Gestion de fichiers

(*File Systems*)

SITE : http://www.sir.blois.univ-tours.fr/~mirian/

Fichier

- Une **unité de stockage logique**.
- Ensemble d'informations en relation entre elles, qui est enregistré sur la mémoire auxiliaire.
- Le **SE établit une correspondance entre les fichiers et les dispositifs physiques**.

Pourquoi nous avons besoin des fichiers?

- Toutes les applications ont besoin d'enregistrer des informations et de les retrouver.
- Un processus peut enregistrer une quantité limitée d'information dans son propre espace d'adressage (virtuel). Problème: espace limité ou mémoire volatile (mémoire vive/cache/registres) et information accessible à un seul processus.
- Plusieurs applications ont besoin de stocker un **grand nombre** d'informations de façon **persistante** (non volatile) et de les rendre **accessibles** à plusieurs processus.

Système de fichiers: Partie du SE responsable de la gestion de fichiers

Deux visions d’un système de fichiers

- **Point de vue de l’utilisateur** : nommage des fichiers, protection et droit d’accès, opération autorisées, etc.
- **Point de vue de l’implantation** : organisation physique d’un fichier sur un disque, gestion des blocs et manipulation des blocs physiques attribués à un fichier, gestion de l’espace libre du disque.

Noms des fichiers (1)

- Un fichier est désigné par son nom (chaîne de caractères).
- Quand un processus crée un fichier, il lui donne un nom. Quand le processus termine, le fichier continue à exister et peut être accédé par son nom.
- Chaque système impose des règles exactes pour les noms de fichier (tous autorisent une chaîne composée de 1 à 8 caractères). **Certains systèmes de fichiers distinguent les majuscules et les minuscules.**
- Nombreux SE gèrent des noms de fichiers en deux parties séparées par un point (Ex: `prog.c`).
  - La partie qui suit le point est une **extension**. Indique, en général le **type** du fichier.
  - Dans MS-DOS taille de l'extension : 1 à 3 caractères.
  - Dans UNIX taille de l'extension : dépend de l'utilisateur. Un fichier peut avoir plusieurs extensions (Ex: `prog.c.Z`).
  - Exemples d'extensions courantes : `.html, .txt, .bak. .ps, .pdf, .tex, .zip, .o`

Noms des fichiers (2)

- Dans certains systèmes (UNIX), les extensions sont simplement **des conventions et ne sont pas imposées par le SE**. D'un autre coté un compilateur C insistera pour compiler des fichiers avec l'extension .c et refusera de faire la compilation dans le cas contraire (quand un compilateur doit lier des programmes C et des programmes assembleur, l'extension lui permet d'identifier le type du programme).
- Windows reconnaît les extensions et leur attribue une action.

Noms des fichiers (3)

SE	Distinction entre les majuscules et les minuscules	Longueur maximale d'un nom de fichier	Longueur maximale d'une extension
Unix/Linux	Oui	255	255
MS-DOS	Non	8	3
Windows 95/98	Non	8	3
Windows NT/2000	Non	255	255

Structures des fichiers

1. Séquence d'octets non structurée
   - Le SE ne connaît pas et ne s'occupe pas du contenu du fichier.
   - Toute signification doit être apportée par les programmes des utilisateurs.
   - UNIX, Windows
   - Offre une flexibilité maximale, car les programmes utilisateurs peuvent mettre ce qu'ils veulent dans les fichiers. Le SE n'aide pas, mais n'impose aucune restriction.
2. Séquence d'enregistrements
   - Un fichier est un ensemble de registres de taille fixe
   - Opérations: *read* (lire 1 registre) et *write* (écrire sur 1 registre)
   - Anciens SE: CP/M
3. Arbre d'enregistrements (pas forcément de la même taille)
   - Chaque registre contient une clé.
   - L'arbre est organisé selon la clé pour permettre un accès rapide.
   - Utilisé avec certains *mainframes*.

Types de fichiers

1- Fichiers ordinaires : Contiennent les informations des utilisateurs

ASCII : contiennent du texte ASCII pur

Ambiguïté de la *fin de ligne* : CR, LF, $CR + LF$ (sous MS DOS)

Peuvent être imprimé tels quels et édité avec n'importe quel éditeur

Binaire : possède une structure interne propre aux programmes qui les exploitent.

L'impression est un charabia aléatoire

Exemple(UNIX) : Le SE exécutera le fichier seulement s'il possède un certain format composé de 5 parties : le *header*, le code, les données, les bits des translation, la table de symboles.

- Le début du *header* est un **nombre magique** qui identifie le fichier comme exécutable.

2- Répertoires :Fichiers système qui conservent la structure du système de fichiers

3- Fichiers spéciaux caractère : Liés aux E/S série

4- Fichiers spéciaux blocs : Liés aux disques

Méthodes d'accès

1. **Accès séquentiel**
   - L'information dans le fichier est traitée en ordre, un enregistrement après l'autre. La lecture dans le désordre n'est pas possible.
   - Éditeurs et compilateurs utilisent cette méthode. Pratique quand le support de stockage était une bande magnétique.
2. **Accès direct**
   - Basé sur le modèle de disques (permettant l'accès direct à n'importe quel bloc de fichier). Le fichier est vu comme une séquence numérotée de blocs d'enregistrement.
   - Les fichiers dont les octets ou les enregistrement peuvent être lus dans n'importe quel ordre sont des **fichiers à accès aléatoire**.
   - Deux méthodes pour spécifier le début de lecture:
     (a) Opération (`read` ou `write`) donne la position n dans le fichier ou la lecture/écriture doit être faite.
     (b) Opération (`seek`) permet de se placer à un endroit donné; ensuite le fichier est lu/écrit séquentiellement avec `read next` ou `write next`.
3. **Dans les SE modernes tous les fichiers sont automatiquement à accès aléatoire**

Attributs des fichiers

- Tous les SE associent (en plus du nom et des données) des informations complémentaires aux fichiers.
- Exemples d'attributs des fichiers : type, emplacement, taille, protection, heure, date, identification de l'utilisateur . . .

Opérations sur les fichiers

- Le SE fournit des appels systèmes pour créer, écrire, lire des fichiers, etc
- Opération le plus courantes:
  1. Créer (Create)
  2. Écire (Write)
  3. Lire (Read)
  4. Ouvrir (Open)
  5. Fermer (Close)
- Exemples des appels systèmes UNIX/Linux relatifs aux fichiers dans la suite ...

... Fichiers

Appels système	Description
`df = open(nom, mode,...)`	Ouverture en lecture, écriture, etc.
`s = close(df)`	Fermeture d'un fichier ouvert
`n = read(df, buffer, nb_octets)`	Lecture d'un nombre d'octets d'un fichier vers un buffer
`n = write(df, buffer, nb_octets)`	Écriture d'un nombre d'octets d'un buffer vers un fichier
`pos = lseek(df, offset, org)`	Déplacement du pointeur de fichier
`s = stat(nom, &buf)`	Obtention d'informations sur le fichier (périphérique, inode, nb de liens durs sur le fichier, propriétaire, groupe, taille, date de création date du dernier accès, date de la dernière modification)
`s = fstat(df, &buf)`	Idem, à partir du descripteur de fichier

Exemple

Un exemple de programme utilisant des appels système de fichiers.

- Programme qui réalise la copie d'un fichier source dans un fichier cible.
- Le programme `copiefichier` peut être appelé avec la commande suivante:
  `copiefichier abc xyz`
  pour copier le fichier `abc` dans le fichier `xyz`. Si le fichier `xyz` existe il est remplacé.

Partitions (1)

- Mini-disque, volumes.
- Partition $\equiv$ disques virtuels.
- Système de fichiers est découpé en partitions.
- Généralement, chaque disque du système contient au moins une partition, qui est une structure de bas niveau dans laquelle résident les répertoires et les fichiers.
- L'utilisateur est concerné seulement par la structure logique des fichiers et des répertoires et peut ignorer complètement les problèmes d'allocation d'espace physique aux fichiers.
- **Un but d'avoir plusieurs partitions pour un SE dans un disque**: sécurité. La division du disque dur en partitions permet un regroupement et une séparation des données. En cas d'accident, seulement les données dans la partition concernant l'accident sont endommagées alors que les autres données sont (en général) préservées.
- Les partitions peuvent aussi être formatées différemment.

Partitions (2)

- Deux **types de partitions principales en Linux**:
  - **Partition de données** : *normal Linux system data* y compris la partition *root*.
  - **Partition swap** : expansion de la mémoire sur le disque dur.
- Linux peut aussi avoir autres types de systèmes de fichier (certains d'entre eux disponibles dans des SE propriétaires).

Répertoires (1)

- Pour conserver une trace des fichiers, les systèmes de fichiers possèdent des répertoires qui sont eux mêmes des fichiers dans nombre des systèmes.
- Un répertoire contient une entrée par fichier.

games	attributs
mail	attributs
work	attributs

Chaque entrée dans la table contient le nom du fichier, ses attributs et l'adresse disque

games	→
mail	→
work	→

Chaque entrée contient le nom et un pointer pour une structure de données contenant les attributs et l'adresse disque

Répertoires (2)

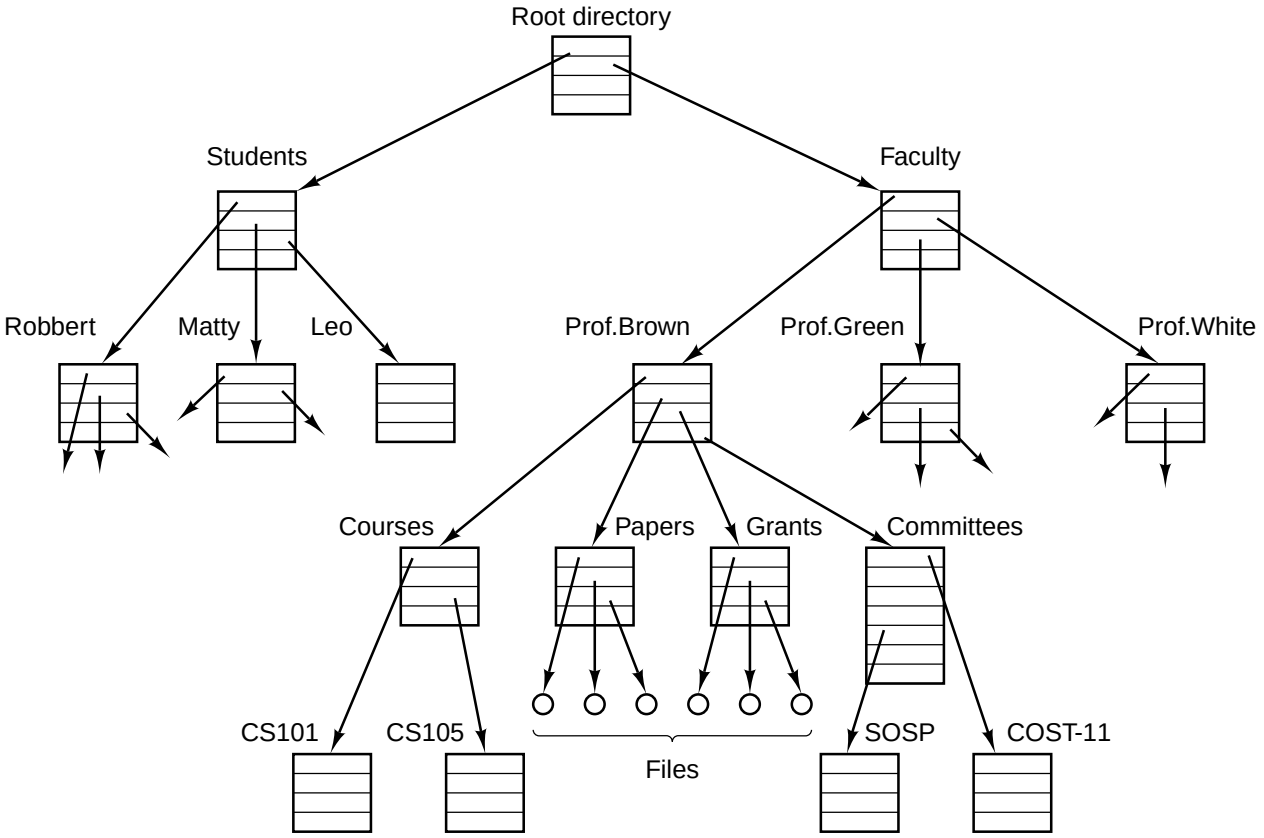
- Sur les SE modernes les fichiers sont organisés en une structure hiérarchique.
- L'identification d'un fichier se fait par son nom précédé d'un **chemin absolu** (c-à-d par rapport à la racine de l'arborescence) ou **relatif** (c-à-d par rapport au répertoire courant).
- Une structure hiérarchique n'est pas forcément un arbre, ça peut être un graphe acyclique (à cause des liens multiples).
- Exemples des chemins absolus (les séparateurs peuvent changer selon le SE)

```
Windows:  \usr\jlb\courrier
Linux :   /usr/jlb/courrier
```

- Le chemin relatif fonctionne conjointement avec le concept de **répertoire de travail** ou **répertoire courant**. **Tous les chemins d'accès qui ne commence pas à la racine sont relatifs au répertoire courant.**
- Exemple (Linux): Les commandes
  `cp /usr/ast/mailbox /usr/ast/mailbox.bak` et
  `cp mailbox mailbox.bak`
  font exactement la même chose si le répertoire de travail est `/usr/ast`

Répertoires (3)

- La plupart des SE qui possèdent une structure hiérarchique de répertoires ont deux entrées particulières dans chaque répertoire :
  1. “.” le répertoire courant
  2. “..” le répertoire parent


Root directory
Students
Faculty
Robbert
Matty
Leo
Prof.Brown
Prof.Green
Prof.White
Courses
Papers
Grants
Committees
CS101
CS105
Files
SOSP
COST-11

Répertoires (4)

Appels système	Description
`s = mkdir(chemin, mode)`	Création d'un nouveau répertoire.
`s = rmdir(chemin)`	Suppression d'un répertoire
`s = link(fichier, lien)`	Création d'un lien
`s = unlink(chemin)`	Suppression d'un lien (et éventuellement du fichier)
`s = chdir(chemin)`	Changement du répertoire courant
`dir = opendir(chemin)`	Ouverture d'un répertoire en lecture
`s = closedir(dir)`	Fermeture d'un répertoire
`dirent = readdir(dir)`	Lecture d'une entrée du répertoire
`rewinddir(dir)`	Retour au début du répertoire pour une nouvelle lecture

Les Liens (1)

- L'appel `link` permet au même fichier d'apparaître sous plusieurs noms, souvent dans des répertoires différents.
- Permet un partage de fichier. Par exemple, dans une équipe, chacun a l'impression d'avoir le fichier dans son répertoire.
- **Il s'agit d'un partage et non d'une duplication**. Tout changement par un des membres de l'équipe étant instantanément visible des autres.

Exemple

`/home/jules`

16	mail
81	games
40	test

`/home/jim`

31	bin
70	memo
59	progA
38	prog

```
link ("/home/jim/memo" , "/home/jules/note")
```

Les Liens (2)

- Le fichier `memo` de `jim` devient visible dans le répertoire de `jules` sous le nom `note`.

`/home/jules`	
16	mail
81	games
40	test
70	note

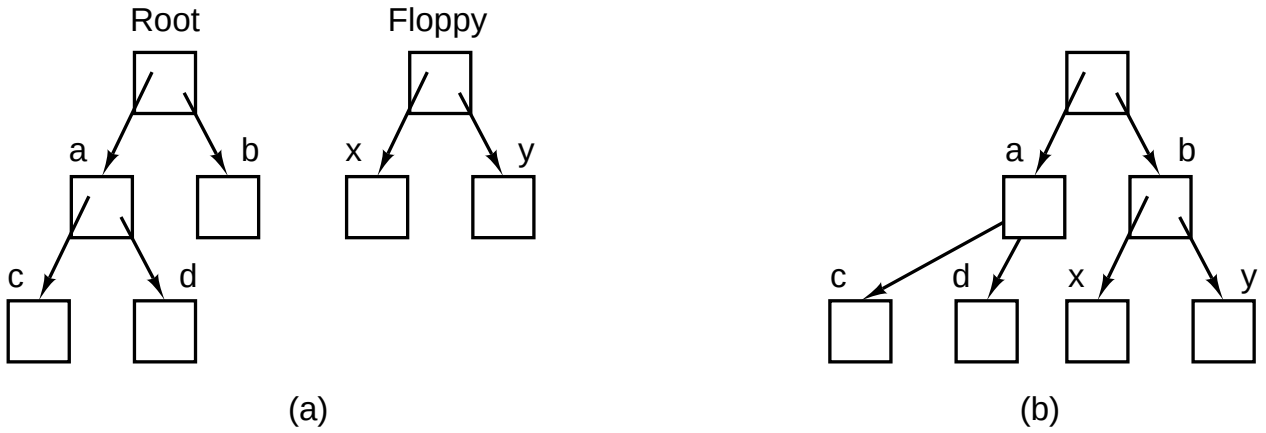
`/home/jim`	
31	bin
70	memo
59	progA
38	prog

- Chaque fichier UNIX/Linux possède un identificateur entier unique appelé **i-number**. Cet entier est un index dans une table d'**i-nodes** qui indique le placement d'un fichier dans le disque, son propriétaire, etc.
- Un répertoire est un fichier contenant un ensemble de pairs *(i-number, ASCII name)*. Exemple: le i-number de `mail` est 16.
- `link` crée un nouvelle entrée de répertoire et reprend un i-node existant
- Exemple: deux entrées ont le même i-number (70) et référencent le même fichier.
- Si l'une des entrées est supprimée à l'aide de `unlink`, l'autre demeure. Si la second disparaît également, UNIX/Linux voit que plus aucune entrée ne

Le `mount` (1)

- `mount` permet de fusionner deux systèmes de fichiers en un seul.
- Situation typique: utilisateur introduisant une disquette (ou un clé usb) dans le lecteur de la machine.
- L'appel `mount` permet d'attacher le système de fichiers de la disquette (clé) sur l'arborescence principale.
- Exemple: `mount ("dev/fd0", "/mnt", 0)`
- Paramètres:
  1. nom d'un fichier spécial en mode bloc (pour le périphérique 0).
  2. point d'encrage du second système de fichiers sur le premier.
  3. indique si le montage se fait en lecture seule ou en lecture et en écriture
- Après l'appel `mount` un fichier sur le périphérique 0 peut être accédé simplement en donnant son chemin complet depuis la racine, sans référence au périphérique d'origine.
- Quand on a plus besoin d'un système de fichier on peut le démonter (`umount`)

Le `mount` (2)



Fichiers et répertoires importants (1)

- **Kernel**: le coeur du SE.
- **Shell** (Wikipédia)
  Le shell du système d'exploitation peut prendre deux formes distinctes :
  - interpréteur de lignes de commandes (CLI, pour *Command Line Interface*) : le programme fonctionne alors à partir d'instructions en mode texte;
  - shell graphique fournissant une interface graphique pour l'utilisateur (GUI, pour *Graphical User Interface*).

  Comment savoir quel shell ont est en train d'utiliser?

  ```
  echo $SHELL
  ```

Fichiers et répertoires importants (2)

- **Home directory**
  - Destination défaut, en général un sous répertoire de `/home`
  - Le chemin correct pour son *home directory* est stocké dans la variable d'environnement $HOME$. Pour connaître sa valeur:

```
echo $HOME
```

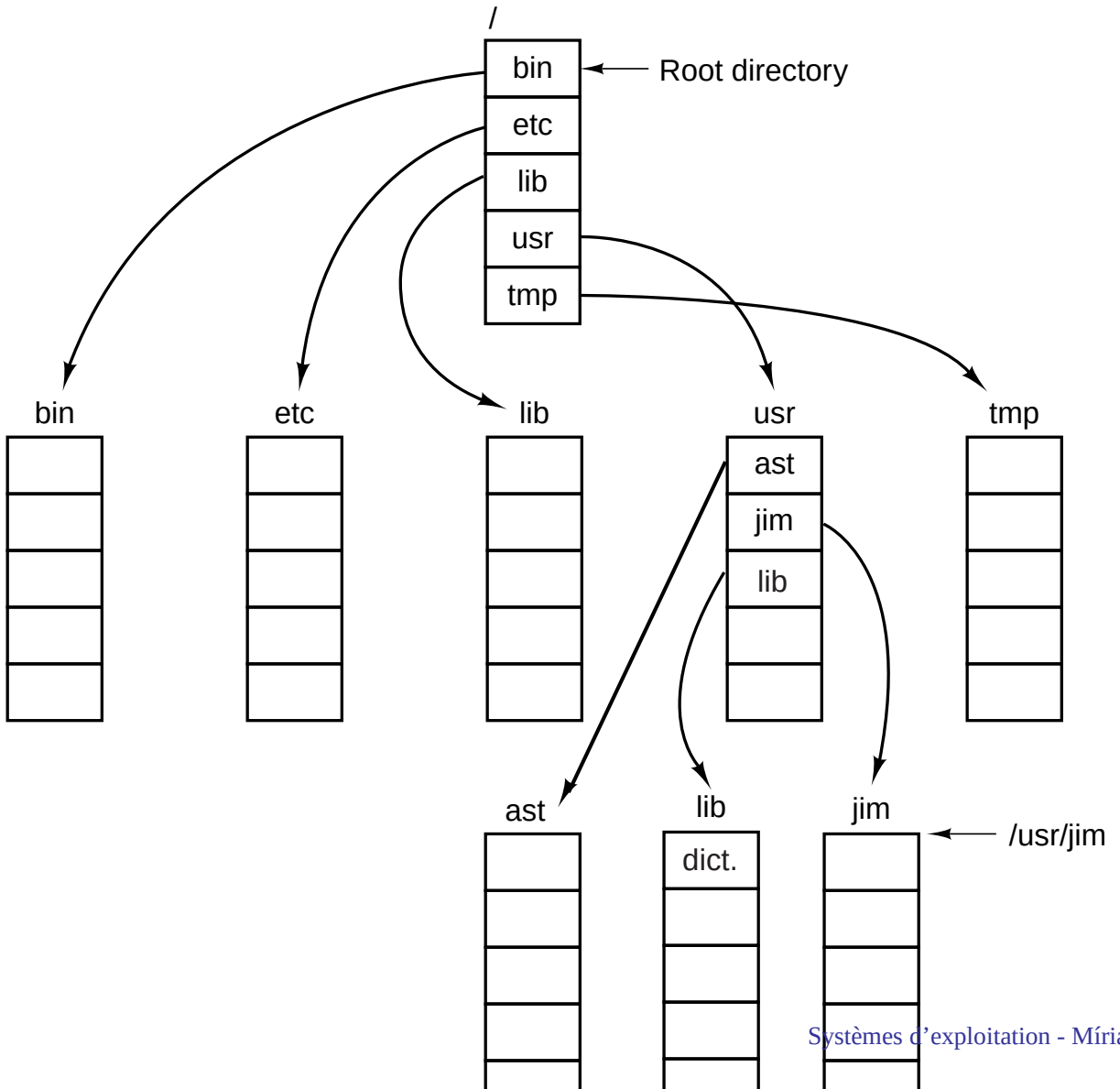
Organisation du système de fichier Linux (1)

Quelques sous répertoires du répertoire `root`

Répertoire	Contenu
`/root`	*Home directory* de l'administrateur système. Attention à la différence entre le répertoire root (/) et le répertoire de l'utilisateur *root*.
`/bin`	Programmes communs, partagés par le système, l'administrateur système et les utilisateurs
`/boot`	Les fichiers de démarrage et le *kernel* (`vmlinux`)
`/dev`	Références à tous les périphériques qui sont représentés par des fichiers ayant des propriétés spéciales
`/etc`	Importants fichiers de configuration
`/home`	*Home directories* des utilisateurs ordinaires
`/initrd`	(on some distributions) Information pour le *boot*.
`/lib`	Libraire. Contient des fichiers nécessaires pour différents types de programmes

Organisation du système de fichier Linux (2)

Répertoire	Contenu
`/lost+found`	Toutes les partiaitons on ce répertoire. Fichiers sauvegarder pendant des failles
`/misc`	Miscellaneous
`/mnt`	Point de montage standard mount pour les systèmes de fichiers extérieurs
`/proc`	Contient information sur les ressources systèmes.
`/sbin`	Programmes pour l'utilisation du systèmes et de l'administrateur
`/tmp`	Espace de stockage temporaire pour le système Nettoyer à chaque *reboot*
`/usr`	Programmes, libraries, documentations, etc. pour tous les programmes des utilisateurs
`/var`	Stockage pour toutes les variables de fichier et des fichiers temporaire crées par l'utilisateur (log files, files downloaded, print spooler area, et


/
bin
etc
lib
usr
tmp
Root directory
bin
etc
lib
usr
tmp
ast
jim
lib
ast
lib
jim
dict.
/usr/jim

Shell - Commandes de base

Commande	Action
`ls`	Lister les fichier d'un répertoire
`passwd`	modifier le mot de passe de l'utilisateur courant
`file` filename	montrer le type du fichier *filename*
`cat` textfile	montrer le contenu du fichier texte à l'écran
`man` command	lire la documentation sur la commande en question

Shell - manipulation des fichiers et répertoires

Commande	Action

`mv` file1 file2
`mv To_Do done`
`mv ../rep[1-4].doc rep/Resto/`
	Déplacer un fichier (ou changer son nom)

`cp` fileSource newFile	Copier un fichier ou un répertoire
`rm` file	Supprimer un fichier

PATH: variable d'environnement

- Quand nous voulons exécuter une commande, en général, nous ne voulons pas donner le chemin entier (correspondant à son emplacement).
  Exemple: Nous faisons seulement `ls` et non `/bin/ls`
  En sachant que la commande `ls` est dans le répertoire `bin`.
- Pour pouvoir faire cela la variable d'environnement PATH doit être établie.
- La variable d'environnement PATH garde les répertoires où les exécutables doivent être cherchés (pour simplifier la tâche de l'utilisateur)
- Pour voir ce que vous avez dans PATH:

  ```
  echo $PATH
  ```

- Le SE cherche un exécutable dans les répertoires de PATH et s'arrête dès qu'il le trouve (il ne cherche pas dans les répertoires suivants).

Exemple

```
jumper:~> wc -l test

(Ctrl-C)
jumper:~> which wc
wc is hashed (/home/jumper/bin/wc)

jumper:~> echo $PATH
/home/jumper/bin:/usr/local/bin:/usr/local/sbin:/usr/X11R6/bin:\
/usr/bin:/usr/sbin:/bin:/sbin
```

- Utilisateur *jumper* veut utiliser `wc` (*word counter*) pour compter les linges d'un fichier. Rien se passe et il arrete la procédure avec `Ctrl+C` .
- Il fait un `which` pour savoir quel `wc` a été utilisé : celui du répertoire `bin` dans son répertoire personnel. Il ne comprend pas l'entrée donnée!
- En regardant le PATH, le `bin` de l'utilisateur est consulté avant le `bin` général.

Shell - Recherche de fichiers(1)

- **which**:
  - Cherche le fichier demandé dans les répertoires listés dans PATH.
  - Comme PATH contient seulement les répertoires contenant des exécutables, `which` ne marche pas pour les fichiers ordinaires.

```
tina:~> which acroread
/usr/bin/which: no acroread in (/bin:/usr/bin:/usr/bin/X11)
```

Une solution: Changer PATH

```
tina:~> export PATH=$PATH:/opt/acroread/bin

tina:~> echo $PATH
/bin:/usr/bin:/usr/bin/X11:/opt/acroread/bin
```

Le changement n'est pas permanent: Le changement fait avec la commande shell `export` est valide seulement pendant une session (il permet son effet après un `logout`). **Un changement permanent est possible seulement en changement les fichiers de configuration**.

Shell - Recherche de fichiers(2)

- **find**:
  - Cette commande permet la recherche d'un fichier par son nom et peut aussi accepter d'autres critère de recherche: taille du fichier, date de la dernière modification, etc...
  - Regarder les pages de `man` pour les options. La forme plus utilisée est `find <path> -name <searchstring>`

```
peter:~> find . -size +5000k
psychotic_chaos.mp3

[mirian@home2 ~]$ find /home/mirian/HDR/Memoire  -name "*.pdf"
/home/mirian/HDR/Memoire/hdr1.pdf
/home/mirian/HDR/Memoire/hdr.pdf
```

Shell - Recherche de fichiers(3)

- Avec `find`, nous pouvons aussi appliquer des opérations sur les fichiers trouvés.

```
peter:~>  find . -name "*.tmp" -exec rm {} \;

peter:~>
```

- Cherche les fichiers avec extension `tmp` et les supprime (Tester avant sans `exec`!)

Shell - Recherche de fichiers(4)

- **locate** ou **slocate**:
  - `slocate` est la version sécurisée de `locate`.
  - Donne la liste des fichiers (leur *path*) dont le nom correspond (partiellement) au motif donné.
  - Plus facile/rapide que `find`, mais plus restreint. La sortie se base sur l'index d'une base qui est mise à jour une fois par jour

```
tina:~> locate acroread
/usr/share/icons/hicolor/16x16/apps/acroread.png
/usr/share/icons/hicolor/32x32/apps/acroread.png
/usr/share/icons/locolor/16x16/apps/acroread.png
/usr/share/icons/locolor/32x32/apps/acroread.png
/usr/local/bin/acroread
/usr/local/Acrobat4/Reader/intellinux/bin/acroread
/usr/local/Acrobat4/bin/acroread
```

Les répertoires qui ne sont pas dans `bin` ne contiennent pas des exécutables.

Shell - `grep`

- `grep` < *expression*> <*fichiers*> cherche dans les fichiers cités et imprime chaque ligne de l'entrée contenant l'expression.
- Très utile pour trouver les occurrences d'une variable dans un programme, ou des mots dans in documents, etc.

Montre les lignes de `/etc/passwd` qui contienne le mot `root`.

```
cathy ~> grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

Montre les lignes en indiquant leur numéro.

```
cathy ~> grep -n root /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
12:operator:x:11:0:operator:/root:/sbin/nologin
```

Compte le nombre de compte qui possède `/bin/bash` comme shell

```
cathy ~> grep -c bash  /etc/passwd
7
```

Expressions régulières (1)

- Une expression régulière = un motif qui permet de décrire un ensemble de chaînes de caractères.
- Les expressions régulières sont massivement utilisées dans les systèmes d'exploitation avec différentes commandes pout chercher des motifs. Il faut toujours regarder dans le `man`

Exemples

`grep 'l{2}' Install`
Afficher toutes les lignes du fichier `Install` contenant des mots avec deux *l*.

`grep 'sys[0-9]' f1`
Afficher toutes les lignes du fichier `f1` contenant la chaîne `sys` suivie d'un chiffre

Opérateurs des expressions régulières

Opérateur	Signification
.	Matches any single character.
?	The preceding item is optional and will be matched, at most, once.
*	The preceding item will be matched zero or more times.
+	The preceding item will be matched one or more times.
$\{N\}$	The preceding item is matched exactly N times.
$\{N,\}$	The preceding item is matched N or more times.
$\{N, M\}$	The preceding item is matched at least N times, but not more than M times.
–	represents the range if it's not first or last in a list or the ending point of a range in a list.
^	Matches the empty string at the beginning of a line; also represents the characters not in the range of a list.
$	Matches the empty string at the end of a line.

Opérateurs des expressions régulières

Opérateur	Signification
$\backslash b$	Matches the empty string at the edge of a word.
$\backslash B$	Matches the empty string provided it's not at the edge of a word.
$\backslash <$	Match the empty string at the beginning of word.
$\backslash >$	Match the empty string at the end of word.

Expressions régulières (2)

Les caractères suivants perdent leur signification habituelle dans les expressions régulières simples. Utiliser la version avec *backslash*

```
"?", "+", "{", "|", "(", and ")"
```

```
"\?", "\+", "\{", "\|", "\(", and "\)".
```

Exemples

```
cathy ~> grep ^root /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Affiche les lignes qui commencent par `root` du fichier `/etc/passwd`.

```
cathy ~> grep :$ /etc/passwd
news:x:9:13:news:/var/spool/news:
```

Si nous voulons trouver les comptes sans un `shell`, nous cherchos les lignes finissant par : dans `/etc/passwd`.

```
cathy ~> grep [yf] /etc/group
sys:x:3:root,bin,adm
tty:x:5:
mail:x:12:mail,postfix
ftp:x:50:
nobody:x:99:
floppy:x:19:
xfs:x:43:
nfsnobody:x:65534:
postfix:x:89:
```

Exemples

```
[mirian@home2]$ more titi
ano 1900
mes 10
dia 27 28
Ola pessoal
onde estamos?

[mirian@home2]$ grep '[[:digit:]]' titi
ano 1900
mes 10
dia 27 28
```

Exemples: attention!

Et si nous oublions les ’?

```
[mirian@home2]$ ls
toto tata titi
[mirian@home2]$ grep [[:digit:]]  titi
ano 1900
mes 10
dia 27 28

[mirian@home2]$ ls
toto tata titi 9
[mirian@home2]$ grep [[:digit:]] titi
ano 1900
```

C’est d’abord le SHELL qui interprète le motif. Dans ce cas il va chercher dans le répertoire de travail un fichier dont le nom est un chiffre et passe cette chaîne (le nom du fichier 9) à `grep` !!!!

Sécurité des fichiers

- Il est souhaitable de pouvoir gérer l'accès aux fichiers. Les droits possibles sont : **lecture**, **écriture** et **exécution**.
- Dans un système UNIX/Linux les utilisateurs sont partagés en trois groupes : l'utilisateur lui même, son groupe et les autres.
- Chaque fichier est la propriété d'un **utilisateur** et du **groupe** duquel il fait partie.
- Pour chaque catégorie d'utilisateur il est possible d'établir des droits spécifiques.
- La commande **ls** peut nous informer les droits d'accès sur les fichiers.
  - Les droits de chaque type d'utilisateur sont présenté par un groupe de trois caractères.
  - De gauche à droite sont représentés : l'utilisateur, son groupe et les autres.
  - Les droits sont toujours représentés dans l'ordre: *read*(r), *write*(w) et *execute*(x).
  - Le premier caractère à gauche indique si le fichier est un répertoire (d) ou un simple fichier ($-$).

Exemples

```
marise:~> ls -l To_Do
-rw-rw-r--    1 marise  users      5 Jan 15 12:39 To_Do
```

Le fichier `To_Do` appartient à l'utilisateur `marise` et au groupe `users` qui ont droit de lecture et écriture sur lui. Ils ne peuvent pas l'exécuté. Tous les autres utilisateurs peuvent lire ce fichier, mais il ne peuvent pas le modifier (écriture) ou l'exécuter.

```
marise:~> ls -l /bin/ls
-rwxr-xr-x    1 root    root   45948 Aug  9 15:01 /bin/ls*
```

Il s'agit d'un fichier exécutable; tout le monde peut l'exécuter, mais il faut être `root` pour pouvoir le changer.

Nom de l’utilisateur

- Commande `id` informe le nom de l’utilisateur et de ses groupes

```
tilly:~> id
uid=504(tilly) gid=504(tilly)groups=504(tilly),100(users)
```

- Le nom d’utilisateur est stocké dans la variable d’environnement USER

```
tilly:~> echo $USER
tilly
```

Codes d'accès

Pour faciliter l'utilisation des commandes, il existe des codes pour les droits et pour les groupes d'utilisateurs.

Codes des droits d'accès

Code	Droit
0 ou -	Le droit d'accès supposé être à cette position n'est pas assuré
4 ou r	Le droit de lecture est donné à la catégorie de l'utilisateur définie dans cette position
2 ou w	Le droit de écriture est donné à la catégorie de l'utilisateur définie dans cette position
1 ou x	Le droit d'exécution est donné à la catégorie de l'utilisateur définie dans cette position

Code utilisateur

Codes des catégorie des utilisateurs

Code	Catégorie
u	permission pour l'utilisateur
g	permission pour le groupe
o	permission pour les autres

Shell - commande chmod

- La commande chmod permet de faire des changement du droit d'accès.
- Cette commande peut être utilisée avec des options numériques ou alphanumériques.
- L'opérateur $-$ sert à supprimer un droit alors que l'opérateur $+$ sert à ajouter un droit. Il est possible de faire des combinaisons (séparées par des virgules)

Shell - commande chmod

Exemple alphanumérique:

```
asim:~> ./hello
bash: ./hello: bad interpreter: Permission denied

asim:~> cat hello
#!/bin/bash
echo "Hello, World"

asim:~> ls -l hello
-rw-rw-r--    1 asim    asim    32 Jan 15 16:29 hello

asim:~> chmod u+x hello

asim:~> ./hello
Hello, World

asim:~> ls -l hello
-rwxrw-r--   1 asim    asim    32 Jan 15 16:29 hello*
```

Shell - commande chmod

Exemple: Transforme le fichier *hello* en fichier privé de *asim*

```
asim:~> chmod u+rwx,go-rwx hello

asim:~> ls -l hello
-rwx------    1 asim    asim    32 Jan 15 16:29 hello*
```

Shell - commande chmod

Exemple pour les répertoires:

- r: le droit de faire un `ls` sur le répertoire
- w: le droit de supprimer/ajouter des fichiers dans le répertoire
- x: le droit de rentrer dans ce répertoire

```
[mirian@home2 Test]$ ls -al
total 24
drwxrwxr-x  3 mirian mirian 4096 Sep 11 21:12 .
drwx--x--- 73 mirian mirian 4096 Sep 11 21:12 ..
d--x--x--x  2 mirian mirian 4096 Sep 11 21:12 peter
[mirian@home2 Test]$ cd peter
[mirian@home2 peter]$ ls
ls: .: Permission denied
[mirian@home2 peter]$ cat > file1
bash: file1: Permission denied
```

Shell - commande chmod

Suite de l'example - on ajoute des droits...

```
[mirian@home2 Test]$ chmod u+r peter
[mirian@home2 Test]$ ls -al
total 24
drwxrwxr-x  3 mirian mirian 4096 Sep 11 21:12 .
drwx--x--- 73 mirian mirian 4096 Sep 11 21:12 ..
dr-x--x--x  2 mirian mirian 4096 Sep 11 21:12 peter
[mirian@home2 Test]$ cd peter
[mirian@home2 peter]$ ls
[mirian@home2 peter]$ cat > toto
bash: toto: Permission denied
```

Shell - commande chmod

- Pour utiliser chmod avec l'option numérique nous avons besoin de grouper les droits pour chaque catégorie d'utilisateur.
- Chaque catégorie est représentée par un tableau de trois "bits" qui sont allumés (si le droit est assuré) ou non. Le résultat est la somme des "bits" allumés.

  Exemple du calcul pour **une catégorie d'utilisateur**

2^2	2^1	2^0	
1	0	0	$2^2 + 0 + 0 = 4$
1	0	1	$2^2 + 0 + 2^0 = 5$

  Exemple: 400 indique que l'utilisateur a droit de lecture sur un fichier et que son groupe et les autres n'ont aucun droit sur ce fichier.

  Exemple: 540 indique que l'utilisateur a droit de lecture et d'exécution sur le fichier, le groupe a seulement droit de lecture et les autres n'ont aucun droit.

- Un numéro avec moins que 3 chiffres indique des zéros à gauche.

Shell - commande chmod

Exemples

Command	Meaning
`chmod 400` file	To protect a file against accidental overwriting.
`chmod 500` directory	To protect yourself from accidentally removing, renaming or moving files from this directory.
`chmod 600` file	A private file only changeable by the user who entered this command.
`chmod 644` file	A publicly readable file that can only be changed by the issuing
`chmod 660` file	Users belonging to your group can change this file, others don't have any access to it at all.
`chmod 700` file	Protects a file against any access from other users, while the issuing user still has full access.
`chmod 755` directory	For files that should be readable and executable by others, but only changeable by the issuing user.
`chmod 775` file	Standard file sharing mode for a group.
`chmod 777` file	Everybody can do everything to this file.

Login dans un autre groupe (1)

```
asim:~> id
uid=501(asim) gid=501(asim) groups=100(users),501(asim),3400(web)

asim:~> grep asim /etc/passwd
asim:x:501:501:Asim El Baraka:/home/asim:/bin/bash

asim:~> grep 501 /etc/group
asim:x:501:
```

- L'utilisateur `asim` a comme **groupe privé** (celui où il est le seul membre) le groupe 501. Le nom de ce groupe est le nom de login du l'utilisateur.
- Nous pouvons trouver le numéro du groupe privé de `asim` dans le fichier `/etc/passwd`. Dans le fichier `/etc/group` nous trouvons le nom de ce groupe.
- La commande `id` informe aussi que `asim` peut participer d'autres groupes, à savoir, le groupe *users* et le groupe *web*.

Login dans un autre groupe (2)

```
asim:/var/www/html> newgrp web

asim:/var/www/html> id
uid=501(asim) gid=3400(web) groups=100(users),501(asim),3400(web)
```

- *asim* veut créer des fichiers dans le groupe *web*. Pour cela il peut utiliser la commande `newgrp` (et utiliser la `gpasswd` d'abord pour établir un mot de passe du groupe)
- À partir de là, tous les fichiers que `asim` va créer seront du groupe *web*.

Mask

- Un nouveau fichier stocké dans un répertoire reçoit les **droits par défaut**. Des fichiers sans droit n'existent pas en UNIX/Linux.
- Les **droits par défaut** sont mis en place par le **mask** à la création des nouveaux fichiers.
- La valeur de ce mask peut être affichée via la commande **umask**

Commande umask

```
bert:~> umask
0002
```

- Dans UNIX/Linux à chaque fois qu'un nouveau fichier est créé (par *downloading* à partir d'Internet, par sauvegarder d'un fichier, etc) une fonction de création de fichiers est lancée.
- Cette fonction crée des fichiers et des répertoires en donnant les permissions suivantes:
  Répertoire : Tout le monde a le droit de lire, écrire et exécuter
  Un répertoire a le droit 777 **ou** ***rwxrwxrwx***

  Ficher : Tout le monde a le droit de lire et écrire
  Un fichier a le droit 666 **or** ***rw-rw-rw-***

  Cela AVANT l'application du mask

Mask

- La **valeur *umask*** est soustraite des droits par défaut après la création du nouveau fichier ou répertoire par la fonction

- Répertoire :

	777	valeur par défaut
−	002	mask (trois derniers chiffres)
	775	valeur umask

- Fichier :

	666	valeur par défaut
−	002	mask (trois derniers chiffres)
	664	valeur umask

Exemple: Mask

```
bert:~> mkdir newdir

bert:~> ls -ld newdir
drwxrwxr-x    2 bert    bert 4096 Feb 28 13:45 newdir/

bert:~> touch newfile

bert:~> ls -l newfile
-rw-rw-r--    1 bert    bert    0 Feb 28 13:52 newfile
```

- Un répertoire possède plus de droits qu'un fichier : **ne pas avoir le droit d'exécuter une répertoire signifie ne pas pouvoir d'accéder!**
- Le mask pour **root** est plus contrainant.

```
[root@estoban root]# umask
0022
```

Changement de propriétaire

- La commande **chown** peut être appliquée pour **changer l'utilisateur et le groupe** propriétaire d'un fichier.
- La commande **chgrp** peut changer seulement le groupe propriétaire.
- Les changements sont possibles seulement si l'utilisateur a le droit de le faire.
- **chown** et **chgrp** permettent le changement récursif.
- Usage restreint.

Exemple

```
jacky:~> id
uid=1304(jacky) gid=(1304) groups=1304(jacky),2034(project)

jacky:~> ls -l my_report
-rw-rw-r--  1 jacky   project       29387 Jan 15 09:34 my_report

jacky:~> chown jacky: my_report
```

- Les deux points (:) après le nom d'utilisateur `jacky` indique une demande de changement de utilisateur ET group propriétaire (sans : on change seulement l'utilisateur)
- Autre possibilité pour changer le groupe est le `chgrp`.

```
jacky:~> ls -l report-20020115.xls
-rw-rw---- 1 jacky   jacky   45635 Jan 15 09:35 report-20020115.xl

jacky:~> chgrp project report-20020115.xls
```

Modes spéciaux (1)

Sticky bit mode

- L'administrateur sytème peut souhaiter avoir des répertoires partagés où chaque utilisateur a des droits d'écriture.
- **Il ne serait pas souhaitable de laisser un utilisateur supprimer ou renommer un fichier appartenant à un tiers.** Pour éviter cela le **sticky bit** est utilisé.
- Le **sticky bit** est indiqué par un t à la fin du champ qui donne la permission pour un fichier.

```
mark:~> ls -ld /var/tmp
drwxrwxrwt   19 root    root   8192 Jan 16 10:37 /var/tmp/
```

- Le **sticky bit** indique que seul le propriétaire du répertoire et le propriétaire d'un fichier qui s'y trouve ont le droit de supprimer ce fichier.
- Le **sticky bit** est établi avec une de deux commandes
  `chmod o+t` *directory*
  `chmod 1777` *directory*
  (le 1 au début indique l'utilisation du *sticky bit*)

Modes spéciaux (2)

SUID (*Set User ID)* et SGID (*Set group ID*)

- Un programme est exécuté **avec les droits de l'utilisateur et du group possèdant le programme** et non les droits de l'utilisateur qui a lancé le programme.
- En d'autres termes, ces modes donnent à l'utilisateur normal la possibilité d'effectuer des tâches qui normalement il n'aurait pas le droit dû au droit d'accès stricts établi par UNIX/Linux.
- Ces modes sont utilisés avec précautions.
- Représenté par le caractère *s* dans le champs de l'utilisateur ou du group (nous traitons ici seulement le cas du *s* dans le champs utilisateur).

Exemple: Changement de mot de passe

- Les utilisateurs souhaitent pouvoir changer de mot de passe sans demander l'administrateur système.
- Les noms des utilisateurs et leur passwd sont stockés dans `/etc/passwd` dont les droit d'accès est restreint.

```
bea:~> ls -l /etc/passwd
-rw-r--r--    1 root   root   1267 Jan 16 14:43 /etc/passwd
```

- Les utilisateurs ont besoin de changer leur information dans le fichier `passwd`. **Cela est fait en donnant une permission spéciale pour le faire au programme passwd**.
- Quand le programme **passwd** est appelé il sera exécuté avec les droits de **root**, permettant ainsi à l'utilisateur normal d'écrire dans le fichier *password* qui appartient à l'administrateur système.

```
[mirian@home2 ~]$ which passwd
/usr/bin/passwd
[mirian@home2 ~]$ ls -al /usr/bin/passwd
-rwsr-xr-x 1 root root 22932 Jul 17  2006 /usr/bin/passwd
```

Implementation des systèmes de fichiers

- Les disques fournissent la mémoire auxiliaire de grande capacité sur laquelle on maintient un système de fichiers
- Pour améliorer l'efficacité des E/S, les transferts entre la mémoire et le disque s'effectuent en unités de *blocs* (1 ou + secteurs)
- Deux **caractéristiques importantes des disques**:
  1. Il est possible de lire un bloc, le modifier et le récrire au même endroit
  2. L'accès direct à un bloc est possible
- Afin de **fournir un accès efficace et pratique** au disque, le SE impose un **système de fichiers** pour permettre de **stocker, localiser et récupérer facilement des données**
- Deux **questions liées à la conception des systèmes de fichiers**
  1. Définir l'allure du système de fichiers (pour l'utilisateur).
  2. Créer les algorithmes et les structures de données pour établir la correspondance entre le système logique de fichiers et les dispositifs physiques de mémoire auxiliaire.

Méthodes d'allocation (1)

Allocation contiguë

- Chaque fichier occupe un ensemble de blocs contigus sur le disque. Exemple: VMS/CMS (IBM)
- **Avantages**
  - Simple: pour savoir l'emplacement du fichier il suffit de savoir la place du premier bloc
  - Très bonne performance: le fichier entier peu être lu dans une seule opération. Accéder au bloc $b + 1$ après le bloc b (aucun déplacement de tête)
- **Incovénients**
  - Savoir la taille maximale d'un fichier pour pouvoir réserver la place.
  - Problème de fragmentation externe (gaspillage de espace qui pourrait être utilisé autrement)

Méthodes d'allocation (2)

Allocation par liste chaînée

- Chaque fichier est une liste chaînée de blocs de disque; les blocs disque peuvent être n'importe où sur le disque. **Chaque bloc contient un pointeur sur le bloc suivant.**
- **Avantages**
  - Chaque bloc du disque peut être utilisé, pas de fragmentation externe (fragmentation interne du dernier bloc)
  - Pour l'entrée du répertoire, il suffit de conserver l'adresse du premier bloc (ensuite suivre les pointeurs).
  - Il n'est pas nécessaire de connaître la taille d'un fichier pour le créer.
- **Inconvénients**
  - L'accès direct est inefficace: pour accéder au bloc n le SE doit lire les $n-1$ blocs précédents.
  - L'espace requis pour les pointeurs

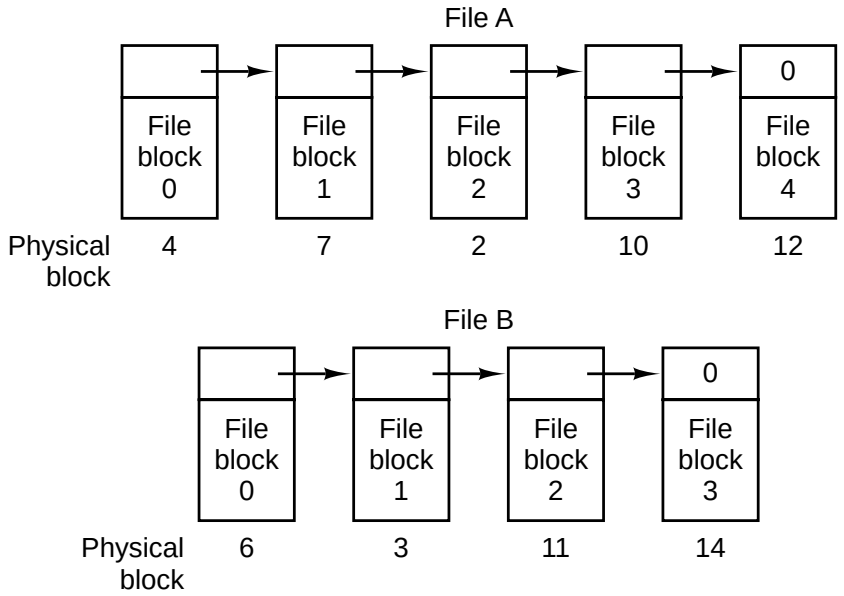


Fig. 6-13. Storing a file as a linked list of disk blocks.

Méthodes d'allocation (3)

Allocation par liste chaînée utilisant une table en mémoire

- Les deux inconvénients d'une liste chaînée peuvent être éliminés en prenant le pointeur de chaque bloc du disque pour le ranger dans une table en mémoire (FAT - *File Allocation Table*).
- L'entrée de répertoire contient le numéro du premier bloc du fichier.
- Exemple: MS-DOS et OS/2
- Au début de chaque partition, une section du disque est réservé pour la FAT
- **Avantages**
  - Bloc disponible en sa totalité pour les données.
  - L'accès aléatoire est facilité: comme la chaîne est totalement en mémoire, bien qu'il faille la parcourir pour trouver un déplacement donné a l'intérieur du fichier, cela peut être fait sans accès à disque.
- **Inconvénients**
  1. FAT en mémoire primaire $\Rightarrow$ cela peut être lourd.
  2. Nombre significatif de positionnement de tête.

Exemple FAT

0	
1	
2	10
3	11
4	7
5	
6	3
7	2
8	
9	
10	12
11	14
12	0
13	
14	0
15	

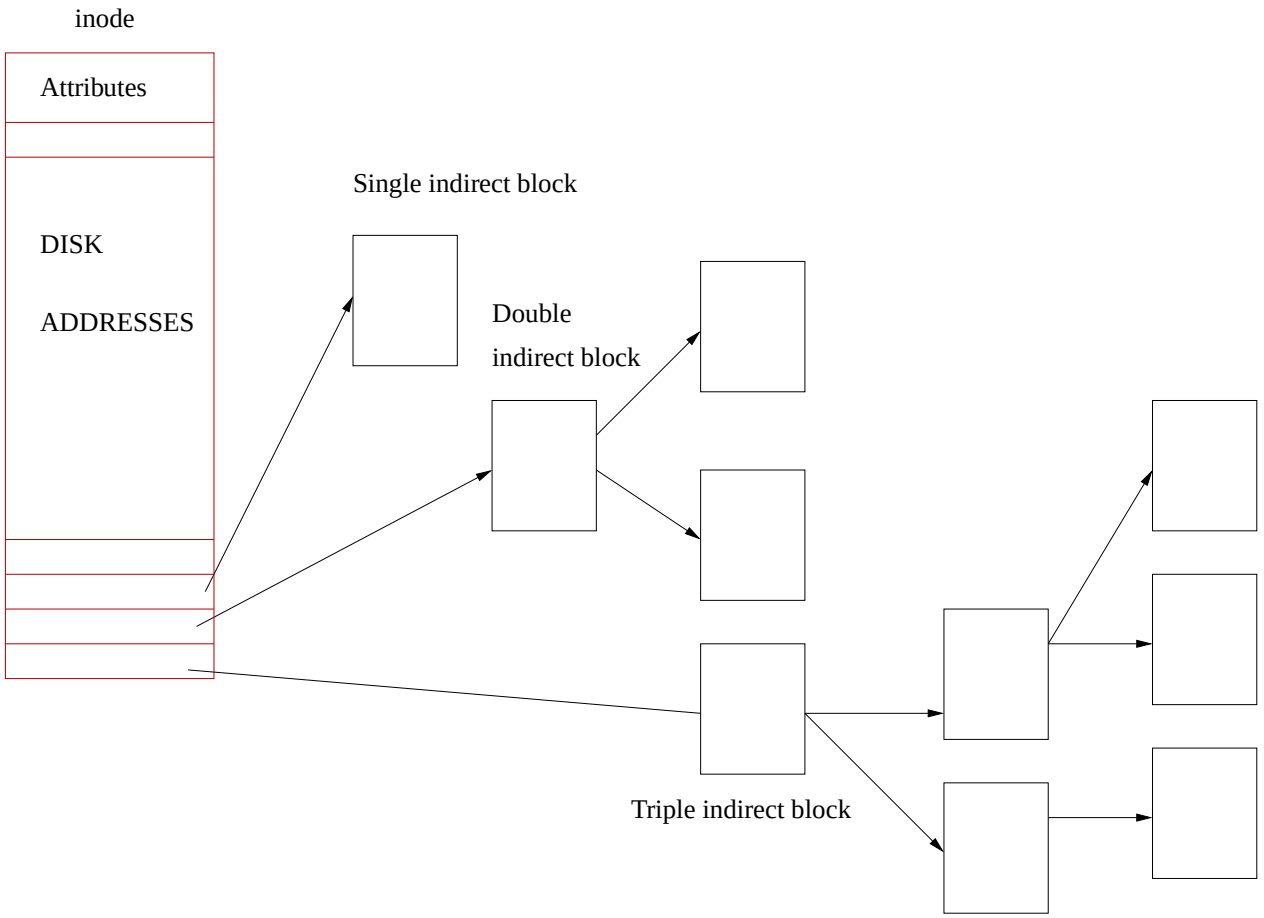
Méthodes d'allocation (4)

I-nodes (noeuds d'information

- Chaque fichier est associé à une table (*i-node*) qui contient les attributs et les adresses des blocs dans le disque.
- Les premiers adresses sont stockées dans le *i-node* lui même. Ainsi, pour les petits fichiers toutes les informations nécessaires sont dans le *i-node*.
- Pour des fichiers plus grands, une adresse dans le *i-node* est l'adresse d'une d'un bloc d'adresses (*single indirect bloc*). Et... ainsi de suite jusqu'au *triple indirect bloc*.
- Avantages

  La taille de la FAT est proportionnelle à celle du disque. Le concept de *i-node* requiert un tableau en mémoire dont la taille est proportionnelle au nombre maximum de fichiers qui peuvent être ouverts simultanément.

Méthodes d'allocation (5)



Implementation des répertoires

- L'entrée d'un répertoire fournit les informations nécessaires pour trouver les blocs de disque. **La fonction principale du répertoire est de faire correspondre le nom ASCII du fichier à une information nécessaire pour localiser les données**.
- L'emplacement des attributs:
  - Directement dans l'entrée du répertoire. Dans ce cas un répertoire est une liste d'entrées (chacune correspond à un fichier) dont la taille est fixée.
  - Pour les systèmes avec *i-node*, il est possible de stocker les attributs dans le *i-node*.

Implementation des répertoires

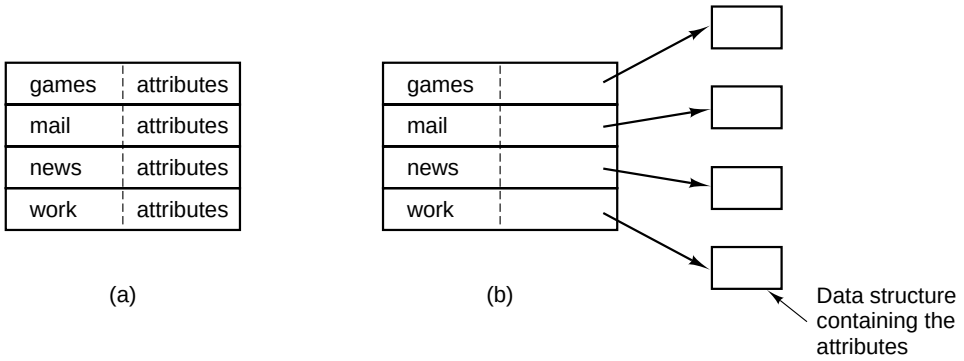


Fig. 6-16. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

Exemples

Répertoire MS-DOS et Windows 95

8 bytes	3	1	10	2	2	2	4
Filename	Ext.	Att.	Reserved	Time	Date	1st block	Size

Répertoire Unix

Bytes	2	14
	i-node nb	Filename