Administration Système
Université Libre de Bruxelles - INFO 151

Marco Codutti
Marco-Codutti@tiscali.be

6 mai 2003

Résumé

Ces notes servent de support pour le cours d'*Aministration Système* donné en Licence en Informatique à l' *Université Libre de Bruxelles.* Il s'agit d'une première version ; elle comporte donc bon nombre d'oublis, imprécisions et erreurs. Le livre de référence qui a été utilisé est *Unix, guide de l'administrateur*, E. NEMETH, G. SNYDER, S. SEEBASS ET T.R. HEIN, CampusPress.

Chapitre 1

Introduction

1.1 Les tâches de l'administrateur

Un administrateur système effectue toutes les tâches
- sur un ordinateur
- essentiellement logicielles (parfois un peu matérielles)
- qui demandent une *connaissance technique*
- qui gagnent à être *centralisées*
- qui sont critiques au niveau *sécurité*

Cela comprend :
- Installer les machines et les systèmes
- Ajouter et supprimer des utilisateurs
- Ajouter et supprimer du matériel
- Effectuer des sauvegardes
- Installer de nouveaux logiciels
- Vérifier le système (services en fonction, bonne gestion des ressources, ...)
- Effectuer les réparations diverses (matérielles et logicielles)
- Tenir à jour la documentation
- Evaluer et gérer la sécurité
- Aider les utilisateurs (souvent pour tout et n'importe quoi)

1.2 Les pré-requis

- Faut-il être un bon utilisateur ? **Oui**
  - Pour comprendre les problèmes des utilisateurs et leurs souhaits
  - Un administrateur reste un utilisateur (privilégié certe). Il doit bien connaître son utilisation au quotidien pour accroître son efficacité.

- Faut-il être un bon programmeur ? **Un Peu**
  - Connaître les bases permet de s'en sortir lors d'une compilation/installation qui ne se passe pas comme prévu.
  - On est souvent amenés à écrire soi-même des scripts
- Faut-il être connaître les rouages internes du système ? **Un Peu**
  - Connaître les bases permet de mieux identifier et comprendre le pourquoi de comportements anormaux (lenteurs, ...)

1.3 Les aptitudes d'un bon administrateur

- Administrer une machine est une tâche critique. Une mauvaise manipulation pour empêcher un grand nombre de personnes de travailler, peut faire perdre de l'argent à l'employeur. Il requiert donc une bonne capacité de *gestion du stress*.
- Un administrateur est en interaction avec une grande quantité d'interlocteurs dont aucun n'a le temps d'attendre. Il faut être capable d'appliquer intensement le *multi-tâches*.
- Les interlocuteurs ont une vision locale et très propre du système d'information de l'entreprise. Leurs demandes peuvent être irréalistes, en contradiction avec d'autres, les échéances peuvent être trop courtes. Il faut pouvoir faire le tri, être diplomate pour expliquer l'impossibilité de satisfaire une demande ou l'attente engendrée par des tâches plus prioritaires
- Les utilisateurs n'ont pas toujours conscience du travail qu'effectue un administrateur système. Il faut donc pouvoir accepter qu'on sera peu remercié du travail accompli mais qu'on nous en voudra souvent pour ce qui ne tourne pas correctment. Une bonne communication est alors primordiale.
- Les problèmes ne surgissent pas toujours aux heures normales de travail. Une certaine *souplesse horaire* est requise.
- Un administrateur a des pouvoirs dangereux si ils sont mal utilisés (atteinte à la vie privée, comportement non éthiques). Un administrateur doit garder la tête froide et adopter un comportement sans reproche. Il doit éventuellement pouvoir refuser d'exécuter certains ordres. Un administrateur a des pouvoirs dangereux si ils sont mal utilisés (atteinte à la vie privée, comportement non éthiques).
- Un administrateur ayant une bonne connaissance de l'informatique et son rôle étant souvent mal compris, on peut le solliciter pour tout et n'importe quoi (utilisation de Word par exemple). Il faut pouvoir rendre service sans que cela ne détourne des vraies tâches prioritaires

d'administration pure)

1.4 Systèmes étudiés dans ce cours

- Nous nous atarderons surtout sur `Unix`
- Windows abordé de matière succinte
- On étudiera toutefois l'intégration des 2 systèmes dans un même réseau
- On n'abordera pas d'autres systèmes comme `OS/400`, l'OS des mainframes IBM ou celui du Mac.

1.5 L'histoire d'`Unix`

Pas de standard pour `Unix`. Existence de plusieurs `Unix` qui se ressemblent plus ou moins.

1969 Naissance dans les laboratoires `AT&T`

1976 Distribué gratuitement aux Universités

1977 Naissance de la version `BSD` de Berkeley (diverge de la version `SysV`)

1991 `Linux` créé par Linus Torvald

1.6 Présentation de `Solaris`

- Système d'exploitation de Sun Microsystems
- En fait, package incluant `SunOS`(un peu comme Linux et une distribution)
- `SunOS` 3 & 4 (`Solaris` 1) étaient très `BSD`
- `SunOS` 5 (`Solaris` 2 à 8) est très `SysV`

1.7 Présentation de `Linux`

- Implémentation gratuite du système `Unix`
- Crée par L. Torvald à partir d'un travail de A. Tanembaum
- Désigne à proprement parler le noyau
- Une distribution `Linux` inclut un noyau et une série de logiciels ; le tout intégrés correctement.

1.8 Les sources d'informations

1.8.1 Les livres

- *Unix, Guide de l'administration*, Evi Nemeth, Garth Snyder, Scott Seebass et Trent R. Hein

1.8.2 L'aide locale

- Les *pages man*
- Les *pages info*
- Les *How-to*
- Les systèmes propriétaires

1.8.3 Les sources Internet

- Les sites dédicacés
- Les *news*

Ces sources peuvent
- être dédiées à un OS
- ou à une distribution
- parler d'administration de plusieurs systèmes

Première partie

Unix isolé

Chapitre 2

Les pouvoirs du *root*

Chaque utilisateur est identifié par un nom (*login name*, *username*). Chaque fichier et chaque processus est rattaché à un utilisateur. Il existe un utilisateur spécial (le *root* ou *superutilisateur*). Cet utilisateur a plus de pouvoir que les autres. L'ensemble des utilisateurs permis sur un système sont définis dans le fichier `/etc/passwd`. C'est là la base de la sécurité sous Unix.

2.1 Droit d'accès aux fichiers et aux processus

Chaque fichier possède à la fois un *propriétaire* et un *goupe de propriétaires*. Le propriétaire est le seul (à part *root*) capable de modifier les permissions associées à un fichier. Un groupe d'utilisateurs peut, évidemment, comporter plusieurs utilisateurs et un utilisateur pourra faire partie de plusieurs groupes. Les groupes existants sur une machine sont définis par le fichier `/etc/group`. En fait, au niveau interne, les utilisateurs et les groupes sont des nombres (le UID et le GID). Ils n'apparaissent sous forme de texte que lors du dialogue avec un utilisateur. La correspondance numéro-nom est explicitée dans le fichier `/etc/passwd`.

Les processus lancés par un utilisateur s'exécutent normalement avec les permissions associées à l'utilisateur. Celà peut être modifié grâce au mécanisme du *setuid*. Nous verrons cela en détail plus tard.

2.2 Le superutilisateur

Il s'agit du compte ayant le UID 0. Son nom est traditionellement *root* et ne devrait pas être changé. Le superutilisateur peut se faire passer pour n'importe quel autre uilisateur. Il peut donc effectuer toute opération valide sur les fichiers et les processus. Il est également le seul à pouvoir exécuter certaines commandes comme :

- modifier la racine d'un processus
- créer des fichiers périphériques
- modifier l'horloge système
- manipuler les quotas et les priorités
- modifier le nom d'une machine et la configuration des interfaces réseau.

2.3 Choisir son mot de passe

Plus que quiconque, le superutilisateur doit veiller à bien choisir son mot de passe. Un mot de passe doit contenir au moins 8 caractères. Il doit être suffisament compliqué que pour ne pas être découvert facilement mais aussi suffisament mnémotechnique que pour ne pas devoir être écrit. Des programmes spécialisés tentent de découvrir un mot de passe par essais à partir de *dictionnaires* et de règles modifiant ces mots. Tout bon mot de passe devrait contenir à la fois des chiffres, des lettres et des signes de ponctuation.

Une première approche est de transformer un mot existant en

- ajoutant des chiffres
- mélangant les majuscules et les minuscules
- modifiant certaines lettres par des signes de ponctuation

Exemple 2.1 `vanessa` *deviendrait* `V@neSSa46`.

Mais les processeurs devenant plus puissants, les programmes de recherche de mot de passe peuvent se permettre d'essayer plus de combinaisons encore. Actuellement, on recommande de condenser une phrase que vous retenez facilement.

Exemple 2.2 `La nuit est Rousse sur la baie de Cabourg` *deviendrait* `LneR/lb2C`.

Changez-le dès qu'un doute s'installe sur son intégrité.

2.4 Devenir root

Le compte *root* étant un véritable compte, on peut se connecter en tant que super-utilisateur comme on le fait pour un utilisateur normal. Cela n'est tout de fois pas recommandé car on ne sait pas alors qui s'est vraiment connecté (cas de plusieurs administrateurs, cas d'un intrus). Une meilleure approche est d'utiliser la commande `su`

2.4.1 `su`

Vous devez vous connecter en tant qu'utilisateur normal. Cette commande permet alors de passer en root. La différence est qu'une indication est reprise dans le fichier journal. On sait alors qui est devenu root. Sur certains systèmes, il existe l'option `-` qui permet d'exécuter les fichiers de configuration pour un evironnement adéquat. Cette commande permet également de devenir n'importe quel uilisateur si on connait son mot de passe. Le root peut devenir n'importe quel utilisateur sans devoir spécifier de mot de passe. Cela peut s'avérer pratique pour bien comprendre un éventuel problème rencontré par un utilisateur.

On recommande de taper le chemin explicite de la commande.

2.4.2 `sudo`

Cette variante permet d'exécuter une seule commande avec les pouvoirs du root. On la retrouve de plus en plus dans les systèmes Unix. Un fichier de configuration (`/etc/sudoers`) permet de définir qui peut faire quoi et sur quel machine.

Les immenses avantages sont :

- Les commandes exécutées sont enregistrées dans un journal
- On peut déléguer des responsabilités à discrétion
- Le mot de passe du root n'est plus connu que par peu de personnes
- On peut supprimer des privilèges à quelqu'un sans devoir changer le mot de passe du root
- Le fichier de configuration est centralisé
- Il reprent clairement qui peut faire quoi sur le système.
- On minimise la possiblité de laisser un session root ouverte

Il y a néanmoins des inconvénients

- Les utilisateurs à qui on a délégué un peu de responsabilité doivent veiller à la sécurité de leur propre compte
- Le plus grand soin doit être apporté au fichier de configuration sinon c'est la porte ouverte aux failles. (exemple : `sudo csh`)

2.5 Les autres pseudo-utilisateurs

Un examen du fichier `/etc/passwd` montre des utilisateurs prédéfinis qui ne correspondent pas à des personnes physiques. Il s'agit de pseudo utilisateurs qui sont reconnus par le noyau avec un statut particulier. Souvent d'ailleurs, leur mot de passe esr définit de telle façon que personne ne puisse les utiliser. On peut trouver

daemon Le proriétaire de logiciels systèmes ne nécessitant aucun privilège.

bin De plus en plus rare. Propriétaire des dossiers possédant les commandes.

nobody Utilisateur sans pouvoir utilisé pour les root distants dans le cas de partage réseau de fichiers.

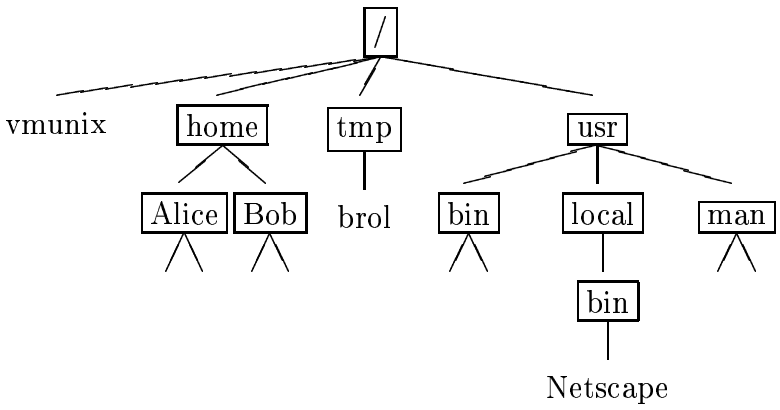
Chapitre 3

Le système de fichiers

Au départ, le système de fichiers (*filesystem* en anglais) est le concept par lequel on accède à toutes les informations stockées sur un support permanent (disques, ...). Cela comprend bien sûr tous les fichiers d'informations (textes, programmes, ...). Mais il a également été étendu pour permettre d'accéder à d'autres élements du système comme des *pseudo-fichiers* qui vont faire le lien avec les composants hardwares.

3.1 Structure

Le filesystem a la structure suivante



On peut le voir comme un arbre dont les noeuds sont des directory et les feuilles des *fichiers* au sens large (cela peut être des directory vides ou des fichiers spéciaux).

Un fichier du filesystem est identifié par le chemin à parcourir à partir de la racine.

Exemple 3.1 *On accèdera au fichier* `netscape` *via* `/usr/local/bin/netscape`.

Il est également possible de spécifier le chemin relativement à une *directory courante*.

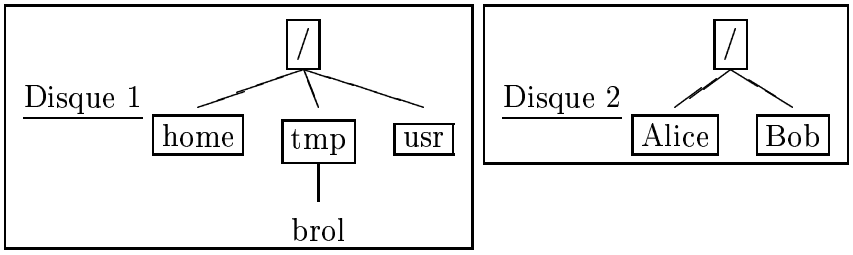
Exemple 3.2 *Si la directory courante est* `/home`*, on peut se contenter de* `Bob`.

Il existe peu de contraintes sur les noms des fichiers et des dossiers.
- Il y a une limite sur les tailles mais elle est grande.
- Un nom ne peut contenir le caractère `/`
- Par contre même les espaces peut être utilisé ce qui peut poser des problèmes à certaines commandes de par la façon dont le shell les interprète.

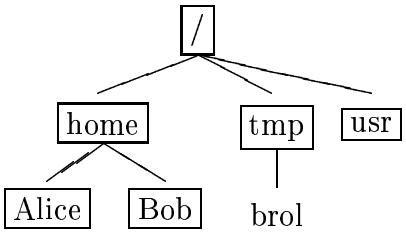
3.2 Les partitions

Si d'un point de vue logique il n'existe qu'un filesystem, physiquement il en va autrement. Une machine peut posséder plusieurs disques durs, un lecteur de CD-ROM, un lecteur de disquettes. Tous ces supports d'informations conteniennent un filesystem. Tous ces filesystems vont être regroupés pour n'en former qu'un seul.

Exemple: soit 2 disques durs contenant les filesystems suivants



Ces deux filesystems sont *montés* en un seul lors du démarrage de la machine pour donner



Notons que le disque 1 doit contenir une directory `home` pour que le deuxième puisse s'y raccrocher. Notons également que ce deuxième commence par `/` comme tout filesystem. Il n'y a aucune référence à `home` et il pourrait d'ailleurs être monté ailleurs. La manière dont les filesystems sont combinés est déterminée par un fichier de configuration que nous verrons à une autre occasion.

Notons enfin que tout disque dur peut être divisé en *partitions* qui sont des disques durs logiques (max. 8) au sein d'un même disque dur. Ce partitionnement aura pour intérêt de séparer des fichiers qui ont des objectifs différents. Nous verrons cela lorsque nous parlerons de l'installation d'un OS.

3.3 Monter et démonter des partitions

La partition principale est choisie au moment du démarrage. Il s'agit de celle contenant le noyau. Pour le reste, les partitions peuvent être ajoutées à volonté à la hiérarchie (via `mount`) et enlevées (via `umount`)

3.3.1 mount

Cette commande permet de *monter* un système de fichiers à la hiérarchie globale. La syntaxe de base est `mount from to`.

Exemple 3.3 *`mount /dev/sd1c /home` permet de monter le filesystem se trouvant sur la troisième partition du premier disque SCSI comme dossier des fichiers utilisateurs.*

3.3.2 liste des systèmes de fichiers

Lors du démarrage, tous les systèmes de fichiers sont automatiquement montés pour créer la hiérarchie globale par défaut. Les informations pour ce faire sont reprises dans un fichier (`/etc/fstab`, `/etc/vfstab` ou encore `/etc/checklist`)

Ce fichier permet également aux administrateurs d'avoir une vue globale de l'organisation des fichiers. Enfin, il permet certains raccourcis dans les commandes (comme `mount`)

Exemple 3.4 *Voici un exemple de fichier `/etc/fstab` sur `Linux`.*

```
/dev/hda6 / ext3 defaults 1 1
/dev/hda3 /home ext3 defaults 1 2
/dev/hda1 /mnt/nt ntfs iocharset=iso8859-15,ro,umask=0 0 0
/dev/hda7 swap swap defaults 0 0
```

On indique
- La localisation physique du système de fichiers
- Le point de montage
- Le type de système de fichiers
- Les options de montage
- Une indication de la nécessité de sauvegarder ce système de fichiers (peu utilisé)
- L'ordre de vérification des systèmes de fichiers (ou la non vérification)

Il existe encore d'autres options liées à d'autres systèmes de fichiers et d'autres façon de monter. Nous verrons cela plus loin.

3.3.3 umount

Cette commande permet de détacher un système de fichiers de la hiérarchie. Le dossier caché redevient accessible. La syntaxe est `umount dossier`.

3.3.4 Et si on ne peut démonter

On ne pourra démonter un filesystem si il est *occupé* pour l'instant. Cela sera le cas si il contient :
- Un fichier ouvert par un processus
- Le répertoire courant d'un processus
- Le répertoire racine d'un processus
- Un programme en cours d'exécution
- Un fichier en mémoire (bibliothèque partagée par exemple)

Pour vérifier cela, on dispose de deux commandes : `fuser` et `lsof`

3.3.5 fuser

Cette commande est disponible sur de nombreux système mais avec des options différentes. Elle permet d'identifier les processus utilisant un fichier ou un système de fichiers.

Exemple 3.5 *Voici le résultat de la commande* `fuser -mv /home` *sur* `Linux` *(extrait)*

```
                     USER        PID ACCESS COMMAND
/home                marco      2305 f.c..  startkde
                     marco      2485 f.c..  kdeinit
                     marco      2504 f.c..  artsd
                     marco      2507 f.c..  kwrapper
                     root       2533 f.c..  kdesud
```

```
marco      2535 f.c..  kalarmd
marco      2536 f.c..  evolution
marco      2541 f....  wombat
marco      2577 f....  evolution-mail
marco      2581 f....  gconfd-2
marco      2592 f.c..  kile
marco      2594 ..c..  bash
root       2710 ..c..  su
marco      2858 f....  xmms
```

3.3.6 lsof

Il s'agit d'un utilitaire gratuit disponible par défaut sur certains systèmes. Cette commande fonctionne à contrario de la précédente et donne les fichiers utilisés par un processus.

Exemple 3.6 *Sur Linux :*

```
[root@localhost /]# lsof -p 1
COMMAND PID USER   FD   TYPE DEVICE    SIZE   NODE NAME
init      1 root  cwd    DIR    3,6    4096      2 /
init      1 root  rtd    DIR    3,6    4096      2 /
init      1 root  txt    REG    3,6   31384 701828 /sbin/init
init      1 root  mem    REG    3,6  539887 750724 /lib/ld-2.2.5.so
init      1 root  mem    REG    3,6 1167240 962882 /lib/i686/libc-2.2.5.so
init      1 root   10u  FIFO    0,6            733 /dev/initctl
```

Détaillons certaines colonnes.

FD Utilisation du fichier

- **cwd** Dossier courant
- **rtd** Dossier *racine*
- **txt** Fichier texte
- **mem** Fichier en mémoire
- **numéro** Numéro du descripteur de fichier
- **r,w,u** Accès en lecture, écriture ou les deux

TYPE Type de fichier (dossier, régulier, socket, ...). Il en existe des dizaines

3.4 Vérifier l'occupation des partitions

Voyons à présent deux commandes qui permettent de vérifier l'occupation des disques.

La commande `df` (`df -k` sur `AT&T`) donne l'occupation des différentes partitions.

Exemple:

```
mcodutti@cso8:df
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/dsk/c0t0d0s0      19047   13032    4115    77%    /
/dev/dsk/c0t0d0s6     480919  344046   88783    80%    /usr
/dev/dsk/c0t0d0s3      76967   23926   45351    35%    /var
/dev/dsk/c0t0d0s5    1085262  851078  125664    88%    /opt
```

Examinons le résultat de cette commande (certaines lignes ont été supprimées pour simplifier notre propos).

Filesystem fichier spécial du file system relié à une partition physique. Nous verrons à un autre moment comment lire cela. Ici, nous avons les partitions 0, 6, 3 et 5 du disque 0.

kbytes l'espace total disponible sur la partition

used l'espace actuellement occupé

avail l'espace disponible

capacity l'occupation exprimée en pourcentage

Mounted on l'endroit où cette partition est montée sur le filesystem

Vous aurez remarqué que `used`+`avail` $\neq$ `kbytes`. Un certain pourcentage d'une partition (souvent 10%) est gardé en réserve. Il pourra être utilisé mais la capacité indiquera 100% avant et des messages d'alerte apparaîtront.

Une commande complémentaire est `du`, elle donne l'espace total occupé par une directory.

Exemple:

```
root@lit1:du -s /var/*
5000    /var/adm
790     /var/mail
345     /var/spool
1259    /var/tmp
```

La taille est souvent exprimée en Kilobytes mais parfois aussi en 1/2 Kbytes (surtout `AT&T`)

3.5 Types de fichiers

Après avoir étudié le filesystem dans son ensemble, voyons à présent le type de fichiers que l'on peut y rencontrer. Officiellement, il en existe 7.

Dossier

Fichier ordinaire. On entend par là la plupart des fichiers. Aussi bien les textes, les exécutables, les inputs et outputs de programmes, ...

Fichier de périphérique en mode caractère ou bloc. Il s'agit de pseudo-fichiers qui sont, en fait, un lien vers un périphérique (Disque, terminal, sortie parallèle, ...). Ils sont tous dans la directory `/dev` ou `/devices`.

Lien symbolique. Comme le lien physique, il permet de donner plusieurs noms à un même fichier et, par extension, de rencontrer un même fichier à plusieurs endroits dans le filesystem. On le distinguera du lien physique.

Tubes nommés et sockets. Peu utile pour l'administrateur.

3.5.1 Directory

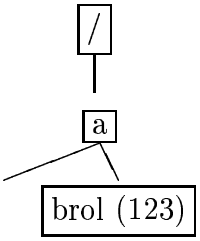
Il faut savoir que, dans le système, un fichier est identifié par un *inode*, numéro **unique** dans la partition. Une directory est un fichier contenant une table associant un nom à un inode

Exemple:

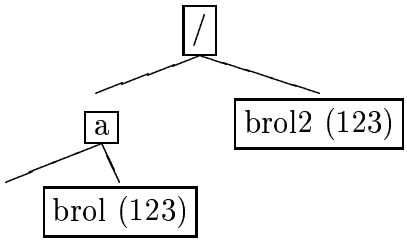
Nom	inode
`.`	123 *(directory courante)*
`..`	126 *(directory parent)*
`vmunix`	345
`etc`	12
`bin`	7

3.5.2 Liens physiques

Prenons comme exemple le file system suivant



où le nombre entre parenthèses indique le inode associé au fichier. La commande `ln /a/brol /brol2` aura pour effet de créer un fichier pointant vers le même fichier physique.

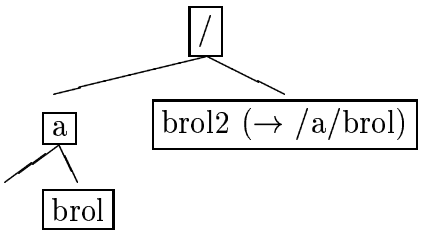


A partir de là, toute action sur `brol2` sera répercutée sur `brol`. Si on supprime `brol`, `brol2` existera toujours. Pourquoi ? Un fichier physique a un compteur reprenant le nombre de références dans le filesystem. Effacer un fichier dans le filesystem revient à *décrémenter* le compteur. Ce n'est que lorsqu'il tombe à 0 que le fichier est physiquement effacé.

Attention : ce mécanisme ne fonctionne qu'à l'intérieur d'une même partition et ne permet pas de créer des liens entres des fichiers de partitions différentes (ce que peuvent faire les liens symboliques).

3.5.3 Liens symboliques

Reprenons l'exemple précédent. La commande `ln -s /a/brol /brol2` va créer un fichier ASCII `/brol2` dont le contenu sera le fichier pointé `/a/brol`



Lorsqu'une action sera intentée sur `/brol2`, le système constatera qu'il s'agit d'un lien symbolique, lira le chemin du fichier pointé et répercutera l'opération vers celui-ci.

Ce genre de liens peut exister même si le fichier pointé n'existe pas encore ou plus et, surtout, il fonctionne **entre les partitions**. Ces liens permettent de dissocier l'emplacement physique d'un fichier de son emplacement logique.

Exemple 3.7 *Vous décidez d'installer tous les fichiers associés au logiciel Netscape dans une même directory pour ne pas les éparpiller. Mais pour que l'exécutable soit trouvé par le système, il doit se situer dans une certaine autre directory. Pas de problème, il suffit de créer un lien symbolique de la directory des exécutables vers la directory netscape.*

Attention : On peut spécifier un chemin relatif pour le fichier pointé. On entend alors relatif par rapport à l'endroit du fichier pas par rapport au dossier courant au moment de la création du lien.

Exemple 3.8 *La commande* `cd /a ; ln -s ./brol /brol2` *n'a pas le même effet que la version sans chemin relatif vue précédemment. Elle devrait plutôt s'écrire* `cd /a ; ln -s ./a/brol /brol2`

3.6 Les permissions

Chaque fichier est possédé par un *propriétaire*, appartient à un *groupe* et est associé à des *permissions* représentées par 12 bits. En voici leur description

r,w,x read, write et execute. Il existe 3 triplets. Un pour le propriétaire, un pour le groupe et un pour le reste des utilisateurs.

setuid modifie les permissions d'un exécutable

setgid modifie le groupe d'un exécutable en action

sticky l'ancien sens de ce bit devient obsolète et les différents OS l'ont récupéré pour des usages divers. En `Solaris`, par exemple, il permet de créer une certaine privacité dans des directory autrefois publiques comme `/tmp`

3.6.1 Effets de r,w et x

Le sens de ces permissions est différents pour les fichiers normaux et les directory

- Pour un fichier
  - **r** permet de lire le fichier
  - **w** permet d'écrire (modifier, append, tronquer)
  - **x** indique qu'il s'agit d'un programme (binaire). Peut être exécuté.
- Pour une directory
  - **r** permet de voir le contenu

w permet d'écrire dans la directory (ajouter, effacer, renommer un fichier)

x permet de traverser la directory

Voici quelques exemples pour mieux comprendre les liens entre les permissions et les opérations permises. Supposons que dans la directory où nous sommes, existe une directory `brol` contenant le fichier `brol2`. Dans le tableau nous indiquons diverses opérations avec les permissions minimales pour que cette opération puisse se réaliser.

brol	brol2	action
r		`ls brol`
x		`cd brol`
x	r	`cat brol/brol2`
wx		`rm brol/brol2` (demande une confirmation si brol2 n'a pas w)
x	w	modifier `brol/brol2`
wx		`touch brol/brol3`
x	w	`cat brol/brol3 >brol/brol2`
wx		`mv brol/brol3 brol/brol2`

Le résultat le moins intuitif est que l'on peut supprimer un fichier qui ne possède pas de permission en écriture s'il se trouve dans une directory qui, elle, possède cette permission.

3.6.2 Cas des scripts

Un script est un cours programme sous forme texte qui sera interpreté par le programme adéquat. On trouve en `Unix` beaucoup de scripts *Shell* mais également des scripts *Perl*, *awk*, ...

La première ligne de ce fichier doit indiquer l'interpréteur à utiliser. Le format est `#!programme` (`#!/usr/bin/perl` par exemple). L'absence de cette ligne indique un script *shell* standard.

Pour être exécuté, ce script doit posséder à la fois les bits `x` et `r`.

3.6.3 Le setuid

Prenons un exemple pour comprendre le fonctionnement de ce bit. Supposons qu'il existe un programme `Nettoye` appartenant à l'utilisateur `untel` tel que `Nettoye dir` détruise la directory `dir`. Supposons, en outre, qu'il existe une directory `brol` appartenant à ce même `untel` et que lui seul peut la détruire.

Si l'utilisateur `telautre` tape `Nettoye brol`, il se verra refuser l'opération car le programme va s'exécuter avec les permissions de `telautre`. Sauf si le *setuid* bit est mis, auquel cas il va s'exécuter avec les permissions de `untel`, le propriétaire du fichier `Nettoye`, et la destruction sera acceptée.

3.7 Quelques commandes pour manipuler les permissions

Voyons comment lire et modifier les permissions d'un fichier ainsi que le propriétaire et le groupe et comment définir les permissions par défaut.

3.7.1 La commande `ls`

```
mcodutti@cso8:ls -l
total 135
drwx------   2 root   other       512 Jan 10  1996 Mail/
-rw-r--r--   1 root   other     42932 Dec 13 09:33 core
```

Avant de décrire comment sont affichées les permissions, voyons les autres informations données par la commande `ls`.

- Le nombre en deuxième position indique le nombre de références vers le fichier physique (cf liens physiques),
- vient ensuite le proriétaire du fichier et le groupe auquel il appartient,
- nous avons ensuite la taille (en bytes) du fichier,
- puis la date,
- et enfin, le nom du fichier (suivi d'un `/` dans le cas d'une directory).

Les permisssions, quant à elles, sont représentées par 10 caractères dont la signification est la suivante :

Le premier indique le type de fichier

- pour un fichier normal

d pour une directory

l pour un lien symbolique

c pour un character device file

b pour un block device file

Les suivants indiquent les permissions (r,w et x) pour le propriétaire, le groupe et le reste du monde (dans cet ordre). Le troisième caractère (`x`) peut toutefois apparaître différemment pour indiquer d'autres permissions

- Le bit `x` du propriétaire peut apparaître comme un `s` pour indiquer que le *setuid* est mis (`S` si le `x` n'est pas mis)

- Le bit `x` du groupe suit la même modification pour le *setgid*
- Le bit `x` du reste du monde apparaît comme un `t` si le *sticky bit* est mis (à nouveau, `T` si le `x` n'est pas mis)

3.7.2 Modifier les permissions

Les permissions associées à un fichier peuvent être modifiées via la commande `chmod perm file` où `file` est le fichier en question et `perm` indique les permissions. Celles-ci peuvent être indiquées de 2 façons :

numériquement. Notation octale des 12 bits de permissions dans l'ordre : setuid, setgid, sticky, r owner, w owner, x owner, r group, w group, x group, r world, w world, x world.

Exemple: 4755 indique `rwxr-xr-x` + setuid = `rwsr-xr-x`

symboliquement par l'expression `who op perm` où

who est `u` (owner), `g` (group), `o` (world) ou `a` (all)

op est + pour ajouter une permission, - pour en enlever ou = pour mettre exactement des permissions (idem approche numérique)

perm est `r, w, x, s` ou `t` ou encore `u, g, o` pour reprendre les permissions d'une autre classe

3.7.3 Modifier le propriétaire et le groupe

Cela se fait via les commandes

```
chown owner file
chgrp group file
chown owner:group file (sur certains systèmes)
```

Attention : Cette commande est généralement réservée au `root` même si certains OS permettent également que le propriétaire actuel d'un fichier les utilise. Ce qui est dangereux. Cela permet par exemple de contourner le système de quotas.

3.7.4 Permissions par défaut

La commande `umask` spécifie les bits qui sont mis à 0 lors de la création d'un fichier.

Exemple: Avec la commande `umask 022` une directory, normalement créée avec les permissions 777, aura 755 et un fichier, normalement créé avec les permissions 666, aura 644.

Ce masque est généralement défini dans le fichier d'initialisation du shell.

3.8 Organisation standard du système de fichiers

La structure arborescente permet d'organiser les fichiers pour pouvoir facilement les retrouver et comprendre leur sens. Elle est également utile pour l'utilisaiton de *partitions*. Cette organisation n'est toutefois pas facile et, de par l'évolution de l'informatique et du système `Unix`, ce qui était une bonne organisation ne l'est plus forçément. Chaque dialecte a fait ses choix ce qui fait qu'on peut se sentir à l'aise sur un système et ne plus retrouver les fichiers sur un autre.

Afin de garder une certaine homogénéité à travers les systèmes, le *Filesystem Hierarchy Standard Group* s'est créé. Il s'est donné pour but de standardisé l'organisation des fichiers sous Unix. Nous vous recommandons la lecture du document *Filesystem Hierarchy Standard*. Nous donnons ici l'essentiel de ce document

3.8.1 Les catégories de fichiers

Ce groupe a d'abord introduit deux dichotomies. D'une part celle distinguant les fichiers partageables et non partageables. Ensuite celle entre les fichiers statiques et variables.

Les fichiers **partageables** sont ceux qui peuvent être utilisés par plusieurs machines en même temps

Les fichiers **variables** sont ceux dont le contenu ainsi que la taille peut fortement varier au cours de la vie du système.

On obtient ainsi quatre catégories de fichiers. Voici un exemple de chacun d'entre-eux.

	partageable	non partageable
statique	`/usr`	`/etc`
variable	`/var/mail`	`/var/lock`

Cette distinction a son importance. Elle va faciliter le partage d'informations entre les machines.

3.8.2 Les dossiers de premier niveau

La base de la hiérarchie ne devrait contenir que ce qui est nécessaire au démarrage du système et à sa réparation. Certains de ses sous-dossiers seront montés d'un autre système de fichiers. On y rencontre essentiellement les dossiers suivants :

`/bin` Les exécutables essentiels qui peuvent être utilsés quand le système de fichiers n'est pas encore complétement monté.

`/boot` Contient ce qui est nécessaire pour charger le noyau.

`/dev` Contient les fichiers liés au périphériques.

`/etc` Contient les fichiers de configuration propres à la machine.

`/home` Les fichiers des utilisateurs.

`/lib` Les librairies essentielles.

`/mnt` Point de montage temporaire.

`/opt` Emplacement pour les logiciels non système. Ce dossier a été introduit il y a quelques années mais ne s'est pas imposé partout. Certains continuent d'utiliser `/usr/local`. D'autres répartissent les fichiers d'une logiciel à travers le système et se basent sur des système comme *RPM* pour s'y retrouver.

`/root` Les fichiers personnels du *root*. Devrait contenir un minimum d'informations.

`/sbin` Les binaires réservés à l'administrateur et utile lors du démarrage.

`/tmp` Les fichiers temporaires.

`/usr` Section importante du système de fichier. Construit pour être statique et partageable même si ce n'est pas encore toujours le cas. On va y retrouver une structure proche de celle de la racine. Nous la détaillons plus loin.

`/var` Section des fichiers variables. Il s'agit d'une introduction assez récente et pas encore toujours bien respectée.

3.8.3 La structure de `/usr`

Historiquement, les fichiers étaient séparés entre la racine et ce dossier pour des raisons d'efficacité. Un petit dique rapide était consacré aux fichiers essentiels. Les autres se retrouvaient dans ce dossier sur un disque plus grand et moins rapide. Aujourd'hui, cette distinction a disparu. Ce dossier devrait servir de base à tous les fichiers partageables et statiques. On y trouve

`/usr/bin` Les exécutables non essentiels qui peuvent n'être disponibles qu'une fois le système en fonctionnement.

`/usr/include` Les fichiers *include* nécessaires à la compilation. Il faut se rappeler que `Unix`a été écrit en C.

`/usr/lib` Les librairies non essentielles.

`/usr/local` Son rôle n'est pas toujours très clair. On devrait y retrouver tout ce qui est local, c'est à dire ce qui est développé et/ou installé localement par l'administrateur : les petits outils, les scripts propres au site, ...On peut aussi y mettre tous les logiciels installés sur la machine.

`/usr/sbin` Les binaires réservés à l'administrateur mais pas nécessaires au démarrage.

`/usr/share` Tout ce qui est indépendant de l'architecture (mais pas de l'OS précis ni de sa version). On y trouvera par exemple les manuels.

`/usr/src` On y trouve tous les sources, notamment ceux du système nécessaires si on veut le recompiler.

Chapitre 4

La documentation

Historiquement, la documentation *en ligne* sous `Unix` se présentait sous la forme de *man pages*. Il s'agit de textes courts et peu formattés accessibles directement via un nom. Ce système reste très présent car il permet un accès rapide à l'information de base. Plus tard, les *info pages* ont été introduites. D'accès moins rapide, elles offrent néanmoins un début d'*hypertexte* et permettent ainsi une documentation plus riche. Dans les années 90, Sun a introduit un système plus graphique, l'*answerbook*. Au départ basé sur *Postscript*, il s'est adapté à l'évolution technologique et est à présent construit sur *SGML* automatiquement converti en *HTML*. De nos jours, la documentation peut venir sous de nombreuses formes mais sera le plus souvent en *HTML* ou en *PDF*. Sous `Linux`, on la trouvera dans le dossier `/usr/share/doc`.

4.1 La pages de manuel

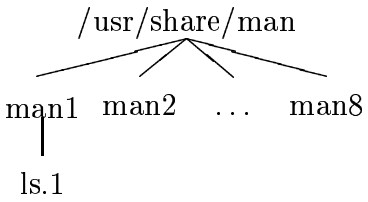
Conçues à la crétion d'`Unix`, ces pages de manuels sont des textes utilisant le format *NROFF* qui permet une mise en page élémentaire (gras, italique, ...). Lorsqu'on y fait appel, elle sont formatées pour obtenir un texte pur et présentée comme on si on avait fait un `more` d'un fichier. On y accède via la commande `man`

Exemple 4.1 *La commande* `man ls` *fournit un mini manuel sur le fonctionnement de la commande* `ls`

Elles ont l'immense avantage de nécessiter peu de ressources pour être visualisées ce qui permet d'y avoir accès sur la console, avec un système déficiant. Elle respectent également une mise en page standard ce qui permet de rapidement trouver l'information.

Son organisation de base a peu évolué depuis sa création. Tout au plus, son emplacement s'est adapté aux nouveaux standard d'organisation et elles se sont internationalisées.

L'organisation de base est la suivante



Ces pages sont réparties dans divers dossiers en fonction du type d'information dispensée. Chaque page de manuel est reprise dans un fichier séparé dont le nom est le nom qu'il faudra donner pour accéder à la page. L'extension est identique à celle reprise dans le nom de dossier.

man1 : les commandes utilisateurs

man2 : les appels systèmes

man3 : fonctions de la librairie standard

man4 : fichiers spéciaux comme les gestionnaires de périphériques

man5 : les formats de fichier

man6 : les jeux

man7 : inclassables

man8 : administration système

manl : manuels locaux

Quelques remarques

- De légères variations par rapport à la liste ci-dessus existent sur chaque système.
- Certains systèmes possèdent une facilité de *cache*. Une page formatée sera sauvée dans un dossier connu pour être récupérée lors de la prochaine invocatin, évitant ainsi de devoir reformater le texte. Ce dossier est souvent *cat* suivi d'un numéro et se trouvera proche des dossiers *man*.
- Souvent aussi, les pages sont compressées et décompressées au vol lors de leur affichage.
- Des pages de manuel peuvent se trouver ailleurs, et même à plusieurs endroits différents. La commande `man` ira explorer chacun des dossiers.
- Son comportement est configuré via le fichier `/etc/man.config`
- La variable d'environnement `MANPATH` permet également d'indiquer les dossiers de recherche de pages.

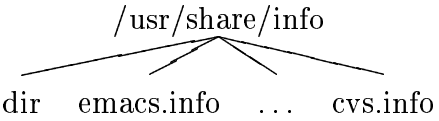
– Lorsque les pages existent en plusieurs langues, le dossier principal est composé d'un dossier par langue. La configuration du système (ou une variable d'environnement) indique alors laquelle aller chercher.

4.2 La pages d'infos

Vous serez peu souvent confrontés à ce système qui tombe en désuétude. Toutefois, certains logiciels, toujours fort utilisés, comme *Emacs* ou *Tex* ont fournit une documentation importante sous ce format. On y accède via la commande `info`

Exemple 4.2 *La commande* `info emacs` *fournit un mini manuel sur le fonctionnement de l'éditeur* `emacs`

L'organisation de base est la suivante



Quelques remarques
– Si on tape juste `info`, le système recherche et affiche le fichier `dir` qui est censé contenir une liste des infos proposées.
– Le format des fichiers est propre au système *info* et contient des hyperliens pour voyager d'une aide à l'autre.
– Une série de touches permet de naviguer à travers le texte (descendre, monter, suivre un lien, revenir en arrière, ...)
– On pourra trouver des versions compressées qui sont décompressées au vol.

Chapitre 5

Les utilisateurs

Dans ce chapitre, nous voyons comment ajouter ou supprimer des utilisateurs dans le système. Créer un compte pour un utilisateur implique 3 étapes

1. L'enregistrer dans le système (login, passwd, ...)
2. Lui attribuer une *home_directory*, espace réservé sur le disque dur.
3. Configurer son compte.

Comment ? Comme souvent, cela peut se faire via un outil graphique, via une commande (`useradd`) ou en manipulant les fichiers. Ici, nous décrivons comment le faire au niveau le plus bas. Nous introduisons tous les concepts qui permettent de comprendre instantanément la méthode graphique.

5.1 Enregistrer un utilisateur

Un utilisateur est caractérisé par une ligne dans le fichier `/etc/password`. Voici sa structure

```
login:passwd:uid:gid:comment:home:shell
```

et sa signification

login. Il s'agit du nom de l'utilisateur. Le règle de base indique qu'il doit s'agir d'un alphanumérique de 8 caractères maximum. De nos jours, certains systèmes acceptent des noms plus longs ainsi qu'une plus grande variété de caractères. On conseille toutefois de s'en tenir aux règles anciennes pour une plus grande compatibilité lors de la mis en réseau. Pour cette même raison, on évitera les majuscules.

passwd. Il apparaît dans ce fichier sous forme codée. Nous donnons plus d'explication dans la section suivante du document.

uid. Il s'agit d'un numéro **unique** de l'utilisateur. Il est compris entre 0 et un maximum dépendant du système. On recommande de rester en dessous de 32767 pour être sûr. Les premiers numéros sont utilisés par le système mais aucune norme n'existe. Pour ne pas interférer avec ceux-ci, on commencera à 100 pour les utilisateurs normaux.

gid. Numéro de groupe. Chaque utilisateur appartient à un groupe principal. Il pourra également appartenir à des groupes secondaires. Cette notion de groupe interviendra au niveau des permissions sur les fichiers.

comment. Nom complet de l'utilisateur. Appelé encore champ *GECOS*. Nous donnons plus d'explications plus loin.

home. Chemin complet de la directory attribuée à l'utilisateur

shell. Chemin complet du shell, le programme qui interagit avec l'utilisateur et qui permet de taper des commandes (`csh, sh, bash, tcsh, ...`).

Pour éditer ce fichier, il est recommandé d'utiliser la commande `vipw`, au lieu de `vi`, ce qui assure que vous êtes le seul à modifier le fichier (cohérence).

5.2 Mot de passe

Le mot de passe apparait codé dans le fichier `/etc/passwd`. Vous devez dès lors laisser à une valeur bidon (`*` par exemple) et utiliser une commande pour le définir. Cette commande est souvent

```
passwd <login>
```

qui définit ou modifie le mot de passe de *`login`*. D'autres variantes, comme `npasswd` existent. Ces dernières effectuent plus de vérificationsquand à la pertinence du mot de passe choisi.

Le mot de passe doit être composé de 6 à 8 caractères (au delà, ils ne sont pas toujours pris en compte). Voici quelques règles pour bien choisir un mot de passe

- Mélanger chiffres, lettres et caractères spéciaux.
- Utiliser les premières lettres des mots d'une phrase facile à retenir.
- Utiliser des approximations phonétiques ou visuelles.
- Ne pas utiliser des dérivés du login.

Exemples de bons mots de passe : `per6val`, `Tyfo1de`, `Aster1*`, `l'Il2l^E`.

Exemples de mauvais mots de passe : `barbara`, `stones`, `kepler1`, `Batman`, `godgod`, `Palermo`.

Qu'est-ce qui permet de dire qu'un mot de passe est mauvais ? Le fichier `/etc/passwd` est public (toute personne qui a un compte sur la machine peut

le lire). Grâce à cela, ont été développés des logiciels (comme `crack`) qui, à partir d'un dictionnaire et de règles de transformations, codent des milliers de mots de passe jusqu'à trouver une version codée qui correspond à celle se trouvant dans le fichier `/etc/passwd`. On trouve ainsi le mot de passe en clair. Un mauvais mot de passe est donc un mot de passe découvert par un tel logiciel. Il vaut toujours mieux pour un administrateur lancer un tel programme avant que quelqu'un d'autre ne le fasse. Nous en reparlerons dans un chapître consacré à la sécurité.

5.3 Le champ *GECOS*

Définit à l'origine pour des besoins particuliers, ce champ est aujourd'hui utilisé, par convention, pour contenir les entrées suivantes, séparées par des virgules : le nom complet, le numéro de bureau, le numéro de téléphone et le numéro de téléphone personnel. Souvent, on se contente du nom.

5.4 Home Directory

La home directory est destinée à accueillir tous les fichiers d'un utilisateur. Pour la créer, la séquence ressemble à

```
mkdir /home/bob
chown bob /home/bob
chgrp 100 /home/bob
```

5.5 Shell

La création d'un compte implique également de placer dans la home directory des fichiers de configuration dont le fichier associé au shell (`.cshrc`, `.profile`, `.basdrc`, ...). Parfois, des versions de base de ce fichier sont disponibles (par exemple dans `/etc/skel`). Sur certains systèmes, la commande `chsh` permet à un utilisateur de modifier son shell. Pour être accepté comme tel, il devra être repris dans le fichier `/etc/shells`.

5.6 Le fichier `/etc/shadow`

Nous avons dit que le fichier des mots de passes étai tlisible par tous et que cela comportait un danger au niveau de la sécurité du système. Pour contrecarrer cette faille, certains systèmes (surtout `AT&T`) ont introduit le

fichier `/etc/shadow` (lisible uniquement par `root`) qui ne contient que le mot de passe qui disparaît alors de `/etc/passwd`. Si on ajoute un utilisateur à la main, cela implique d'éditer les 2 fichiers.

La plupart des systèmes utilisant ce fichier en on profité pour y introduire de nouveaux champs permettant un contrôle plus fin des mots de passe.

Par exemple, en introduisant également le concept de *Password Aging* qui permet d'imposer une durée de vie limitée à un mot de passe et ainsi forcer les utilisateurs à en changer régulièrement. Ou encore, en définissant à l'avance une date limite à partir de laquelle le compte sera désactivé.

Notons enfin que certains systèmes sont exigeants lorsqu'un mot de passe est introduit et utilisent quelques règles pour refuser des mots de passe trop faciles. Ce système existe par exemple sur les machines du centre de calcul.

5.7 Le fichier `/etc/group`

Si l'utilisateur doit appartenir à d'autres groupes que son groupe principal, il faut l'ajouter dans ce fichier. Son groupe principal doit aussi s'y trouver et certains préconisent également de l'indiquer avec son groupe principal.

5.8 Supprimer un utilisateur

Pour supprimer temporairement un utilisateur ou lui empêcher momentanément l'accès, il suffit de remplacer le mot de passe codé dans `/etc/passwd` par le caractère `*` (il s'agit là d'une convention, tout autre caractère peut convenir). Comme aucun mot de passe ne peut donner ce caractère une fois codé, cela bloque l'accès au compte. Cette méthode possède des lacunes. Notamment, la personne ne reçoit pas de message clair (juste que son mot de passe est incorrect). Une autre possibilité est de changer son shell en un programme qui affiche un message indiquant pourquoi l'accès est refusé avant de quitter.

Pour supprimer définitivement un compte, il faut

1. *Backuper* la home
2. Détruire la home
3. Supprimer l'utilisateur dans `/etc/passwd`
4. Supprimer les quelques fichiers qui peuvent traîner hors de sa home (notamment la mailbox)

Il est conseillé de ne jamais réutiliser un *uid*, ce qui pourrait avoir pour conséquence d'attribuer à un nouvel utilisateur des fichiers de l'ancien qui traînent encore dans le système.

5.9 Les commandes toutes prêtes

La plupart des systèmes offrent à présent des commandes textes effectuant tout ou une partie du travail. Ces commandes auront pour nom : `useradd`, `userdel`, `usermod`.

5.10 Particularités FreeBSD

Ce système a adopté une approche originale. Toute l'information est reprise dans le fichier `/etc/master.passwd`, accessible uniquement par *root*. Pour des raisons de compatibilité, un fichier `/etc/passwd` est crée automatiquement lors de toute modification du fichier maître.

Chapitre 6

Les processus

Un **processus** est une instance d'un programme en cours d'exécution. Il est caractérisé par du *code*, des *données* et des attributs dont

PID (Process ID) Numéro du processus (**unique** à un moment donné)

PPID (Parent Process ID) PID du processus parent

UID (User ID) Le numéro du propriétaire qui a lancé le processus

GID (Group ID) Le numéro du groupe qui a lancé le processus

EUID (Effective UID) Propriétaire effectif. Sera utilisé pour déterminer les permissions accordées au processus

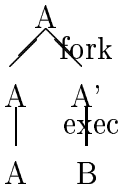
EGID (Effective GID) Idem pour le groupe

état prêt à tourner ?

priorité politique d'attribution du CPU

6.1 Création d'un processus

Les processus naissent par duplication/différenciation. Voyons par exemple comment un processus A va créer un processus B



Le processus A' est identique au processus A sauf en ce qui concerne les points suivants (non exhaustif)

– A' reçoit un nouveau PID
– Son PPID est le PID de A
– L'accounting est réinitialisé

Voyons un exemple de code C mettant en évidence les appels systèmes qui effectuent pratiquement cette création.

```
A:    int kidpid;
      kidpid = fork();
      if( kidpid==0 ) { /* fils */ exec(B); }
      /* père. kidpid donne le pid du fils */
```

6.2 Les états d'un processus

Le processeur exécute chaque processus *prêt* par petits intervalles. Macroscopiquement, c'est comme si plusieurs processus avançaient en parallèle. Mais un processus n'est pas toujours prêt. Voici quelques uns de ses états possibles

CPU c'est l'heureux élu. Il tourne

Runnable prêt à être exécuté

Sleeping en attente d'un événement, d'une ressource

Zombie prêt à mourir

Stoppé interdit de CPU

6.3 Les signaux

Les **signaux** permettent d'agir sur un processus (arrivée d'une donnée, temps écoulé, ...). Lorsqu'un processus reçoit un signal, 3 possibilités s'offrent à lui

1. Il peut avoir prévu comment le gérer
2. Il peut avoir décidé de l'ignorer (si possible)
3. Une action par défaut est entreprise

Lorsqu'un signal est reçu, et s'il n'est pas ignoré, le contrôle est passé automatiquement à la routine de gestion de ce signal (fournie par le processus dans le cas 1 ou par le système dans le cas 3). Une fois ce traitement fini, on revient à l'endroit exact dans le processus avant l'interruption (sauf si le traitement du signal a entrainé la mort de ce processus).

6.3.1 Quelques signaux utiles

On reprend ici une liste des signaux les plus utiles du point de vue administration. Nous indiquons si un processus peut décider de l'ignorer et s'il peut fournir sa propre routine de gestion.

Num.	Nom	Description	Défaut	Ignorer ?	Gérer ?
1	HUP	Eveil	Terminer	Oui	Oui
2	INT	Interrompre (CTRL-C)	"	"	"
3	QUIT	Quitter	"	"	"
9	KILL	Tuer	"	Non	Non
15	TERM	Tuer	"	Oui	Oui
	STOP	Stop	Stop	Non	Non
	TSTP	Stop Clavier (CTRL-S)	"	Oui	Oui

Pour certains signaux, le numéro n'est pas indiqué car il varie d'un système à l'autre.

6.4 Les priorités

Le processeur est attribué par tranches aux processus *prêts* en fonction de leur priorité. Cette priorité dépend de

classe du processus (système, real-time, time-sharing, intercatif, ...)

temps CPU déjà utilisé

Nice paramètre modifiable par l'utilisateur

Plus *nice* est petit, plus la priorité est grande. En `BSD` elle varie de -19 à 19. En `AT&T` elle va de 0 à 39.

Un utilisateur ne peut qu'augmenter la valeur *nice* d'un processus lui appartenant, alors que le super-user a tous les pouvoirs.

6.5 Le swap

Le **swap** (ou **mémoire virtuelle** en français) est une mémoire auxilliaire sur le disque dur. Il est utilisé comme extension à une RAM trop petite.

Il faut savoir qu'un processus occupe des *pages* de mémoire (dont la taille dépend de l'OS). Les pages occupées mais non utilisées (parce que le processus n'est pas prêt ou se trouve pour le moment dans une autre partie du code par exemple) peuvent être transférées sur le swap afin de soulager ainsi la RAM

(c'est ce qu'on appelle le **swap out**). Si nécessaire, elles seront rappelées automatiquement en RAM (**swap in**).

Ce mécanisme est très utile dans un environnement multi-utilisateur où plein de processus *inactifs* cohabitent.

Pour vérifier l'occupation du swap, la commande varie d'un OS à l'autre. Par exemple `swap` (IRIS et Solaris), `free` (Linux), `swapinfo` (HP-UX), `pstat -s` (SunOS).

6.6 Les démons

L'OS n'est pas monolithique mais composé d'une multitude de processus spécialisés (*modularité*). Cela permet notamment de ne charger que ce qui est nécessaire ou d'upgrader facilement une partie.

Un **démon**, c'est cela : un processus du système tournant en tâche de fond et remplissant un rôle spécifique.

Ex: `sendmail` qui envoie et reçoit les mails ou `in.rlogind` qui accepte les connections rlogin

Dans les OS modernes, ils peuvent être lancés automatiquement si nécessaire.

6.7 Commandes pour gérer les processus

Voyons à présent quelques commandes pour visualiser les processus et modifier leur caractéristiques.

6.7.1 ps sous BSD

La syntaxe de la commande `ps` est différente sous `BSD` et `AT&T` . Nous les voyons donc séparément. Commençons par la version `BSD` .

```
root@cso8:ps -aux | head
USER       PID %CPU %MEM   SZ  RSS TT       S    START  TIME COMMAND
tcouss    3319 93.5  5.0 6752 6288 pts/9    R 11:15:54  5:01 mapleV -p 7,6
mcodutti 28320  3.0  7.410928 9376 ?        S 17:17:10  2:04 xemacs
```

La signification des colonnes est la suivante (pour les moins évidentes)

% CPU Moyenne utilisation du CPU

% MEM Pourcentage de mémoire RAM occupée

SIZE Taille totale du processus

RSS Taille RAM effectivement occupée (le reste est en swap)

TIME temps total CPU attribué (en min :sec)

6.7.2 ps sous AT&T

```
root@cso8:/usr/bin/ps -ef | more
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0   Dec 22 ?        0:00 sched
    root     1     0  0   Dec 22 ?        4:22 /etc/init -
```

La signification des colonnes est la suivante (pour les moins évidentes)

PPID PID du parent

STIME date de début du processus

C lié à la priorité

TTY terminal attaché au processus

6.7.3 Top

`top` est un programme qui donne des informations utiles sur les processus les plus actifs. Il n'est pas disponible par défaut sur tous les processus mais il est *gratuit*.

Par rapport à `ps`, il a les avantages suivants

- rafraichissement dynamique de l'écran
- infos sur le swap, la RAM
- infos sur la charge CPU
- permet de voir/modifier le *nice*

Cette commande est pratique pour surveiller un système. Toutefois elle alourdit le système et il ne faut pas l'utilsier de façon systématique.

6.7.4 `kill`

Permet d'envoyer un signal à un processus

Exemple 6.1 `kill -HUP 1` *pour envoyer le signal 1 au processus 1.*

6.7.5 `killall`

Sur certains système, permet de tuer tous les processus portant un même nom

Exemple 6.2 `killall nfsd`

A utiliser avec précaution

6.7.6 Nice

Pour voir le *nice* d'un processus, utiliser la commande `top`. Pour modifier ce *nice*, utiliser `top` ou `renice` (`BSD`). Pour donner une priorité quand on lance un processus, `nice valeur commande`.

6.8 Nohup

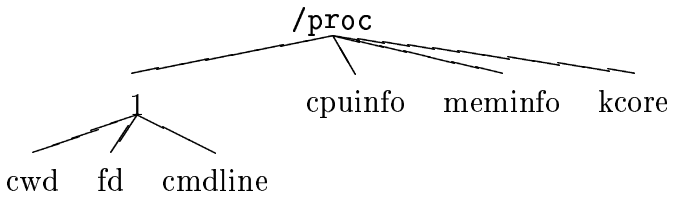
Lorsqu'un utilisateur se déconnecte, le shell envoye un signal `SIGHUP` à tous les descendants. Souvent (si le processus ne le capte pas ou ne l'ignore pas) cela amène à tuer un processus en background. La solution est d'utiliser la commande `nohup` qui va lancer le processus avec une option pour ignorer le `SIGHUP`.

Ex: `nohup batch &`

Notons toutefois que certains shells (dont les dérivés du C-shell) font un `nohup` automatique si on utilise le `&`.

6.9 /proc

Il s'agit d'une *pseudo-directory* reprenant les caractéristiques des processus (pour consultation). Cette directory se retrouve essentiellement sous `AT&T` mais également sous `Linux`. Elle ressemble à



Elle contient une directory par processus dont le nom est son PID. Elle contient également des fichiers reprenant des infos sur le système.

Il est vivement recommander de ne pas y toucher.

Chapitre 7

Le shell

Un **shell** est un programme servant d'interface avec l'utilisateur. Il accepte des commandes et lance leur exécution. Il y a en fait plusieurs shells qui se ressemblent tous un peu avec quelques fonctionnalités différentes et quelques différences de syntaxe

`sh`	Bourne Shell (début des années '70)
`csh`	C-Shell (fin des années '70)
`ksh`	Korn-Shell. `sh` + facilités du `csh`
`bash`	Bourne Again SHell. Soutenu par GNU. `sh` + facilités interactives (comme les flèches)
`tcsh`	`csh` + flèches

7.1 Les scripts

Un **script** est une suite de commandes (dans un fichier) que le shell va exécuter comme si on les tapait interactivement.

Pour lancer un script, il y a deux possibilités.

1. `source script` (en `(t)csh` et `bash`)
   `. script` (en `(k)sh` et `bash`)
2. Le rendre *exécutable* (permission `x`) et taper simplement son nom.

Les deux solutions ne sont pas tout-à-fait identiques en ce sens que, dans la deuxième, un sous-shell, distinct du shell courant, est lancé pour exécuter ces commandes.

Si on veut exécuter un script avec un autre shell (ou un tout autre programme) que le shell courant, il faut l'indiquer par une première ligne bien précise dans le fichier.

```
#!/bin/sh
```

en tête d'un fichier indique que le programme `/bin/sh` (le Bourne shell ici) doit être lancé pour traiter le fichier.

7.2 Initialisation

Lorsqu'il démarre, le shell lit un script d'initialisation dans la HOME de l'utilisateur. On peut ainsi configurer le shell et l'environnement UNIX.

`sh`

`.profile` (au login)

`(t)csh`

`.login` (au login)
`.(t)cshrc`
`.logout` (au logout)

`bash`

`.bash_profile` ou `.bash_login` ou `.profile` (au login)
`.bashrc` (si ce n'est pas un shell associé à un login)
`.bash_logout` (au logout)

Sur certains OS, il y a également lecture d'un fichier global avant. Si ce n'est pas le cas, on peut simuler cette faculté en faisant commencer le fichier local par un `source` du script commun.

7.3 Les variables

Il faut faire la distinction entre les variables locales, qui ne sont accessibles que dans le shell, et les variables globales (on dit plutôt *d'environnement*) qui sont passées à tous les fils lancés par le shell (comme les commandes)

`(t)csh`

L'assignation d'une variable locale se fait par `set VAR=valeur` et celle d'une variable globale par `setenv VAR valeur`. La liste des variables locales (globales) connues ainsi que leur valeur courante s'obtient par `set` (`setenv`).

`(ba)sh`

Ici, une variable globale est une variable locale qui a été exportée. L'assignation se fait par `VAR=valeur` et une variable est exportée avec `export VAR`. Pour une liste des variables, `set`. Alors que `env` ne donne que les variables exportées.

Dans tous les shells, on fait référence à la valeur d'une variable par `${VAR}` ou `$VAR` si cela ne provoque pas d'ambiguïté.

Ex: (en Bourne Shell)

```
# var1=1
# var2=2
# export var2
# sh                          (en entre dans un autre shell)
# echo $var1
# echo $var2
2
```

7.3.1 PATH

Cette variable d'environnement défini où le shell cherche les commandes.

1. Si le nom contient le caractère `/`, il est interpreté comme un chemin (relatif ou absolu) Ex: `#/usr/bin/more` ou `#./brol`
2. Sinon, le shell cherche un fichier de ce nom dans un des répertoires spécifiés par `PATH` (dans l'ordre).
   Ex:

   ```
   #echo $PATH
   /bin:/usr/bin:.:/usr/bin/X11
   #more                              (trouvé car dans /usr/bin)
   ```

Pour des raisons de sécurité, il est vivement recommandé de ne pas avoir `.` dans son `PATH`.

En `(t)csh`, cette variable est lue une fois au démarrage pour des raisons d'efficacité. Si on la modifie et si on souhaite que le shell en tienne compte, il faut utiliser la commande `rehash`.

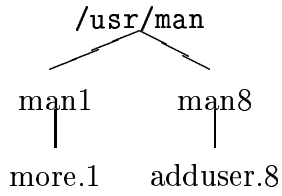
Notons également que certaines commandes sont internes au shell et non pas définies dans des fichiers externes. Elles sont ainsi plus rapides et toujours accessibles.

Ex: `cd`, `source`, `alias`, `echo`, `kill`, ...

7.3.2 MANPATH

Les manuels accessibles via la commande `man` sont regroupés en sections (un répertoire par section).

Ex:



Ces versions sont dans un langage appelé `nroff` (penser à LaTeX). Il sont compilés lorsqu'on y fait appel. Parfois, on dispose également des versions précompilées (répertoires `cat1`, ...).

`MANPATH` définit tous les chemins où l'on peut trouver des manuels

Ex:

```
# echo $MANPATH
/usr/man:/usr/local/man:/usr/X11/man
# man more
```

va chercher dans les répertoires `/usr/man/mani`, `/usr/local/man/mani` et `/usr/X11/man/mani` un fichier de la forme `more.i` où `i` est un symbole parmi une liste dépendant de l'OS (souvent 1 à 9 ainsi que `l` et `n`).

7.3.3 LD_LIBRARY_PATH

Un programme utilise des fonctions définies dans des librairies. Ces fonctions peuvent être incorporées

1. dans l'exécutable lors de la compilation. C'est la méthode classique. On parle de compilation statique.
2. ou lors de l'exécution, dans le processus, à partir de la librairie sur le disque. On parle de compilation dynamique.

La deuxième méthode s'impose de plus en plus. Elle permet d'obtenir un exécutable moins gros et permet des upgrades de la librairie sans recompilation des programmes. Toutefois, cela implique que la librairie doit être présente partout où on lance le programme.

`LD_LIBRARY_PATH` définit où chercher les librairies dynamiques (en plus des chemins par défaut).

Ex:

```
mcodutti@lit1:echo $LD_LIBRARY_PATH
/usr/local/lib:/usr/ucblib:/usr/openwin/lib:/usr/dt/lib
mcodutti@lit1:ldd /bin/more
        libintl.so.1 =>  /usr/lib/libintl.so.1
```

```
        libc.so.1 =>     /usr/lib/libc.so.1
        libw.so.1 =>     /usr/lib/libw.so.1
        libdl.so.1 =>    /usr/lib/libdl.so.1
mcodutti@lit1:ls -l /usr/lib/libc.*
-rw-r--r--   1 bin       1192328 Oct 25  1995 /usr/lib/libc.a
lrwxrwxrwx   1 root           11 Jun  7  1996 /usr/lib/libc.so -> ./libc.so.1
-rwxr-xr-x   1 bin         664048 Oct 27  1995 /usr/lib/libc.so.1*
```

La commande `ldd` indique toutes les librairies qui vont être impliquées par l'exécution d'un programme. La version en `.a` d'une librairie est la statique alors que la dynamique est un lien symbolique vers une version particulière de la librairie. Un programme ne pourra s'exécuter si la version présente est fort différente de celle prévue.

7.3.4 Le prompt

Le **prompt** (invite en français) est le petit texte que le shell affiche à l'écran pour signaler qu'il est prêt à recevoir la commande suivante. On le définit via la variable `prompt`

Ex: (`tcsh`)

```
# set prompt="%n@%m:%/ "
mcodutti@lit1:/ULB/staff/mcodutti
```

7.3.5 `TERM`

Les opérations particulières sur un terminal (gras, souligné, effacer, se positionner, ...) se font via des caractères spéciaux (différents d'un type de terminal à un autre).

Les caractéristiques de chaque type de terminal sont stockées dans un fichier (`termcap`) ou une arborescence (`terminfo`)

La variable `TERM` donne l'identificateur du terminal utilisé (vt100, tvi925, xterm, ...).

7.3.6 Autres variables

USER login de l'utilisateur

HOME home directory

SHELL le nom du shell courant

PWD la directory courante

HOSTNAME nom de la machine

HOSTTYPE type de machine

EDITOR éditeur préféré (utilisé par certaines commandes)

TAPE nom du tape (utilisé par `mt` par exemple)

PRINTER nom de l'imprimante par défaut (utilisé par `lpr`)

UID user ID

GID group ID

7.4 Les alias

Un **alias** permet de donner un nom à une suite de caractères (un raccourci, en somme).

Pour définir un alias, la syntaxe est `alias nom=valeur` en `(ba)sh` (attention ! pas d'espace autour de `=`) et `alias nom valeur` en `(t)csh`.

Pour voir la liste des alias définis, `alias`.

Pour effacer un alias, `unalias nom`

Ex:

```
# touch brol1 ; touch brol2
# rm brol1
# alias rm "rm -i"
# rm brol2
rm: remove brol2 (y/n)? y
```

Tous les shells sauf `sh` donnent également un sens spécial au caractère `~`.

- `~` indique la home de l'utilisateur
- `~login` indique la home de l'utilisateur ayant ce login

Ex:

```
# cd ~alice/Mail
# pwd
/home/alice/Mail
# whoami
mcodutti
# cd ~/brol
# pwd
/home/mcodutti/brol
```

7.5 Les quotes

Les **quotes** sont des opérateurs qui modifient le sens de leur argument. Il y a en 4 ', \, " et ‘

- ' groupe des caractères et supprime leur interprétation

  Ex:
  ```
  # touch brol1 brol2
  # touch 'brol3 brol*'
  # ls -l
  -rw-r--r--   1 mcodutti        0 Feb 22 19:21 brol1
  -rw-r--r--   1 mcodutti        0 Feb 22 19:21 brol2
  -rw-r--r--   1 mcodutti        0 Feb 22 19:21 brol3 brol*
  ```
- \ idem mais uniquement sur le caractère suivant.

  Ex: `# touch brol3\_brol4`
- " idem à ' mais les variables sont interpretées.

  Ex:
  ```
  # set var=brol
  # touch '${var}1'
  # touch "${var}2"
  # ls -l
  total 6003
  -rw-r--r--   1 mcodutti        0 Feb 22 19:24 ${var}1
  -rw-r--r--   1 mcodutti        0 Feb 22 19:24 brol2
  ```
- ‘ censé englober une commande. Elle est exécutée et remplacée par son résultat (output). Notons qu'il y a substitution de variables.

  Ex:
  ```
  # echo uname
  uname
  # echo ‘uname‘
  SunOS
  # set var=uname
  # echo ‘$var‘
  SunOS
  ```

7.6 Redirections

Chaque fichier ouvert par un processus se voit assigner un numéro (le *file descriptor*). Un processus commence avec 3 descripteurs :

0 standard input (par défaut, relié au clavier)

1 standard output (par défaut, relié à l'écran)

2 standard error (par défaut, relié à l'écran)

On peut toutefois rediriger ces file descriptors. Cette technique est couramment utilisée un UNIX pour lire les données et/ou sauver les résultats dans un fichier. Voyons comment cela fontionne en `(ba)sh` :

- `<file` : redirige l'input vers `file`
- `>file` : redirige l'output vers `file`. créé ou vidé.
- `>>file` : redirige l'output vers `file`. append.
- `2>&1` : associe le file descriptor 2 au fichier référencé par le file descriptor 1 (c-à-d redirige l'erreur vers l'output).
- `cmd1|cmd2` : l'output de `cmd1` est redirigé vers l'input de `cmd2`.

7.7 Substitution de fichiers

Lorsqu'il reçoit une commande, le shell interprete certains caractères et les remplace en fonction des fichiers présents dans le filesystem. Cela permet d'indiquer en raccourci un ou plusieurs fichiers.

- `*` : remplace $n \leq 0$ caractères (pas .)
- `?` : remplace 1 seul caractère (sauf .)
- `[abc]` : remplace `a` ou `b` ou `c`

Ex:

```
# ls - a
. .. .cshrc .login res1 res1b res2 res2b res3
# ls *
res1 res1b res2 res2b res3
# ls res?b
res1b res2b
# ls res[12]
res1 res2
```

7.8 History

Le shell retient les commandes précédentes, ce qui permet d'y faire référence. Ce mécanisme n'existe toutefois pas en `sh`. Dans les autres shells, la portée est déterminée par une variable d'environnement.

- `history` : donne la liste des précédentes commandes
- `!57` : rappelle la commande numéro 57
- `!rm` : rappelle la commande la plus récente débutant par `rm`

– !! : rappelle la dernière commande
– ^s1^s2^ : rappelle la dernière commande où s1 est remplacé par s2
– + beaucoup d'autres ...

Notons qu'en bash et en tcsh, on peut utiliser les flèches pour revenir aux commandes précédentes et les modifier.

7.9 Ordre d'évaluation d'une commande

Lorsque le shell reçoit une commande, il va l'interpreter en fonction de tous les mécanismes que l'on vient de voir (history, alias, ...). Il est bon de connaître plus en détail comment il procède pour mieux comprendre le résultat de certaines commandes.

Voici, par exemple, l'algorithme d'interprétation du C-Shell :

1. Traiter l'history (!, ^)
2. Mettre la commande dans l'history
3. Séparer en mots
4. Alias
5. Traiter les redirections et background (<,>,|,&)
6. Remplacer les variables par leur valeur (pas dans '')
7. Traiter le back quoting (``)
8. Expansion des noms de fichiers
9. Exécuter la commande

7.10 Les paramètres

On peut passer des arguments à un script. Dans celui-ci, on y fait référence par

– $1, ..., $9 pour les arguments 1 à 9
– $* pour tous les arguments
– $0 pour le nom du programme

Ex:

```
# cat brol
echo "$0-$2-$*"
# brol a b c d e
brol-b-a b c d e
```

7.11 Evaluation d'expressions *booléennes*

Il faut savoir que tout processus se termine par un appel à la fonction `exit` qui indique au système que le processus est terminé et peut devenir un *zombie*. Lors de l'*exit*, le processus fournit un nombre (le statut) indiquant comment cela s'est passé, destiné au processus père. Par convention, 0 indique que tout s'est bien passé et un nombre différent de 0 indique un code d'erreur. Ainsi, toute commande renvoie également un code de statut.

La commande `test` evalue une expression *booléenne* et renvoie un statut en conséquence. 0 si l'expression est vraie et une valeur non nulle si elle est fausse. Elle permet de tester des expressions impliquant des nombres ou des chaînes de caractères mais également de tester des propriétés de fichiers (permission, existence,...)

La syntaxe est `test expression` ou de manière équivalente, dans un script, `[ expression]`.

7.12 Structure

Comme tout langage, le shell fournit des structures de branchement et de répétition. Nous n'entrerons pas dans les détails. Voici le canevas de quelques structures du Bourne shell :

```
# pour une alternative
if command;
then
...
else
...
fi

# pour boucler sur une liste
for name in list; do
...
done

# pour répeter une commande
while command; do
...
done
```

Exemple:

```
# Ce script affiche la liste des fichiers executables de la directory courante
for name in *; do
    if [ -x $name ]; then echo $name est executable; fi
done
```

7.13 Commandes utiles

`Unix` est basé sur le principe de commandes qui effectuent une et une seule chose et qui sont facilement combinables pour obtenir un plus grand effet. Toute personne voulant écrire des scripts se doit de connaître les principales. Elle fonctionnent toutes suivant le même principe. Un texte (sur l'entrée standard ou issu de fichiers) est fourni en entrée et un autre texte est fourni en sortie.

7.13.1 head/tail

Ces 2 commandes permettent de ne garder que les premières (dernières) lignes/octets d'un fichier. La commande `tail` est beaucoup plus riche et permet de faire ce que fait `head`.

Exemple: `#tail -20 brol`

7.13.2 grep

Cette commande permet de ne sélectionner que les lignes respectant certains motifs.

- On peut sortir les lignes candidates
- Les faire précéder d'un numéro de ligne
- Les accompagner de quelques lignes autour
- Inverser la sélection
- Travailler sur plusieurs fichiers
- Afficher le nom du fichier correspondant
- N'afficher que le nom des fichiers contenant au moins une ligne reprise

Exemple: `#grep scsi *.h`

7.13.3 sort

Cette commande permet de trier des lignes de texte.

- On peut spécifier les colonnes à traiter
- Ainsi que le délimiteur de colonnes

- On peut trier en ordre inverse
- Indiquer si le tri est numérique ou alphabétique
- On peut juste vérifier si un fichier est trié
- Ou encore joindre plusieurs fichiers triés

Exemple: `#sort -t: -n +3 /etc/passwd`

7.13.4 awk

Cette commande est très puissante mais un peu difficile à appréhender. En gros, elle considère le texte comme étant composé de colonnes (champs). On peut associer une série de commandes à différents motifs rencontrés.

Exemple: Pour extraire une colonne du fichier passwd

```
#cat /etc/passwd | awk -F : '{print $7}'
```

Exemple: Pour obtenir la liste des pourcentages d'utilisation des disques durs (2 versions)

```
#df | awk '!/Sys/ {print $5}'
#df | tail +2 | awk '{print $5}'
```

7.13.5 uniq

Cette commande ne garde qu'une copie de lignes indentiques dans un fichier trié. On peut aussi obtenir un comptage de chaque occurence de ligne.

Exemple: Pour obtenir la liste des shells utilisés sur le système

```
#cat /etc/passwd | awk -F : '{print $7}' | sort |uniq
```

7.13.6 sed

Il s'agit d'un éditeur non interactif. Cela permet de programmer des changements fins dans un texte. Son utilisation n'est tout de fois pas immédiate.

- On peut effectuer à peu près tout ce qu'on peut faire avec `vi`
- Cela peut s'appliquer à tout le texte ou juste une partie

Exemple: Pour remplacer bash par tcsh dans le fichier `/etc/passwd`

```
#cat /etc/passwd | sed s/bash/tcsh/
```

Exemple: Pour ne pas le faire pour la première ligne

```
#cat /etc/passwd | sed '2,$ s/bash/tcsh/'
```

7.13.7 Exercice récapitulatif

Donner une liste triée de tous les shells utilisés sur la machine (du plus utilisé au moins utilisé).

Chapitre 8

Démarrage et arrêt

8.1 Le démarrage

1. Le *moniteur* (en PROM) prend la main
2. Etapes de *bootstrapping*
3. Le chargement et l'initialisation du *noyau*
4. La détection des *périphériques* et leur configuration
5. La création des *processus spontanés*
6. L'intervention de l'opérateur
   (si démarrage manuel)
7. Exécution des *scripts de démarrage*
8. Le passage en *mode multi-utilisateur*

8.1.1 Le moniteur

- Petit programme en PROM dans la machine
- Sur PC, il s'agit du *BIOS*
- A pour rôle d'identifier le périphérique de démarrage et d'initier le bootstrapping.
- Procède en général automatiquement mais il est possible de l'interrompre. On dispose alors d'une interface pour
  - Vérifier les composants hardwares
  - Modifier la séquence de démarrage
  - Indiquer des options de démarrage

8.1.2 Le moniteur PC

- Tout commence par le *BIOS* qui est configuré pour rechercher le système sur certains périphériques dans un certain ordre (avec ses limites).
- Il charge les 512 premiers octets du périphériques sélectionné (le *MBR*)
- Ce *MBR* contient un petit programme qui charge le *boot loader* (2ème niveau).
- Ce *boot loader* charge le noyau

8.1.3 Le moniteur PC et `Linux`

- `Linux` fournit un *boot loader* pour le MBR (*Lilo* ou *Grub*)
- Ils connaissent les autres systèmes d'exploitation
- Il est possible de spécifier des options au démarrage (single, autre noyau)

8.1.4 Le moniteur `Solaris`

- La séquence de touches permettant d'y accéder est `STOP-A`.
- Voici quelques commandes et leur signification :

  `boot disk|cdrom|net` pour booter sur le disque, le cdrom ou le réseau.

  `probe-scsi` pour lister les périphériques SCSI

  `boot -s` pour booter en mode *single-user*

  `boot -r` à utiliser si on a ajouté un périphérique depuis le dernier démarrage.

8.1.5 Le noyau

Le noyau est un programme qui est chargé en mémoire et exécuté. Chaque machine a en ROM un petit programme, le *loader*, qui se charge de cette opération (éventuellement en 2 étapes).

- Le noyau est un programme qui va mettre en place `Unix`.
- Il va diriger les étapes suivantes du démarrage.
- Il restera en mémoire pendant le fonctionnement.
- Possède un nom et un emplacement connu du moniteur
  `/unix`, `/vmunix`, `/boot/vmlinuz`
- Peut être recompilé pour être adapté aux besoins.

8.1.6 Détection des périphériques

- Le noyau est construit avec certains pilotes.

- Au démarrage, ces pilotes détectent la présence effective du matériel géré.
- Peuvent demander des informations complémentaires aux périphériques.
- Certains noyaux peuvent s'enrichir dynamiquement de nouveaux pilotes (modules). Cela permet :
  - de garder un noyau léger
  - de ne pas devoir recompiler à chaque ajout de pilote

8.1.7 Les processus spontanés

- Appelés ainsi parce que non créés par le système `fork` traditionnel.
- Leurs nombres, rôles et noms varient d'un `Unix` à l'autre.
- Il y a toujours `init` avec le PID 1
- C'est ce dernier qui va mettre la système en opération (démarrage des démons, des processus d'ouverture de sessions, ...)

8.1.8 Le mode manuel

- `init` peut passer en mode manuel.
- Utile en phase de récupération/réparation (disque endommagé par exemple)
- Système minimal (au niveau des partitions montées, des processus lancés)
- Demande le plus souvent le mot de passe root
- Nécessité parfois une vérification manuelle des disques (`fsck`)
- Une fois terminé, on peut demander à continuer le démarrage

8.1.9 Les scripts de démarrage

- Ils sont lancés par `init` et vont configurer la machine :
  - Donner un nom à l'ordinateur
  - Définir la zone horaire
  - Vérifier les disques
  - Les assembler (monter en un seul filesystem)
  - Nettoyer `/tmp`
  - Configurer le réseau
  - Lancer les démons
- Le mécanisme mis en oeuvre est très différent entre `BSD` et `SysV`.

8.1.10 Les scripts de démarrage `BSD`

- `init` exécute le seul script `/etc/rc` (shell)

- Ce script s'occupe de lancer les autres
- Il se base sur des fichiers de configuration (`/etc/rc.conf`, `/etc/rc.conf.local`)
- Et lance des scripts spécifiques (`/etc/rc*`)
- Le script `/etc/rc.local` est dédié aux modifications locales apportées par l'administrateur.
- Il peut exister un script (genre `/etc/rc.shutdown`) exécuté lors d'un arrêt.

8.1.11 Les scripts de démarrage `SysV`

- Il existe 7 niveaux d'exécution

  0 : système arrêté

  1 (ou S) : niveau mono-utilisateur

  2 à 5 : différents niveaux utilisateurs

  6 : système redémarré

- Démarrer ou arrêter revient à changer de niveau.
- A chaque niveau sont associés des scripts.
- Le fichier `/etc/initttab` défini les scripts à lancer lorsqu'on entre dans un niveau.
- Ex: extrait de la version `Solaris`

```
s3:3:wait:/sbin/rc3 >/dev/console 2>&1 </dev/console
```

  s3 nom arbitraire

  3 concerne l'entrée dans le niveau 3

  rc3 script à exécuter

  wait il faut attendre que le script soit fini avant de passer à autre chose

- Chaque script `rc`*`i`* a un comportement standard
- A chaque niveau correspond le dossier `/etc/rc`*`i`*`.d`
- Ce dossier contient des scripts de la forme `[K|S]nnnNOM`
  - `S` pour *Start* ; `K` pour *Kill*
  - le numéro permet de les ordonner
  - le nom indique son rôle (démarrage d'un démon ; aspect particulier du système, ...)
- Lorsqu'il entre dans un niveau, le système exécute tous les scripts commençant par `K` (dans l'ordre des numéros, avec l'argument `stop`). Ceci permet de tuer tout ce qui vient des autres niveaux.

- Ensuite, il exécute tous les scripts commençant par `S` (dans l'ordre des numéros ; avec l'argument `start`)
- Presque toujours, il s'agit de liens symboliques vers les scripts déposés dans `/etc/init.d`. D'où l'intérêt du paramètre. Exemple:
- Supposons que l'on a un gestionnaire de licence à démarrer au niveau 3 et à tuer au niveau 0.
- On va créer le script `/etc/init.d/licence` qui ressemblera à

```
#!/bin/sh
case ``$1'' in
   'start')
      # on lance ici le démon
   'stop')
      # on tue ici le démon
esac
```

- Il ne reste plus qu'à créer les liens symboliques qui vont définir son utilisation

```
ln -s /etc/init.d/licence /etc/rc3.d/S20licence
ln -s /etc/init.d/licence /etc/rc0.d/K20licence
```

Particularités `Linux`. Les niveaux d'exécution 2 à 5 sont tous multi-utilisateurs et définis comme suit :

2 : sans réseau

3 : avec réseau

4 : inutilisé

5 : avec réseau et interface graphique

8.2 Arrêter, redémarrer le système

Il existe plusieurs façons d'éteindre un ordinateur.
- La commande `shutdown`
- Les commande `halt` et `reboot`
- Envoyer un signal à `init`
- Tuer `init`
- Couper l'alimentation

Décrivons cela en commençant par le plus sûr.

8.2.1 *shutdown*

- Cette commande est la plus sûre et la plus propre.

- Elle permet de
  - programmer un arrêt à l'avance
  - prévenir les utilisateurs (message explicatif)
  - empêcher les logins si le shutdown est imminent
- Elle possède également des options pour indiquer à quel niveau on veut arriver
  - arrêter la machine
  - passer en mode single user
  - rebooter la machine
- La syntaxe exacte est différente d'un système à l'autre
- Exemple: En `Linux`,

  ```
  shutdown -h 23:00 'Upgrade Systeme'
  ```

  - indique qu'on programme un arrêt à 23 heures.
  - Les utilisateurs seront prévenus par le message indiqué.

8.2.2 *halt* et *reboot*

- Tout aussi sûr que la commande précédente mais peut-être plus brutal.
- Sur certains OS, personne n'est prévenu et l'action est immédiate.
- Sur d'autres, c'est équivalent à `shutdown`.

8.2.3 *fasthalt* et *fastboot*

- Issu de `BSD` .
- n'indique pas que l'arrêt doit être rapide mais bien le prochain démarrage.
- pratiquement, ces commandes écrivent un fichier vide dont le nom est `/fastboot` puis passent la main à leur homologue.
- au démarrage, le système voit la présence de ce fichier et se met en mode *fast*, ce qui équivaut à ne pas vérifier les disques (l'opération la plus longue).
- de plus, le fichier `/fastboot` est détruit.

8.2.4 signal à *init*

- Sur certains OS, envoyer le signal `TERM` au processus `init` permet de passer en mono-utilisateur. (*à déconseiller*)
- Tuer le processus `init` permet de redémarrer. (*brutal*)

8.2.5 Séquence de touches

- Sur certaines machines, une séquence de touches permet de redémarrer.
- A éviter au maximum car peut aboutir à une inconsistance sur le disque (à cause de l'utilisation de caches).
- Ceci n'est pas vrai pour `Linux` où la touche `ALT-CTRL-DEL` est *remappée* sur la commande `reboot` (via le fichier `inittab`).

8.2.6 Eteindre la machine

- En dernier recours uniquement.
- Le risque de perte d'informations et/ou d'inconsistance sur le disque n'est pas faible.

Chapitre 9

Les tâches périodiques

Certaines opérations doivent être effectuées régulièrement (nettoyer le disque, vérifier le système, ...). `Unix` fournit un mécanisme pour lancer des commandes à intervalles réguliers, `cron`. Il s'agit d'un démon tournant en permanence et utilisant des fichiers de configuration (`crontab`) lui indiquant ce qui doit être lancé et quand.

La directory `/var/spool/cron/crontabs` (ou `/var/spool/cron` ou encore `/var/cron/tabs`) contient un fichier par utilisateur. Ce fichier associé à un utilisateur a pour nom son login et contient toutes les commandes à lancer pour son compte. Il a le format suivant

```
# commentaire
minute heure jour mois jour_semaine commande
```

Ex: `0,30 8-20 * * * verif`
indique que la commande `verif` doit être lancée toutes les heures précises et les heures 30, tous les jours entre 8h et 20h30.

Les fichiers de configuration sont gérés via la commande `crontab`

`crontab -l`	liste le contenu du fichier
`crontab -e`	pour éditer (en `vi`) le fichier
`crontab -e user`	pour éditer le fichier de `user` (uniquement par `root`)
`crontab -r`	pour supprimer le fichier

Chaque fois qu'une commande est lancée, l'utilisateur reçoit un mail contenant le résultat (l'output) de la commande.

Qui a le droit d'utiliser ce mécanisme ? Il existe deux fichiers qui indique qui peut et qui ne peut pas invoquer `cronbtab`. Ces fichiers s'appellent `cron.allow` et `cron.deny` (parfois juste `allow` et `deny`) et se trouvent dans le dossier `/etc` (ou `/var/cron`, `/usr/lib/cron` ou encore `/etc/cron.d`).

Si le fichier *allow* existe, la personne doit s'y trouver. S'il n'existe pas, alors la personne ne peut pas se trouver dans le fichier *deny*. Si aucun fichier n'existe, alors tout le monde ou seul le *root* peut l'utiliser (dépendant du site)

Chapitre 10

Les sauvegardes

10.1 Introduction

L'information contenue sur les disques est souvent plus importante que l'ordinateur lui-même. Il est impératif de s'assurer contre la perte d'informations due à

- une défaillance matérielle
- une destruction par un logiciel
- une erreur de l'utilisateur (`rm *`)
- un désastre (incendie, tremblement de terre, raz de marée, ...)

La stratégie adoptée va dépendre de

- la quantité, le roulement et l'importance de l'information
- la somme que l'on est prêt à engager.

Nous allons commencer par aborder les aspects techniques (commandes existantes, supports possibles) pour ensuite parler de stratégie et de bonne pratique.

10.2 Les commandes liées à la sauvegarde

Les commandes les plus répandues pour la sauvegarde dans le monde `Unix`sont `dump` et `restore`. De nombreux logiciels offrent une interface agréable et des possibilités d'automatisation mais utilisent ces 2 commandes en arrière plan.

10.2.1 dump

`dump` est la méthode la méthode la plus appropriée aux backups. En effet,

- on peut tout *backuper* (les anciennes versions de `tar` ne peuvent pas backuper `/dev`)
- respecte les liens (moins facile avec `tar`)
- notion de **backup incrémental**

Un backup se fera à un certain *niveau* (de 0 à 9). Un backup de niveau i consistera à sauver tout ce qui a été modifié depuis le dernier backup à un niveau inférieur. Un backup de niveau 0 revient à tout backuper.

La commande pour un `dump` ressemble à

```
dump 0uvf /dev/nrst0 /dev/rsd0g
```

où

0 niveau (ici full backup)

u mettre à jour `/etc/dumpdates` qui retient les dates et niveaux des backups (essentiel pour les backups partiels)

v verbeux

f on spécifie le nom du tape (si différent tape par défaut)

Attention : On spécifie le pseudo-fichier lié à une partition en mode *raw* (`rsd0g`) et pas le filesystem correspondant (`/usr`). Mais certains OS le permettent.

`Solaris` : la commande est `ufsdump`

Si `dump` se méprend sur le tape (densité, longeur), utiliser des options (s,d) pour l'éclairer.

Il est possible d'effectuer un backup sur le tape d'une autre machine. La syntaxe est

```
rdump 0uf orca:/dev/nrst0 /dev/rsd0g
```

Attention : Il faut les permissions réseau (`.rhosts`)

Cette commande permet de centraliser les backups en écrivant un script qui lance les backups de tous les disques d'un groupe de machines.

Ex: soit 3 machines (A,B,C) avec un tape branché sur B. On peut lancer sur A un script qui ressemble à ce qui suit pour backuper les 3 machines.

```
# script sur A pour lancer le backup
      rdump B:tape partitions
rsh B  dump   tape partitions
rsh C rdump B:tape partitions
```

Remarque : Sur certaines machines, la commande `dump` offre la possibilité d'une sauvegarde en réseau et il n'y a pas de `rdump`.

10.2.2 restore

Pour récupérer des fichiers sauvés sur bande, il faut

1. déterminer sur quelle(s) bande(s) se trouvent les fichiers à restaurer.
2. restaurer dans l'ordre chronologique
3. choisir la directory où on restaure. En effet, la restauration se fait dans la directory courante, d'où
   - dans `/tmp` puis les déplacer
   - directement à la bonne place

On peut restaurer	une partie	`restore xvf /dev/nrst0 fichiers`
	tout	`restore rvf /dev/nrst0`
	réseau	`rrestore xvf orca :/dev/nrst0 fichiers`
	interactif	`restore ivf /dev/nrst0`

La méthode interactive lance un programme qui permet de se ballader dans le filesystem sauvé sur la bande et d'indiquer ce qui doit être restauré. On dispose des commandes :

ls idem `ls` UNIX (contenu)

cd dir idem `cd` UNIX (change directory)

add file—dir ajoute le fichier ou la directory (récursif) dans la liste des fichiers à restaurer. Il apparaîtra avec un `*` lors d'un `ls`.

extract lance la restauration.

10.2.3 tar

`tar` collecte plusieurs fichiers en un seul, ce qui facilite leur manipulation. Il est plus simple et plus souple d'usage que `dump` mais ne propose pas de sauvegarde incrémentale ce qui réduit drastiquement ces possibilités en tant qu'outil sérieux de sauvegarde. De plus, il ne permet pas de sauver sur plusieurs volumes.

créer : `cd / ; tar cf /tmp/home.tar home`

contenu : `tar tf /tmp/home.tar`

extraire : `cd /tmp ; tar xf home.tar home/bob`
(va extraire `bob` dans `/tmp/home/bob`)

sur bande : `cd / ; tar cf /dev/nrst0 home`

compression : `cd / ; tar zcvf /tmp/home.tgz home`

10.2.4 mt

Il est possible de mettre plusieurs enregistrements à la suite sur une bande

Ex: `tar cf /dev/nrst0 /home ; tar cf /dev/nrst0 /usr` donne

/home : /usr :

Le lecteur de bandes met des repères pour pouvoir retrouver les débuts d'enregistrements CEPENDANT

- il n'y a pas de table des matières
- il ne connaît rien du format du contenu (tar, dump, autre, ...)

C'est pourquoi, il faut **tout noter soigneusement**.

Pour se promener sur la bande, on dispose de la commande `mt`

`mt rew`	pour rebobiner
`mt fsf n`	pour avancer de n enregistrements
`mt off`	pour éjecter la bande

Ex:

`mt rew ; mt fsf 2`	on se retrouve au début du 3ème
`tar cf /dev/nrst0 /tmp`	Sauver `/tmp` et fin du 3ème
`mt rew`	début du 1er
`tar xf /dev/nrst0`	extraire `/home` et fin du 1er
`mt fsf 1`	début 2ème (**Attention**)
`tar xf /dev/nrst0`	extraire `/usr` et fin du 2ème

10.3 Les supports de sauvegarde

L'offre en terme de support pour la sauvegarde, est abondante. On y trouve des bandes magnétiques, des Cd, des disques durs. Se référer au livre de Nemeth, Snyder, Seebass et Hein cité en fin de document. Citons-en quelques uns

Disquettes. Trop faible quant au volume et à la fiabilité.

Superdisquettes Les lecteurs Zip et Omega sont présents mais le coût reste élevé.

CD Intéressant mais la capacité reste faible et le prix des réinscriptibles est plus élevé que d'autres solutions.

Bandes Format QIC, 8mm (Exabyte) ou 4mm (DAT). Capacité allant jusqu'à 40 Gb avec un prix de l'ordre de 2 Euros le Gb.

L'exabyte et le DAT sont souvent les meilleurs choix.

10.4 Les stratégies de sauvegarde incrémentale

La stratégie de backup à adopter dépend
- du nombre de bandes disponibles
- de l'époque à laquelle on veut pouvoir remonter et avec quelle précision
- de la facilité de récupération
- du temps de backup

Le système de fichiers est composé de partitions aux caractéristiques fort peu semblables. Il est dès lors normal d'envisager des stratégies différentes pour ces différentes partitions.

Examinons quelques cas et la stratégie associée.

10.4.1 Les fichiers personnels

Il s'agit là d'une partie très sensible qui doit faire l'objet d'une sauvegarde minutieuse. On peut proposer la stratégie suivante.
- Une sauvegarde mensuelle de niveau 0. On effectuera un roulement sur 12 bandes en gardant chaque année une bande en réserve. On peut ainsi remonter mois par mois sur une année puis année par année.
- Une sauvegarde hebdomadaire de niveau 5. On gardera 8 bandes. On peut ainsi remonter 2 mois en arrière de semaine en semaine.
- Une sauvegarde quotidienne de niveau 9. Eventuellement sur disque pour faciliter les opérations.

Lors de la restauration, un maximum de 3 bandes seront nécessaires. Si la sauvegarde quotidienne se fait sur disque, cela accélère la sauvegarde et facilite la restauration d'un fichier si on s'en rend compte rapidement mais cela rend l'information plus vulnérable.

Il ne faut pas oublier que `dump` sauvegarde une partition et que certains des fichiers utilisateurs peuvent se trouver sur une autre partition (mails par exemple). Il faut prévoir uen stratégie pour ces fichiers également.

Ne pas oublier non plus que des fichiers détruits entre deux sauvegardes peuvent réapparaître.

10.4.2 Le système

Si le système de fichiers a été correctement partitionné, les fichiers qui sont suceptibles d'être modifiés sont regroupés (dans /etc qui se trouve probablement dans la partition principale). Ces fichiers sont modifiés à un rythme beaucoup plus faible que les fichiers principaux. De plus, leur volume est peu important. Pour ces fichiers, la commande `tar` exécutée à intervalle régulier (hebdomadaire par exemple) peut s'avérer suffisant.

Le reste du système (et les logiciels tierces) peuvent être sauvegardés de façon discrétionnaire lors d'une installation ou d'une mise à jour.

10.5 Quelques conseils précieux

Reprenons ici quelques conseils bien utiles lorsqu'on à la tâche de sauvegarder les informations.

- Centraliser et automatiser les sauvegardes : pour une plus grande simplicité.
- Etiquetter les bandes et se donner un accès rapide à la table des matières
- Choisir un intervalle de sauvegarde raisonnable
- Choisir les systèmes de fichier avec précaution
- Essayer de faire tenir les sauvegardes sur une seule bande
- Entreposer les bandes à l'extérieur
- Protéger les sauvegardes : l'information sur bande est vulnérable ; il s'agit d'un trou énorme dans la sécurité des données
- Limiter autant que possible les activités pendant les sauvegardes
- Vérifier les bandes ! Vérifier les bandes ! Vérifier les bandes !
- Préparer les scénarios catastrophes.

Chapitre 11

Les fichiers journaux

Un journal (*log* en anglais) est un fichier qui garde une trace de l'activité d'un composant. De nombreux programmes (parties du système) sauvent dans ces journaux des informations sur ce qu'ils font. Ils sont utilisés en cas de problème (idéalement, pour les prévenir). Nous allons voir les principaux fichiers journaux puis le système intégré `syslog`.

11.1 Les fichiers journaux classiques

11.1.1 Intérêts

Ces fichiers peuvent servir pour :
- prévenir une intrusion ou trouver le reponsable
- détecter la source et les causes d'une panne
- analyser les performances du système
- calculer des statistiques sur l'utilisation d'un service

11.1.2 Localisation

Malheureusement, la localisation et les noms de ces journaux sont tout-à-fait incohérents d'un système à l'autre mais également au sein d'un même système. Ainsi, on trouve souvent ces fichiers dans `/var/adm/`, `/var/log/`, `/var/cron/`, `/usr/adm`, ...

Le fichier *log* effectivement utilisé par un programme peut être
- hardcodé
- une option de démarrage
- déterminé par un fichier de configuration
- définit via *syslog*.

11.1.3 Stratégie de conservation

Ces journaux ont tendance à croître rapidement et si rien n'est fait, ils peuvent vite saturer les disques. Quelle stratégie adopter ?

1. On peut décider de ne rien garder
2. On peut les remettre à zéro régulièrement
3. On peut faire une rotation et garder quelques fichiers
4. On peut les archiver

Ne rien garder

Soit en indiquant de ne pas créer ces fichiers journaux, soit en les détruisant régulièrement. Cette dernière technique ne garantit pas de pouvoir remonter loin dans le temps. Hors, c'est souvent utile de pouvoir remonter de quelques semaines. Une intrusion, par exemple, ne se détecte pas toujours tout de suite.

Une rotation des fichiers

Régulièrement, le journal est copié dans un fichier avec un numéro avant d'être réinitialisé. Ces fichiers subissent une rotation (1 devient 2, 2 devient 3, ...). Le dernier est perdu. Ils peuvent être compressés afin de gagner de la place. Par exemple, on peut effectuer une rotation hebdomadaire et garder 4 exemplaires. On peut ainsi remonter jusqu'à 1 mois.

Archiver les journaux

En combinaison avec la stratégie précédente, il est possible de sauver les fichiers sur bande avant de les détruire. Cela sera parfois imposé par le site pour des questions légales ou d'audit.

Remarques

Certains journaux sont gardés ouverts en permanence et il est dangereux de les manipuler directement. Il vaut mieux tuer le processus qui se l'accapare puis le relancer.

Des scripts de domaine public existent pour la rotation des fichiers. Souvent même, ils sont déjà intégrés au système et lancés automatiquement via `cron`.

11.1.4 Quelques fichiers importants

Nous n'allons pas passer en revue tous les fichiers journaux utilisés sur un système mais en voici quelques principaux.

messages est utilisé par le système pour des messages très variés : démarrage du système, de démons, problèmes

syslog est également abondamment utilisé par le système et les autres composants pour renseigner sur leur activité

sulog renseigne sur les `su` qui ont été effectués

sudo.log idem pour les `sudo`

lpd-errs erreur du système d'impression LPD

xdm-erros erreur du système XDM

11.2 Le système intégré `syslog`

syslog est un mécanisme centralisé de gestion des messages de trace. Il fournit

- pour le programmeur, un outil simplifié et standardisé pour la création de traces
- pour l'administrateur, un outil centralisé pour la gestion de ces traces

Ce système est composé de trois parties

- le démon `syslogd` qui répond aux requêtes des programmes d'ajout d'une trace et les dirige au bon endroit
- les appels systèmes utilisés par les programmes
- une commande interactive d'ajout d'une trace

Le démon `syslogd` est piloté par un fichier de configuration qui détermine ce qu'il faut faire des messages en fonction de

- sa provenance (type) : kern, mail, lpr, user, ...
- son importance (niveau) : emerg, error, warning, debug, ...

Un message peut être

- jeté
- envoyé sur les écrans des utilisateurs connectés
- sauvé dans un fichier
- passé à une autre machine (centralisation)

11.2.1 Le fichier de configuration

Le comportement de syslog est déterminé par le fichier `/etc/syslog.conf` qui est un tableau à 2 colonnes. La première indique les messages concernés, la deuxième ce qu'il faut en faire.

Ex:

```
*.emerg                         *
*.crit                          /dev/console
*.warning; daemon,auth.info     /var/adm/messages
local0.*                        @loghost
```

Remarques

- Le niveau donné est celui *à partir duquel* la règle s'applique.
- Certains système, comme `Linux`, offrent plus de souplesse ; on peut indiquer un niveau exact ou prendre la négation de niveaux.
- On ne peut avoir qu'une seule action par ligne mais on peut répeter une même condition sur 2 lignes différentes.
- Si on modifie ce fichier, il faut prévenir `syslogd` par un `kill -1`.

11.2.2 logger

Cette commande fonctionne comme les appels systèmes dans un programme. Elle permet d'envoyer un message au démon `syslogd` qui va le traiter en fonction de son fichier de configuration. Elle est utile :

- pour tester le fichier de configuration
- dans les scripts locaux pour utiliser ce mécanisme centralisé de trace

Ex:

```
[marco@localhost marco]$ logger -p kern.emerg "Formatting hard disks..."
[marco@localhost marco]$
Message from syslogd@localhost at Fri Mar  7 11:09:50 2003 ...
localhost marco: Formatting hard disks...
```

Remarques

On voit que tout le monde peut utiliser ce mécanisme. Il faut donc bien vérifier la source d'un message dans un fichier journal.

Deuxième partie

Unix en réseau

Chapitre 12

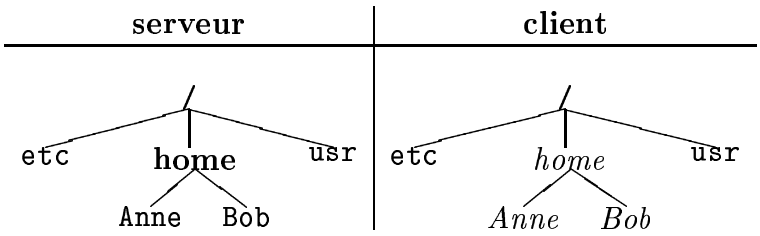
Network File System

NFS, le Network File System permet à une machine de *partager* une partie de ses fichiers, les rendant ainsi accessibles à partir d'autres machines. Il a été introduit dans les années 80 par SUN.

Les avantages sont :

1. un gain de place
2. une même vision des fichiers sur toutes les machines.

12.1 Mécanisme



Le serveur *exporte* un dossier (ici `/home`), rendant ainsi accessible tout ce qu'elle contient.

Le client *monte* explicitement ce dossier. Ensuite, tout est transparent. L'accès à ces fichiers se fait comme si ils étaient présents localement.

Remarques :

- NFS est implémenté sur le protocole RPC (Remote Procedure Call) qui utilisera TCP ou UDP.
- On en est à la version 3 à préférer à la version 2, plus lente.
- Il faut être attentif à la gestion des UID et GID. On rappelle que root sera généralement considéré comme un utilisateur normal au travers de NFS.

12.2 Serveur NFS

Le serveur doit *exporter* (on dit aussi *partager*) ses fichiers. Deux protocoles sont alors mis en place. Le premier pour le montage des dossiers. Et un autre différent pour les accès aux fichiers.

12.2.1 Exportation

Le démon s'occupant du montage de dossier NFS (sur le serveur) est `mountd` (parfois `rpc.mountd`). Il se base sur une fichier de configuration pour savoir quoi exporter et comment. Il existe également une commande qui permet d'ajouter/supprimer dynamiquement des dossiers à exporter.

Le nom du fichier (ainsi que son format) et de la commande peut varier d'un système à l'autre.

Export

Sur de nombreux systèmes, le fichier de configuration est `/etc/exports`

Ex:

```
/usr/csoft      -access=cso5:cso6:cso7:cso8
/usr/share/man -ro
```

où on indique qu'on exporte `/usr/csoft` mais uniquement pour ces 4 machines. Par contre, `/usr/share/man` est exporté au monde entier mais en lecture seule.

La commande est alors `exportfs`. Elle permet de visualiser ce qui est exporté et d'ajouter (supprimer) des ékéments. Après avoir modifié le fichier, il faut demander au démon de se resynchroniser. Cela se fait avec via la comamnde `exportfs -av`.

Sous `Linux` le format du fichier `/etc/exports` est différent. On trouvera plutôt des lignes du genre

```
/usr/csoft cso5(rw) cso8(ro)
```

pour indiquer qu'on donne l'accès en écriture à `cso5` mais uniquement en lecture pour `cso8`

share

`Solaris` a décidé de ne pas faire comme tout le monde et utilise plutôt le fichier `/etc/dfs/dfstab` qui ressemble à

```
share -F nfs -o rw=cso8 /usr/csoft
```

Pour visualiser ce qui est exporté : `share`. Pour demander au démon de relire le fichier : `shareall`. Le fichier contient en fait des commandes.

Options

Parmi les différentes options de ces fichiers et commandes, on trouve
- La possibilité de ne permettre que la lecture ou permettre également l'écriture. Eventuellement en fonction de la machine
- Indiquer les machines pour lesquelles le *root* restera un super utilisateur et pas *nobody* comme c'est normalement le cas
- Empêcher de créer des fichiers avec le SETUID bit à 1 via NFS
- Empêcher de monter des sous-dossiers d'un dossier exporté

12.2.2 Service de fichiers

Le démon qui répond aux requêtes d'accès aux fichiers via NFS est `nsfd` (parfois `rpc.nfsd`). Lorsqu'il est lancé, il prend un argument qui est le nombre de copies de lui-même qu'il va devoir créer. Ce nombre peut avoir son importance sur la qualité de service. Trop petit, il entraînera un retard dans les réponses. Trop grand, cela peut saturer le serveur.

12.3 Client NFS

Pour pouvoir utiliser des fichiers exportés, le client doit d'abord *monter* la directory. Pour cela, la commande est `mount`

Ex: Pour monter la directory `/usr/share/man` de `cso8` sur `cso7`

```
mount cso8:/usr/share/man /usr/share/man
```

Comme toujours avec `mount`, la directory `/usr/share/man` doit exister sur `cso7`. Il s'agira en général d'une directory vide.

Pour visualiser les partitions NFS montées, `mount` ou `df` . Ces commandes donnent des informations sur tous les filesystems montés, pas uniquement via NFS.

Pour démonter un fichier, `umount /usr/share/man`.

12.4 Démarrage

La commande `mount` fonctionne au coup par coup. Pour définir tous les fichiers montés au démarrage, on utilise le fichier `/etc/fstab` (`/etc/vfstab` sous `Solaris`)

Ex:

```
mcodutti@mathpc4:cat /etc/fstab
/dev/hda2    swap     swap    defaults   1   1
/dev/hda3    /        ext2    defaults   1   1
/dev/hda1    /dos     msdos   defaults   1   1
none         /proc    proc    defaults   1   1
cso7:/ULB/staff/mcodutti /ULB/staff/mcodutti nfs defaults 0 0
cso:/var/mail            /var/spool/mail     nfs defaults 0 0
```

Les premières lignes donnent les rôles des différentes partitions du disque dur. Les deux dernières concernent NFS.

Parmi les options, citons :

- La possibilité d'indiquer que si le montage pose problème (serveur pas prêt), on peut passer à la suite et revenir plus tard sur le montage problématique.
- La possibilité de ne pas bloquer un appel à un accès fichier si le serveur pose problème.

12.5 Statistiques

Pour examiner le comportement du système, on peut utiliser la commande `nfsstat`

- `nfsstat -c` donne des statistiques pour le client
- `nfsstat -s` fait de même pour le serveur

12.6 Montage automatique

Avec NFS, tous les dossiers réseaux sont habituellement montés au moment du démarrage de la machine une fois pour toutes. Cela offre quelques désavantages. Ainsi, si un serveur plante, c'est tout un ensemble de clients qui s'en voient affectés même si la dossier n'est pas utilisé. De même, NFS n'offre aucune alternative pour changer de serveur en cas de problème.

Un système de montage automatique vient se greffer sur NFS pour monter les dossiers uniquement en cas de besoin. Ils seront automatiquement

démontés après un certain temps d'inactivité. Le serveur utilisé pourra être choisi parmi une liste. Les deux systèmes les plus utilisés sont `automount` et `amd`.

- `automount`, plus simple, avait toutefois la réputation d'être bogué ; problème à présent résolu
- `amd` est un peu plus complexe à configurer mais plus riche.

`automount` est la solution originelle de SUN. On le trouve sur la plupart des systèmes à quelques variantes prêt. C'est lui que nous allons étudier.

12.6.1 automount

Ce démon se configure via plusieurs types de cartes (des fichiers en fait)

Les cartes indirectes

De nom libre (mais généralement `/etc/auto.xxx`). Permettent de monter plusieurs systèmes de fichier attachés à un même dossier. Ce dossier est indiqué dans un autre fichier. Ex:

```
math math:/home
info info:/home
```

Les cartes directes

Au contraire des cartes indirectes, le point de montage est explicite et absolu. On trouve le plus souvent une seule de ces cartes (de nom `/etc/auto.direct`) Ex:

```
/usr/share/man brol:/usr/share/man
```

La carte principale

Unique, elle liste les cartes directes et indirectes à effectivement utiliser. Pour les cartes indirectes, elle indique également le point de montage. Son nom est généralement `/etc/auto.master`). Ex:

```
/home  /etc/auto.home
/-     /etc/auto.direct
```

12.6.2 Les cartes exécutables

Plus rare mais très puissant. Si le fichier est un exécutable, `automount` va l'exécuter et sa sortie doit être un fichier de configuration. Peut être utilisé par exemple pour fournir un fichier de configuration adapté à la machine à partir d'un fichier de configuration centralisé.

12.6.3 Options

Tous ces fichiers acceptent des options et des variantes. Ainsi,
- on trouve la plupart des options NFS
- on a la possibilité d'indiquer plusieurs serveurs (un ordre est possible) fournissant le même dossier (généralement, en lecture seule)
- des raccourcis existent pour monter, par exemple, tous les dossiers exportés d'une machine.

12.6.4 Les démons et commandes

Sur certaines machines, le démon s'appelle `automountd` et est géré via la commande `automount`. Sur d'autres, le démon est `automount` qui n'a pas de commande associée.

Chapitre 13

Network Information Server

NIS, le Network Information Server permet le partage de fichiers systèmes par plusieurs machines. Les fichiers sont maintenus sur une seule machine, le serveur, et chaque client interroge ce fichier centralisé plutôt qu'une version locale. Cela facilite grandement le travail d'administration.

Quels fichiers partager ? De bons candidats sonts :

- `/etc/passwd` et `/etc/shadow`
- `/etc/hosts`
- `/etc/group`
- `/etc/mail/aliases`
- `/etc/printcap`

NIS n'est pas la seule réponse à ce problème, il existe des systèmes plus simples à mettre en place qui peuvent répondre de manière satisfaisante aux besoins. C'est ce que nous allons d'abord voir.

13.1 `Rdist`

`rdist` permet de copier des fichiers d'une machine centrale (serveur) vers des clients. Un peu comme un `rcp` (remote copy) mais en plus riche et plus souple. Le plus grand apport étant de ne copier que ce qui a été modifié depuis la dernière copie.

Il peut être une bonne alternative à NIS et même à NFS. Par raport à ces 2 systèmes, on épinglera les avantages et les inconvénients suivants.

- Une perte de place puisque le fichier est copié.
- Un retard (configurable) dans la mise à jour des fichiers des clients.
- Une meilleure résistance à des pannes réseau.
- Le devoir de se connecter explicitement sur le serveur pour y modifier un fichier. Cela peut être un inconvénient comme nous le verrons.

– Une moins grande sécurité comme nous le verrons.

Si on utilise NFS pour pouvoir ne gérer qu'une seule copie (qui se modifie peu) mais que l'espace disque n'est pas un problème alors `rdist` est une bonne alternative.

On peut appliquer le même raisonnement avec NIS bien que NIS apporte un plus indéniable pour le partage du fichiers des mots de passe.

Fonctionnement

`rdist` n'est pas standard mais est gratuit et facile à trouver. Il fonctionne de la sorte.

On définit une machine qui contiendra l'original d'un ensemble de fichiers. Les autres machines recevront automatiquement une copie fidèle. Cette copie est effectuée à la demande. Il peut donc y avoir parfois des différences entre les copies et l'original si celui-ci a été modifié depuis la dernière fois que la copie a été demandée. Pour faciliter et accélérer le processus, `rdist` ne copie que les fichiers qui ont étés modifés depuis la dernière copie.

Configuration

Son fonctionnement est déterminé par un fichier de configuration

Ex: pour maintenir une même directory `/usr/local` sur les machines `cso5,cso6,cso7` et `cso8`, on aura sur `cso8` :

```
# cat /etc/distfile
(/usr/local) -> (cso5,cso6,cso7)
    install;
    notify mcodutti@cso.ulb.ac.be
```

Il est alors lancé par la commande `rdist -f /etc/distfile`. Toute modification du système sera notifiée par mail.

Une bonne solution est de le lancer automatiquement toutes les nuits via le *cron*.

Sécurité

`rdist` pose quelques problèmes de sécurité. En effet, il utilise, au niveau du serveur, la commande `rsh` pour copier des fichiers sur les clients. S'il est utilisé pour copier des fichiers systèmes, le serveur doit avoir un accès *root* sur le client ce qui n'est normalement pas possible.

Rappelons comment fonctionne `rsh`.

Sur le client, tourne le démon `rshd`. Celui-ci reçoit les requêtes (pas seulement `rsh`) et les accepte si elles respectent la sécurité qu'on lui a expliquée. Il se base sur les fichiers `/etc/hosts.equiv` (global) et `/.rhosts` (un par utilisateur). Ainsi, avec la configuration suivante

```
cso1#cat /etc/hosts.equiv
cso2
```

n'importe quel utilisateur de cso2 (même `root`) peut se connecter sur cso1.

Ce mécanisme revient un peu à ne pas fermer les portes coupe-feu dans un bâtiment. Si une des machines tombe sous une attaque, les autres tombent aussi vite.

13.2 expect

`expect` est à proprement parler un programme qui permet d'introduire de l'automatisation dans un processus interactif (en mode texte). Il lance un processus interactif et réagit (en simulant un texte tapé) lorsqu'il reconnait le texte envoyé par le procesus. Cette facilité est donc plus large que ce qui nous occupe ici.

Puisque `rdist` a des problèmes de sécurité du fait que c'est au serveur de *pousser* les fichiers sur les clients, on peut imaginer un mécanisme opposé où les clients vont *tirer* les fichiers du serveur.

Le plus simple est de mettre ces fichiers à dispositions sur le serveur via FTP. On utilise ensuite `expect` pour automatiser le transfert. Voici un exemple tiré du livre de référence.

```
spawn /usr/bin/ftp nom_serveur
while 1 {
  expect {
    "Name*: "   {send "nom_user\r"}
    "Password:" {send "mot_de_passe\r"}
    "ftp> "     {break}
    "failed"    {send_user "Accès réfusé!\r"; exit 1}
    timeout     {send_user "Timeout!\r"; exit 2}
  }
}
send "lcd /etc\r"
expect "ftp> " {send "cd pub/etc\r"}
expect "ftp> " {send "get passwd\r"}
expect "ftp> " {send "quit\r"}
exit 0
```

`expect` est une extension du langage `TCL` que vous aurez peut-être reconnu.

13.3 NIS

Venons-en à NIS [1], le Network Information Server introduit par SUN, encore eux, dans les années 80.

NIS est une base de données centralisée pour certains fichiers systèmes. Les plus importants sont `passwd` et `hosts`.

13.3.1 Mécanisme

NIS est composé d'un serveur maître, éventuellement de un ou quelques serveurs esclaves et, bien sûr, de clients. Les fichiers sont

- tenus à jour sur le serveur maître
- répercutés sur les serveurs esclaves
- consultés par les clients

Ce mécanisme

1. permet une gestion plus aisée vu qu'on ne doit maintenir qu'une version des fichiers
2. assure une cohérence entre les machines
3. dans le cas du fichier `passwd`, permet de disposer des mêmes comptes (avec mêmes mots de passe) sur toutes les machines.

Sur le serveur, les fichiers textes sont *compilés* en des fichiers indexés pour un accès rapide à l'information. Les clients vont donc avoir accès aux enregistrements du fichier plutôt qu'au fichier dans son entiereté.

13.3.2 Domaine

Un serveur sert un *domaine*. Ce domaine n'a rien à voir avec le domaine définit par DNS

```
mcodutti@lit1:domainname
lit
# pour modifier le domaine
mcodutti@lit1:domainname brol
```

[1] Pour la petite histoire, NIS s'appelait autrefois *Yellow Pages* (YP). Le nom a du être modifié pour des raisons de copyright. Mais on retrouve encore `yp` dans le nom des commandes.

Sur certains OS, le fichier `/etc/defaultdomain` est lu au démarrage pour déterminer le domaine.

13.3.3 Serveur maître

On peut utiliser comme fichiers distribués

1. les fichiers standards (Ex: `/etc/hosts`)
2. des copies déposées ailleurs (Ex: `/var/yp/nis/hosts`) ce qui est plus propre

Un serveur est généralement configuré lors de l'installation. Après, c'est plus compliqué et je ne vais pas le détailler ici.

Les démons prenant en charge ce service sont

`ypserv`	serveur
`ypxfrd`	propage la base vers les esclaves

Pour créer un serveur maître, il faut introduire la suite d'instructions

```
# cd /var/yp
# domainname brol
# ypinit -m
# ypserv
```

`ypinit` est une programme interactif qui va poser des questions à propos des esclaves, des fichiers à partager, ...

`ypserv` lance les serveurs.

Lorsqu'on modifie un fichier, la commande `make` (ou `ypmake`) met à jour la base et la répercute sur les esclaves.

13.3.4 Serveur esclave

Ils sont optionnels. On les utilise pour décharger le maître et/ou pour pouvoir reprendre le relais en cas de panne de celui-ci.

Pour les créer, il faut introduire la suite d'instructions

```
# cd /var/yp
# domainname brol
# ypinit -s nom_maître
# ypserv
```

13.3.5 Client

Il peut être configuré pour

1. demander à un serveur bien particulier
2. prendre le premier qui répond pour un domaine donné

Le démon du client est `ypbind`

13.3.6 Password

Le cas de fichier `/etc/passwd` est particulier car il est modifié aussi suite à l'intervention des utilisateurs. Dans le cas de NIS, la commande `passwd` n'a plus de sens car le mot de passe n'est plus en local et de toute façons une modification locale serait perdue.

Pour régler ce problème, NIS a introduit la commande `yppasswd` qui remplit le même rôle que `passwd` mais entre en dialogue avec la base sur le serveur (sur lequel tourne le démon `rpc.yppasswd`).

13.3.7 Indiquer que NIS doit être utilisé

Utiliser NIS, ne détruit pas les fichiers locaux qui continuent à exister et qui gardent tout leur sens, soit en cas de panne des serveurs soit, et surtout, lors du démarrage, alors que le client NIS n'est pas encore lancé ou encore parce que le mot de passe du root doit pouvoir rester différent d'une machine à l'autre.

Le système utilise un fichier de configuration `/etc/nsswitch.conf` pour savoir quoi utiliser. Ce mécanisme est d'ailleurs un peu plus vaste puisqu'il permet également de gérer `NIS+` et `DNS`.

En voici un exemple

```
passwd:  files nis
shadow:  files nis
hosts:   nis [NOTFOUND=return] files
```

où l'utilisateur est d'abord cherché en local avant de demander à NIS et où une machine n'est cherchée en local que si NIS est en panne.

Sur certains vieux systèmes, un mécanisme différent était utilisé. Il peut d'ailleurs continuer à cohabiter avec le fichier `/etc/nsswitch.conf`. Avec ce mécanisme, un fichier système qui se termine par `+` indique qu'il faut aller consulter la base NIS si l'information n'est pas trouvée en local. Ce mécanisme est moins souple puisqu'on ne peut pas choisir l'ordre et qu'il ne tient pas compte de DNS.

13.3.8 sécurité

NIS pose quelques problèmes de sécurité. Premièrement, un serveur ne contrôle pas qui sont ses clients. Ce qui permet à n'importe qui de devenir client mais ce n'est en général pas très grave. Deuxièmement, et surtout, certaines versions de NIS ne permettent pas au client d'indiquer qui est son serveur. Dans cette configuration, le client lance un appel *broadcast* sur le réseau et la première machine qui répond est choisie comme serveur même si c'est une machine intrue.

13.3.9 NIS+

NIS+ est le successeur de NIS. Il répond aux mêmes objectifs mais introduit de la sécurité. Toutefois, c'est **TRES COMPLIQUE** et cela ne devient vraiment intéressant que pour de **GRANDS RESEAUX**.

Sun voulant pousser NIS+, dont il est l'instigateur, `Solaris`ne permet plus de configurer la machine en serveur NIS (elle peut toujours être client NIS). Toutefois, un package comblant cette lacune est disponible sur le Web.

Il semble que NIS+ ne va pas s'imposer. On raconte même que SUN, eux-même, ne l'utilisent pas chez eux.

Chapitre 14

Le réseau Internet

14.1 Introduction

Rappelons que différents types d'adresses interviennent dans un réseau IP.

- L'adresse *MAC* (Media Access Control). Ainsi pour Ethernet, un composant sera identifié par une adresse composée de 6 octets.
- L'adresse IP formée de 4 octets et associée à chaque interface réseau.
- Les noms d'hôtes qui sont affectés à des adresses IP et qui sont plus faciles à utiliser pour les humains
- Les ports qui permettent d'adresser un service particulier sur une machine.

14.2 Adresse IP

En IP, une adresse de machine est composée de 4 bytes.

Ex: `134.184.15.9` est l'adresse de `is3`.

Le début de l'adresse indentifie le réseau sur lequel se trouve la machine alors que la fin identifie une machine particulière du réseau. Le tableau suivant reprend les différents découpages entre partie réseau et partie machine. Un `N` indique que le byte fait partie du réseau alors qu'un `H` indique qu'il est dédié à la machine.

classe	byte 1	schéma	
A	1 à 126	`N.H.H.H`	Réseaux majeurs
B	128 à 191	`N.N.H.H`	Sites larges
C	192 à 223	`N.N.N.H`	réseaux de 250 machines
Autres			expérimental

La partie réseau est allouée par des organismes alors que la partie machine est de la responsabilité de l'administrateur du site.

14.3 Subnet

Les sites possédant des adresses de classe B (ou A) peuvent décider de raffiner les adresses en créant un *subnet*.

Ex: à l'ULB, une adresse comme `164.15.125.1` est composée de la partie réseau (`164.15`), de la partie subnet (`125`) et du numéro de la machine (`1`).

Pour distinguer la partie réseau de la partie hôte, on peut donner le *netmask* (`255.255.255.0` dans notre exemple) ou suffixer l'adresse du nombre de bits de la partie réseau (`164.15.125/24` dans notre exemple). Notons que ce nombre n'est pas forçement un multiple de 8.

14.4 Connexion au réseau

La mise en service d'une interface réseau se fait via la commande `ifconfig`.

```
ifconfig eth0 164.15.127.64 up netmask 255.255.255.0
> broadcast 164.15.127.255
```

- **eth0**. nom de l'interface (souvent `ie0, le0, ln0, en0, eth0`). On trouve également `lo` ou `lo0` qui représente une carte fictive, en fait une boucle interne qui crée un réseau composé de la seule machine. Cela lui permet de dialoguer avec elle-même en interne sans passer par le câble.
- **adresse IP**
- **netmask** identifie la partie net/subnet du reste. Ici, on a 3 fois 255 pour indiquer que les 3 premiers bytes font partie de l'adresse réseau (net ou subnet).
- **broadcast** adresse pour les *broadcast*, c-à-d les envois à toutes les machines du même réseau. Utilisé par certains programmes pour poser une question lorsqu'on ne sait pas qui peut répondre ou pour poser une même questions à toutes les machines.

La commande `ifconfig eth0` renseigne sur la configuration courante.

14.5 Le routage

Le *routage* consiste à déterminer le chemin à prendre par un paquet pour aller d'une machine à une autre. Ce n'est pas une tâche facile vu la complexité

du réseau. Il faut tenir compte des divers chemins possibles, réagir en cas de panne d'un noeud, de surcharge, ...

Cela se fait par une suite de décisions locales. Chaque machine dispose d'une table indiquant à qui envoyer le paquet en fonction de l'adresse IP. En général, une machine connaitra les machines proches et un routeur/gateway pour les autres adresses.

La commande `netstat -r` affiche la table de routage

Ex:

```
mcodutti@mathpc4:netstat -r
Kernel routing table
Destination     Gateway         Genmask         Flags M Iface
localnet        *               255.255.255.0   U     0 eth0
loopback        *               255.0.0.0       U     0 lo
default         gate_127.ulb.ac 0.0.0.0         UG    1 eth0
```

A l'ULB, le gateway d'une machine sera toujours une machine dont l'adresse IP est formée des mêmes 3 premiers bytes et dont le dernier est 254. Son nom sera `gate_x` où `x` est le byte 3 de l'adresse de la machine.

Ex: : A partir du résultat de la commande `netstat -r`, déduire la topologie du réseau.

```
#netstat -r -n
Dest            Mask              Gateway          Fl  If
132.236.227.0   255.255.255.0     132.236.227.93   U   eth0
default         0.0.0.0           132.236.227.1    UG  eth0
132.236.212.0   255.255.255.192   132.236.212.1    U   eth1
132.236.220.64  255.255.255.192   132.236.212.6    UG  eth1
127.0.0.1       255.255.255.255   127.0.0.1        U   lo0
```

Pour ajouter un chemin dans la table (Linux) :

```
route add -net 164.15.123.0 eth0
```

Pour définir la route par défaut (Solaris) :

```
route add net default 164.15.125.254 1
```

Sur certains OS (Solaris), le fichier `/etc/defaultrouter` est lu au démarrage pour déterminer le routeur par défaut. Sur d'autres (Linux), ces infos sont reprises dans le fichier `/etc/sysconfig/network`.

Dans les exemples que nous venons de voir, le routage est statique. Il est détetrminé au démarrage via des fichiers de configurations et des commandes. Cette solution convient pour des réseaux simples et des postes de travail. Pour de véritables routeurs dans une topologie complexe, le routage doit devenir dynamique.

Chapitre 15

Domain Name System

Un utilisateur préfère utiliser un nom de machine mais les programmes de bas niveaux fonctionnent avec des adresses IP uniquement. Il faut donc un mécanisme de traduction entre le nom et l'adresse IP.

Cela peut se faire de 2 façons :

1. par un fichier `hosts` (en local ou via *NIS*)
2. par le DNS (Domain Name System)

15.1 Le fichier `hosts`

Le fichier `hosts` reprend les machines et leur adresse IP (1 ligne par machine)

Ex:

```
164.15.125.9      cso8 cso
164.15.123.2      lit1 lit
```

Si on donne plusieurs noms, les suivants sont des alias (noms équivalents)

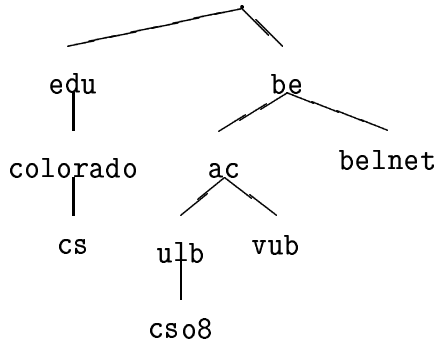
Ce fichier peut exister soit localement (`/etc/hosts`) soit via le réseau (*NIS*).

Avant l'introduction de DNS, cette technique était la seule utilisée. Vu la taille du réseau Internet actuel, un tel fichier ne peut plus contenir toutes les machines du monde. Cela ne serait jamais à jour et des doublons (deux machines ayant le même nom) ne serait pas rare. Il a donc fallu passer à un système plus souple et plus puissant, DNS.

15.2 DNS

Face à l'accroissement du nombre de machines, DNS introduit

1. une hiérarchisation des noms (Ex: `cso.ulb.ac.be`)



   Un noeud correspond à un (sous)domaine alors qu'une feuille correspond à un une machine.

2. une responsabilité distribuée. A chaque niveau, il existe une autorité responsable de l'attribution des noms au niveau inférieur, ce qui assure l'unicité. A chaque niveau, il y a des *serveurs DNS* qui connaissent les machines du niveau inférieur et *la* machine immédiatement supérieure.

Remarquons qu'il existe plusieurs types de serveurs (maître, esclaves, caches, ...) et que tout serveur est aussi client.

L'implémentation la plus utilisée est `BIND`.

15.3 client DNS

Le client est configuré via le fichier `/etc/resolv.conf`. C'est pourquoi on parle de résolveur.

Ex: pour les machines du NO, le fichier doit ressembler à

```
mcodutti@cso8:cat /etc/resolv.conf
domain          ulb.ac.be
nameserver      164.15.125.1
nameserver      164.15.59.200
```

Le fichier `hosts` doit toujours contenir les machines qui sont utilisées avant que DNS ne soit mis en route ou en cas de panne des serveurs

15.4 Ordre

DNS et `hosts` peuvent coexister. Dans ce cas, il s'agit de savoir dans quel ordre ils seront utilisés. En fait, cela dépend fortement du système

`Solaris`

Le fichier `/etc/nsswitch.conf` détermine l'ordre.
Ex: S'il contient la ligne

```
hosts:      nis dns [NOTFOUND=return] files
```

Cela signifie qu'il faut d'abord consulter la NIS puis la DNS. Si la machine n'est pas trouvée, on s'arrête. Si, par contre, on n'a pas eu de réponse pour d'autres raisons (réseau en panne, ...), alors on consulte le fichier local.

`SunOS`

DNS ne peut fonctionner que si NIS fonctionne (à moins d'installer un *patch*). On a toujours l'ordre suivant : NIS puis DNS

`Linux`

L'ordre est déterminé par le fichier `/etc/host.conf` qui contient une ligne qui ressemble à
`order hosts, bind`

15.5 serveur DNS

Pour chaque noeud, on définit

1. un serveur principal qui contient la base de données. A l'ULB, par exemple, c'est `resu1` qui tient ce rôle
2. (optionel) un ou des serveurs auxilliaires qui reprennent une copie de la base. La mise à jour est régulière. Par exemple au NO, on dispose de `plaine1, 164.15.125.1`

Le démon impliqué est `named`. Il se base sur le fichier de configuration `/etc/named.conf`. Ce fichier possède sa syntaxe propre et reprend différentes informations comme

- Le type de serveur
- Les domaines gérés
- Des options globales

Ex: Voici le fichier d'un serveur à l'ESI. (tiré d'un document de Marcel Van Haelen)

```
options {
     directory "/var/named";
     };
zone "labozen.esi.be" in {
     type master;
     file "labozen.hosts";
     };
zone "0.16.172.in-addr.arpa" in {
     type master;
     file "labozen.rev";
     };
zone "0.0.127.in-addr.arpa" in {
     type master;
     file "local.rev";
     };
```

Les informations sont spécifiées dans les fichiers indiqués dans le fichier de configuration, sous forme de texte.

Ex: Voici le fichier d'un serveur à l'ESI. (tiré d'un document de Marcel Van Haelen)

```
;;
$TTL 86400
@  IN SOA toutatis.labozen.esi.be. root.toutatis.labozen.esi.be. (
  99022701
  28800
  7200
  604800
  86400 )
;;
  NS toutatis.labozen.be.

localhost  A     127.0.0.1
toutatis   A     172.16.0.1
           TXT   "Serveur Labo Zen"
www        CNAME toutatis
smtp       CNAME toutatis

kyo        A     172.16.0.2
iznogoud   A     172.16.0.3
```

15.6 Test

Pour tester le bon fonctionnement du DNS, on peut utiliser des outils qui interrogent le DNS et donne l'adresse d'une machine avec des informations sur qui a répondu, ... On a `nslookup` (de moins en moins utilisé), `dig` et `host`.

Ex:

```
mcodutti@cso8:nslookup orca
Server:  plaine1.ulb.ac.be
Address:  164.15.125.1

Name:    orca.vub.ac.be
Address:  134.184.129.10
Aliases:  orca.ulb.ac.be
```

Ex:

```
[marco@pc1 etc]$ dig cso.ulb.ac.be

; <<>> DiG 9.2.1 <<>> cso.ulb.ac.be
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60986
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;cso.ulb.ac.be.                 IN      A

;; ANSWER SECTION:
cso.ulb.ac.be.          5873    IN      A       164.15.130.38

;; AUTHORITY SECTION:
ulb.ac.be.              63215   IN      NS      resu1.ulb.ac.be.
ulb.ac.be.              63215   IN      NS      vnet3.vub.ac.be.

;; ADDITIONAL SECTION:
resu1.ulb.ac.be.        49623   IN      A       164.15.59.200
vnet3.vub.ac.be.        17396   IN      A       134.184.15.13

;; Query time: 22 msec
```

```
;; SERVER: 195.238.2.21#53(195.238.2.21)
;; WHEN: Sat Mar 22 18:04:15 2003
;; MSG SIZE  rcvd: 123
```

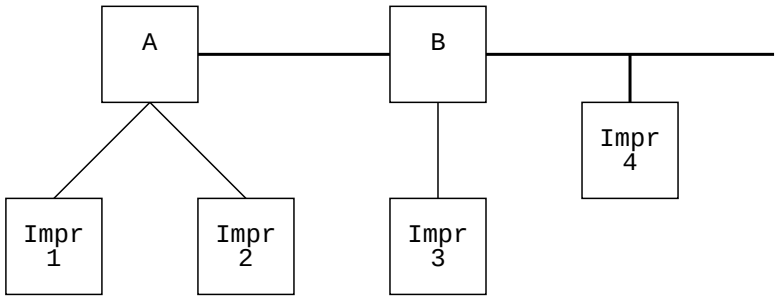
Ex:

```
[marco@pc1 etc]$ host -v cso.ulb.ac.be
Trying "cso.ulb.ac.be"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29308
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;cso.ulb.ac.be.                 IN      A

;; ANSWER SECTION:
cso.ulb.ac.be.          15948   IN      A       164.15.130.38

;; AUTHORITY SECTION:
ulb.ac.be.              84869   IN      NS      vnet3.vub.ac.be.
ulb.ac.be.              84869   IN      NS      resu1.ulb.ac.be.

;; ADDITIONAL SECTION:
vnet3.vub.ac.be.        72984   IN      A       134.184.15.13
resu1.ulb.ac.be.        76679   IN      A       164.15.59.200

Received 123 bytes from 195.238.2.21#53 in 23 ms
```

Chapitre 16

Les impressions

En UNIX, les imprimantes sont facilement partageables d'une machine à l'autre.



Cela sera totalement transparent pour l'utilisateur qui verra toutes les imprimantes de la même façon.

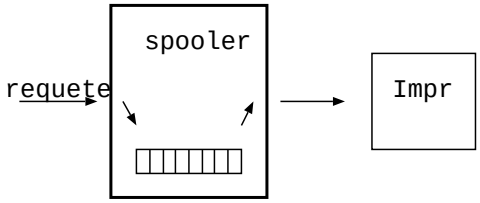
Il existe plusieurs systèmes d'impressions. Traditionnellement, on rencontre celui de `BSD` et celui de `AT&T` , forts différents. Depuis quelques années, d'autres systèmes sont apparus comme LPRng et CUPS, dont nous toucherons un mot.

Voyons quelques aspects communs au deux systèmes historiques.

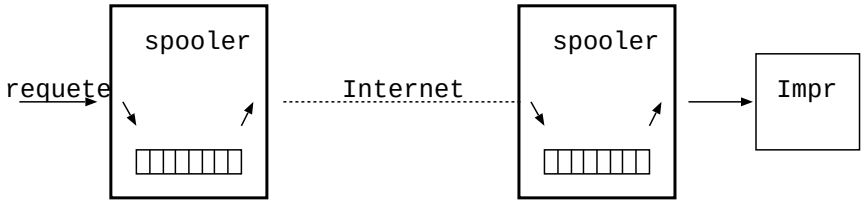
Le spooler

Le **spooler** est un système qui reçoit les requêtes d'impression, les stocke et les envoye à l'imprimante en séquence. Il y a (généralement) un spooler par imprimante.

local



remote



Langages

Il existe différents langages (PDL, Page Description Language) dans le monde des imprimantes. Les plus connus sont Postscript et PCL. Souvent, une conversion du document devra être effectuée avant l'envoi à l'imprimante. Cela se fera via l'utilisation d'un *filtre*.

Les types d'imprimantes

En dehors des imprimantes séries, parallèles et USB, il existe également des imprimantes directement réliée au réseau Internet. Elles possèdent une adresse IP et un spooler, généralement de type BSD.

16.1 Le système `BSD`

Décrivons ici le système d'impression en `BSD` . Donnons d'abord un aperçu des différents composants.

- `lpr` envoie un fichier au spooler
- `lpd` est le démon qui gère le spooler
- Les requêtes sont stockées dans une directory dont le nom ressemble à `/var/spool/lpd/printername/`
- Le fichier `/etc/printcap` décrit les imprimantes
- `lpq` permet de visualiser les fichiers en attente
- `lprm` permet d'enlever un fichier de la file
- `lpc` permet l'administration des imprimantes (stop, restart, ...)

16.1.1 printcap

`/etc/printcap` est un fichier ASCII qui décrit les imprimantes une après l'autre. L'imprimante peut être locale (directement reliée à l'ordinateur), distante (attachée physiquement à un autre ordinateur) ou réseau (directement reliée au réseau)

Ex: imprimante locale

```
math|lp|DEClaser 5100 at Math Gene Bunker:\
        :lp=/dev/ttyS1:\
        :br#19200:\
        :sd=/var/spool/lp/math:\
        :af=/var/spool/lp/math/acct:\
        :lf=/var/spool/lp/math/log:\
        :sh:
```

Ex: imprimante à distance

```
1|cso|Epson ex-1000 at CSO (N4). Connected to cso6:\
        :lp=:\
        :rm=cso6:\
        :rp=cso:\
        :sd=/usr/spool/lp/cso:
```

16.1.2 Commandes orientées utilisateur

Pour imprimer un fichier : `lpr -Pprinter fichier`. Si l'option `-P` n'est pas spécifiée, on prend la variable d'environnement `$PRINTER`. Si elle n'est pas définie, on imprime sur `lp`. Si `lp` n'est pas défini, on imprime sur la première impimante.

Pour visualiser les fichiers actuellement dans la file d'attente : `lpq -Pprinter`.

Ex:

```
mcodutti@lit3:lpq -Pcso
cso is ready and printing
Rank   Owner      Job  Files           Total Size
1st    ssaeedni   657  standard input  1908 bytes
```

Pour effacer un fichier de la file : `lprm -Pprinter job`

Ex:

```
mcodutti@lit3:lprm -Pcso 657
```

16.1.3 Spooler

Le dossier désigné pour le spooling contient des fichiers de log, de statut, ...

Pour chaque fichier envoyé à l'impression, il contient également

- Un fichier de la forme `cfA657...` qui contient des infos sur le fichier (qui, quand, ...)
- Un fichier de la forme `dfA657...` qui contient le fichier à imprimer proprement dit

`lpr` crée ces deux derniers fichiers et prévient `lpd` qui les envoye sur la bonne machine (si remote) ou les incorpore à la liste (si local).

16.1.4 Commande orientée administrateur

La commande est `lpc` dont la syntaxe est : `lpc command printer`, où `command` est

enable/disable pour ouvrir/fermer une file d'attente

start/stop pour déclencher/interrompre l'impression

down/up combine les 2 précédents

clean pour vider la file d'attente

topq pour placer un fichier en tête de liste

restart pour redémarrer le spooler en cas de problème

16.1.5 Permissions

L'impression a distance est régulée via un fichier de permissions. Son nom est `/etc/hosts.lpd`. Il se trouve sur le serveur et liste toutes les imprimantes qui peuvent imprimer sur ses imprimantes. Remarquons que le contrôle se fait au niveau de la machine et pas de l'utilisateur. Le démon `lpd` va également consulter le fichier `hosts.equiv`.

16.1.6 Les filtres

Il est parfois nécessaire de modifier le format du fichier imprimé.

- Pour commencer par une séquene d'initialisation de l'imprimante
- Pour modifier l'ASCII en un format reconnu par l'imprimante
- Pour imprimer plusieurs page sur une feuille
- ...

Pour celà, il faut ajouter l'option `if` dans le fichier de configuration. Ex:

```
math|lp:\
        :lp=/dev/ttyS1:\
        :br#19200:\
        :sd=/var/spool/lp/math:\
        :if=/usr/lib/lpd/filtre1:
```

où `filtre1` est un script recevant le fichier en entrée et fournissant la version modifiée en sortie.

16.1.7 Utilisation non standard

Certains utilisent le système de façon non standard, plus exactement ses capacités de *spooling* pour autre chose que l'impression. Par exemple, on peut l'utiliser pour gérer facilement un journal commun.

```
journal:\
        :lp=/dev/null:\
        :sd=/var/spool/lpd/journal:\
        :if=/usr/local/lib/journal:
```

avec le script

```
#!/bin/bash
cat >> /var/log/journal
```

Il pourra même être centralisé sur une machine.

16.2 Le système SysV

Le module d'impression de SysVn'est guère utilisé que par Solariset HP. A la base, il ne permet pas l'impression via le réseau. Dès lors, ces deux OS lui ont apporté quelques modifications (différentes).

Un avantage de ce système par rapport à celui de BSD est l'introduction de *classes*. Plusieurs imprimantes peuvent être inscrites dans la même classe. Il est alors possible d'imprimer sur une *classe* et l'impression se fera sur l'imprimante la moins chargée de la classe.

Citons les différents composants de ce système.

- `lp` envoye un fichier au spooler
- `lpsched` est le démon qui gère le spooler
- `lpadmin` permet de configurer le système
- Quelques commandes comme `lpshut`, `lpstat`, ...permettent de gérer les imprimantes
- Les requêtes sont stockées dans une directory dont le nom ressemble à `/var/spool/lp/request/printername/`

16.2.1 Configuration

Bien qu'il existe des fichiers texte pour configurer le système, il est recommandé de ne pas les manipuler directement mais via la commande `lpadmin`. En voici quelques exemples.

Ex: Pour ajouter l'imprimante `printer1` de type `model1` connectée sur `/dev/term/a`

```
lpadmin -pprinter1 -v/dev/term/a -mmodel1
```

Ex: Même situation mais la configuration est copiée de l'imprimante `printer2`

```
lpadmin -pprinter1 -v/dev/term/a -eprinter2
```

Ex: Pour ajouter une imprimante à une classe (qui sera créée si elle n'existe pas encore)

```
lpadmin -pprinter1 -cclass1
```

Ex: Pour définir l'imprimante par défaut

```
lpadmin -dprinter1
```

Et il y a bien d'autres options.

16.2.2 Impression

Pour imprimer un fichier : `lp -d printer fichier`. Si l'option `-d` n'est pas spécifiée, on regarde la variable d'environnement `$LPDEST`. Si elle n'est pas définie, on prend l'imprimante par défaut (définie par `lpadmin -d`).

16.2.3 Gestion

- `lpstat` donne des infos sur le statut d'une imprimante, l'état du démon, le contenu des classes, ...
- `cancel` permet d'enlever un fichier d'une file
- `disable` et `enable` pour suspendre/reprendre l'impression
- `lpmove` pour déplacer un travail d'une file à l'autre

16.3 LPRng

Il s'agit d'un système d'impression plus récent tentant de fusionner le meilleur des deux précédents systèmes. Il introduit également une plus grande sécurité via *Kerberos* ou *PGP*.

D'un point de vue pratique, il fournit une compatibilité avec les commandes `BSD` et `SysV`.

16.3.1 Configuration

La configuration de *LPRng* se fait via différents fichiers

- Pour la configuration du système d'impression : `/etc/lpd.conf`
- Pour la configuration des permissions : `/etc/lpd.perms`
- Pour la définition des imprimantes : `/etc/printcap`

Le fichier de configuration des permissions permet d'accepter (ou refuser) l'impression (ou le contrôle du spool ou l'administration) sur une imprimante à des utilisateurs bien précis sur des machines bien précises. Un exemple de ligne est :

```
ACCEPT SERVICE=P,R,M,Q REMOTEHOST=lit REMOTEUSER=mcodutti PRINTER=litpr1
```

où les codes de service correspondent à, respectivement, l'impression, l'envoi dans le spool, la suppression d'un fichier à soi du spool, l'interrogation du spool.

Le fichier de configuration `printcap` est compatible avec les versions `BSD`. Il introduit toutefois quelques nouveautés comme la possibilité d'appliquer un filtre à un fichier qui est destiné à une imprimante réseau. Il fournit également la commande `checkpc` qui permet de vérifier la validité du fichier et l'existence des fichiers qui y sont mentionnés.

16.3.2 Utilisation

La commande est identique à celle du système `BSD` avec quelques options en plus. Citons la possibilité d'imprimer sur une imprimante réseau qui n'a pas été définie dans `printcap`

```
lpr -Plitpr1@lit1%8552 brol
```

16.4 Quelques outils intéressants

Ghostscript permet de convertir un fichier Postscript en de nombreux formats différents. Dans notre cas, l'utilité des de le convertir en PCL pour une impression Postscript.

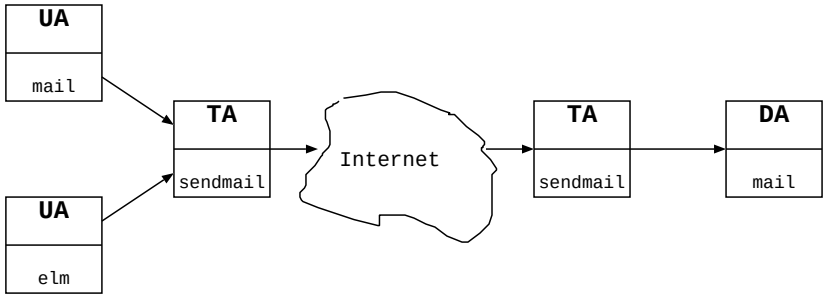
mpage convertit de l'ASCII en Postscript. Permet également une certaine mise en page du Postscript (2 pages sur une seule par exemple).

enscript est un outil équivalent.

Chapitre 17

Le courrier électronique

Voici un schéma décrivant le fonctionnement du courrier électronique.



On y trouve

Agent Utilisateur : interface entre l'utilisateur et le système

Agent de Transport : s'occupe du transport du mail de machine en machine jusqu'à destination

Agent de Réception : délivre le mail sur la machine : boîte aux lettres d'un utilisateur, fichier, programme.

Boîte de réception : la boîte aux lettres d'un utilisateur sera `/var/mail/login` ou `/var/spool/mail/login`. Une solution sous forme de base de données est en train de se développer pour des gros systèmes (fournisseur d'accès par exemple)

Agent d'accès : Permet d'accéder à un boîte aux lettres distantes (IMAP, POP)

Agent d'envoi : Plus récent, il s'insère entre AU et AT et permet de soulager le travail de AT en vérifiant la valider des mails. Un programme comme `sendmail` fait les 2 sur 2 ports différents (25 et 587)

17.1 Adresses

Avant que les adresses se présentent sous la forme qu'on connait, il y avait d'autres systèmes d'adresses, le plus souvent relatives (l'adresse spécifiait l'ensemble des machines par lequel le mail devait passer pour arriver à destination). Une grande part de la complexité des AT actuels (comme sendmail) provient du fait qu'ils gèrent cette disparité liée aux vieux systèmes. Mais ils tombent en obsolescence et sont même parfois désactivés volontairement pour des raisons de sécurité.

Actuellement, toute personne disposant d'un compte `Unix`possède automatiquement une adresse électronique de la forme `login@host.domain`.

Ex: Une personne possèdant le compte `mcodutti` sur `orca.ulb.ac.be` a pour adresse `mcodutti@orca.ulb.ac.be`.

17.2 Structure interne d'un courriel

Un message est composé de trois parties

1. l'enveloppe : détermine la destination du message.
2. les en-têtes : ensemble de paires proriétés/valeurs qui renseignent sur le message (par où il est passé, le sujet, la date, sons statut, ...). Certains sont standards, d'autres pas (ils commencent par `X-`
3. Le corps même du message

Voici un exemple d'un message. Les parties received permettent d'identifier le chemin emprunté.

```
Return-Path: <elevy@litpc49.ulb.ac.be>
Delivered-To: marco@localhost.pc1.localdomain
Received: from localhost (localhost.localdomain [127.0.0.1]) by
        pc1.localdomain (Postfix) with ESMTP id 33C83346E5
        for <marco@localhost>;
        Tue, 22 Apr 2003 08:01:24 -0400 (EDT)
Received: from pop.tiscali.be [62.235.14.101] by localhost with POP3
        (fetchmail-6.1.0) for marco@localhost (single-drop);
        Tue, 22 Apr 2003 14:01:24 +0200 (CEST)
Received: from guppy.vub.ac.be (134.184.129.2)
        by mail.tiscali.be (6.7.015) id 3E9BA4650002B753
        for Marco-Codutti@tiscali.be;
        Tue, 15 Apr 2003 12:56:20 +0200
Received: from mach.vub.ac.be (mach.vub.ac.be [134.184.129.3]) by
```

```
        guppy.vub.ac.be (8.9.1b+Sun/3.17.1.ap (guppy)) id MAA00137;
        Tue, 15 Apr 2003 12:56:20 +0200 (MET DST)
        for <Marco-Codutti@tiscali.be>
Received: from litpc49.ulb.ac.be (litpc49.ulb.ac.be [164.15.123.99]) by
        mach.vub.ac.be (8.9.3/3.13.3.ap (mach)) id MAA27651;
        Tue, 15 Apr 2003 12:56:18 +0200 (MET DST)
        for <Marco-Codutti@tiscali.be>
Subject: cotes projets
From: Eythan Levy <elevy@litpc49.ulb.ac.be>
To: Marco Codutti <Marco-Codutti@tiscali.be>
Content-Type: multipart/mixed; boundary="=-YgewUQqO99c1Gdo2aQkN"
X-Mailer: Ximian Evolution 1.0.8-3mdk
Date: 15 Apr 2003 12:55:00 +0200
Message-Id: <1050404100.9364.36.camel@litpc49.ulb.ac.be>
Mime-Version: 1.0
X-Evolution-Source: mbox:/var/spool/mail/marco
```

17.3 Les alias

Les alias peuvent être définis dans différents endroits

1. Dans le fichier de configuration de certains AU
2. Dans le fichier de configuration de l'AT
3. Dans le fichier de transmission d'un utilisateur (`.forward`)
4. Le champ MX d'une base DNS

17.3.1 aliases

Le fichier s'appelle `/etc/aliases` (ou `/etc/mail/aliases` ou `/usr/lib/aliases`) Il définit des règles d'alias pour modifier le destinataire local d'un mail. Les alias sont de la forme

```
marco:          mcodutti
techniciens:    mcodutti@ulb.ac.be, gpaquet@ulb.ac.be
prof:           :include:/etc/aliases.prof
suggestions:    "/dev/null"
info:           "|programme"
```

On peut donc rediriger vers un autre utilisateur local, une autre adresse, une liste d'adresses (éventuellement définie dans un fichier séparé), un fichier (il sera ajouté en fin de fichier) ou un programme (attention à la sécurité).

Sur de nombreux systèmes, le fichier est compilé pour accélérer l'accès. Après une mise à jour, il faut resynchroniser la version compilée avec `newaliases`

17.3.2 forward

Un utilisateur peut décider de *forwarder* ses mails reçus dans une boîte aux lettres c-à-d les envoyer automatiquement à une autre adresse. Cela permet, par exemple, de tout concentrer dans une seule boîte si on en possède plusieurs. Cela se fait via le fichier `.forward` dans la home

Ex:

```
# cat ~/.forward
mcodutti@cso.ulb.ac.be, \mcodutti@ulb.ac.be
```

Le \ indique que le message envoyé à cette adresse ne devra plus passer par le mécanisme du `.forward` et permet ainsi d'éviter des boucles.

17.3.3 DNS

Pourquoi un courriel addressé à `mcodutti@ulb.ac.be` arrive-t-il ? DNS contient également un champ MX qui permet d'indiquer une redirection pour le mail pour n'importe quel niveau de hiérarchie.

Ex:

```
[marco@pc1 marco]$ dig ulb.ac.be MX
(...)
;; QUESTION SECTION:
;ulb.ac.be.                     IN      MX

;; ANSWER SECTION:
ulb.ac.be.              71802   IN      MX      150 mailhost.vub.ac.be.
ulb.ac.be.              71802   IN      MX      100 mailhost.ulb.ac.be.
(...)
```

17.4 sendmail

`sendmail` est le MTA le plus utilisé. D'autres MTA parfois rencontrés sont `postfix` et `qmail`.

`sendmail` implémente le protocole SMTP ou son extension ESMTP qui ajoute quelques fonctionnalités comme le MIME 8 bits, le cryptage et les recommandés.

Il est configuré via le fichier `sendmail.cf` qui a la réputation, pas vraiment usurpée, d'être très compliqué (un livre de 700 pages le décrit). En pratique, plus personne n'écrit un fichier `sendmail.cf` à partir d'une page vierge. On part toujours d'un fichier d'exemple qui se rapproche de notre situation et on l'adapte.

Depuis la version 8 de `sendmail`, on peut même utilier la commande de macro `m4` pour générer un fichier de configuration à partir d'une version plus lisible.

Le fichier de configuration reprend l'emplacement d'autres fichiers comme
- `/etc/aliases` : la liste des alias
- `/etc/aliases.db` : version compilée
- `/var/spool/mqueue/` : file d'attente des messages

Mais son but principal est de définir comment les adresses sont comprises et de déterminer qui peut s'en charger. Cela se fait via des *règles* qui sont regroupées en ensembles. Certains ensembles ont des rôles prédéfinis
- L'ensemble 0 résout un MDA à partir d'une adresse
- L'ensemble 1 traite l'expéditeur
- L'ensemble 2 traite le destinataire
- L'ensemble 3 effectue un pré-traitement sur toute adresse
- L'ensemble 4 effectue un post-traitement sur toute adresse
- L'ensemble 5 traite les adresses locales

Par exemple, on peut imposer un domaine pour l'adresse de l'expéditeur.

```
S1
R$+<@$*> $:$1<@ulb.ac.be>
```

où
- `$+` correspond à 1 à plusieurs mots
- `$*` correspond à 0 à plusieurs mots
- `$:` stoppe l'utilisation de la règle pour éviter les cycles
- `$1` correspond au premier match de la partie gauche
- Les `<` et `>` ont étés ajoutés par les règles 3

17.5 Tester

On peut obtenir des infos en parlant directement à un AT

Ex:

```
mcodutti@cso1:telnet resu1 25
Trying 164.15.59.200...
Connected to resu1.ulb.ac.be.
```

```
Escape character is '^]'.
220 resu1.ulb.ac.be ESMTP sendmail 8.8.8/3.17.1.ap (resu) ready at Sat, 26 Apr 2
ehlo cso1
250-resu1.ulb.ac.be Hello cso1.ulb.ac.be [164.15.130.31], pleased to meet you
250-EXPN
250-VERB
250-8BITMIME
250-SIZE 16000000
250-ONEX
250-ETRN
250-XUSR
250 HELP
mail from:<mcodutti@cso.ulb.ac.be>
250 <mcodutti@cso.ulb.ac.be>... Sender ok
rcpt to:<mcodutti@cso1.ulb.ac.be>
250 <mcodutti@cso1.ulb.ac.be>... Recipient ok
data
354 Enter mail, end with "." on a line by itself
test 2
.
250 QAA06040 Message accepted for delivery
quit
221 resu1.ulb.ac.be closing connection
Connection closed by foreign host.
```

La commande `sendmail` contient des options où elle passe en mode test. Ici, on vérifie les adresses et la machine du destinataire.

Ex:

```
mcodutti@cso1:/usr/lib/sendmail -bv mcodutti@ulb.ac.be
mcodutti@ulb.ac.be... deliverable: mailer relay,
  host mailhost.ulb.ac.be, user mcodutti@ulb.ac.be
```

17.6 Les listes de diffusions

Le fichier `/etc/aliases` permet déjà de gérer une liste de diffusion mais il lui manque beaucoup de fonctionnalités et de facilités.

- Il faut être `root` pour gérer la liste (une solution simple existe)
- Tout le monde peut envoyer à la liste
- Pas de notion de modérateur
- Pas de solution automatique pour s'ajouter/s'enlever de la liste

– Si une adresse est mauvaise, c'est l'expéditeur qui est prévenu pas le gestionnaire de la liste

C'est pourquoi, on a développé des systèmes spécialisés de gestion de liste comme `majordomo`.

Voici un exemple d'entrées à mettre dans le fichier `/etc/aliases`

```
test:         "|/usr/local/majordomo-1.94.5/wrapper resend -l test test-list"
test-list:    :include:/usr/local/majordomo-1.94.5/lists/test
owner-test:   jarchie
test-owner:   jarchie
test-request:  "|/usr/local/majordomo-1.94.5/wrapper majordomo -l test"
```

Chapitre 18

Intégration Windows/Unix

Pour cette partie, nous vous renvoyons au texte de Pierre Bettens : *"Intégration Windows/Linux dans le cadre du cours d'administration des réseaux"* (`http://pittt.free.fr/index.php/esi.html`)

Crédits

Ce document a été réalisé en LaTeX sous Linux avec les outils suivants

- Kyle comme éditeur
- dvips et ps2pdf pour la production des formats Postscript et PDF.
- acroread (Adobe) pour la visualisation

Table des matières

1 Introduction **1**
1.1 Les tâches de l'administrateur 1
1.2 Les pré-requis 1
1.3 Les aptitudes d'un bon administrateur 2
1.4 Systèmes étudiés dans ce cours 3
1.5 L'histoire d'`Unix` 3
1.6 Présentation de `Solaris` 3
1.7 Présentation de `Linux` 3
1.8 Les sources d'informations 4
1.8.1 Les livres 4
1.8.2 L'aide locale 4
1.8.3 Les sources Internet 4

I Unix isolé **5**

2 Les pouvoirs du *root* **6**
2.1 Droit d'accès aux fichiers et aux processus 6
2.2 Le superutilisateur 7
2.3 Choisir son mot de passe 7
2.4 Devenir root 8
2.4.1 `su` 8
2.4.2 `sudo` 8
2.5 Les autres pseudo-utilisateurs 9

3 Le système de fichiers **10**
3.1 Structure 10
3.2 Les partitions 11
3.3 Monter et démonter des partitions 12
3.3.1 mount 12
3.3.2 liste des systèmes de fichiers 12

3.3.3 umount . . . 13
3.3.4 Et si on ne peut démonter . . . 13
3.3.5 fuser . . . 13
3.3.6 lsof . . . 14
3.4 Vérifier l'occupation des partitions . . . 15
3.5 Types de fichiers . . . 16
3.5.1 Directory . . . 16
3.5.2 Liens physiques . . . 16
3.5.3 Liens symboliques . . . 17
3.6 Les permissions . . . 18
3.6.1 Effets de r,w et x . . . 18
3.6.2 Cas des scripts . . . 19
3.6.3 Le setuid . . . 19
3.7 Quelques commandes pour manipuler les permissions . . . 20
3.7.1 La commande `ls` . . . 20
3.7.2 Modifier les permissions . . . 21
3.7.3 Modifier le propriétaire et le groupe . . . 21
3.7.4 Permissions par défaut . . . 21
3.8 Organisation standard du système de fichiers . . . 22
3.8.1 Les catégories de fichiers . . . 22
3.8.2 Les dossiers de premier niveau . . . 22
3.8.3 La structure de `/usr` . . . 23

4 La documentation **25**
4.1 La pages de manuel . . . 25
4.2 La pages d'infos . . . 27

5 Les utilisateurs **28**
5.1 Enregistrer un utilisateur . . . 28
5.2 Mot de passe . . . 29
5.3 Le champ *GECOS* . . . 30
5.4 Home Directory . . . 30
5.5 Shell . . . 30
5.6 Le fichier `/etc/shadow` . . . 30
5.7 Le fichier `/etc/group` . . . 31
5.8 Supprimer un utilisateur . . . 31
5.9 Les commandes toutes prêtes . . . 32
5.10 Particularités FreeBSD . . . 32

6 Les processus **33**
6.1 Création d'un processus . . . 33
6.2 Les états d'un processus . . . 34
6.3 Les signaux . . . 34
6.3.1 Quelques signaux utiles . . . 35
6.4 Les priorités . . . 35
6.5 Le swap . . . 35
6.6 Les démons . . . 36
6.7 Commandes pour gérer les processus . . . 36
6.7.1 `ps` sous `BSD` . . . 36
6.7.2 `ps` sous `AT&T` . . . 37
6.7.3 Top . . . 37
6.7.4 `kill` . . . 37
6.7.5 `killall` . . . 37
6.7.6 Nice . . . 38
6.8 Nohup . . . 38
6.9 `/proc` . . . 38

7 Le shell **39**
7.1 Les scripts . . . 39
7.2 Initialisation . . . 40
7.3 Les variables . . . 40
7.3.1 `PATH` . . . 41
7.3.2 `MANPATH` . . . 41
7.3.3 `LD_LIBRARY_PATH` . . . 42
7.3.4 Le prompt . . . 43
7.3.5 `TERM` . . . 43
7.3.6 Autres variables . . . 43
7.4 Les alias . . . 44
7.5 Les quotes . . . 45
7.6 Redirections . . . 45
7.7 Substitution de fichiers . . . 46
7.8 History . . . 46
7.9 Ordre d'évaluation d'une commande . . . 47
7.10 Les paramètres . . . 47
7.11 Evaluation d'expressions *booléennes* . . . 48
7.12 Structure . . . 48
7.13 Commandes utiles . . . 49
7.13.1 head/tail . . . 49
7.13.2 grep . . . 49
7.13.3 sort . . . 49

7.13.4 awk 50
7.13.5 uniq 50
7.13.6 sed 50
7.13.7 Exercice récapitulatif 51

8 Démarrage et arrêt **52**
8.1 Le démarrage 52
8.1.1 Le moniteur 52
8.1.2 Le moniteur PC 53
8.1.3 Le moniteur PC et `Linux` 53
8.1.4 Le moniteur `Solaris` 53
8.1.5 Le noyau 53
8.1.6 Détection des périphériques 53
8.1.7 Les processus spontanés 54
8.1.8 Le mode manuel 54
8.1.9 Les scripts de démarrage 54
8.1.10 Les scripts de démarrage `BSD` 54
8.1.11 Les scripts de démarrage `SysV` 55
8.2 Arrêter, redémarrer le système 56
8.2.1 *shutdown* 56
8.2.2 *halt* et *reboot* 57
8.2.3 *fasthalt* et *fastboot* 57
8.2.4 signal à *init* 57
8.2.5 Séquence de touches 58
8.2.6 Eteindre la machine 58

9 Les tâches périodiques **59**

10 Les sauvegardes **61**
10.1 Introduction 61
10.2 Les commandes liées à la sauvegarde 61
10.2.1 dump 61
10.2.2 restore 63
10.2.3 tar 63
10.2.4 mt 64
10.3 Les supports de sauvegarde 64
10.4 Les stratégies de sauvegarde incrémentale 65
10.4.1 Les fichiers personnels 65
10.4.2 Le système 65
10.5 Quelques conseils précieux 66

11 Les fichiers journaux **67**
11.1 Les fichiers journaux classiques 67
11.1.1 Intérêts 67
11.1.2 Localisation 67
11.1.3 Stratégie de conservation 68
11.1.4 Quelques fichiers importants 69
11.2 Le système intégré `syslog` 69
11.2.1 Le fichier de configuration 69
11.2.2 logger 70

II Unix en réseau **71**

12 Network File System **72**
12.1 Mécanisme 72
12.2 Serveur NFS 73
12.2.1 Exportation 73
12.2.2 Service de fichiers 74
12.3 Client NFS 74
12.4 Démarrage 75
12.5 Statistiques 75
12.6 Montage automatique 75
12.6.1 automount 76
12.6.2 Les cartes exécutables 77
12.6.3 Options 77
12.6.4 Les démons et commandes 77

13 Network Information Server **78**
13.1 `Rdist` 78
13.2 expect 80
13.3 NIS 81
13.3.1 Mécanisme 81
13.3.2 Domaine 81
13.3.3 Serveur maître 82
13.3.4 Serveur esclave 82
13.3.5 Client 83
13.3.6 Password 83
13.3.7 Indiquer que NIS doit être utilisé 83
13.3.8 sécurité 84
13.3.9 NIS+ 84

14 Le réseau Internet **85**
14.1 Introduction 85
14.2 Adresse IP 85
14.3 Subnet 86
14.4 Connexion au réseau 86
14.5 Le routage 86

15 Domain Name System **88**
15.1 Le fichier `hosts` 88
15.2 DNS 89
15.3 client DNS 89
15.4 Ordre 90
15.5 serveur DNS 90
15.6 Test 92

16 Les impressions **94**
16.1 Le système `BSD` 95
16.1.1 `printcap` 96
16.1.2 Commandes orientées utilisateur 96
16.1.3 Spooler 97
16.1.4 Commande orientée administrateur 97
16.1.5 Permissions 97
16.1.6 Les filtres 97
16.1.7 Utilisation non standard 98
16.2 Le système `SysV` 98
16.2.1 Configuration 99
16.2.2 Impression 99
16.2.3 Gestion 99
16.3 LPRng 100
16.3.1 Configuration 100
16.3.2 Utilisation 100
16.4 Quelques outils intéressants 100

17 Le courrier électronique **102**
17.1 Adresses 103
17.2 Structure interne d'un courriel 103
17.3 Les alias 104
17.3.1 aliases 104
17.3.2 forward 105
17.3.3 DNS 105
17.4 `sendmail` 105

17.5 Tester 106
17.6 Les listes de diffusions 107

18 Intégration Windows/Unix **109**