Quatrième partie

Fichiers

Contenu de cette partie

Gestion des données conservées en mémoire secondaire

- Fichiers
  - point de vue utilisateur : interface, modèle
  - mise en œuvre : implantation, accès
- Organisation et désignation des fichiers
  - point de vue utilisateur : répertoires, liens
  - mise en œuvre
- Annexes : points techniques

Plan

Gestion des données rémanentes

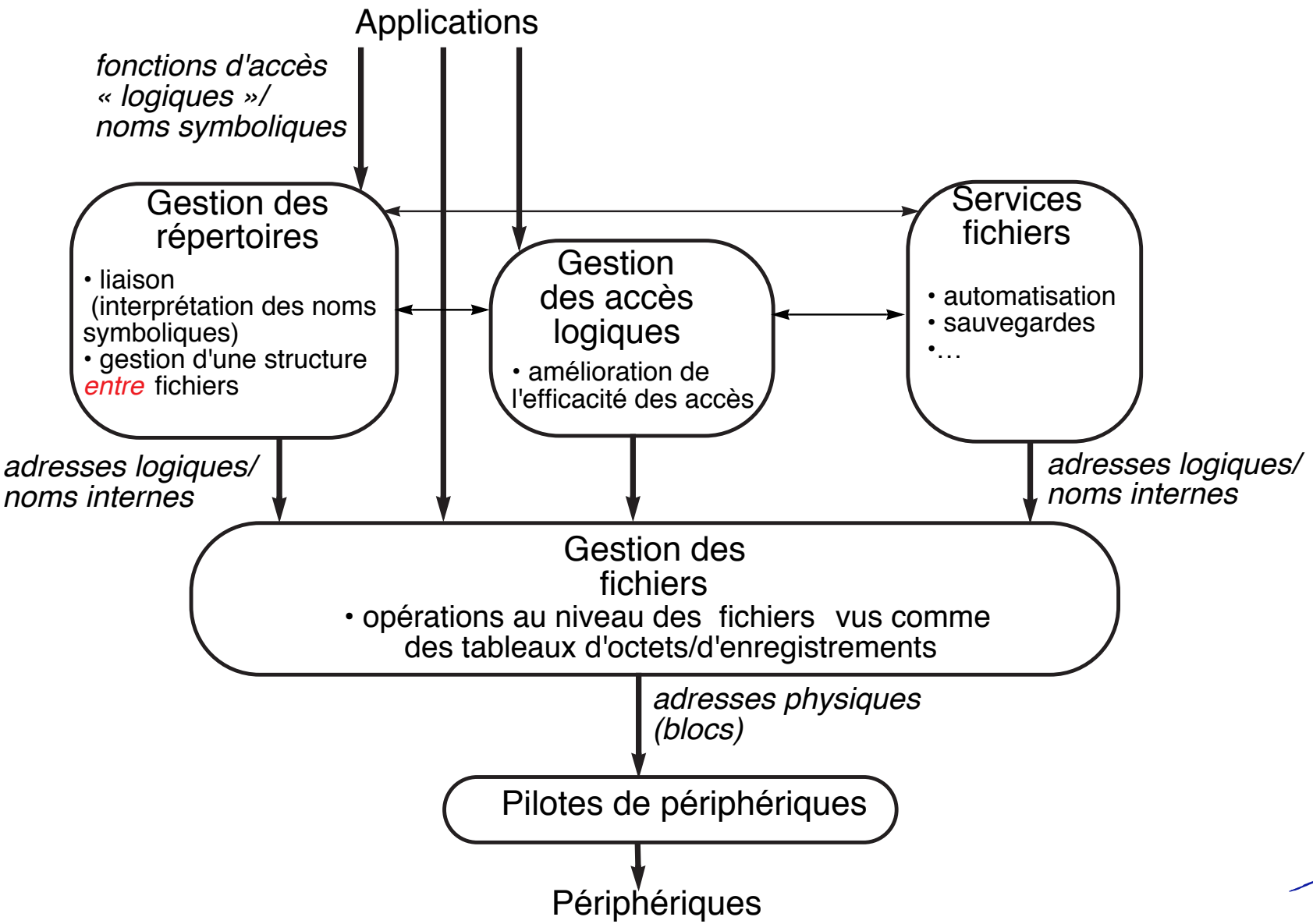
Intérêt de la mémoire secondaire (disques, bandes...)

- Rémanence (les données peuvent être conservées)
- Capacité de stockage

Aide à l'utilisation des données stockées en mémoire secondaire :

- Abstraction
  notion de fichier : modèle simple d'accès aux données stockées
- Partage et protection des fichiers
  droits d'accès, contrôle de concurrence
- Organisation logique des fichiers
  service de gestion de répertoires (SGR)
- Autres services
  - Contrôles et services liés au « type », au contenu des fichiers (binaires, textes...)
  - Accélération des accès (index...)
  - Sauvegarde

Structure des services de gestion des données rémanentes



Plan

Gestion des échanges avec les périphériques

Abstraction des E/S

- périphériques blocs : disques...
  - données structurées (regroupées) dans des blocs (souvent de taille fixe)
  - chaque bloc est identifié par une adresse
  - l'accès aux données est déterminé par le numéro du bloc
- périphériques caractères : clavier...
  - données non structurées, vues comme des suites (flots) d'octets
  - l'accès aux données doit suivre l'ordre du flot

Caches

- *But* : optimiser les échanges en conservant dans un tampon (cache) en mémoire rapide une copie des données échangées
- *Difficulté* : gestion de la cohérence
  entre copie en cache et copie sur le périphérique

Notion de fichier

Abstraction pour un ensemble de données

- conservées indépendamment de l'exécution des applications (rémanence)
- ayant un lien logique (pour l'utilisateur)
- différenciées par un identifiant (clé) ou un indice (position)

$$\rightarrow \text{fichier} \triangleq \text{suite} \begin{cases} \text{d'octets} \\ \text{d'enregistrements} \end{cases}$$

Opérations

Abstraction : support, implantation, opérations de bas niveau

Métaphore : bande magnétique

- créer/détruire (Unix : `creat`.../`unlink`...)
- ouvrir/fermer (Unix : `open`/`close`)
- accès (lire, écrire, naviguer) (Unix : `read` ,`write`, `lseek`)

Utilisation des fichiers

Fichiers ordinaires

Sous Unix, le contenu d'un fichier est une suite d'octets, sans autre structure. L'interprétation de ce contenu dépend de l'utilisation :

Programmes exécutables

Commandes du système ou programmes créés par un usager

Exemple

`gcc -o prog prog.c`	produit le programme exécutable (fichier prog)
`./prog`	exécute le programme prog

Fichiers de données

- Documents, images, programmes sources, etc.
- *Convention* : il est pratique d'ajouter au nom un suffixe indiquant la nature du contenu (usage en Unix, nécessaire sous Windows)
  - *Exemples* : .c (programme C), .o (binaire translatable), .gif (format d'images), .ps (PostScript), .pdf...
  - *Remarque :* la commande Unix `file` fournit dans tous les cas une indication sur la nature du fichier.

Utilisation des fichiers

Autres fichiers

La plupart des ressources se présentent sous Unix comme des fichiers afin d'offrir une interface (un modèle) unique à l'utilisateur :

- tubes (pipes) : accès à un tampon mémoire
  vu comme une paire de fichiers (dépôt/retrait)
- répertoires
- liens (alias)
- fichiers spéciaux : les périphériques appraissent sous Unix comme des fichiers (souvent situés dans le répertoire `/dev`)
  - Les périphériques sont désignés par des noms de fichiers
  - Interface d'usage (commandes, primitives) : celle des fichiers
    - *open/close* : connexion/déconnexion au périphérique
    - *read* : lecture (si cela a un sens par rapport au périphérique)
    - *write* : écriture (si cela a un sens par rapport au périphérique
    - *ioctl* : action spécifique.
  - Les mécanismes de protection sont les mêmes.
  - À un fichier spécial correspond un pilote de périphérique
    (Davantage de détails sont fournis en annexe)

Protection et partage des fichiers

La protection (ou sécurité) recouvre plusieurs aspects

- confidentialité :
  informations accessibles aux seuls usagers autorisés
- intégrité : pas de modifications non désirées
- contrôle d'accès :
  seuls certains usagers sont autorisés à faire certaines opérations
- authentification :
  garantir qu'un usager est bien celui qu'il prétend être

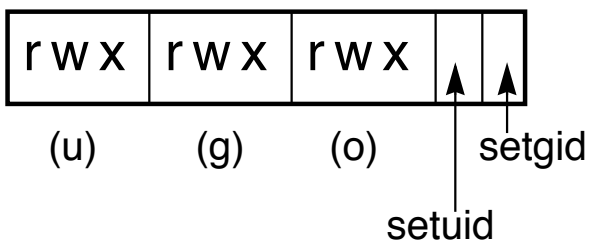
Note : quelques compléments sur la mise en œuvre de l'intégrité et du contrôle d'accès sont fournis en annexe.

Sécurité des fichiers (sous Unix) : droits d'accès

On définit

- des types d'opérations sur les fichiers : lire (`r`), écrire (`w`), exécuter (`x`)
- des classes d'usagers :
  usager propriétaire du fichier (`u`), groupe propriétaire (`g`), autres (`o`)

Fichiers ordinaires



Exemple (fichier `fich`) :

`rwx r-- r--` : tout accès pour le propriétaire, lecture seule pour tous les autres

`chmod go+w fich` : droit w donné au groupe et aux autres

`chmod o-w fich` : retire le droit w aux autres

Répertoires

Même chose, mais le droit x signifie « recherche dans le répertoire »

Mécanisme de délégation

- *Problème* : partager un programme dont l'exécution nécessite des droits d'accès que n'ont pas les utilisateurs du programme
- *Moyen* (`setuid` ou `setgid`) : accorder les droits du propriétaire à l'utilisateur (uniquement) pour la durée de l'exécution du programme

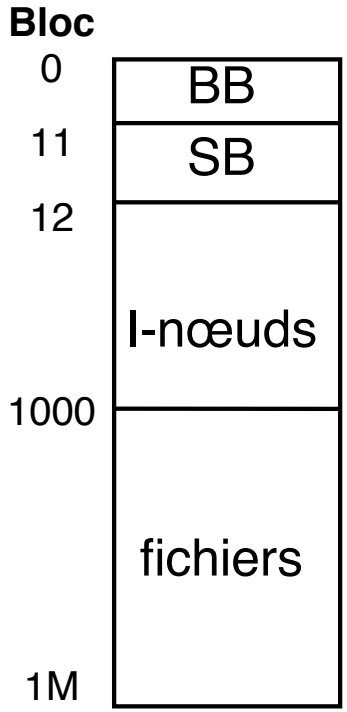
Plan

Système (logique) de fichiers

Implante

- un ensemble (non structuré) de fichiers
- vus comme des tableaux d'octets (sans structure interne)
- dans un espace physique vu comme un disque (logique) (tableau de blocs de taille fixe)

Structure type d'un disque logique



- blocs d'amorce
  UNIX : `bootblocks (BB)`
- informations sur la structure du disque : adresse, type des ≠ zones qui le constituent, localisation des blocs libres
  UNIX : `superbloc (SB)`
- catalogue du disque : liste des *descripteurs* des fichiers présents sur le disque logique
  UNIX : `table des I-nœuds`
- fichiers

Structure type d'une entrée de catalogue (i-nœud Unix)

- propriétaires(s)
- droits d'accès
- date(s) de dernier accès, de création...
- taille
- type
- référence(s) vers l'implantation du fichier sur disque
- données pour gérer la fiabilité : checksum, référence vers une copie...

Interface avec l'organisation physique : partitions (disques virtuels)

But : indépendance par rapport à la configuration physique

- un disque virtuel peut utiliser une portion d'un disque physique,
- ou un groupe de disques physiques
- chaque disque virtuel a la structure décrite précédemment

Terminologie

minidisks (VM), volumes (MS-DOS), partitions (MacOS), disques logiques

Exemple : Unix propose 2 types de disques logiques

- swap : utilisé pour la régulation et la gestion mémoire
- systèmes de fichiers : utilisé pour... le stockage des fichiers

Implantation des partitions : structure type d'un périphérique

- bloc d'amorce : code d'amorce + table des partitions
- table des partitions : taille et adresse des différentes partitions
- réserve de blocs (remplacement des blocs devenant véreux)

Interconnexion des disques logiques

≈ greffe d'arborescences contenues dans des disques logiques

- au niveau de la racine : Windows
- insertion d'une sous-arborescence quelconque en un point quelconque d'une autre arborescence (montage)

  Exemple : Unix (et NFS)
  - un système de fichiers (SF) est privilégié : le SF « système »
  - les autres SF peuvent s'y greffer (instruction `mount` (montage logique))
  - mécanisme utilisé par NFS pour la connexion de fichiers/répertoires distants

Représentation de l'espace libre (1/2)

But

garder une trace des blocs qui peuvent être (ré)alloués
→ « liste » des blocs libres

Vecteur de bits

carte de l'occupation du disque logique : pour chaque bloc du disque, un bit indique si le bloc est libre

- Avantages :

  simple et efficace

- Inconvénient :

  . . . à condition que le vecteur tienne en mémoire centrale

→ utilisation

  - par les systèmes/micro-ordinateurs (MacOS)
  - lorsque la partition comprend relativement peu de blocs (groupes de cylindres BSD 4.x)

Représentation de l'espace libre (2/2)

Liste chaînée

carte de localisation de l'espace libre :

- L'espace libre est vu comme une suite de blocs.
- Chaque bloc libre contient l'adresse du suivant

Problème

inefficace (allouer n blocs $\rightarrow$ n E/S)

Amélioration 1

le premier bloc libre contient les adresses des n suivants (dont n-1 sont directement utilisables)

Amélioration 2

si n blocs libres contigus, stocker n et l'adresse du premier

Implantation des fichiers sur le disque

Allocation contiguë

Chaque fichier est rangé dans des blocs d'adresses successives

- L'entrée du catalogue pour le fichier contient l'adresse du premier bloc et la taille du fichier
- Stratégies de placement : First fit, Best fit, Worst fit

Avantages

- permet l'accès direct et l'accès séquentiel
- temps d'accès minimal, pour un fichier

Inconvénients

- fragmentation externe (blocs libres nombreux mais dispersés) → recompactage coûteux
- gestion lourde, lorsque la taille des fichiers augmente

Implantation des fichiers sur le disque

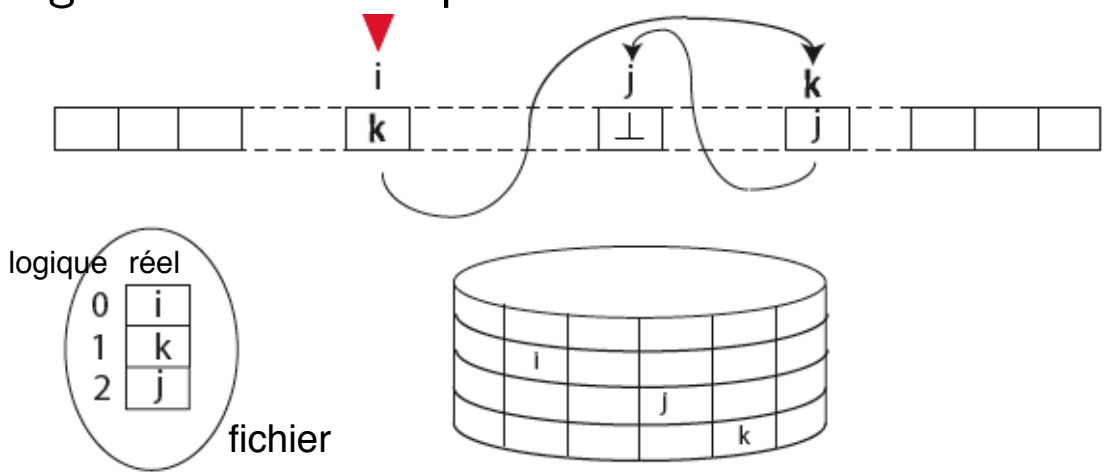
Allocation fragmentée

Pas de relation entre les adresses (la localisation) des blocs du fichier

Table d'allocation (FAT) (∼ carte de l'occupation du disque)

La table d'allocation est indexée par les numéros de blocs disque.

- *Idée* : regrouper toutes les adresses→ moins d'E/S
- Le catalogue contient le numéro du 1er bloc alloué à chaque fichier.
- Chaque entrée de la table d'allocation contient
  - le numéro du bloc suivant, si le bloc correspondant est alloué à un fichier
  - 0 si le bloc correspondant est libre
  - EOF si le bloc correspondant est le dernier du fichier
- Le chaînage des entrées exprime celui des blocs alloués



Implantation des fichiers sur le disque

Allocation chaînée

Les blocs alloués à un fichier sont chaînés

- Le catalogue contient l'adresse du premier bloc et celle du dernier bloc de chaque fichier
- Chaque bloc contient l'adresse de son successeur

Avantages

- pas de fragmentation externe
- pas de problème lié à l'évolution de la taille d'un fichier

Inconvénients

- accès direct très inefficace
- fiabilité : perdre un bloc $\approx$ perdre tout ce qui suit dans le fichier $\rightarrow$ chaînage double $\rightarrow$ surcroît de gestion

Implantation des fichiers sur le disque

Allocation indexée

Le premier bloc de chaque fichier est (le premier bloc de) la table des adresses des blocs alloués au fichier

Avantages

- Cf allocation chaînée
- Accès direct possible

Inconvénient : perte de place pour les petits fichiers

→ Idée : gérer un ensemble de blocs d'index (de taille réduite)

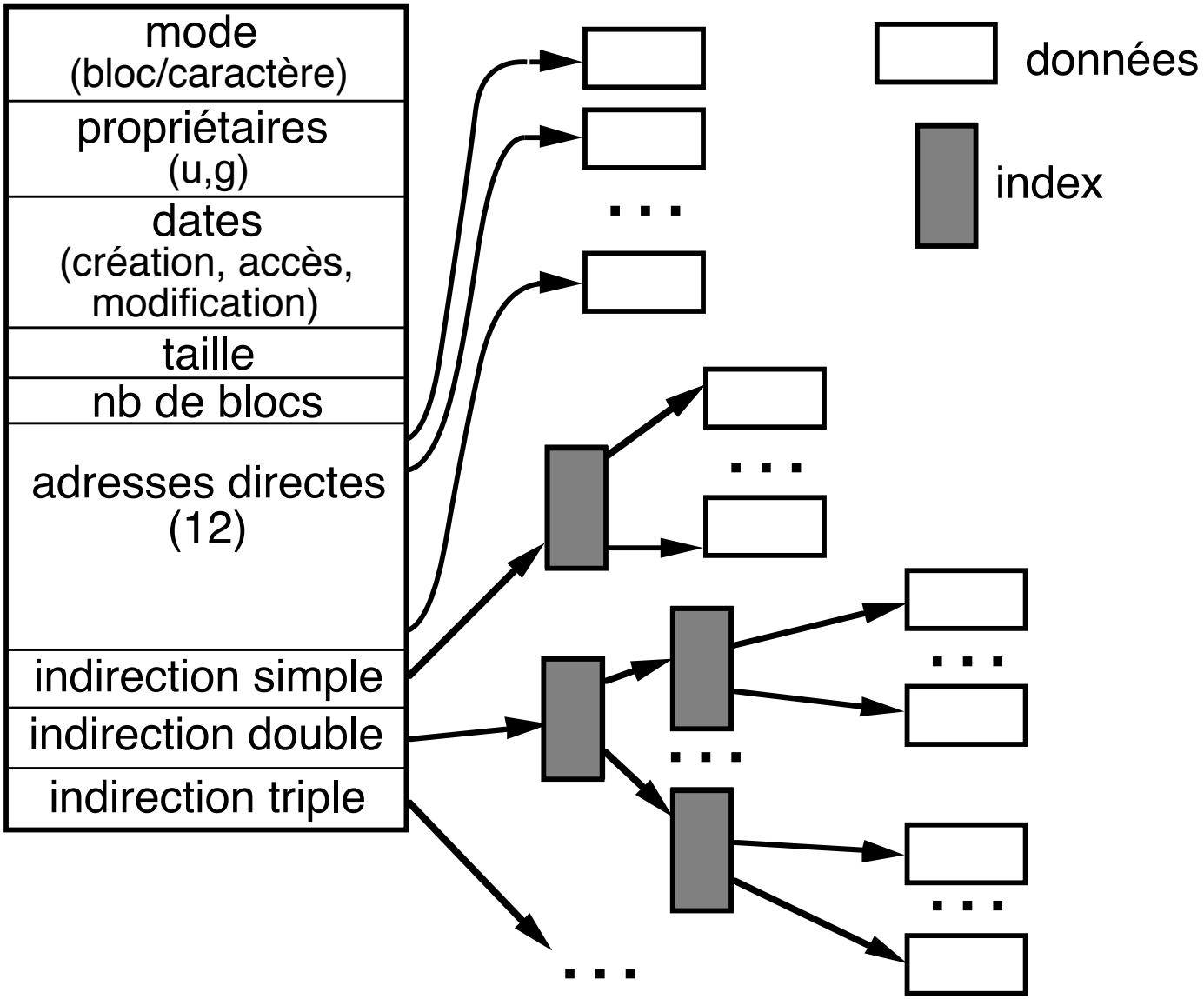
- plus compact
- mais temps d'accès accru

Méthodes

- Liste chaînée
- Arborescence (le bloc d'index référence des blocs d'index qui. . .)
- Méthode hybride (Unix) : l'entrée du catalogue pour un fichier (i-nœud) contient 15 adresses de blocs :
  - 12 adresses directes (48K adressables)
  - 3 adresses indirectes

    indirections simple (1 Mo adressable), double (1 Go), triple (1 To)

I-nœud UNIX


mode
(bloc/caractère)
propriétaires
(u,g)
dates
(création, accès,
modification)
taille
nb de blocs
adresses directes
(12)
indirection simple
indirection double
indirection triple
données
index

Plan

Interface utilisateur pour la manipulation des fichiers

Service fourni

- permettre une désignation « familière » (noms externes)
- structurer l'espace de noms/d'objets défini par l'utilisateur

→ notion de répertoire :
ensemble de fichiers nommés et regroupés au choix de l'utilisateur

Structuration minimale : répertoire « plat »

Simple liste de noms de fichiers

Difficultés

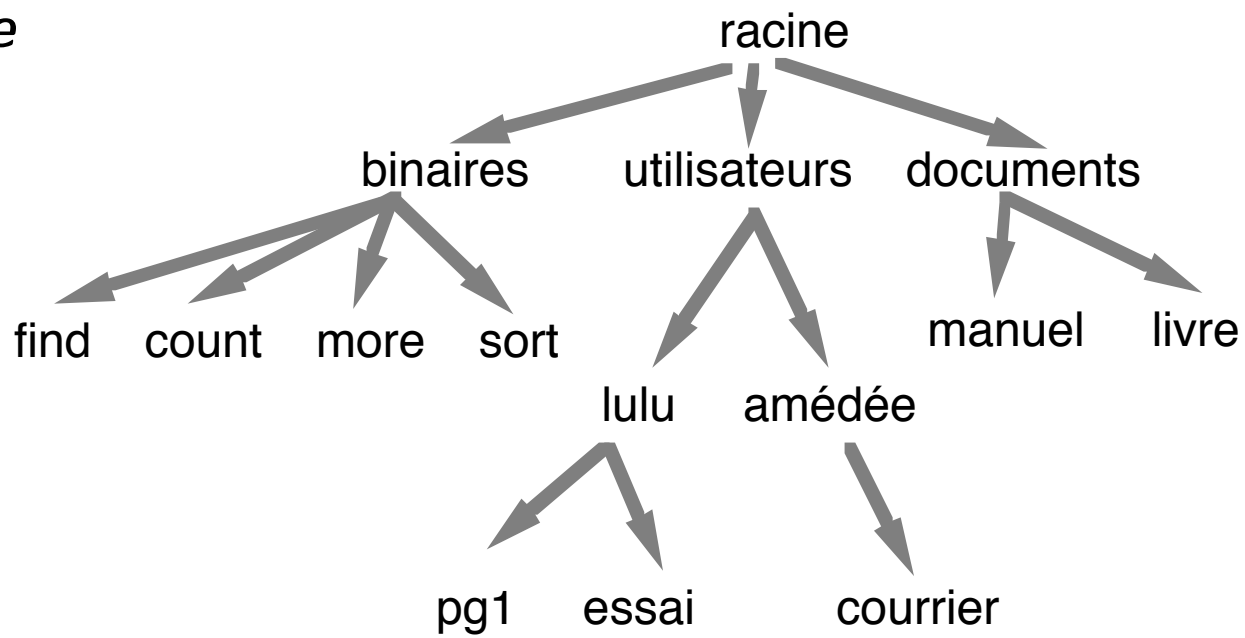
- accumulation des fichiers → allongement de la liste
- conflits entre noms

Structuration classique : arborescence

Principe : inclure des répertoires dans les répertoires

Organisation purement logique, choisie par l'utilisateur

Exemple



Intérêt

- organisation définie par l'utilisateur :
  fichiers regroupés par sujet/usage/utilisateur
- règle les conflits de noms (un fichier est désignable par le chemin (unique) le reliant à la racine)
- contrôle des droit d'accès

Modèle d'utilisation : navigation dans l'arborescence

- Nom unique : chemin absolu (chemin reliant la racine au fichier)
  Exemple (Unix) : `/utilisateurs/lulu/pg1`
- Raccourcis
  - chemins relatifs :
    utilisation d'un préfixe implicite (répertoire de travail/courant)
    - *Exemple* : si le répertoire courant est `/utilisateurs/lulu`, le fichier précédent peut être désigné par `pg1`
    - *Généralisation* :
      liste de préfixes (chemins) permettant l'emploi de noms locaux
      Exemple : variable `PATH` en Unix
    - Opérations sur le répertoire courant
      - initialisation : répertoire privé (ou répertoire de connexion)
      - affectation : `cd`
      - affichage : `pwd`
  - abréviations spécifiques à une application
    Exemple (shell Unix) : `.`, `..`, `~xxx`
  - synonymes : liens

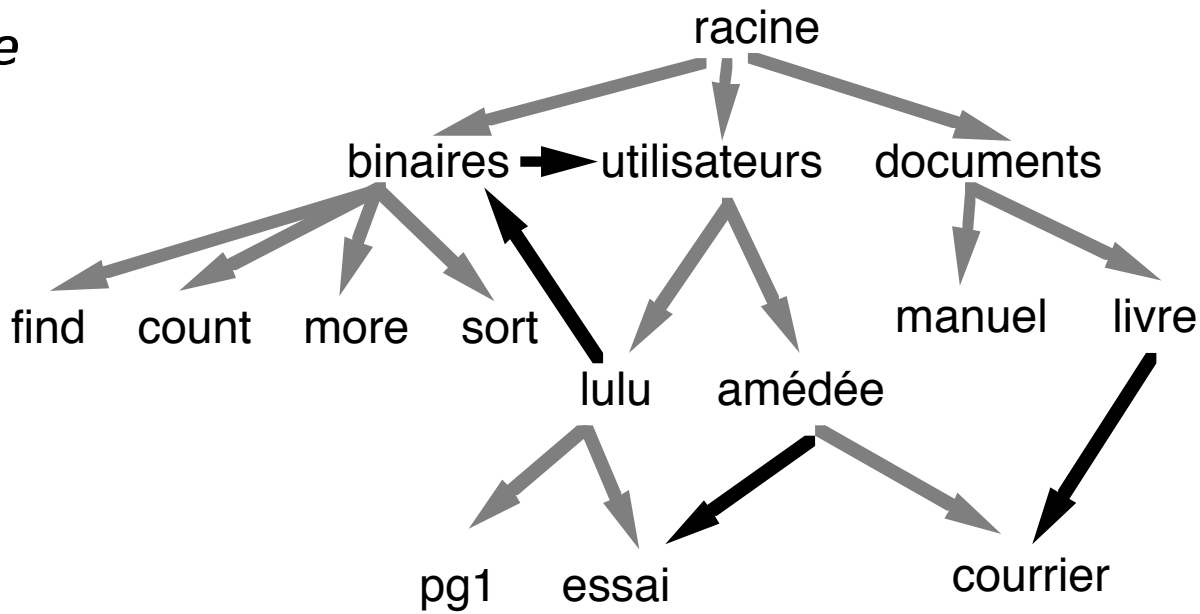
Structure de graphe : liens logiques

But

désigner (sans dupliquer) un même fichier depuis plusieurs répertoires

- le fichier/le répertoire apparait comme une entrée « ordinaire » dans chacun des répertoires
- structure de graphe

Exemple



Non unicité des désignations/des chemins
→ navigation moins naturelle (cycles, prédécesseurs multiples. . .)

Plan

Mise en œuvre des répertoires

répertoire = table de correspondance :
nom externe (utilisateur) $\leftrightarrow$ nom interne (SX)

pour Unix, un répertoire est juste un fichier contenant cette table

Mise en œuvre des liens (Unix)

- liens physiques (nom interne) (commande `ln f ./f_bis`) :
  ajout d'une entrée `f_bis` au répertoire courant,
  associée au même i-nœud que `f`
- liens symboliques (chemin d'accès) (`ln -s f ./f_bis`) :
  le chemin du fichier `f` est stocké dans le fichier `f_bis` du répertoire courant
  - $\rightarrow$ problème de mise en cohérence du contenu des fichiers de référence, en cas d'évolution

Exemple (sous UNIX, dans l'arborescence précédente)

création d'un lien symbolique vers binaires, sous le nom `programmes`, depuis le répertoire `lulu`.

```
>pwd
/utilisateurs/lulu
>ls
essai pg1
>ln -s /binaires programmes
>ls
essai pg1 programmes
# un fichier "/utilisateurs/lulu/programmes" a été créé,
dont le contenu est "/binaires"
>cd programmes
>ls
count find more sort
```

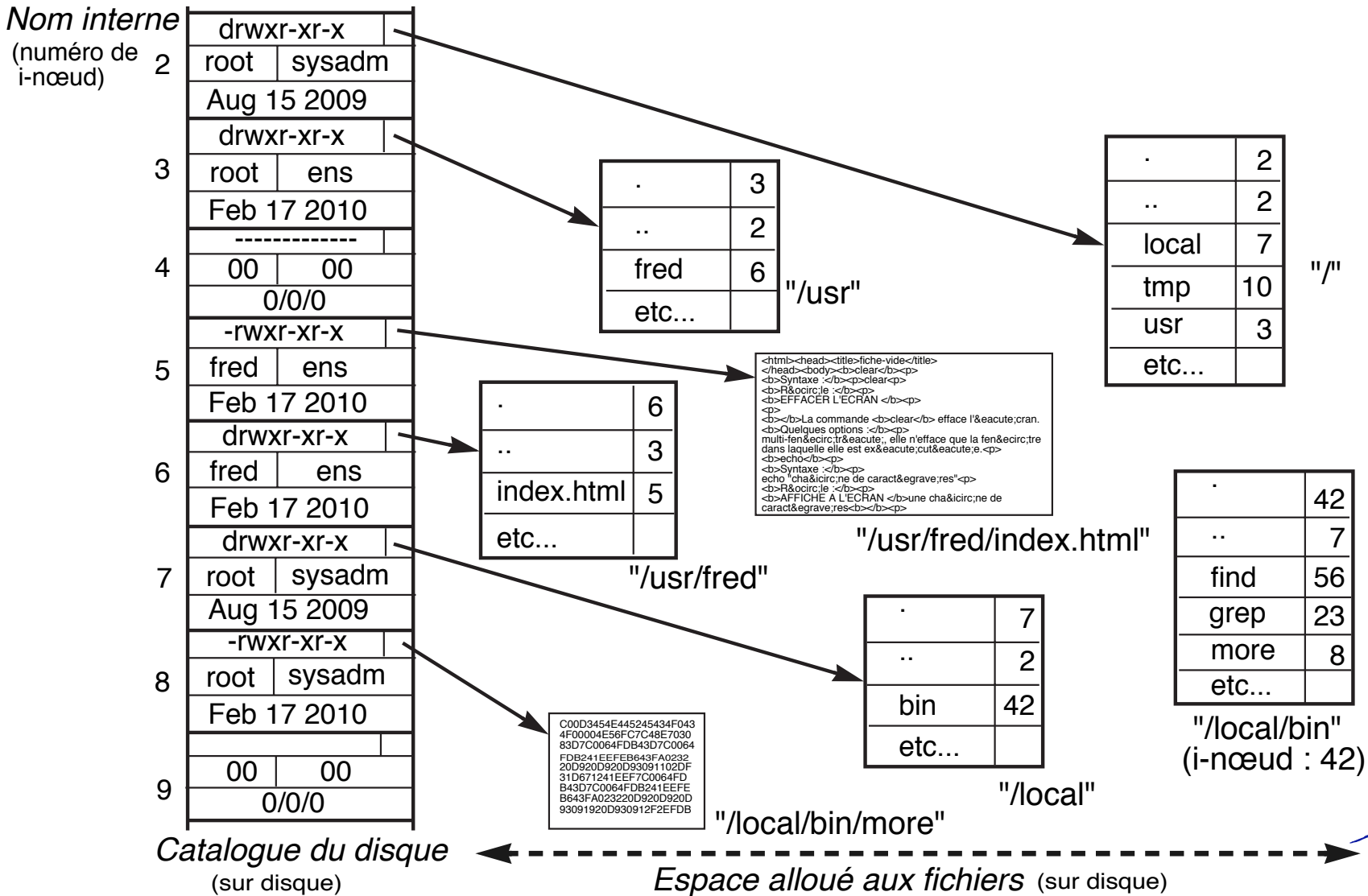
Contrôle d'existence

Problème

maintenir la cohérence des liens vers un objet,
si celui-ci est supprimé/déplacé

- Politique minimale : évaluation paresseuse
  lien vers un objet disparu = référence erronée
  Exemple : liens symboliques Unix
- Politique classique : conserver l'objet tant qu'il est référencé
  - Conserver le nombre de références (liens) vers chaque objet
    Exemple : liens physiques Unix
    $\rightarrow$ un fichier se peut être supprimé que si son compteur est nul
    Problème : test invalide en cas de cycles dans les références
  - Ramasse-miettes (périodique)
    - marquer les objets accessibles depuis la racine
    - supprimer les objets non marqués
  - Datation : supprimer les objets non accédés depuis longtemps

Interprétation des chemins d'accès par le service de répertoires



Plan

Le sous-système d'E/S d'UNIX BSD

But : proposer une interface homogène,
pour masquer les particularités des divers périphériques

→ Unix présente les périphériques

- à un niveau abstrait comme des `fichiers séquentiels` avec lesquels s'échangent des flots.
  - leur interface recouvre celle des fichiers
  - prise en compte des particularités
    → opération supplémentaire : `ioctl()`
- à un niveau fin, via une table de points d'entrée qui normalise les opérations réalisées par chaque pilote de périphérique

Identification d'un périphérique

- classe (bloc/caractère)
- numéro majeur : identifiant du pilote (type du périphérique)
- numéro mineur : identifie un périphérique parmi ceux de son type

Interface pilote/noyau

- routines d'initialisation
- routines de traitement des interruptions
- points d'entrée (opérations d'E/S)
  2 tables, en fonction de la classe :
  - `bdevsw` : périphériques mode bloc
  - `cdevsw` : périphériques mode caractère

  Chaque entrée de la table correspond à un pilote (numéro majeur) et contient une série de pointeurs (points d'entrée) vers les routines de service du périphérique (accès, contrôle...)

Communication avec le système

tampons (blocs/caractères)

Structure et interface

<table>
<tr><td colspan="7">Interface logicielle</td></tr>
<tr><td rowspan="2">sockets</td><td rowspan="2">fichier simple</td><td rowspan="3">interface corrigée</td><td rowspan="3">interface brute</td><td rowspan="3">interface données brutes</td><td>interface données corrigées</td><td>mémoire virtuelle</td></tr>
<tr><td rowspan="2">protocoles</td><td rowspan="2">SGF</td><td>gestion du terminal</td><td>gestion de l'espace de swap</td></tr>
<tr><td>interface réseau</td><td colspan="3">pilotes de périphériques mode bloc</td><td colspan="2">pilotes de périphériques mode caractère</td></tr>
<tr><td colspan="7">Interface matérielle</td></tr>
</table>

Périphériques bloc

- disques, bandes... (taille courante d'un bloc : 1Ko-8Ko)
- utilisent une antémémoire (un cache) : « tampons de blocs »

Périphériques caractère

- tous les périphériques (sauf réseau) n'utilisant pas le tampon de blocs
  - `/dev/mem`, `/dev/null`
  - mémoire virtuelle
  - interface « brute » pour périphériques blocs (ex : `/dev/rdsk`)
- échanges par (flots d') octets
- gestion des tampons par le pilote lui-même (*C-listes*)

Système de fichiers standard Linux (FSSTND) (`man hier`)

Cette arborescence définit une organisation type pour les fichiers sous Linux qui se retrouve, avec quelques variantes, sur la plupart des systèmes Unix. Principaux répertoires de cette arborescence :

- **/bin** → programmes et commandes utilisateur de base : cat, cp, mv, rm...
- **/boot** → copie des données utiles au démarrage (blocs d'amorce...)
- **/etc** → fichiers de configuration locaux, comme passwd, group, ou utilisés parTCP/IP, comme services, inetd.conf, exports
  - **/etc/skel** → répertoire type d'un nouvel utilisateur
  - **/etc/rc.d** → scripts appelés par init au démarrage du système
  - **/etc/X11** → fichiers de config. X11 (Xconfig, Xmodmap, xinitrc...)
- **/conf** → fichiers de configuration. Si ce répertoire est présent, les fichiers de /etc sont des liens vers les fichiers de /conf
- **/dev** → pilotes de périphériques : console (console système), mouse, hda ((1er) disque IDE), ...,lp0 ((1ère) imprimante), ..., null (poubelle (sans retour)), sda (1er) disque SCSI)), sda1 (1ère partition de sda), sda2 (2e partition), ..., sdb ((2e) disque SCSI), ..., tty (terminal virtuel)...
- **/home** → répertoires des utilisateurs. Souvent sur une partition distincte.
- **/install** → informations sur les programmes installés
- **/lib** → code des bibliothèques partagées
- **/mnt** → montage de disquettes, de systèmes de fichiers via NFS...

- **/sbin** → commandes de base de démarrage et d'administration système : fdisk, fsck, getty, init, update, ifconfig, ping, lilo...
- **/tmp** → espace de manœuvre par tous les utilisateurs
- /var → fichiers variant souvent : traces, journaux, spoule (/var/spool/), verrous (/var/lock/)
- **/usr** → principaux programmes non nécessaires durant le démarrage. L'idée est de permettre de partager ce répertoire entre machines.
- **/usr/bin** → commandes utilisateur et programmes système. La séparation entre /usr/bin,/bin, et /sbin n'est pas toujours claire : il est préférable que ces répertoires contiennent des liens réciproques
- **/usr/bin/X11** → programmes du système de fenêtrage X
- **/usr/doc** → documentations qui ne sont pas au format man
- **/usr/include** → fichiers d'include des bibliothèques C
- **/usr/lib** → librairies statiques de divers langages, talons de liaison avec les bibliothèques partagées, et divers fichiers de configuration
- **/usr/lib/X11** → données X11 (polices, tables de couleur...)
- **/usr/man** → pages du manuel en ligne
- **/usr/src** → codes sources des programmes système
- **/usr/local** → installation de programmes supplémentaires. Comprend souvent des répertoires bin, etc, include, sda, et man
- **/usr/TeX** → installation du logiciel d'édition TeX

Plan

Disques durs mécaniques

<rédaction réservée>

Disques mémoire flash (SSD)

<rédaction réservée>

Disques RAID

<rédaction réservée>

Plan

Sûreté de fonctionnement

But : garantir l'intégrité des données conservées

- fiabilité (résistance aux pannes)
- sécurité(contrôles d'accès, intégrité, authentification, confidentialité)

Fiabilité

Moyen : redondance

Sauvegarde périodique

- complète/de reprise
- différentielle/par incréments :
  limitée aux fichiers modifiés depuis la dernière sauvegarde

Principe : combiner plusieurs fréquences de sauvegarde

Exemple

fréquence	nature	conservation
quotidienne	incrémentale	semaine
hebdomadaire	complète	mois
mensuelle	complète	an

Fiabilité : redondance interne

Principe : fournir plusieurs moyens d'établir une information

Exemples

- dupliquer les descripteurs
- stocker l'identifiant du fichier dans chacun des blocs du fichier
- chaînage double des blocs d'un fichier

Remarques

- Il est souvent préférable que les copies soient « éloignées »
- Un principe général est de combiner
  - une information centralisée (efficace) (ex : table d'allocation)
  - et une information « dispersée » (robuste)
    (ex : informations au niveau des blocs)
- La redondance peut être exploitée
  - statiquement (en cas de défaillance)
  - dynamiquement (calculs d'accès en double)

Sécurité

Moyens

- cryptage + clés (session, fichiers, répertoires, opérations)
- listes d'accès
  matrice : opérations permises, par utilisateur, et par fichier
  - raccourcis : droits par défaut, groupes d'utilisateurs
  - opérations élémentaires (lire/écrire/exécuter...)
- capacités
  - clé autorisant l'accès, attribuée en fonction des identificateurs de l'objet/de l'utilisateur
  - structure d'une capacité

  - pour être robuste, le contrôle doit être fait à chaque accès (et non à l'ouverture/au premier accès seulement)

Mise en œuvre des listes d'accès sous UNIX : ACLs (Access Control Lists)

Documentation complémentaire : `https://www.usenix.org/legacy/publications/library/proceedings/usenix03/tech/freenix03/full_papers/gruenbacher/gruenbacher_html/main.html`

Généralisation du mécanisme de droits d'accès Unix (`ugo/rwx`)

Objectif

Mise en place de politiques de sécurité plus fines :

- Possibilité d'allouer des droits distincts au sein d'un même groupe
- Allocation de droits sur une base individuelle, pour chaque fichier

Interface

- ACL = suite d'Access Control Entries (ACE)
- Forme d'une ACE : entrée : [uid/gid] : permissions
- Deux commandes :
  - Lecture : getfacl *fichier*...
  - Mise-à-jour : setfacl -x|-m|-s [ACL] *fichier*...
- Politique de sécurité : seuls le propriétaire et l'usager `root` peuvent modifier l'ACL d'un fichier

ACLs : ACE

Permissions classiques

Propriétaire :	user : :rwx
Groupe du propriétaire	group : :rwx
Autres	other : :rwx

Permissions étendues

Usager nommé	user : nom :rwx
Groupe nommé	group : nom : rwx
Masque	mask : :rwx

Rôle du masque : limiter les permissions des usagers/groupes nommés

Héritage d'ACL entre répertoires

default : <déclaration d'ACE>

Sémantique

- Un répertoire hérite de la liste *default* à la fois en tant que liste d'ACL effective et liste *default* pour lui-même ;
- La liste *default* inhibe le filtre *umask*.

ACLs : exemple

```
$ umask 027
$ mkdir dir
$ ls -dl dir
drwxr-x-- ... agruen suse ... dir

$ getfacl dir
# file: dir
# owner: agruen
# group: suse
user::rwx
group::r-x
other::--

$ setfacl -m user:joe:rwx dir
$ getfacl -omit-header dir
user::rwx
user:joe:rwx
group::r-x
mask::rwx
other::--

$ ls -dl dir
drwxrwx--+ ... agruen suse ... dir
```

Plan

Méthodes d'accès

Objectif

organiser un ensemble de données pour améliorer l'efficacité des accès, en fonction des patrons (motifs) d'accès aux éléments de cet ensemble

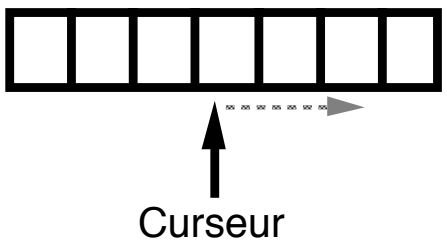
- Parcours de l'ensemble complet → accès séquentiel
- Accès « ponctuel »
  - à des enregistrements identifiés → accès indexé
  - par le contenu → accès associatif (listes inverses)

Accès séquentiel

Métaphore : bande magnétique

Structure de données ≈ file

- Les enregistrements sont vus comme une suite
- Un curseur (ou index) désigne l'enregistrement accessible



- Opérations de base :
  - ouverture/revenir_au_début : curseur sur position 1
  - lecture/passer_au_suivant
    - renvoie la valeur de l'enregistrement désigné par le curseur
    - l'index progresse d'une position
  - ajout/suppression (écriture)
    - normalement en fin de fichier
    - certaines organisations offrent la possibilité de fixer la position du curseur et d'écrire à partir de celle-ci (flots Unix)

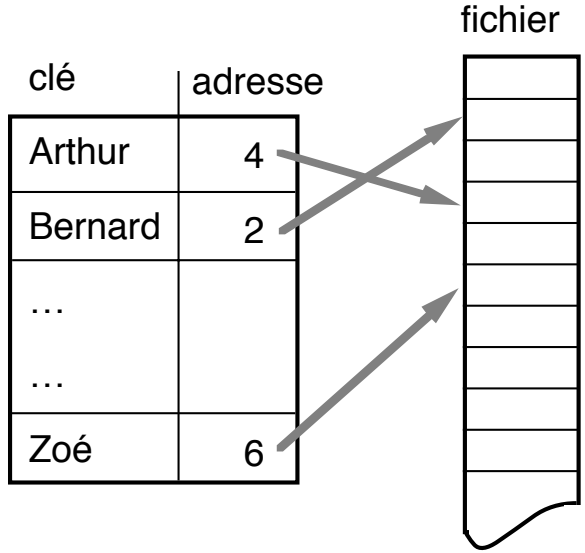
Accès direct

But : retrouver un enregistrement à partir (d'une partie) de son contenu

Clés ordonnées : accès indexé

Clé $\triangleq$ partie de l'enregistrement qui permet de l'identifier de manière unique

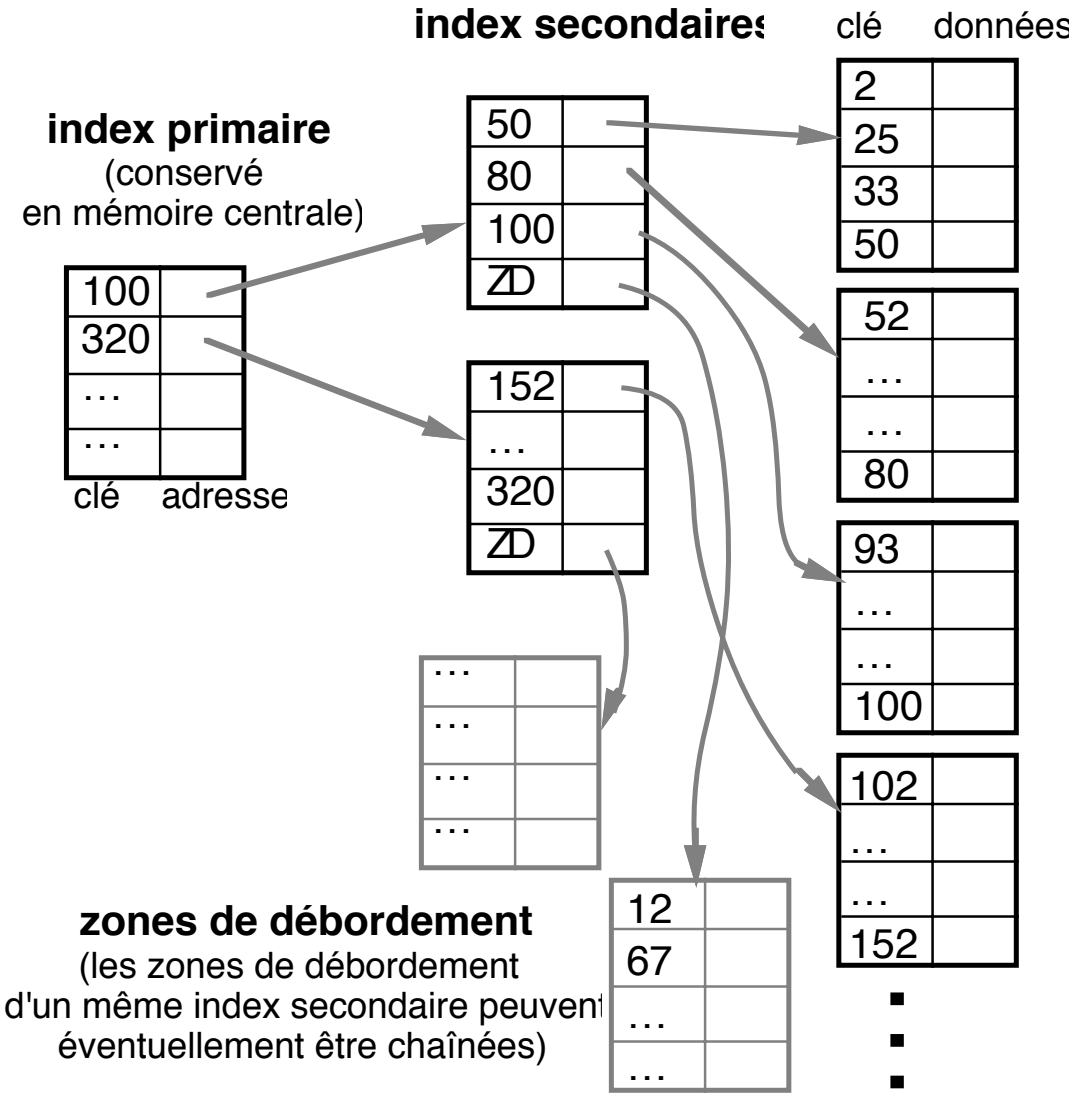
$\rightarrow$ accès à l'enregistrement via une table : < clé, adresse >



Accélérer la recherche d'une clé $\rightarrow$ structuration arborescente

arbres équilibrés (B-arbres) : toutes les branches restent de hauteur voisine, quelle que soit l'évolution de l'arbre

Faciliter les accès séquentiels : séquentiel indexé



- structuration arborescente
- niveau logique « réduit » (adresses physiques)
- un bloc contient des enregistrements dont les clés se suivent (par rapport aux valeurs existant dans le fichier)
- nombre de niveaux réduit/fixe → zones (blocs) de débordement

Adressage dispersé (hachage, «hash-code»)

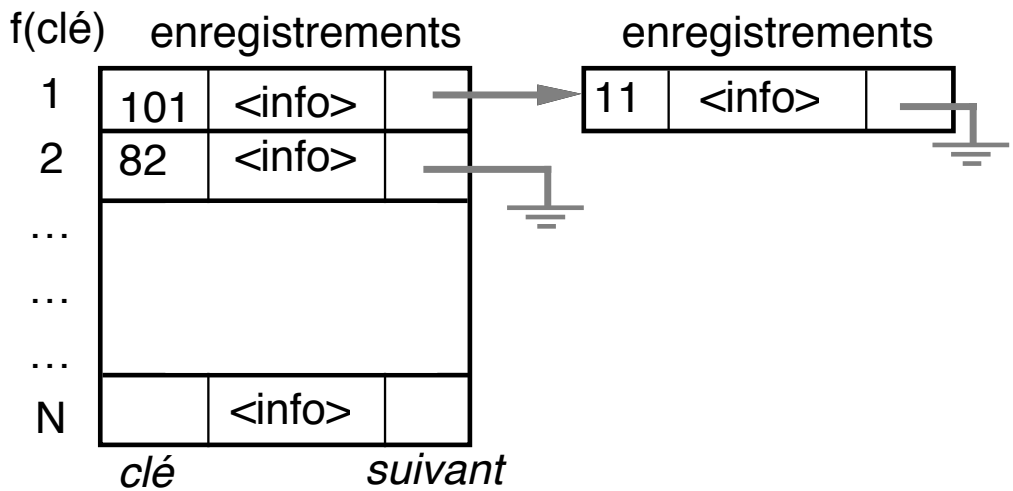
But : éviter la gestion/le stockage d'une table ordonnée <clé,adresse>

→ déterminer l'adresse à partir de la clé :
adresse = f(clé) (f : fonction de dispersion (de hachage))

Idéal

Pour un fichier de N enregistrements,
bijection entre les valeurs de clés présentes et 1,2...,N

Réalité : collisions (f non injective) → f renvoie vers une (tête de) liste



Remarque : modulo N est une fonction facile à mettre en œuvre

Accès associatif : listes inverses

Généralisation de l'accès indexé : permettre de retrouver les enregistrements ayant une valeur donnée, pour un champ donné

Multiliste

Pour chacun des champs pour lequel un accès par la valeur est souhaité

- ajouter un champ « pointeur » à chaque enregistrement
  $\rightarrow$ une liste regroupe les enregistrements de même valeur
- ajouter une table (index «secondaire»)
  $<$ valeur, pointeur vers la tête de liste associée à la valeur $>$

Fichier inversé

Pour chacun des champs pour lequel un accès par la valeur est souhaité

- un index regroupe, pour chaque valeur,
  la liste des clés (ou des adresses) des enregistrements dont le champ possède cette valeur
- l'enregistrement ne contient pas de valeur,
  mais un pointeur vers l'entrée de l'index contenant la valeur