) la syntaxe du C.

```
if (condition) action [else action]
while (condition) action
for (expression ; condition ; expression) action
break
continue
variable=expression
printf format [,liste-d'expressions] [> expression]
print [liste-d'expression] [ > expression]
next
exit
```

De plus, l'action `skip` indique que l'on ne traite pas, pour la ligne courante, les actions qui devraient être déclenchées dans les lignes de programme suivantes. `Exit` signifie l'arrêt du traitement.

L'action du `BEGIN` n'est effectuée qu'une seule fois avant de commencer l'action de traitement du fichier.

L'action du `END` n'est effectuée qu'une seule fois après l'action de traitement du fichier.

Variables prédéfinies : `$1 $2 ... $9` représentent les différents champs de la ligne courante, `$0` représente la ligne entière. Notons que pour `$n`, si n=4, équivaut bien à `$4`. Il est possible d'affecter une valeur à un champ même non existant, `$(NF+2) = 5` créera aussi les champs intermédiaires (et provoquera la mise à jour de NF). NF représente le nombre de champs de l'enregistrement courant (Number of Fields) et NR le nombre d'enregistrements en entrée vu jusqu'ici (Number of Records). FS (Field Separator), défini par défaut à espace, est le séparateur de champs en entrée. RS (Record Separator) définit le séparateur entre enregistrements (changement de ligne par défaut). OFS (Output Field Separator) permet de définir le séparateur de champs pour les impression par `print`, de même que ORS (Output Record Separator) sépare les enregistrements en sortie.

Tableaux. Ils sont associatifs dans awk. Cela permet des opération de test ou de boucle avec l'operateur spécial `in`.

Exemples :

```
%cat FIC1
BULL M. Tata Bernard 03.10.15.30.25
SUN M. Toto Gerard 01.10.20.30.40
HP M. Truc Sylvestre 02.20.01.17.12

%awk \
  '{printf "%s\t%s\tMonsieur %s\n" ,$1,$4,$3}'\
  FIC1
BULL 03.10.15.30.25 Monsieur Tata
SUN 01.10.20.30.40 Monsieur Toto
HP 02.20.01.17.12 Monsieur Truc

# si la liste des commandes est longue,
# on peut les mettre dans un fichier...
%cat factorielle.awk

function factorielle(n) {
  if (n <= 1) return 1
  else
    return n * factorielle(n - 1)
}
{print $1, factorielle($1)}

%awk -f factorielle.awk << #fin
5
6
#fin
5 120
6 720
%awk \
  '{t["moi"]="a"; t["lui"]=13; OFS=":";
   for (val in t) print val,t[val];}' FIC1
moi:a
lui:13

# le nombre de lignes du fichier toto
%awk 'END {print NR}' toto
```

6 Sécurité

L'accès aux fichiers est conditionné par le numéro d'identité de l'utilisateur (« User Identity » ou UID) et son numéro de groupe (« Group Identity » ou GID). Chaque utilisateur possède un UID et un GID. Par défaut, il partage son GID avec éventuellement d'autres utilisateurs qui font alors partie du même groupe. L'UID et le GID avaient des valeurs comprises entre 0 et 65535 (cette limite a été largement augmentée de nos jours). L'UID 0 (généralement appelé root) est réservé et donne les privilèges suivants :

- Accès à la totalité des fichiers du système quelles qu'en soient les permissions
- Droit d'exécuter la totalité des appels système

Nous avons déjà évoqué précédemment, les attributs d'un fichiers. On peut donc très simplement assurer une protection totale entre utilisateurs, ou entre groupes d'utilisateurs grâce à ce mécanisme.

Par défaut, lorsqu'un utilisateur créé un fichier, le système consulte la variable `umask` restreignant les droits par défauts que veut donner l'utilisateur. Les droits effectivement donnés sont umask & droits demandés (le & représente le « et binaire »). Avec `umask` on utilise une valeur octale. Par exemple, pour avoir les droits équivalents à l'octal 750, on donnera un masque égal à 027, voici pourquoi :

Exemple :

`rwx r-x ---` droits désirés pour la création d'un fichier/répertoire standard

`111 101 000` Protection codée en binaire (=750 en octal)

`000 010 111` Complément à 1 donne le masque binaire (=027 en octal)

donc un umask de 027 (positionné par la commande `umask 027`) permettra de créer des fichiers et des répertoires avec des droits limités à ceux indiqués en première ligne (sauf le droit d'exécution pour les fichiers qui n'est jamais automatique).

Remarques : dans beaucoup de systèmes la valeur par défaut est 022 : cette valeur peut être trop permissive sur un serveur en production.

6.1 /etc/passwd

Permissions conseillées : uniquement en lecture pour tout le monde et écriture pour root qui est sont propriétaire).

Ce fichier décrit l'ensemble de tous les utilisateurs du système. Il se modifie au moyen d'un simple éditeur de texte. Sur System V, on conseille l'utilisation de `vipw` (`vi` spécial pour passwd) ou géré avec des commandes telles : `adduser`, `deluser`,... Toujours sous System V, la commande `pwck` permet de contrôler la cohérence du fichier.

Il contient exactement une ligne par utilisateur. Cette ligne comporte 7 champs séparés par le caractère : (deux-points). La syntaxe est la suivante :

`logname:mot_de_passe:uid:gid:commentaires_libres:rep_par_defaut:shell`

- `logname : nom de login de l'utilisateur`
- `mot_de_passe` : mot de passe chiffré. En fait, on initialise ce champ avec le caractère * (ou *LK*) (ce qui bloque l'entrée) et on change cette valeur ultérieurement avec la commande passwd. Si dans ce champ on trouve « x » c'est que dans ce système le mot de passe crypté est probablement stocké dans `/etc/shadow` (voir 6.3).
- `uid` : numéro d'utilisateur associé au nom de login. Les valeurs d'UID inférieures à 500 sont généralement par convention réservées au système ou à des logins spéciaux. L'uid 0 confère les privilèges d'administration (super-utilisateur, appelé root).
- `gid` : numéro de groupe dont fait partie l'utilisateur. De même, les valeurs inférieures à 500 sont généralement réservées pour le système.
- `commentaires_libres` dit champ `GECOS` pour *General Electric Comprehensive Operating System* souvent utilisé pour conserver le vrai nom de l'utilisateur, son numéro de poste, etc. Voir les commandes `chfn`, `finger`, etc.
- `rep_par_defaut` : répertoire initial sous lequel l'utilisateur sera logué. La variable HOME de l'utilisateur est initialisée à cette valeur.
- `shell` : est le nom de l'interpréteur de commandes. On choisit généralement `/bin/bash` ou `/bin/tcsh` suivant que l'on utilise l'un ou l'autre. On peut toutefois indiquer dans ce champ le chemin d'accès de n'importe quelle commande. Si le champ est vide, alors le shell sera `/bin/sh`. Ce champ peut être modifié par l'utilisateur par la commande `chsh`. Ce champ positionné à `/bin/false` peut être utilisé pour préciser que l'utilisateur décrit n'est qu'un pseudo-utilisateur (login spécial) auquel ne fournit pas la possibilité de se loguer.

6.2 /etc/group

Permissions conseillées : `-rw-r--r--`

De même que pour le fichier précédent, ce fichier peut être modifié par un éditeur de texte quelconque ou géré à l'aide de commandes telles : `addgroup`, `delgroup`, etc. Sous System V, sa cohérence est vérifiée par la commande `grpck`. Le fichier `/etc/group` comprend une ligne par groupe. Cette ligne comporte 4 champs séparés par des caractères : (deux-points).

`groupname:passwd:gid:logname1,logname2,logname3,...`

- `groupname` : est le nom du groupe (en général max. 8 caractères)
- `passwd` : le mot de passe du groupe. Ce champ existe mais n'est pas utilisé pour les versions Unix BSD. S'il est positionné à « x », il faut consulter le fichier `/etc/gshadow` (voir section 6.4).
- `gid` : numéro d'identification du groupe
- `logname1,..` : liste des logins des utilisateurs de ce groupe (parfois aucun).

6.3 /etc/shadow

Permissions maximales conseillées : `-rw-r-----`

Ce fichier ne doit pas être accessible en lecture par les utilisateurs normaux afin de maintenir la sécurité des mots de passe, en particulier contre les attaques par dictionnaires (voir section 6.5). Pour transformer une gestion par fichier de groupe classique `/etc/passwd` en une gestion par `/etc/passwd` et `/etc/shadow` il existe la commande `pwconv` (et inversement par `pwunconv`.

Ce fichier est modifiable par la commande `passwd`. Il contient exactement une ligne par utilisateur. Cette ligne comporte 9 champs séparés par le caractère : (deux-points). La syntaxe est la suivante :

`logname:mot_de_passe_crypté:Nb_70:Nb_chmin:Nb_chmax:Nb_exp:Nb_des:Nb_des70:Champs réservé`

- `logname` : Nom de login
- `mot_de_passe_crypté` : mot de passe crypté. Le mot de passe crypté comprend 13 à 120 caractères pris dans l'alphabet réduit `a-z, A-Z, 0-9, . et /`. Suivant les systèmes, l'algorithme de chiffrement utilisé est DES, MD5, ou autre (si c'est `md5` le mot de passe commence par `$1$`, `sha256` : `$5$`, `sha512` : `$6$`, `blowfish` : `$2a$`, voir `pam_unix`).
- `Nb_70` : date de dernière modification du mot de passe donné par le nombre de jours écoulés depuis le 1er janvier 1970. Si on met la valeur 0 dans ce champ, l'utilisateur est obligé de changer de mot de passe au prochain login. Un champ vide signifie que la gestion de l'âge des mots de passe (*password aging*) est désactivée.
- `Nb_chmin` : Nombre de jours à attendre avant de pouvoir changer de mot de passe.
- `Nb_chmax` : Nombre de jours max. après lesquels le mot de passe doit être changé. Après ce délai, le mot de passe est encore valide jusqu'à ce que l'utilisateur se logue mais il est alors obligé de changer de mot de passe.
- `Nb_exp` : Nombre de jours avant l'expiration (`Nb_chmax`) du mot de passe impliquant l'avertissement de l'utilisateur. 0 ou un champ vide signifie que l'on avertit pas l'utilisateur.
- `Nb_des` : Nombre de jours après l'expiration provoquant la désactivation du compte. Après ce délai plus `Nb_chmax`, le compte est verrouillé.
- `Nb_des70` : date de pose du verrou sur le compte, exprimée en nombre de jours depuis le 1er janvier 1970.
- `Champ réservé` : destiné à un usage futur.

Notons que ces paramètres sont modifiables par la commande `chage`. Un compte avec `Nb_chmax` à 99999 et les champs suivants vides est « éternel ». Les valeurs par défaut utilisées par `useradd` sont prises dans `/etc/login.defs`

6.4 /etc/gshadow

Permissions maximales conseillées : `-rw-r-----`

Ce fichier ne doit pas être accessible en lecture par les utilisateurs normaux afin de maintenir la sécurité des mots de passe, en particulier contre les attaques par dictionnaires (voir 6.5). Pour transformer une gestion par fichier de groupe classique `/etc/group` en une gestion par `/etc/group` et `/etc/gshadow` il existe la commande `grpconv` (et inversement par `grpunconv`.

Ce fichier est modifiable par les commandes `groupadd`, `groupdel`, `groupmod` et surtout `gpasswd`. Il peut être vérifié par `grpck`. La syntaxe est la suivante :

`nomGroupe:mot_de_passe_crypté:liste_des_admins_du_groupe:liste_des_membres_du_groupe`

- `nomGroupe` : Nom du groupe.
- `mot_de_passe_crypté` : mot de passe crypté. S'il est vide, seuls les membres du groupe peuvent utiliser la commande newgrp. (dans une précédente version, le mot de passe est demandé et un simple retour chariot suffit). S'il est égal à juste un caractère (tel « ! », le groupe est verrouillé). Si un mot de passe est défini alors seuls les membres du groupe peuvent utiliser la commande `newgrp` sans entrer de mot de passe, les non-membres doivent le fournir (il semble que dans certaines versions les non-membres ne sont même pas invités à le fournir !).
- `liste_des_admins_du_groupe` : peut être vide. La commande `gpasswd -A` permet à l'administrateur de définir les administrateurs de ce groupe et avec `gpasswd -M` les membres. Un administrateur de groupe peut ajouter ou supprimer des utilisateurs en utilisant respectivement les options -a et -d de `gpasswd`. Note : un administrateur de groupe n'est pas forcément membre de ce groupe !
- `liste_des_membres_du_groupe` : Liste des membres.

6.5 Chiffrement du mot de passe

Le chiffrement d'un mot de passe consomme environ une 1 miliseconde de temps CPU sur une machine de 1000 mips (1 mips = 1 million d'instructions par seconde). Pour chiffrer toutes les combinaisons possibles (soit $2^{56+12} \approx 7,2^{16}$ pour DES [1]) cela demanderait un temps de calcul de l'ordre de plus de 9000 ans.

Par contre le chiffrement d'un dictionnaire de 36 000 mots prend 36 secondes, le résultat du calcul peut être alors confronté aux mots de passe du système... C'est ce que l'on appelle l'attaque par dictionnaire.

La probabilité de la découverte d'un ou plusieurs mots de passe dépends donc surtout du choix des mots de passe.

La solution pour pallier les attaques par dictionnaire consiste à modifier la commande passwd et à rendre obligatoire l'introduction d'un chiffre ou d'un caractère spécial dans le mot de passe. Cette modification existe en standard sous System V release 2.2. Mais cela reste insuffisant par rapport à la puissance des ordinateurs actuels.

Une autre possibilité est de déporter le mot de passe dans un autre fichier qui n'est pas en lecture publique (« shadow passwords » stockés dans `/etc/shadow` et les mots de passe de groupe dans `/etc/gshadow`).

1. 56 bits de clé formé à partir des 7 bits de poids faible des 8 premiers caractères du mot de passe + un «sel» aléatoire sur 12 bits

L'algorithme DES a de nombreuses faiblesses cryptographiques qui font que désormais son utilisation est fortement déconseillée même pour le chiffrement de mots de passe. C'est pourquoi on lui préfère désormais l'algorithme MD5 ou autre (voir 6.3 pour une liste des algorithmes les plus utilisés et leurs codes).

6.6 Choix du mot de passe

L'administrateur système ne doit laisser aucun login (lançant l'exécution d'un shell) sans mot de passe. En général, Unix ne prend en compte que les 8 premiers caractères (tant que l'on utilisait que l'algorithme DES) d'un mot de passe. La commande passwd réclame souvent des mots de passe qui comportent au moins 5 ou 6 caractères.

On choisira donc des mots de passe qui comportent des majuscules, des minuscules et des caractères spéciaux. On évitera absolument les mots qui font partie du vocabulaire, des noms et prénoms usuels.

Les bons mots de passe comportent divers caractères ASCII, mais sont aussi simples à mémoriser. Par exemple, des mots de passe phonétiques comme 13Zor, E20c, ... ou dérivés d'une phrase comme 29aDL (29 avenue Division Leclerc). Enfin, on adoptera une fréquence raisonnable de changement des mots de passe (pas trop souvent car on risque alors d'oublier le mot de passe !).

6.7 Mécanisme d'expiration du mot de passe

Ce mécanisme, s'il est mis en oeuvre, impose le renouvellement périodique du mot de passe. Il n'est pas mis oeuvre sur tous les systèmes Unix. Il est proposé sous SunOS 4.1 (et suivants) et la plupart des systèmes dérivés de System V (et tous ceux utilisant les shadow passwords).

Il est possible d'interdire deux changements successifs afin d'éviter que l'utilisateur ne puisse réutiliser le mot de passe « usé ». Seul l'administrateur système peut définir ces paramètres.

SunOs utilise les options -n, -x et -d de la commande passwd pour gérer l'âge du mot de passe, d'autres, tel Linux, utilisent la commande `chage`. Lorsque le mécanisme d'expiration est mis en service, il est indispensable que l'utilisateur soit averti, lors de chaque login, de la durée de validité résiduelle de manière à ne pas être surpris le jour où le mot de passe doit être effectivement changé.

6.8 Substitution d'identité (bit `s`)

La permission `x` sur un fichier peut être remplacée par la permission `s` que l'on peut affecter au propriétaire et/ou au groupe du fichier. Cette permission indique que le fichier est exécutable et que pendant son exécution le processus prend les droits du propriétaire et/ou du groupe du fichier. On dit d'un tel fichier exécutable qu'il est SUID et/ou SGID.

En fait, lorsqu'un processus s'exécute, ses droits d'accès sont conditionnés par l'UID effectif (EUID) et par son GID effectif (EGID). Normalement, EUID=UID et EGID=GID de l'utilisateur qui a lancé l'exécution du processus.

Lorsque la permission `s` est positionnée sur le propriétaire du fichier exécutable alors EUID = UID du fichier.

De même, lorsque la permission `s` est positionnée sur le groupe du fichier exécutable, alors EGID=GID du fichier.

Par exemple, la commande passwd permet à tout utilisateur de changer son mot de passe contenu dans `/etc/passwd` (ou pire dans `/etc/shadow`) bien que seul root (UID 0) ait le droit d'écriture dans ce fichier. Les permissions de la commande et du fichier sont donc :

```
-rwsr-xr-x 3 root root 24576 May 3 1989 /bin/passwd
-rw-r--r-- 1 root wheel 2087 Dec 3 12:50 /etc/passwd
```

La notion de groupe est différente entre System V et BSD 4 mais elle fait toujours appel aux informations stockées dans `/etc/group`.

System V : Un processus à un moment donné possède les droits d'accès d'un seul groupe défini par son EGID. Il possède donc les droits d'accès de ce groupe et tous les fichiers qu'il crée sont créés avec un groupe égal à l'EGID du processus créateur. L'utilisateur peut changer de groupe à l'aide de la commande `newgrp`. Il ne peut le faire que s'il est membre du nouveau groupe. Cette commande a pour effet de changer son GID pour celui du groupe choisi.

Remarque : la commande `newgrp` est une commande interne (built-in) du Bourne shell mais pas du C-shell ni de BASH. Cela signifie qu'en C-shell ou en BASH on peut revenir sur le shell précédent avec le GID précédent.

BSD 4 : L'utilisateur peut créer des fichiers qui appartiennent à n'importe quel groupe du système. En effet, lorsque l'on crée un fichier, ce fichier appartient au même groupe que le groupe du répertoire dans lequel il est créé. On possède à un instant donné tous les droits d'accès pour tous les groupes dont on est membre. La commande newgrp n'existe alors pas car elle n'est pas nécessaire.

SunOs 4 et suivants, Linux : Ces Unixes proposent la fusion des deux mécanismes précédents. Par défaut, ils se comportent comme System V. Mais, si on veut obtenir le comportement BSD, il suffit de positionner le droit `s` sur le groupe du **répertoire** concerné. Cette dualité de comportement impose donc l'existence de la commande `newgrp` pour la compatibilité avec System V.

Le droit SUID est codé par la valeur octale 4000, donc l'exécutable `/bin/passwd` a le droit octal 4755. Le droit SGID est codé par la valeur octale 2000.

6.9 Droits d'accès : algo

L'accès aux fichiers par un processus est déterminé par l'algorithme suivant (System V) :

```
Si (euid_processus == uid_fichier) alors
  accès selon les droits du propriétaire
sinon si (egid_processus == gid_fichier) alors
  accès selon les droits du groupe
sinon
  accès selon les droits des autres
finsi
```

Pour BSD 4.[23], l'accès groupe est différent : on vérifie que
`egid_processus == gid_fichier` où `gid_fichier` appartient au `group_access_list` (liste des groupes dont l'utilisateur est membre).

6.10 Sticky bit (bit `t`)

- Sur fichier exécutable :
  Cet attribut spécial, noté `t`, indique que le segment texte (code binaire) d'un fichier exécutable n'est « swapped out » qu'une seule fois et qu'il est conservé dans l'espace disque de swap même une fois la commande terminée. Cela permet un chargement plus rapide lors de la prochaine invocation de la commande. Cet attribut est sans effet sous Linux.
- Sur répertoire (BSD 4.3, Linux) :
  Le bit `t` sur un répertoire dont le mode est `drwxrwxrwt` a pour effet de protéger les fichiers qu'il contient contre la suppression : seul le propriétaire ou root peuvent supprimer un fichier d'un tel répertoire. Cette notion a été introduite pour protéger des répertoires en accès publics tels `/tmp` ou `/var/spool/mail`.

Le sticky bit est codé par la valeur octale 1000, donc `/tmp` a le droit octal 1777.

6.11 sudo

`sudo` permet de configurer très simplement les droits d'exécution de certaines commandes des utilisateurs du système. En clair avec sudo, vous pouvez définir quel utilisateur aura le droit de faire un shutdown, monter un cd-rom,.... Cela peut permettre d'éviter un usage excessif du bit `s`.

6.11.1 Configuration

Le fichier de configuration est `/etc/sudoers`. Un exemple de fichier est le suivant :

```
# sudoers file.
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the sudoers man page for the details on how to write a sudoers file.

# Cmnd alias specification
Cmnd_Alias    HALT=/sbin/halt

# User privilege specification
root    ALL=(ALL) ALL

# Samples
# %users  ALL=/sbin/mount /cdrom,/sbin/umount /cdrom
# %users  localhost=/sbin/shutdown -h now

mayero  ALL=NOPASSWD:/sbin/ifdown,/sbin/ifup,/usr/bin/cdrecord,/usr/bin/cdrecord-dvd,HALT
```

Remarques :
- La syntaxe générale pour une ligne qui n'est pas une définition d'alias est la suivante :
  `qui sur_quelle_machine = (avec_quel_user) quoi1,quoi2,...`
- Pour éditer et le modifier le fichier `/etc/sudoers` il est préférable de lancer, en tant que root, `visudo`. C'est un éditeur de texte avec les commandes de vi, qui permet de détecter les erreurs de syntaxe et de faire prendre en compte automatiquement les modifications à sudo.
- Si l'on ne mets pas `NOPASSWD`, sudo demande le mot de passe de l'utilisateur et le conserve en mémoire (par défaut 5 mn).

6.11.2 Utilisation

`sudo -l` permet de voir la liste des commandes super utilisateur auquel on a droit. Il suffit ensuite de taper la commande précédée de sudo (sans passer root).

`sudo -u user cde` précise avec quel user lancer la commande.

`sudo -k` force l'oubli du mot de passe éventuellement mémorisé.

6.12 chroot

`chroot` redéfinit l'univers de fonctionnement d'un programme. Plus précisément, il détermine un nouveau répertoire « / » pour un programme ou une session. Schématiquement, tout ce qui est à l'extérieur du répertoire « chrooté » n'existe pas pour un programme ou un shell. Si une faille dans un service « chrooté » permet l'obtention des droits de ce service (par exemple à distance) dans votre serveur, l'intrus ne pourra pas voir la totalité de l'arborescence de votre système. Le fait de ne pas voir vos fichiers limite les commandes qu'il peut lancer et ne lui donne pas la possibilité d'exploiter des fichiers qui seraient vulnérables. Attention, la "prison" chroot n'est pas la sécurité absolue, avec les droits de root on peut en sortir...

6.13 Contrôle du login comme administrateur système

Le login sous root peut être limité au moyen du fichier `/etc/securetty` pour BSD 4.2, du fichier `/etc/ttys` pour BSD 4.3 et du fichier `/etc/ttytab` pour SunOs 4.x.

Ce fichier (quel que soit son nom) sert à préciser les droits des terminaux : on peut considérer que tel terminal est d'usage trop risqué et interdire la possibilité d'y faire un login root.

La commande `su` permet, dans une session ouverte, de prendre l'identité d'un autre utilisateur. Elle identifie puis authentifie l'utilisateur de la même manière que login.

Ainsi, si le fichier des droits des terminaux interdit les accès login root pour tous les terminaux, cela impose l'utilisation de la commande `su` pour pouvoir passer root. L'intérêt étant que la commande `su` garde une trace de son utilisation.

7 Les services et démons

Une machine (un hôte) peut héberger un ou plusieurs services réseaux (ou autre). Exécuter un service, c'est lancer un ou plusieurs « démons », processus exécutés en mémoire, en tâche de fond, et qui scrutent en permanence les demandes de connexion qui leur sont adressées, en provenance du réseau.

7.1 Définition de daemon ou démon

Un DAEMON (Disk And Execution Monitor) est un processus qui s'exécute en arrière plan. Ce terme a été créé par les inventeurs d'UNIX et a été repris sur les autres systèmes d'exploitation. Les démons sont souvent démarrés lors du chargement du système d'exploitation, et servent en général à répondre à des requêtes du réseau, à l'activité du matériel ou à d'autres programmes en exécutant certaines tâches. Un daemon n'est attaché à aucun terminal il doit donc écrire en cas de besoin dans un fichier, on parle alors de fichiers de traces ou logs.

Dans le cas des services réseaux, pour distinguer et écouter les demandes émanant de machines clientes, chacun de ces processus est doté d'au moins un numéro de « port ». La liste des correspondances normalisées entre le nom du service, le protocole TCP (ou UDP) qui le contrôle et son numéro de port standard, se trouve dans le fichier `/etc/services`. Conventionnellement, les ports de numéros inférieurs à 1024 (dits privilégiés), sont écoutés par des services démarrés uniquement par root.

Par exemple, les services web (processus httpd), ssh (processus sshd) écoutent habituellement sur les ports 80 et 22 respectivement.

De son coté, un programme client, pour s'adresser convenablement à un serveur doit préciser :

- son adresse ip (ou un nom de domaine qui sera résolu en adresse ip par un serveur DNS accessible)
- le numéro du port sur lequel ce serveur « écoute ». Ensuite s'engage un dialogue d'identification mutuelle, qui aboutit à une connexion de durée variable.

Pour plus de détails, voir le cours de réseaux.

Nous allons présenter maintenant le principe de démarrage d'un système Linux, comment les processus sont lancés et dans quel ordre et comment configurer le système pour, par exemple, lancer des services personnalisés. Il existe plusieurs mécanismes différents suivant les Unix. Historiquement le plus ancien est *init System V*. De nos jours, le modèle *systemd* commence à s'imposer (voir section 7.4).

7.2 Niveaux et scripts de démarrage (init System V)

Sous LINUX (et UNIX d'une manière générale) on peut définir un niveau de démarrage qui correspond à un certain nombre de services (logiciels) qui seront lancés. A tout moment Linux se trouve dans un niveau de démarrage particulier, on peut très bien passer d'un niveau de démarrage à un autre, ce qui aura pour effet d'arrêter ou de lancer d'autres services. Un certain nombre de niveaux de démarrage ont été prédéfinis sur le système, on peut les voir dans le fichier `/etc/inittab` (sur la distribution Linux Ubuntu on peut trouver un nouveau système « Upstart », remplaçant `/sbin/init` utilisant des fichiers de configuration dans `/etc/init`, un par service, listant les niveaux dans lesquels ils doivent être démarrés mais ajoutant la notion de dépendance, plus fine que simplement un ordre de démarrage).

7.2.1 /etc/inittab

Voici ce qu'on peut trouver au début de ce fichier :

```
# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
id:5:initdefault:
```

Note : La répartition des niveaux de démarrage dépend du système utilisé. Par exemple, sous Linux Debian, les niveaux 2,3,4 et 5 ne forment qu'un seul niveau (le 2). Dans `/etc/inittab` on y trouve en particulier les lignes :

```
# /etc/inittab - Debian
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.
```

Prenons le cas plus général à 6 niveaux de démarrage. Le niveau 0 réalise un arrêt (shutdown) de la machine, 1 pour rentrer en mode single user (uniquement root de connecté), 2 en mode multiutilisateur sans NFS (partage de SGF en réseau), 3 le mode multiutilisateur complet, 4 inutilisé, 5 mode multiutilisateur avec l'interface graphique X lancée automatiquement, et 6 pour rebooter la machine. initdefaut définit l'état de marche par défaut de la machine en fonctionnement normal.

Au démarrage, le noyau exécute `/sbin/init`, qui est le premier processus à être lancé. Pour arrêter la machine, par exemple, il est possible de taper manuellement `init 0`. Mais dans la pratique, il va lire séquentiellement le fichier `/etc/inittab` pour lancer les autres services. Voici le fichier `/etc/inittab` avec les commentaires associés (la syntaxe est : `étiquette:niveaux de démarrage:action:commande`) :

```
# The default runlevel is defined here
id:3:initdefault:
# First script to be executed,
#if not booting in emergency (-b) mode
si::bootwait:/etc/init.d/boot

l0:0:wait:/etc/rc.d/rc 0

l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3

l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
# what to do when CTRL-ALT-DEL is pressed
ca::ctrlaltdel:/sbin/shutdown -r -t 4 now

# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon

# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -h +1

# Single user mode

~~:S:wait:/sbin/sulogin
```

Niveau de démarrage par défaut = 3

Au démarrage c'est le script `/etc/rc.d/rc.sysinit` qui va être lancé en premier, il permet d'initialiser le système et réalise les tâches suivantes (liste non exhaustive) : initialisation du réseau, de la swap, vérification des disques (fsck éventuel), montage des disques, chargement des modules,etc. Les messages que l'on voit au boot sont générés par ce fichier.

On ne lit pas cette ligne car le niveau de démarrage par défaut est 3

idem

idem

Le niveau de démarrage 3 lance tous les services qui sont dans `/etc/rc.d/rc3.d`, voir l'exemple suivant. Le wait signifie que tant que tous les scripts de lancement ne se sont pas achevés, on ne va pas plus loin dans le fichier. On ne lit pas les 3 lignes suivantes :

Si l'on appuie sur les touches ctrl+alt+del, on déclenche un shutdown (arrêt de la machine) après 4s (-t 4) à partir de l'affichage du warning, -r redémarrage après arrêt, now pour dire immédiatement.

Lancement de consoles en mode texte suivant le niveau de démarrage défini en début de ligne. respawn signifie que le processus sera relancé automatiquement si jamais il est interrompu.

Si l'on passe au niveau de démarrage 5, lancement de l'interface graphique

Si on détecte une panne de courant par un système de type UPS on déclenche un arrêt propre dans 1 heure quel que soit le niveau de démarrage.

Si l'on démarre en mode simple utilisateur (ce qui peut être forcé au niveau du chargeur de boot, grub ou lilo pour Linux) alors demander un mot de passe. L'étiquette est `~~` pour être sûr que ce soit la dernière dans l'ordre lexicographique.

Exemple : passage au niveau de démarrage 3 : le script consulte le répertoire `/etc/rc.d/rc3.d`.

7.2.2 Scripts init system V

On trouve dans les répertoires `/etc/rc.d/rc`$< n >$`.d`, par exemple :

```
lrwxrwxrwx  1 root root 10 2008-10-08 01:54 K13nfs -> ../nfs
lrwxrwxrwx  1 root root 10 2008-10-08 01:54 K15portmap -> ../portmap
lrwxrwxrwx  1 root root  9 2008-10-08 01:44 S03resmgr -> ../resmgr
lrwxrwxrwx  1 root root 12 2008-10-08 01:42 S04boot.udev -> ../boot.udev
lrwxrwxrwx  1 root root 10 2008-10-08 02:06 S05network -> ../network
lrwxrwxrwx  1 root root  9 2008-10-08 01:42 S06syslog -> ../syslog
lrwxrwxrwx  1 root root  7 2008-10-08 01:53 S12sshd -> ../sshd
```

```
lrwxrwxrwx  1 root root  9 2008-10-22 21:14 S18autofs -> ../autofs
lrwxrwxrwx  1 root root  7 2008-10-08 01:53 S20cron -> ../cron
```

Le système va arrêter les services dont les liens commencent par K puis lancer ceux commençant par S, par ordre croissant (notons qu'il existe une version plus moderne ou l'on donne à l'intérieur même du script les dépendances). Si les services commençant par K ne sont pas lancés, ils ne seront pas (ré)arrêtés, et de même, si les services commençant par S sont déjà lancés, ils ne seront pas (re)lancés. Pour s'en convaincre, il suffit de lire le script `/etc/rc`, qui est le script qui est exécuté avec l'option du niveau concerné. On peut y trouver en particulier `runlevel` qui est le niveau choisi :

```
# First, run the KILL scripts.
for i in /etc/rc$runlevel.d/K* ; do
        check_runlevel "$i"  || continue

        # Check if the subsystem is already up.
        subsys=${i#/etc/rc$runlevel.d/K??}
        [ -f /var/lock/subsys/$subsys -o -f /var/lock/subsys/$subsys.init ] \
                || continue

        # Bring the subsystem down.
        if LC_ALL=C egrep -q "(killproc |action )"  $i ; then
                $i stop
        else
        else
                action "Stopping $subsys:"  $i stop
        fi
done

# Now run the START scripts.
for i in /etc/rc$runlevel.d/S* ; do
        check_runlevel "$i"  || continue
...
```

7.2.3 Créer son script de lancement de service dans Initscripts System V

Voici un exemple de mise en place et de lancement d'un script personnalisé, appelons le totod. On suppose que l'on souhaite lancer le service uniquement à l'état de marche 3, 4 et 5 et l'arrêter lorsque l'on passer à l'état de marche 6, 2 et inférieur :

- il faut commencer par écrire un script totod dans `/etc/rc.d/init.d`. en recopiant un script existant
- on créer un lien dans `/etc/rc.d/rc3.d` :
  `ln -s /etc/rc.d/init.d/totod /etc/rc.d/rc3.d/S99totod`
  Le service sera donc celui qui sera lancé en dernier à l'état 3. Faire de même dans les répertoires rc4.d et rc5.d.
- Ensuite tapez pour l'état de marche 2 :
  `ln -s /etc/rc.d/init.d/totod /etc/rc.d/rc2.d/K01totod`
  Le service sera donc celui qui sera arrêté en premier à l'état 2. Faire de même dans les répertoires rc0.d, rc1.d et rc6.d.

Rappelons que quand le lien commence par S le script est appelé avec l'argument start et par K avec l'argument stop.

Un autre moyen consiste à utiliser la commande `chkconfig` (attention cette commande n'existe pas sur tous les unix, en particulier pas sur Debian utilisez plutôt `update-rc.d`). Il faut rajouter en entête du fichier (après la ligne `#!/bin/sh`) :

```
# chkconfig:  60 40
```

Pour lancer la commande aux états de marche 3, 4 et 5, il suffira de taper :

```
chkconfig --level 345 totod on
```

Et pour l'arrêter aux états de marche 0, 1, 2 et 6 il suffira de taper :

```
chkconfig --level 0126 totod off
```

Les liens S60totod et K40totod seront ainsi créés.

La commande `chkconfig` utilisée seule (ou avec l'option `--list`) donne également la liste des services activés et désactivés.

En général, les services cherchent leurs fichiers de configuration dans le répertoire `/etc`.

7.2.4 Les services réseaux « inetd »

Contrairement au daemon en *standalone* (démarrages autonomes), qui tourne alors dans un processus dédié, et afin d'alléger la charge mémoire et CPU, une méthode a été mise au point : elle consiste à séparer les tâches d'écoute et de services proprement dit. Ainsi, les serveurs a priori moins sollicités (telnet, ftp, pop ...) ne sont démarrés qu'en cas de besoin, en présence d'une requête émanant d'un client autorisé. Ceci a été réalisé par un système de sous-traitance qui permet la séparation des taches : un superviseur, inetd ou xinetd, écoute l'ensemble des ports d'entrée, vérifie si nécessaire les droits du client (selon sa provenance, l'heure etc.) et démarre une instance du serveur particulier auquel s'adresse la requête.

Pour contrôler le super-serveur inetd, la commande est comme d'habitude pour les services : `/etc/init.d/inetd start | stop`.

La nature de la requête est identifiée selon le protocole qu'elle utilise et le port serveur auquel elle s'adresse ; le service demandé est connu en regardant dans la base de données stockée dans les fichiers `/etc/protocols` et `/etc/services`. `inetd` utilise `/etc/inetd.conf` pour savoir si le service demandé est un de ces « services » inscrits. Il est donc recommandé pour la sécurité du système de désactiver les services non utilisés dans `/etc/inetd.conf`.

Dans certains Unix, inetd a été remplacé par xinetd. Le principe est très similaire, à la différence que, dans `/etc/xinetd.d`, chaque service dispose de son propre fichier de configuration comportant des options de sécurité plus évoluées telle que la possibilité d'ouvrir le service qu'à certaines heures du jour. Il faut noter que ces services sont généralement désactivés par défaut. Pour rendre actif un service, il faut remplacer : `disable = yes` par `disable = no` dans le fichier de configuration correspondant.

7.3 Les fichiers de trace d'exécution ou logs (syslogd)

Un aspect incontournable de la sécurité consiste à pouvoir garder des traces des « opérations » effectuées par le système. Cette possibilité est donnée grâce au démon `syslogd`. Il permet de rediriger vers des fichiers ou des terminaux les messages envoyés par d'autres démons, services ou par le noyau Linux. Le démarrage se fait par `/etc/init.d/syslog start`. Il est d'usage de mettre les fichiers de log dans le répertoire `/var/log` et ce répertoire dans une partition propre (afin d'éviter qu'une saturation de ce répertoire n'entraîne un arrêt du système tout entier).

Les différentes catégories de services (« facility ») sont :

auth ou security	Messages de sécurité et d'authentification.
authpriv	La même chose que précédemment, mais logs plus privés
cron	Messages de crontab
daemon	Messages systèmes générés par un daemon
ftp	Messages du serveur ftp
kern	Messages du noyau
lpr	Messages du serveur d'impression
mail	Messages du serveur de messagerie
news	Messages du serveur de news
syslog	Messages de syslog lui-même
user (défaut)	Messages générés par le programme en cours d'un utilisateur
uucp	Messages UUCP (Unix to Unix Copy)

Remarques : UUCP (Unix to Unix Communication Protocol) est un ancien protocole qui permet à deux machines d'échanger des fichiers et d'exécuter des commandes sur la machine distante en passant par une ligne téléphonique (modem), mais aussi sur une couche TCP/IP, voire via un câble série direct (null modem).

En plus de la catégorie de service, est associé une « sévérité », classée de la moins grave à la plus grave.

debug (7)	Messages de debogage
info (6)	Messages d'information
notice (5)	Messages un peu plus importants que les messages info
warning ou warn (4)	Messages d'avertissement
err (3)	Messages d'erreur
crit (2)	Situation critique
alert (1)	Situation critique nécessitant une intervention immédiate
emerg ou panic (0)	Système inutilisable

Le fichier de configuration est `/etc/syslog.conf` dont voici un exemple :

```
# Tous les messages du noyau vers la console.
# En plus de les envoyer sur la console on les dirige vers le fichier
# /var/log/kernel.
kern.*                                          /dev/console
kern.*                                          -/var/log/kernel
# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none        /var/log/messages
# envoyer tous les messages sur la console 12
*.*                                             /dev/tty12
# The authpriv file has restricted access.
authpriv.*                                      /var/log/secure
# Log all the mail messages in one place.
mail.*                                          /var/log/maillog
# Log cron stuff
cron.*                                          /var/log/cron
# Emergencies are sent to everybody logged in.
#
*.emerg                                         *
```

Commentaires :
- Le signe « - » est utilisé devant les noms de fichiers les moins critiques pour améliorer les performances en écriture au risque de perdre des données en cas de crash du système (pas de synchronisation des fichiers à chaque message)
- Log de tous les messages dans le fichier `/var/log/messages` à partir du niveau info inclus, sauf (none) les messages de type mail, que l'on place dans `/var/log/maillog`, les messages authpriv et cron (qui vont dans `/var/log/cron`)
- On dirige tous les messages vers la console 12 (accessible par CTRL+ALT+F12)
- Log de tous les messages d'urgence rendant le système instable dans tous les fichiers et dans toutes les consoles utilisateurs existantes.
- La commande `logger` permet d'enregister des messages dans les logs par exemple à partir de scripts (voir aussi printk en C).
- Diverses versions de syslog coexistent désormais, citons syslog-ng et rsyslog

Remarque : la commande `tail -f` permet de lire un fichier de log en temps-réel.

7.4 Systemd, journald, logind

Cette section est encore en construction [2]. Systemd est en passe de remplacer peu à peu init System V présenté à la section précédente. Les distributions suivantes l'ont activé par défaut : Fedora (v15 Mai 2011), Mageia (v2 Mai 2012), openSUSE (v12.2 Sept 2012), Red Hat Entreprise (v7.0 Juin 2014), Debian 8 "Jessie" (2014), Ubuntu (planifié), Gentoo (utilisé pour GNOME 3), CentOS (v7 Oct 2014).

7.4.1 Systemd

Systemd est un gestionnaire de service qui permet l'accélération du démarrage (par parallélisation des démarrages de services indépendants, voire même interdépendants en utilisants des sockets). Il ne repose pas sur des scripts shell mais sur une multitude de petits fichiers de configuration. Systemd intègre journald (remplaçant de syslogd pour la gestion des logs) et logind (pour la gestion des sessions utilisateurs). Systemd intègre aussi le code de udev depuis 2012 pour la gestion à chaud des périphériques.

Les éléments gérés s'appellent des **unités**. Une **unité** peut être un **service**, un **target**, un **montage de SGF**, un **socket** (remplace inetd), un **device** (périphérique),... La configuration de systemd repose sur l'arborescence `/usr/lib/systemd/system`. Les fichiers des unités qui sont dans ce répertoire peuvent être masqués par une configuration personnalisée dans `/etc/systemd/system`. Les fichiers d'unité peuvent être complétés par un répertoire `<nom>.targets.wants` listant les dépendances.

Un **service** s'écrit nom.service, par exemple `sshd.service` :

2. Une référence intéressante à consulter est l'article sur ce sujet du numéro 153 de Linux Magazine, consultable à l'adresse `http://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-153/Systemd-vainqueur-de-Upstart-et-des-scripts-System-V`

```
[Unit]
Description=OpenSSH server daemon
After=syslog.target network.target auditd.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStartPre=/usr/sbin/sshd-keygen
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target
```

Un **target** regroupe un ensemble d'unités avec des dépendances et des points de synchronisation. Par exemple, `default.target` est défini pour préciser ce qu'il faut démarrer par défaut en donnant le nom de **target** à démarrer.

```
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
After=multi-user.target
Conflicts=rescue.target
Wants=display-manager.service
AllowIsolate=yes

[Install]
Alias=default.target
```

Un **montage de SGF** décrit ce qui doit être monté et où. Par exemple `tmp.mount` :

```
[Unit]
Description=Temporary Directory
Documentation=man:hier(7)
Before=local-fs.target

[Mount]
What=tmpfs
Where=/tmp
Type=tmpfs
Options=mode=1777,strictatime
```

Le pilotage des services à chaud repose sur l'usage de la commande `systemctl`. Voici quelques usages de cette commande :

systemctl	description	*init System V*
systemctl reboot	redémarrer	init 6
systemctl poweroff	arrêter le système	init 0
systemctl suspend	suspendre (en mém.)	
systemctl list-units	lister les **unités**	chkconfig –list
systemctl start < *unit* >	démarrer l'unité < *unit* >	service < *unit* > start
systemctl stop < *unit* >	arrêt de l'unité	service < *unit* > stop
systemctl status < *unit* >	état de l'unité	service < *unit* > status
systemctl enable < *unit* >	active cette unité pour le prochain démarrage	chkconfig < *unit* > on
systemctl disable < *unit* >	désactive cette unité	chkconfig < *unit* > off
systemctl list-dependencies [< *unit* >]	liste les dépendances de cette unité ou toutes	
systemctl restart < *unit* >	redémarrer telle unité, utile pour les services	service < *unit* > restart
systemctl daemon-reload	reload systemd, recherche d'unités nouvelles ou modifiées	
systemctl –failed	liste les unités qui n'ont pas pu démarrer	
systemctl -H user@host	option permettant de contrôler un systemd distant via ssh.	

7.4.2 Systemd-journald

Journald a pour finalité d'être un remplaçant de syslog (ou rsyslog) (voir man systemd-journald) faisant partie intégrante de systemd. Syslog gère les traces d'exécution de façon linéaire (dans le temps) dans des fichiers textes alors que journald gère des index sur des fichiers binaires. L'objectif de journald est de rendre cette sauvegarde des «logs» sécurisée et infalsifiable. Par conséquent il faut passer par l'exécutable `journalctl` pour voir le contenu des logs.

journalctl	description	*Syslog*
journalctl -f	option pour suivre l'évolution du journal à chaud	`tail -f /var/log/messages`
journalctl -p <prio>	option pour afficher uniquement ce niveau de priorité (peut être un intervalle 3..5 par exemple)	
journalctl executable	affiche les trace concernant cet executable, attention il faut le chemin complet	
journalctl _SYSTEMD_UNIT=<nom.service>	affiche les logs de ce service	
journalctl _PID=<num>	affiche les logs concernant ce processus	
journalctl <expression> + <expression>	le «+» signifie l'un ou l'autre (le «et» est un simple espace)	

Le fichier de configuration est `/etc/systemd/journald.conf`. Rsyslog est encore nécessaire pour certaines actions telle que renvoyer les logs sur un serveur de log. Il faut donc encore connaître `/etc/rsyslog.conf` et `/etc/syslog.conf`. Notons également que les configurations de journald et rsyslog sont disjointes, ce qui signifie que les logs peuvent être différents suivant que l'on utilise journald (`journalctl -f`) ou rsyslog (`tail -f /var/log/messages`).

7.4.3 Logind

Logind fait partie intégrante de systemd et remplace entre autres getty pour le lancement de plusieurs consoles au démarrage. Son fichier de configuration est `/etc/systemd/logind.conf`, on y trouve entre autres la possibilité de paramétrer le nombre de consoles à démarrer, que tuer comme processus quand un utilisateur se délogue etc. Logind définit la notion de «seat» pour différentier les utilisateurs et les consoles. Il peut être utilisé à distance suivant autorisations.

La commande `loginctl` permet un paramétrage à chaud de logind :

loginctl list-users	liste les utilisateurs actuellement connectés
loginctl show-session <numero>	présente des détails sur telle session (qui est logué, comment,...), sans numéro donne des infos sur logind
loginctl activate <numero>	bascule en tache de premier plan la ssession indiquée
loginctl lock-session	

7.5 Planification de tâches : cron et crontab

Il s'agit de faire de la planification grâce au service `crond` (`/etc/init.d/crond start`). La commande `crontab` permet de modifier le fichier de configuration qui permettra de lancer des commandes à intervalles réguliers ou à certaines dates.

Tout utilisateur autorisé par `/etc/cron.allow` et non refusé par `/etc/cron.deny`, il faut lancer la commande crontab avec l'option -e. L'administrateur a toujours le droit de le faire. L'éditeur par défaut est alors appelé. Il ne reste plus qu'à entrer les différents champs, dans un ordre particulier :

1. minute (0 à 59).
2. heure (0 à 23).
3. jour du mois (1 à 31).
4. mois (1 à 12)
5. jour de la semaine (0 à 6 : 0 = Dimanche, 1 = Lundi...ou mon, tue ...).
6. commande, telle qu'elle serait saisie sous l'interpréteur mais de préférence avec le chemin complet.

Une méthode très utilisée consiste à écrire son propre fichier crontab (au lieu de l'éditer avec la commande `crontab -e`) puis de régénérer la table crontab avec la commande `crontab nom-fichier`. Pour supprimer la table crontab on utilisera l'option « -r ». L'option « -l »permet d'imprimer la table crontab en cours sur la sortie standard.

Exemple de fichier crontab (qui se trouve dans `/var/spool/crontab/utilisateur` pour le crontab d'un utilisateur)

```
10 8 * * *    cd sau-lipn; make current
0 23 * * *    cd CerPAN/cerpan/www; make daily
```

A différentier du format du fichier système `/etc/crontab` (notons surtout l'ajout par rapport au format précédent d'un nom d'utilisateur qui sera utilisé pour lancer la commade : ici root pour chaque ligne).

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root nice -n 19 run-parts --report /etc/cron.hourly
02 4 * * * root nice -n 19 run-parts --report /etc/cron.daily
22 4 * * 0 root nice -n 19 run-parts --report /etc/cron.weekly
42 4 1 * * root nice -n 19 run-parts --report /etc/cron.monthly
```

Ce qui signifie que les répertoires `/etc/cron.hourly` etc. contiennent respectivement les scripts a lancer toutes les heures, tous les jours (à 4H02 du matin), etc. La commande run-parts exécute tous les fichiers exécutables situés dans le répertoire donné en paramètre. L'option –report sert à afficher les noms des exécutables qui produisent une sortie (erreur ou standard).

7.6 Configuration de DHCP

Il faut commencer par installer les paquetages dhcp-common et dhcp-server. Il y a un seul fichier de configuration : `/etc/dhcpd.conf`.

Attention le Daemon du serveur DHCP va écouter par défaut toutes les interfaces réseaux actives, il va falloir lui préciser l'interface que l'on veut qu'il écoute.

Pour cela il nous faut éditer le script dhcpd (qui se trouve généralement dans `/etc/rc.d/init.d/`). Éditer la ligne comme suit :

```
# Define INTERFACES to limit which network interfaces dhcpd listens on.
# The default null value causes dhcpd to listen on all interfaces.
#INTERFACES="eth0"
INTERFACES=""
```

Modifier la dernière ligne en précisant le nom de l'interface qui doit être écouté. Par exemple : `INTERFACES="eth0"`.

Notons que cette solution n'est pas la meilleure qui soit, suivant les distributions il existe un fichier permettant d'éviter de changer le script de démarrage : voir `/etc/sysconfig/dhcp*.conf` et `/etc/default/dhcp*.conf` etc.

Reste maintenant à configurer le fichier dhcpd.conf

Voici une configuration de base pour un petit réseau local :

```
# Forcage de la mise à jour des IP fixes
update-static-leases on;
ddns-domainname \og maison.mrs\fg ;
max-lease-time 3600;
default-lease-time 3600;

# Les clients du réseau seront tous reconnus mais
# si on ne connaît pas leur adresse mac.
allow unknown-clients;

# Durée de vie du bail
max-lease-time 3600;
default-lease-time 3600;

# Les options qui seront transmisent aux clients
option domain-name-servers 192.168.0.254;
option domain-name \og reseau.net\fg ;
option routers 192.168.0.254;

# La définition de la plage d'IP à distribuer.
subnet 192.168.0.0 netmask 255.255.255.0 {
range 192.168.0.1 192.168.0.10;
}
```

Voici maintenant une configuration plus complexe avec une mise à jour automatique des DNS, et la possibilité d'attribuer des adresses ip fixe (grâce a l'adresse mac des machines) et des adresses ip dynamiques.

```
# méthode de mise à jour du DNS :
ddns-update-style interim;
# mise à jour autorisée
ddns-updates on;
# Forcage de la mise à jour par le serveur DHCP
ignore client-updates;
# Forcage de la mise à jour des IP fixes
update-static-leases on;
ddns-domainname \og maison.mrs\fg ;
max-lease-time 3600;
default-lease-time 3600;

# Les options qui seront transmisent aux clients
option domain-name-servers 192.168.0.254;
option subnet-mask 255.255.255.0;
option routers 192.168.0.254;
# La définition de la plage d'IP à distribuer
subnet 192.168.0.0 netmask 255.255.255.0 {

# Définition des adresses dynamiques
range 192.168.0.64 192.168.0.127;

# définition des adresses ip fixe.
host moteyo {
hardware ethernet 17:03:4E:63:E9:F2;
fixed-address 192.168.0.150; } host chaudron {
hardware ethernet 12:25:31:F5:E2:98;
fixed-address 192.168.0.151; } host foz {
hardware ethernet 06:21:56:9A:C9:45;
fixed-address 192.168.0.152; } host titax {
hardware ethernet 01:56:42:8F:2B:4E;
fixed-address 192.168.0.153; }
```

```
}

# Pour la mise à jour dynamique du DNS local
allow unknown-clients;
zone reseau.net. {
primary 127.0.0.1; } zone 0.168.192.in-addr.arpa. {
primary 127.0.0.1; }
```

Une fois ces configurations effectuées il faut relancer le serveur DHCP avec la commande : `/etc/init.d/dhcpd restart`.

7.7 Configuration de Apache

Ce paragraphe décrit les principaux paramètres pour mettre en place un service HTTP minimum, avant de lancer le service serveur. Le fichier de configuration d'Apache est httpd.conf (variable suivant les distributions).

- `port 80` indique quel est le port utilisé par le service (par défaut 80). Il est possible d'utiliser un autre port, par contre vous devrez spécifier au navigateur quel est le port utilisé par le serveur. Si vous configurez par exemple le port 8080 sur une machine www.domaine.fr, vous devrez spécifier dans le navigateur www.domaine.fr :8080, pour que le serveur reçoive et traite votre requête.
- `user <pseudo_user>` et `group <groupe_pseudo_user>` spécifient souvent le pseudo compte utilisé par le serveur une fois qu'il est lancé. En effet, pour accéder aux ports inférieurs à 1024, le serveur utilise un compte administrateur, ce qui présente des dangers. Une fois le processus actif, il utilisera l'UID d'un autre compte. Ce compte doit pouvoir lire les fichiers de configuration et ceux de la racine du serveur HTTP. Certaines distributions utilisent le compte « www-data », « nobody »ou encore « apache ».
- `ServerAdmin <adresse mail>` précise quel est le compte qui reçoit les messages. Par défaut le compte administrateur sur la machine locale.
- `ServerRoot <repertoire>` indique l'adresse du répertoire racine du serveur, où sont stockés les fichiers de configuration du serveur HTTP. Cette adresse peut être modifiée.
- `ErrorLog <fichier>`, journalisation des erreurs. L'adresse est calculée à partir de ServerRoot.
- `ServerName <www.domaine.fr>` indique le nom ou l'alias avec lequel la machine est désignée. Par exemple, l'hôte ns1.domaine.fr, peut avoir le nom d'alias www.domaine.fr. Voir le cours réseaux sur la résolution de nom avec un DNS.
- `DocumentRoot <repertoire>` indique l'emplacement par défaut des pages HTML quand une requête accède au serveur. Par exemple, si `DocumentRoot` est `/home/httpd/html`, l'URL `http://www.domaine.fr/a.html` correspond au fichier `/home/httpd/html/a.html`
- `ScriptAlias /cgi-bin/ /usr/lib/cgi-bin`, de la forme « ScriptAlias FakeName RealName », indique où sont physiquement situés les scripts sur le disque, ainsi que l'alias utilisé par les développeurs pour le développement des scripts et des pages.
- `UserDir public_html`, ce paramètre décrit le processus utilisé pour accéder aux pages personnelles d'une personne, si ces pages sont stockées dans son répertoire personnel.
  Attention, vérifier que le répertoire personnel a les droits de lecture et parcours pour les autres, car le serveur web tourne souvent sous une identité non root.
- `Alias /CheminVu/ /CheminRéel/`, par exemple : « /icons/ /usr/share/apache/icons/ », ce paramètre permet de renommer, à la manière d'un lien logique, un emplacement physique avec un nom logique.
  Exemple : vous voulez que www.MonDomaine.edu/test/index.html, ne corresponde pas physiquement à un répertoire sur la racine du serveur HTTP mais à un emplacement qui serait /usr/local/essai. Vous pouvez mettre dans le fichier de configuration d'Apache un alias de la forme : alias /test/ /usr/local/essai/
- `DirectoryIndex` donne le ou les noms des fichiers que le serveur doit rechercher si le navigateur passe une requête sur un répertoire. Par exemple sur une requête http ://www.domaine.fr, le serveur va rechercher dans l'ordre s'il trouve un fichier index.html, index.stml, index.cgi... en fonction des paramètres de cette variable.
- Les fichiers `.htaccess` : Apache permet de sécuriser les accès répertoire par répertoire. Il est possible de définir, le nom du fichier qui contiendra les possibilités d'accès par un utilisateur à un répertoire donné. Par défaut la valeur est `.htaccess`. Ce paramètre est modifiable.
- Limitations de la sécurité par répertoire : ce procédé alourdit la charge du serveur. En effet, si une requête est passée sur www.domaine.fr/rep1/rep2/index.html, le serveur va vérifier dans chaque répertoire rep1, rep2... l'existence d'un fichier .htaccess. Ce sont les règles du dernier fichier qui seront appliquées. Ce processus est mis en œuvre pour chaque accès. Cette directive est donc à utiliser avec beaucoup de parcimonie car elle crée une surcharge pour le serveur.

- La directive `AllowOverride None`, permet de désactiver l'utilisation des fichiers .htaccess dans les niveaux inférieurs. La directive AllowOverride peut être utilisée avec d'autres options par exemple : AuthConfig. Les fichiers .htaccess peuvent, s'ils sont présents spécifier leurs propres directives d'authentification
- La directive `ExecCGI`, permet l'exécution de scripts cgi dans ce répertoire.

Restreindre l'accès à certains répertoires. Pour chaque répertoire « rep », sur lequel on désire avoir une action, on utilisera la syntaxe suivante :

```
<Directory UnRepertoire>
...Ici mettre les actions...
</Directory>
```

Tout ce qui est entre les balises s'applique au répertoire « rep ». Exemple : On désire autoriser l'accès du répertoire « /intranet »aux machines du réseau d'adresse 192.168.1.0/24 et de nom de domaine domaine.fr, et l'interdire à tous les autres :

```
<Directory /intranet>
#Ordre de lecture des règles
order allow,deny
deny from all
allow from 192.168.1 #ou encore allow from .domaine.fr
</Directory>
```

Il importe de préciser dans quel ordre les règles de restriction vont être appliquées. Cet ordre est indiqué par le mot réservé « order », par exemple « order deny,allow »(On refuse puis on alloue l'accès à quelques adresses, c'est à dire que toutes les régles deny vont être lues d'abord, puis ce sera le tour de toutes les règles allow) ou « order allow,deny »(on accepte généralement les accès mais il sont refusés pour quelques adresses : ici, on prend en compte en premier lieu toutes les règles allow dans l'ordre trouvé, puis ensuite toutes les règles deny).

Exemple : On désire que l'accès soit majoritairement accepté, sauf pour un ou quelques sites :

```
<directory /home/httpd/html>
AllowOverride none
Order deny,allow
deny from pirate.com badboy.com cochon.com
allow from all
</directory>
```

Authentifier l'accès à un répertoire : ce procédé va permettre de sécuriser l'accès à un répertoire ou à des fichiers. L'accès sera autorisé à une ou plusieurs personnes ou encore à un ou plusieurs groupes de personnes. AuthName, définit ce qui sera affiché au client pour lui demander son nom et son mot de passe.

AuthType, définit le mode d'authentification et d'encryptage « basic »avec HTTP/0 ou « MD5 »par exemple avec HTTP/1.

AuthUserFile, définit le fichier qui contient la liste des utilisateurs et des mots de passe. Ce fichier contient deux champs (Nom d'utilisateur, Mot de passe crypté). Vous pouvez créer ce fichier à partir du fichier /etc/passwd (attention ! faille de sécurité potentielle : il n'est pas forcément avisé d'avoir le même mot de passe pour accéder à Linux et pour accéder à un dossier Web) ou avec la commande « htpasswd »d'Apache.

Exemple du mode d'utilisation de la commande « htpasswd » :

```
root@mr:/home# htpasswd --help
        htpasswd [-cmdps] passwordfile username
 -c  Create a new file.

#> htpasswd -c /etc/apache/users mlx
```

Ici on crée le fichier /etc/apache/users et on ajoute un compte. N'utiliser l'option « -c »que la première fois.
AuthGroupFile définit le fichier qui contient la liste des groupes et la liste des membres de chaque groupe.
Require permet de définir quelles personnes, groupes ou listes de groupes ont une permission d'accès.
Exemple de fichier AuthUserFile :

```
dodo:aesvsjdd<HJSbsbzu
didi:Gbhndzkjdb.jsk
```

Exemple de fichier AuthGroupFile :

```
users: toto tata titi tutu
```

Exemple d'autorisation :

```
require user toto titi /* toto et titi ont un accès */
require group users /* le groupe users à un accès */
require valid-user /* toute personne existant dans AuthUserFile */
```

Exemple d'accès sécurisé sur un répertoire :

```
<Directory /home/httpd/html/intranet/>
AuthName PatteBlanche
AuthType basic
AuthUserFile /etc/httpd/conf/users
AutGroupFile /etc/httpd/conf/group
     <Limit GET POST>#Ici il faudra un mot de passe
        require valid-user
     </Limit>
</Directory>
```

La déclaration d'un accès authentifié sur un répertoire est faite dans le fichier de configuration d'Apache, ou alors en créant un fichier « .htaccess »dans le répertoire que l'on souhaite sécuriser. Le fichier « .htaccess »contient les mêmes directives que celles qui auraient été déclarées dans le fichier httpd.conf.

Attention, si vous mettez les clauses d'accès restreint pour un répertoire dans le fichier de configuration d'Apache, les clauses seront incluses entre 2 balises :

```
<Directory ...>
</Directory ...>
```

Si vous mettez les clauses de restriction dans un fichier « .htaccess », vous n'avez pas besoin de mettre ces balises.

Pour démarrer, voir l'état ou encore arrêter le serveur web :

```
service apache start
service apache status
service apache stop
```

Test de la configuration : lancez le navigateur et tapez l'url `http://localhost`. Vous devriez pouvoir utiliser indifféremment l'adresse IP ou le nom d'hôte de votre machine. Vous devez également pouvoir accéder à partir des autres machines du sous-réseau.

7.8 Synchronisation d'horloge

Le meilleur protocole est sans conteste NTP : Network Time Protocol. Voici un exemple de fichier de configuration : `/etc/ntp.conf` :

```
# deny-by-default policy
restrict default ignore

# syncronize with european time servers
server europe.pool.ntp.org
# this one will not be the same!
server europe.pool.ntp.org

# By default, exchange time with everybody, but don't allow configuration.
restrict -4 default kod notrap nomodify nopeer noquery

# local fudge if network servers not available
server  127.127.1.0     # local clock
fudge   127.127.1.0 stratum 10
```

```
# Local users may interrogate the ntp server more closely.
restrict 127.0.0.1

# track wander (leading directories will need to exist!)
driftfile /var/spool/ntp.drift
```

8 Appel de procédure à distance et applications : RPC, NFS et NIS

Les RPC (développées par sun) permettent l'exécution de procédures sur une machine distante. Nombreuses sont les applications ayant recours aux RPC (Remote Procedure Call). On trouve notamment NFS, NIS mais également la plupart des r-commandes rlogin, rsh, rcp, rusers, rwall, etc...

Les procédures (ou fonctions) RPC sont regroupées en programmes et identifiées par des numéros. Les programmes se voient aussi attribuer un numéro ainsi qu'un numéro de version. C'est par le biais de ce triplet qu'un client peut appeler une procédure particulière. Les numéros sont attribués d'une façon stricte (idem ports TCP et UDP). Ainsi mountd se voit attribuer le numéro 100005. Le programme serveur RPC contenant les procédures distantes utilise les ports éphémères, pas les ports bien connus. Ceci nécessite l'existence d'un standard conservant trace des ports éphémères utilisés. Le processus assurant cette fonction, c'est `portmap` (ou sunrpc ou rpcbind), "l'organiseur de ports". Il est lui-même un programme serveur RPC de numéro 100000, de version 2 et écoutant sur les ports 111 TCP et UDP.

Remarque : conjointement aux RPC et au-dessus (dans le modèle OSI), on utilise le protocole XDR (eXternal Data Representation). XDR gère la mise en forme des données : cryptage, conversion EBCDIC/ASCII. Il définit un standard de représentation des types sur le réseau, afin notamment de palier la multiplicité des représentations utilisées (big endian, little endian, ...).

Les RPC se trouvent au niveau 5 du modèle OSI au dessus de TCP/UDP, tandis qu'XDR se trouve au niveau 6. Ces appels sont gérés par le processus portmap via les numéros vus plus haut. Ces numéros sont recensés dans le fichier `/etc/rpc`. Ce fichier contient les services constructeurs, ce n'est pas un fichier de configuration, il joue le même rôle que `/etc/services` dans la correspondance nom d'application/port associé pour la programmation RPC.

Par exemple :

```
# This file contains user readable names that can be used in place of rpc program numbers.

portmapper      100000  portmap sunrpc rpcbind
rstatd          100001  rstat rup perfmeter rstat_svc
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
etherstatd      100010  etherstat
...
```

Voici le schéma de fonctionnement :

1. Portmap est lancé et est en écoute sur les ports TCP et UDP 111.
2. Un serveur de RPC s'enregistre auprès de portmap : par son numéro de programme, sa version et son protocole de transport.
3. En retour, portmap lui attribue un numéro de port à écouter.
4. Lorsqu'un client veut solliciter le serveur, il contacte portmap (111/UDP ou TCP) avec les paramètres numéro de programme et version.
5. portmap lui retourne alors le port d'écoute du serveur correspondant pour le protocole, le numéro de programme et la version précisée.
6. la communication peut maintenant se faire directement entre le client et le serveur de RPC

La commande rpcinfo permet de connaître les "inscriptions" en cours auprès de portmap :

```
[toto@mach1:~] rpcinfo -p
   program no_version protocole  no_port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100007    2   udp    733  ypbind
    100007    1   udp    733  ypbind
    100007    2   tcp    736  ypbind
    100007    1   tcp    736  ypbind
...
```

Si la commande `rpcinfo -p` donne :

```
rpcinfo: ne peut contacter l'aiguilleur de ports: RPC: erreur système sur
l'hôte cible - Connexion refusée
```

C'est que le démon portmap n'est pas lancé...

8.1 NFS (Network File System)

NFS fournit un accès transparent aux systèmes de fichiers d'un serveur distant. Toute application travaillant avec un fichier local, peut travailler identiquement sans aucune modification avec un fichier distant via NFS. NFS n'accède qu'aux parties du fichier que référence un processus.

Lorsqu'un processus client demande un accès à un fichier :

1. il sollicite le noyau
2. le noyau détermine si le fichier est local ou distant
3. si le fichier est distant, le noyau passe au client NFS les références du fichier distant
4. le client sollicite le serveur concerné,
5. portmap sur le serveur recherche sur quel port l'application concernée (serveur NFS) écoute. En fait (cf. man nfs), si le démon NFS de l'hôte distant n'est pas enregistré dans le gestionnaire de ports, ou si le port 111 est filtré, le port standard pour NFS (2049) peut être utilisé directement. L'implémentation initiale de NFS utilisait UDP, il existe maintenant des implémentations TCP (en particulier NFSv4 qui intègre des possibilités d'authentification et de confidentialité via kerberos). On rappelle que UDP est sans états. L'inconvénient étant l'absence de gestion efficace des congestions, d'où l'intérêt d'utiliser TCP.
6. le serveur NFS opère la requête via les routines d'accès aux fichiers locaux.

Pour qu'un serveur exporte une partie de ses systèmes de fichiers, il doit :

1. démarrer portmap (puisque mountd utilise les RPC)
2. démarrer ensuite mountd, le démon de montage
3. mountd s'enregistre alors auprès de portmap

Pour qu'un client monte un système de fichiers distants appartenant à un serveur NFS en service :

1. mount doit faire une requête auprès du portmap distant
2. le portmap distant retourne le numéro de port à contacter pour joindre mountd
3. mount contacte ensuite mountd pour opérer le montage
4. mountd retourne à mount un descripteur de fichier dans le cas favorable
5. la consultation du système de fichiers distants est maintenant possible

8.1.1 Configuration d'un serveur NFS

1. Comme nous l'avons vu précédemment, les services portmap qui gère les connexions RPC, et nfs ont dû être installés initialement.
2. Le fichier `/etc/exports` :
   ce fichier (à créer s'il est absent) contient la liste des exportations. Sur chaque ligne, on précise un répertoire du système de fichiers, suivi par la liste des machines distantes clientes autorisées à les monter. Si cette liste est vide, toutes les stations accessibles sont autorisées.
   Exemples d'exportation déclarées dans le fichier `/etc/exports` sur le serveur p00 :

```
# repertoire   liste-machines  (liste-options)
/home/jean  pc2(ro)  pc3(rw)
/usr/bin  pc2(ro) pc3(ro)
/var/www/html  *.toto.fr (ro) pc3 (rw)
```

Pour valider un changement opéré dans ce fichier de configuration : `# exportfs -a`.

8.1.2 Configuration d'un client NFS

1. on effectue le montage, sur le point de montage, de la ressource. Par exemple :

   ```
   [root@pc3 /]# mkdir /mnt/nfs
   [root@pc3 /]# mount -t nfs p00:/home/httpd/html /mnt/nfs
   ```

   On a effectué le montage, sur le point de montage précédent, de la ressource `/var/www/html` (qui a été exportée par p00). L'utilisateur sur pc3 pourra alors mettre à jour le site WEB distant sur p00.
   La syntaxe générale est : `mount -t nfs nom-machine:arborescence point-montage`
2. Respects par nfs des droits :
   Bien sûr les permissions des fichiers importés s'appliquent vis à vis de l'utilisateur, notamment en ce qui concerne la directive (rw). On ne pourra mettre à jour sur la station cliente, un fichier exporté que s'il possède la permission w vis-à-vis de l'utilisateur.
3. Automatisation du montage :
   Pour cela, il suffit d'ajouter le contenu de la commande précédente dans une ligne du fichier `/etc/fstab` : `p00:/home/httpd  /mnt/nfs  nfs  auto, user`.
   Autres paramètres de montage : `intr` interrompt une requête NFS en cas de serveur indisponible, permet d'éviter un blocage.

8.2 NIS

Le service NIS (Network Information Service), permet de centraliser les connexions sur un réseau local. L'objectif central de tout serveur de fichiers d'un réseau local est de permettre aux utilisateurs du réseau de se connecter au serveur de fichier sous un compte centralisé au niveau du réseau, et non pas défini machine par machine et aussi d'accéder à ses fichiers (répertoire personnel, ...). La connexion et l'authentification sont du ressort d'un service d'annuaire tel que NIS, tandis que les accès aux répertoires personnels et partagés sont permis par exemple par le service NFS.

NIS maintient un ensemble de base de données (ou annuaire) centralisé pour un groupe de machines appelé domaine NIS. Supposons que le nom de domaine NIS attribué soit toto. Ces informations sont alors stockées dans le répertoire /var/yp/toto, sous forme d'un ensemble de fichiers de base de données (voir makedbm) binaires appelés cartes ou maps.

Les types d'informations que les stations clientes viennent chercher sont essentiellement les correspondances entre noms et adresse IP des machines du réseau, les vérifications des noms de login, mots de passe et groupe d'appartenance des comptes utilisateurs existants sur le serveur. Les réponses sont contenues dans 6 maps usuels, situés dans `/var/yp/toto`, et appelés hosts.byname, hosts.byaddr, passwd.byname, passwd.byuid, group.byname et group.bygid.

En fait il est possible de faire une hiérarchie de serveurs pour les grands domaines par le biais d'un serveur maître et de plusieurs serveurs esclaves (voir ypinit).

Remarque : NIS était appelé Yellow Pages, d'où les commandes commençant par yp... Mais "Yellow Pages" est une marque déposée, il a donc fallu adopter un autre nom.

8.2.1 Configuration du serveur NIS

Du côté serveur, les services à lancer sont portmap, ypserv (le serveur NIS) et yppasswd (le service spécialisé dans le changement des mots de passe) :

```
/etc/rc.d/init.d/portmap start
/etc/rc.d/init.d/ypserv start
/etc/rc.d/init.d/yppasswd start
```

Configuration :

1. Choisir un nom de domaine NIS, soit toto par exemple, indépendamment du nom de domaine du réseau (titi.fr). Supposons de plus que le serveur ait pour adresse IP : 10.100.200.1 avec un masque 255.255.255.0

2. Déclaration du domaine NIS :
   Éditer le fichier `/etc/sysconfig/network` (attention dépendant de la distribution linux), et y ajouter cette ligne : NISDOMAIN=toto
   Relancer le serveur par `/etc/rc.d/init.d/ypserver restart`
   Vérification : la commande domainname doit obtenir comme réponse ce nom de domaine NIS.
3. Préciser les machines autorisées à accéder au service NIS (optionnel si on configure `/etc/ypserv.conf`)
   Éditer le fichier `/var/yp/securenets` et insérer les lignes :

```
# pour permettre l'accès sur le serveur même
255.0.0.0   127.0.0.0
# pour permettre l'accès de toutes les machines du sous-réseau (masque et
# adresse réseau)
255.255.255.0   10.100.200.0
```

4. Préciser les informations que NIS doit gérer :
   Éditer le fichier `/var/yp/Makefile` et lister sur la ligne commençant par all :
   les données à gérer : `all: passwd group hosts`
5. Générer les cartes :
   Il s'agit maintenant de créer les 3 cartes (maps) correspondant aux 3 fichiers /etc/passwd, /etc/group et /etc/hosts.
   Faire `/usr/lib/yp/ypinit` pour créer le serveur NIS. On peut aussi faire `make` dans `/var/yp`. Il y a alors création d'un sous-répertoire `/var/yp/toto` (portant le nom du domaine NIS) contenant les 6 fichiers binaires de permissions 600 : hosts.byname, hosts.byaddr, passwd.byname, passwd.byuid, group.byname et group.bygid
6. Renseigner le fichier de configuration de NIS :
   Éditer le fichier `/etc/ypserv.conf` et indiquer l'adresse IP du réseau comme ci-dessous :

```
# Host          : Map              : Security   : Passwd_mangle
#
10.100.200.     : passwd.byname    : port       : yes
10.100.200.     : passwd.byuid     : port       : yes
```

7. Relancer le serveur : `/etc/rc.d/init.d/ypserv restart`

Le serveur devrait être fonctionnel. Vérifions :

```
# ps ax | grep yp
root  550    .....   ypserv
root  823   ......   rpc.yppasswdd
```

8.2.2 Configuration d'une machine cliente

Du côté client, les services à lancer sont portmap, ypbind :

```
/etc/rc.d/init.d/portmap start
/etc/rc.d/init.d/ypbind start
```

Configuration :

1. Dans `/etc/sysconfig/network`, comme sur le serveur il faut déclarer le nom du domaine en ajoutant la ligne

```
NISDOMAIN=toto
```

2. Éditer `/etc/yp.conf`, et y ajouter les 2 lignes :

```
domain toto server  10.100.200.1
ypserver titi
```

3. Éditer `/etc/nsswitch.conf`, et veillez à la présence des lignes :

```
passwd:     files  nis
group:      files  nis
hosts:      files  nis  dns
```

4. Lancer le service client : `/etc/rc.d/init.d/ypbind start`

Vérifications :

`ps ax |grep yp` ou `ypwhich` doit donner le nom complet de la machine qui héberge le serveur. `ypcat passwd` permet d'afficher la carte des comptes utilisateurs.

`yppasswd` permet à l'utilisateur de changer son mot de passe comme s'il se trouvait à la console du serveur.

9 La virtualisation

D'après Wikipedia, la virtualisation est "*l'ensemble des techniques matérielles et/ou logicielles qui permettent de faire fonctionner sur une seule machine plusieurs systèmes d'exploitation et/ou plusieurs applications, séparément les uns de autres comme s'ils fonctionnaient sur des machines physiques distinctes.*"
Les bases de la virtualisation ont été données par Gerald J. Popek et Robert P. Goldberg en 1974.

9.1 Les différents domaines de la virtualisation

Le mot *virtualisation* est utilisé pour plusieurs domaines et à des niveaux (logiciels, matériels) différents :
- La virtualisation de réseaux : il s'agit de partager une même infrastructure physique au profit de plusieurs réseaux virtuels isolés. On parle généralement de VLAN (Virtual Local Area Network).
- La virtualisation de stockage : il s'agit de séparer la représentation logique et la représentation physique de l'espace de stockage. Une couche logicielle va permettre de regrouper plusieurs espaces de stockage (physiques) pour ensuite découper cet espace en espaces virtuels (partitions logiques).
- La virtualisation d'environnement : elle consiste à encapsuler l'application et son contexte d'exécution système dans un environnement cloisonné. On l'appelle parfois virtualisation d'applications. Exemples : Wine (permet d'exécuter des applications Windows sur une plateforme Linux), un environnement chroot (voir section 6.12).
- La virtualisation de serveurs : elle consiste à masquer les caractéristiques de chaque machine physique. Grâce à un logiciel, un serveur physique va être divisé en plusieurs environnements virtuels isolés les uns des autres. Exemple : VirtualBox, VMware.

Nous nous intéresserons principalement à la virtualisation de serveurs. Il existe deux principales méthodes de virtualisation de serveurs : la virtualisation totale et la paravirtualisation.

Dans la virtualisation totale, il s'agit de faire croire au système d'exploitation qu'il s'exécute sur une machine physique alors que ce n'est pas le cas. Il est souvent dit que, dans ce cas, on utilise un hyperviseur de type 2.

Dans la paravirtualisation, au lieu de chercher à faire croire aux systèmes d'exploitation qu'ils s'exécutent sur une machine physique, il est possible d'adapter le système d'exploitation à la couche de virtualisation. La paravirtualisation vise donc à modifier les systèmes d'exploitation pour communiquer avec un hyperviseur au lieu de communiquer avec une machine physique. On parle alors souvent d'hyperviseur de type 1.

La Figure 5 résume les deux technologies.

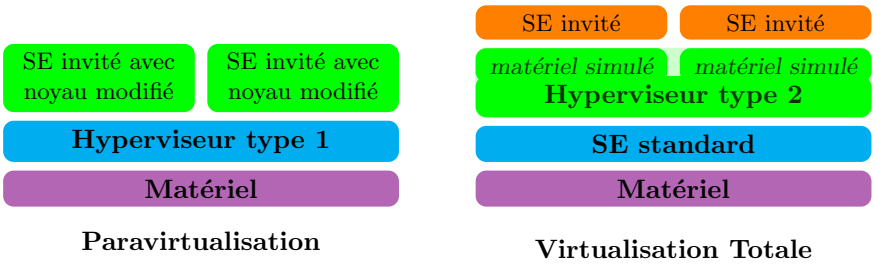


FIGURE 5 – Les deux principaux principes de la virtualisation

Exemples Parmi les logiciels (hyperviseurs) les plus connus, on trouve :
1. Paravirtualisation : Xen, Hyper-V, ESX server (de VMware), ...
2. Virtualisation totale : VMware server, VirtualBox, Microsoft Virtual PC, QEMU, ...

9.2 Principes techniques

Les bases : Le logiciel de virtualisation (souvent également appelé hyperviseur) sert à transformer les instructions du système invité (la machine virtuelle) en instructions pour le système hôte (la machine physique).

Un système d'exploitation doit normalement manipuler le matériel à un niveau très bas. La machine virtuelle émule donc de manière logique (c'est-à-dire avec un programme) tout le matériel habituel de l'architecture de l'ordinateur cible. En pratique, le disque dur de la machine virtuelle est la plupart du temps géré comme un (volumineux) fichier pour le système hôte, alors que la mémoire vive dont le système invité dispose est réservée par le programme de la machine virtuelle. Le reste de l'architecture de l'ordinateur peut varier grandement selon les implémentations, mais on retrouve généralement au moins une carte réseau standard, un clavier 105 touches standard et une carte graphique minimale.

Le système s'exécutant dans la machine virtuelle est un système d'exploitation à part entière, tel qu'on pourrait en installer sur une machine physique : Microsoft Windows, GNU/Linux, Mac OS X, etc. Cette particularité est la caractéristique principale de la virtualisation complète : les systèmes invités n'ont pas à être modifiés pour être utilisés dans une machine virtuelle utilisant une technologie de virtualisation.

Les difficultés : Les problèmes actuels de la virtualisation sont principalement liés à l'omniprésence de l'architecture x86 au sein des ordinateurs. Cette architecture se prête mal à la virtualisation car elle possède 18 instructions dites critiques. Elles sont en effet accessibles à partir d'une application alors qu'elles permettent d'accéder aux ressources physiques.

Il ne faut en aucun cas qu'un système virtuel ait la possibilité de modifier les ressources physiques de la machine. Il est donc du ressort de l'hyperviseur d'intercepter ces instructions critiques de manière à simuler leur comportement.

9.3 Avantages et inconvénients de la virtualisation

La virtualisation a pris une place importante au sein des entreprises/établissements et ce sont principalement les serveurs qui sont concernés. Les principaux **avantages** sont les suivants :

- une optimisation de l'infrastructure : on estime que la charge moyenne d'un serveur est d'environ 10% (selon VMware). Il s'agit de récupérer les ressources restantes pour d'autrees applications.
- une réduction du nombre de machines physiques et des économies d'énergie : économies d'espaces, de frais de ventilation, ...
- une réduction des interruptions de service : sauvegardes plus simple des machines virtuelles,...
- une facilité de migration : sauvegardes plus simple des machines virtuelles,...
- une compatibilité : pas de dépendance entre le serveur virtuel et la machine physique.
- un cloisonnement : permet de faire des tests sans risquer de compromettre d'autres applications.

Il existe également des **inconvénients**, dont certains sont importants :

- une machine physique unique : c'est le point le plus vulnérable, si cette machine physique tombe en panne, ce sont plusieurs serveurs qui sont impactés.
- des performances moindres (suivant la puissance de la machine physique).
- une difficulté supplémentaire pour détecter et résoudre les problèmes, puisque la couche de virtualisation vient s'ajouter aux autres.
- la virtualisation est parfois impossible : par exemple les bases de données ont recours à de nombreux accès disques et le delai d'accès supplémentaire introduit par la couche de virtualisation peut être rédhibitoire.
- suivant les outils de virtualisation, il n'est parfois possible de virtualiser que des systèmes d'exploitation prévus pour la même architecture matérielle que le processeur physique.