

# ASR2 Système Système de Fichier

Stéphanie Moreaud

Département d'informatique  
IUT Bordeaux 1

# Plan

- 1 Fichiers et système de fichiers
- 2 Organisation du système de fichiers
- 3 Représentation des répertoires
- 4 Autres caractéristiques

# Bibliographie



TANENBAUM (A.), *Systèmes d'exploitation*. Pearson Education.



BILLAUD (M.), *Cours de ASR2-Système*.

<http://www.labri.fr/perso/billaud/>.

D'après les transparents de Michel Billaud.

<http://www.labri.fr/perso/billaud/>

# Plan

- 1 Fichiers et système de fichiers
  - Système de Gestion de Fichiers
- 2 Organisation du système de fichiers
- 3 Représentation des répertoires
- 4 Autres caractéristiques

# Fichiers

Sauvegarde (pérenne) de l'information sous forme de **fichiers**

- lot d'informations portant un nom,
- le plus souvent concervés en mémoire de masse

→ un **contenu**

→ des **méta-données** (attributs du fichier)

- taille
- propriétaire
- droits d'accès
- date de création
- date de dernier accès
- ...

# Fichiers

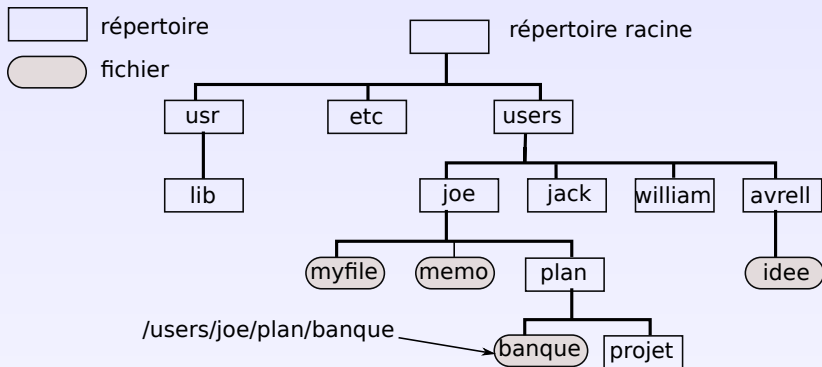
Les fichiers sont classés dans des répertoires

- chaque répertoire peut contenir d'autres répertoires

→ organisation arborescente

→ structure de données appelée système de fichiers.

# Système de fichiers



Pour l'utilisateur : arborescence

Fichiers/répertoires accessibles par leur nom

→ chemin d'accès

# Système de Gestion de Fichiers

Fonction : gestion des structures de fichiers

- Manipulation des fichiers : création, destruction, etc...
- Allocation sur la mémoire secondaire
- Localisation des fichiers : accès au contenu
- Sécurité et contrôle des fichiers
- Fiabilité en cas de panne
- ...

# Plan

- 1 Fichiers et système de fichiers
- 2 Organisation du système de fichiers
  - Allocation contiguë
  - Allocation par liste chaînée
  - Tables des blocs Unix
- 3 Représentation des répertoires
- 4 Autres caractéristiques

# Organisation du système de fichiers

Fichier = unité d'information abstraite

Organisation des données varie selon les systèmes de fichiers

- allocation disques sur des **blocs** de données
- choix des blocs qui composent chaque fichier

Quelques représentations possibles

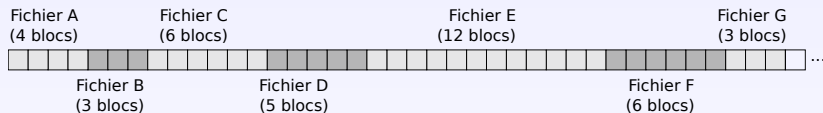
- catalogues de fichiers
- tables d'allocation (FAT)
- tables de blocs

# Allocation contiguë

Stockage de chaque fichier dans une **suite de blocs consécutifs**.

Exemple : Allocation d'un fichier de 24 Ko

- taille des blocs sur le disque = 2 Ko
- ➔ utilisation de 12 blocs consécutifs.



# Catalogue de fichiers

Mise en œuvre : catalogue de fichiers

Exp : VTOC = Volume Table of Contents (IBM)

- pas de répertoires,
- fichiers composés de blocs contigus,
- fichiers alloués les uns à la suite des autres sur le disque

# Catalogue de fichiers

Au début du disque, une **table de fichier** qui indique

- le **nom** de chaque fichier
- son emplacement (**position premier bloc, taille**)

<b>nom</b> du fichier	position du <b>premier bloc</b>	<b>taille</b>
CLIENTS	10	50
PRODUITS	60	500
FACTURES	560	2000
...	...	...

# Catalogue de fichiers

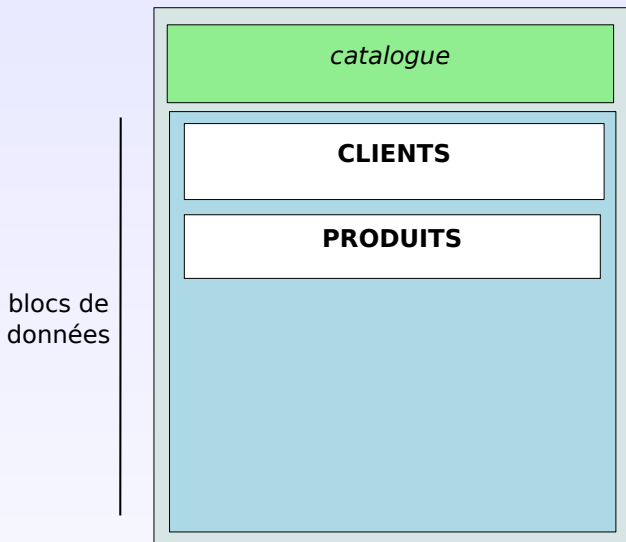
Dans le catalogue :

- table des fichiers
- liste des **espaces libres**

Reste du disque : contenu des fichiers

- blocs de données du premier fichier,
- puis blocs du second,
- etc...

# VTOC : occupation du disque



# Gestion de l'espace

## Espaces contigus

- **Réservation** d'espace à la création d'un fichier
- **Restitution** quand le fichier est supprimé
- **Extension** des fichiers ?

# Avantages/Inconvénients

## Avantages

- simplicité
  - information nécessaire : adresse premier bloc, nb de blocs
- performances
  - déplacements de bras minimisés

## Inconvénients

- perte de place
  - réservations non utilisées
- espaces contigus de taille variable → fragmentation

# Solutions

Allocation dans plusieurs zones

- 1 fichier = plusieurs zones
- allouées au besoin (dynamiquement)

Exemple : réservation d'un fichier de 20 Ko + 5 extensions de 10 Ko

Utilitaire de réorganisation du disque

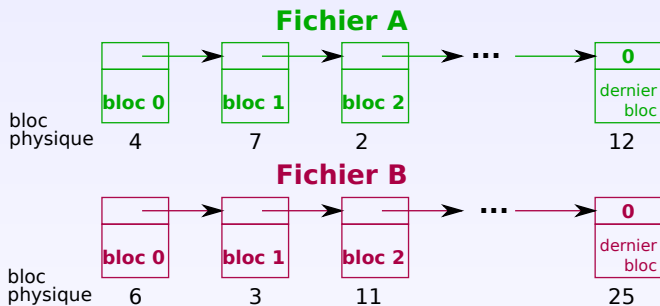
Remarque : allocation contiguë adaptée à d'autres média

- CD-ROM

# Allocation par liste chaînée

Fichiers conservés sous forme d'une liste chaînée de blocs disque

- premier mot de chaque bloc = pointeur sur le bloc suivant



# Allocation par liste chaînée

## Fichiers non contigus

- entrée du catalogue contient adresse du premier bloc
- extension facile à gérer
- pas de perte d'espace
  
- taille des données d'un bloc = puissance de 2 - taille pointeur
- parcours séquentiel des blocs → multiples accès aléatoires sur le disque

Solution : utilisation d'une **table d'allocation**

# Table d'allocation

Table contenant la position du bloc suivant

bloc physique	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
index bloc suivant			12		7			2					-1		

↑  
début fichier A
↑  
fin fichier A

- Pas de pointeurs au début des blocs
- Table chargée en mémoire principale
  - parcours de la table avant tout accès disque
  - inconvénient : espace mémoire occupé par la table
    - exp : disque de 200 Go, blocs de 1 Ko  
→ 200 millions d'entrées

→ Système **FAT** (*File Allocation Table*)

# Catalogue de fichiers

- Table des fichiers

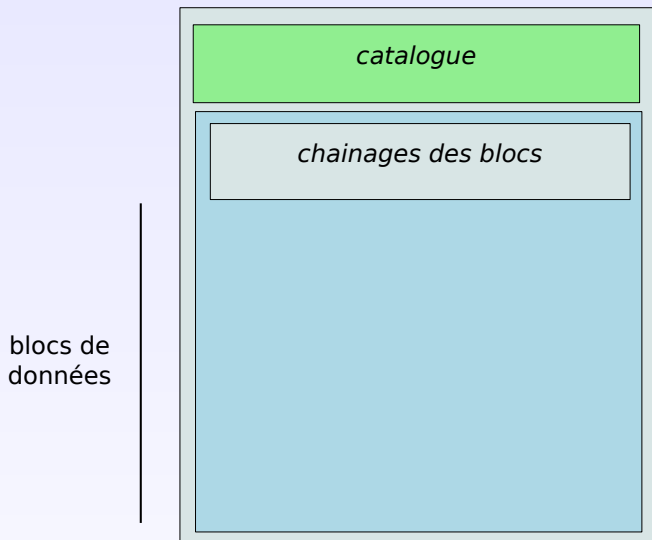
nom du fichier	position du premier bloc	taille
CLIENTS	10	50
PRODUITS	12	500
FACTURES	15	2000
...	...	...

- Table de chaînage des blocs

	...	10	11	12	13	...
bloc suivant	...	11	20	13	14	...

- puis blocs de données

# FAT : occupation du disque



# Autre représentation

Table des blocs intégrée dans le catalogue

nom	taille	B1	B2	B3	B4	....	B16
CLIENTS	3	10	11	22	-		-
PRODUITS	4	20	11	42	-		-
...							
FACTURES	20	101	102	103	104	...	116
...							
FACTURES	-	117	118	119	120	...	-
...							

→ utilisation de "lignes de continuation"

Représentation utilisée dans CP/M

# Utilisation de nœuds d'index (UNIX)

A chaque fichier est associé une **structure de données** appelée *i-node* (index-node) qui contient :

- les attributs (taille, propriétaire, droits...)
- les adresses de ses premiers blocs de données

# Utilisation de nœuds d'index (UNIX)

A chaque fichier est associé une **structure de données** appelée *i-node* (index-node) qui contient :

- les attributs (taille, propriétaire, droits...)
- les adresses de ses premiers blocs de données
- l'adresse d'un **bloc d'indirection simple**
  - contient d'autres adresses de **blocs de données**

# Utilisation de nœuds d'index (UNIX)

A chaque fichier est associé une **structure de données** appelée *i-node* (index-node) qui contient :

- les attributs (taille, propriétaire, droits...)
- les adresses de ses premiers blocs de données
- l'adresse d'un **bloc d'indirection simple**
  - contient d'autres adresses de **blocs de données**
- l'adresse d'un **bloc d'indirection double**
  - contient des adresses de **blocs d'indirection simple**

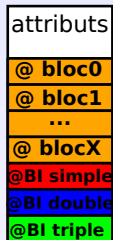
# Utilisation de nœuds d'index (UNIX)

A chaque fichier est associé une **structure de données** appelée *i-node* (index-node) qui contient :

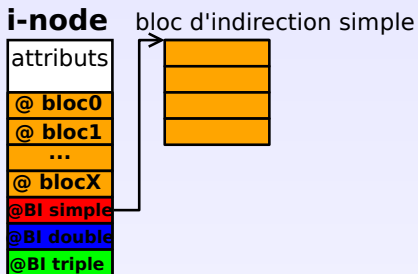
- les attributs (taille, propriétaire, droits...)
- les adresses de ses premiers blocs de données
- l'adresse d'un **bloc d'indirection simple**
  - contient d'autres adresses de **blocs de données**
- l'adresse d'un **bloc d'indirection double**
  - contient des adresses de **blocs d'indirection simple**
- l'adresse d'un **bloc d'indirection triple**
  - contient des adresses de **blocs d'indirection double**

# I-nodes et blocs d'indirection

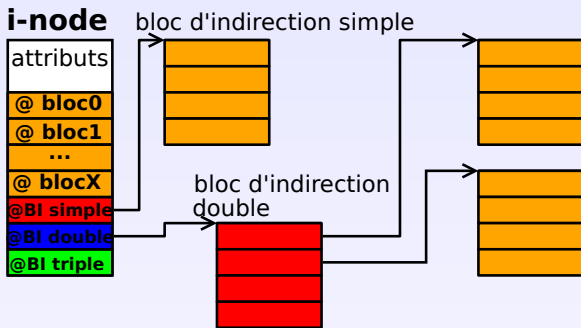
## **i-node**



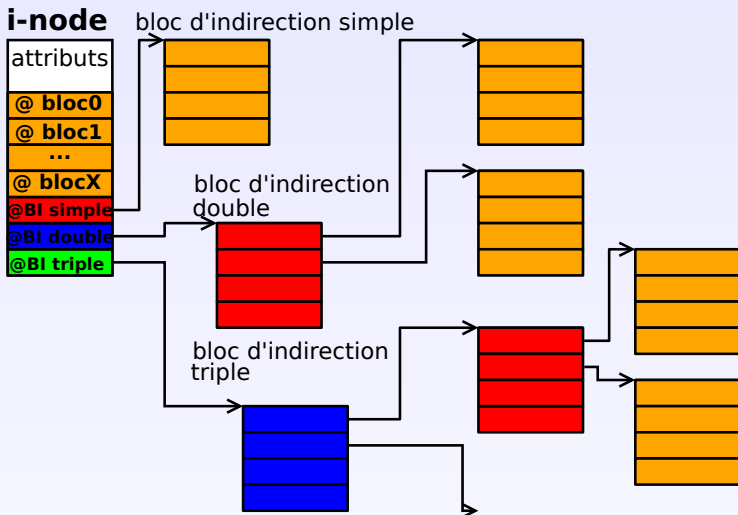
# I-nodes et blocs d'indirection



# I-nodes et blocs d'indirection



# I-nodes et blocs d'indirection



# Chiffrage

Supposons :

- des blocs de 4 Ko ( $2^{12}$ )
- des adresses sur 32 bits

Capacité maximale du disque ?

# Chiffrage

Supposons :

- des blocs de 4 Ko ( $2^{12}$ )
- des adresses sur 32 bits

Capacité maximale du disque ?

Avec 32 bits on peut adresser

$2^{32}$  blocs

# Chiffrage

Supposons :

- des blocs de 4 Ko ( $2^{12}$ )
- des adresses sur 32 bits

Capacité maximale du disque ?

Avec 32 bits on peut adresser

$2^{32}$  blocs

le disque peut ainsi contenir *en théorie*

$$2^{32} \times 2^{12} = 2^{44} \text{ octets} = 16 \text{ Tera octets}$$

# Chiffrage

Supposons :

- des blocs de 4 Ko ( $2^{12}$ )
- des adresses sur 32 bits

Capacité maximale du disque ?

Avec 32 bits on peut adresser

$2^{32}$  blocs

le disque peut ainsi contenir *en théorie*

$$2^{32} \times 2^{12} = 2^{44} \text{ octets} = 16 \text{ Tera octets}$$

Question : taille maximum d'un fichier ?

# Taille maximum d'un fichier

- une adresse = 32 bits = 4 octets
- un bloc = 4 Ko

# Taille maximum d'un fichier

- une adresse = 32 bits = 4 octets
- un bloc = 4 Ko
  - peut contenir  $4096 = 1024$  adresses.

# Taille maximum d'un fichier

- une adresse = 32 bits = 4 octets
- un bloc = 4 Ko
  - peut contenir  $4096 = 1024$  adresses.

Donc

- un **bloc d'indirection simple** conduit à 1024 blocs de données

# Taille maximum d'un fichier

- une adresse = 32 bits = 4 octets
- un bloc = 4 Ko
  - peut contenir  $4096 = 1024$  adresses.

Donc

- un **bloc d'indirection simple** conduit à 1024 blocs de données soit  $1024 \times 4\text{Ko}$  soit 4 Mo de données

# Taille maximum d'un fichier

- une adresse = 32 bits = 4 octets
- un bloc = 4 Ko
  - peut contenir  $4096 = 1024$  adresses.

Donc

- un **bloc d'indirection simple** conduit à 1024 blocs de données soit  $1024 \times 4\text{Ko}$  soit 4 Mo de données
  - un **BI doubles** conduit à 1024 BI simple (4 Go)
  - un **BI triple** conduit à 1024 BI double (4 To)
- Pour la plupart des accès, une indirection suffit

# Bilan

## Fichiers contigus :

- temps d'accès : très bonne performances
- problème de gestion des espaces libres
- convient très bien à des supports en lecture seulement (CD, DVD)

## Blocs chaînés, table des blocs, i-nodes

- gestion souple et efficace de l'espace
- problèmes de performance si les données sont dispersées

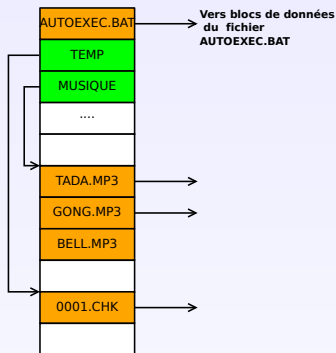
# Plan

- 1 Fichiers et système de fichiers
- 2 Organisation du système de fichiers
- 3 Représentation des répertoires**
  - Comme des catalogues de fichiers
  - Comme des fichiers de données
    - Exemple
  - Gestion des blocs libres
    - Vérification du système de fichiers

- 4 Autres caractéristiques

# Comment représenter les arborescences ?

- Comme des catalogues de fichiers
- catalogue arborescent,
  - entrées spéciales dans la table des fichiers



# Catalogues de fichiers

Répertoires matérialisés dans le catalogue par des lignes spéciales qui renvoient vers d'autres lignes

- ne permet pas d'avoir des liens, seulement des *raccourcis*
- solution adoptée par CP/M, MS/DOS, Windows...

types de ligne	information
fichier	taille, blocs
vide	
répertoire	numéro de ligne
raccourci	chemin destination

# Représentation des répertoires

→ Comme des fichiers de données

Solution utilisée sous Unix

Un système de fichiers contient

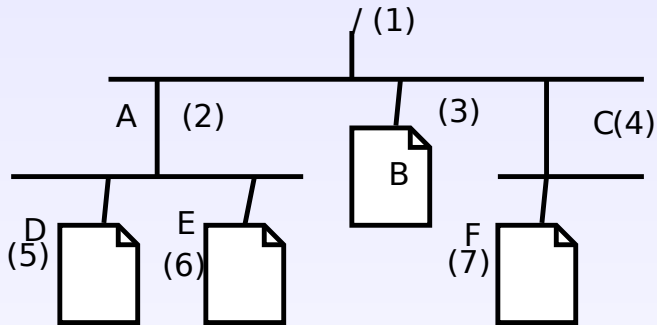
- une *table d'i-nodes* (noeuds d'information)
- des *blocs de données* liés à ces i-nodes

Un fichier/répertoire est identifié par son numéro d'i-node

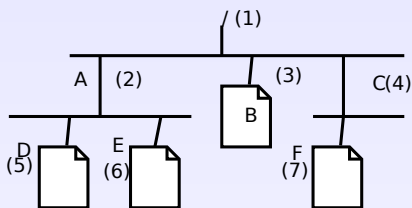
# Différents types d'i-nodes

types	donnée
fichiers	Blocs = contenu du fichier
répertoires	Blocs = table de noms, n° i-node
liens symboliques	chemin d'accès
périphériques	type, majeur, mineur
...	

# Exemple d'arborescence



# Table des inodes



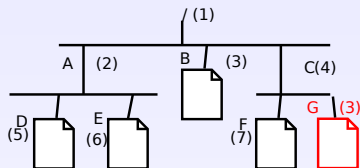
N°	type	CR	contenu des blocs
1	d	4	..=1, .=1, A=2, B=3, C=4
2	d	2	..=1, .=2, D=5, E=6
3	f	1	"coucou"
4	d	2	..=1, .=4, F=7
		...	

CR = compteur de références

# Compteur de référence

Indique combien de fois l'objet est référencé  
 CR = 0, on peut récupérer l'espace qu'il occupe.

Après ln /B /C/G



N°	type	CR	contenu des blocs
1	d	4	..=1, .=1, A=2, B=3, C=4
2	d	2	..=1, .=2, D= 5, E=6
3	f	2	"coucou"
4	d	2	..=1, .=4, F=7, G=3
		...	

# Gestion des blocs libres

Le système possède

- une liste des blocs libres
- un tableau de marquage des blocs occupés

# Vérification du système de fichiers

Utilitaire `fsck`, descente de l'arborescence :

- 1 vérification des i-noeuds, des blocs et des tailles
- 2 vérification de la structure des répertoires
- 3 vérification de la connectivité des répertoires
- 4 vérification des compteurs de référence
- 5 vérification de l'information du sommaire de groupe

→ peut être très long.

# Vérification du système de fichiers

Vérification des i-noeuds, des blocs et des tailles

- i-noeuds non-détruits  $\Rightarrow$  blocs alloués.
- blocs alloués  $\Rightarrow$  i-noeud.

Vérification de la structure des répertoires

- numéro d'i-noeud cité dans un répertoire  $\Rightarrow$  i-noeud existant

Vérification de la connectivité des répertoires

- i-noeuds actifs  $\Leftrightarrow$  accessibles depuis la racine

Vérification des compteurs de référence

- Nombre de références recalculé = nombre de références indiqué dans l'i-noeud

# Plan

- 1 Fichiers et système de fichiers
- 2 Organisation du système de fichiers
- 3 Représentation des répertoires
- 4 **Autres caractéristiques**
  - Journalisation
  - Snapshots (clichés)
  - Conclusion

# Autres caractéristiques : Journalisation

## Systemes de fichiers journalisés, journal :

- garde une trace des opérations d'écriture non terminées
  - permet de les reprendre en cas d'arrêt brutal
- Pas de pertes d'informations
- reprise sur incidents plus rapide (évite le *fsck*)

# Snapshots (clichés)

Pendant la durée d'une sauvegarde,

- on ne veut pas que le système de fichiers soit modifié
- on ne veut pas arrêter l'exploitation

**Cliché** : copie de l'état du système de fichiers à un moment donné

On ne copie que *ce qui a changé* à partir du moment du cliché.

# Conclusion

## Objectifs

- API pour accès aux fichiers/répertoires
- indépendance vis à vis du support
- fiabilité

## Fonctionnalités

- arborescences
- droits d'accès
- accès concurrents, ...

## Performances

- liées à l'implémentation
- liées au contexte d'usage