

# Les systèmes d'exploitation

Wadoud BOUSDIRA<sup>1</sup>  
wadoud.bousdira@univ-orleans.fr

<sup>1</sup>LIFO, University of Orléans  
**Orléans, France**

Orléans, 2023

## Deux visions

- utilisateur :
  - ▶ arborescence de répertoires et fichiers
  - ▶ créer
  - ▶ libérer
  - ▶ modifier
- SE
  - ▶ unité de base = flot de données.

## Fichier logique

Vue que l'utilisateur de la machine a de la conservation de ses données

- un type de données standard défini dans les langages de programmation, sur lequel un certain nombre d'opérations spécifiques peuvent être réalisées
- un ensemble d'enregistrements ou d'articles.
  - ▶ enregistrement : type de données regroupant des données de types divers liées entre elles par une sémantique inhérente au programme qui les manipule,
  - ▶ constitue pour le programme une unité logique de traitement,
  - ▶ différents modes d'accès : **séquentiel**, **indexé**, **direct**.

## Méthodes d'allocation de la mémoire secondaire

- fichier logique  $\longleftrightarrow$  fichier physique : les enregistrements doivent être écrits dans les secteurs composant les blocs du disque qui le forment
- Fichier physique = {blocs physiques alloués au fichier}

Il faut connaître à tout moment l'ensemble des blocs libres  $\rightsquigarrow$  **gérer l'espace libre sur le disque.**

# Les fichiers

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

public class CopierSansBlanc {
    public static void copier(String fSource, String fDest) throws IOException {
        PrintWriter out = new PrintWriter(new FileWriter(new File(fDest)));
        Scanner in = new Scanner(new File(fSource));
        while (in.hasNextLine()) {
            String ligne = in.nextLine();
            Scanner sc = new Scanner(ligne);
            String ligneRes = "";
            if (sc.hasNext()) ligneRes = sc.next();
            while (sc.hasNext()) ligneRes = ligneRes + " " + sc.next();
            out.println(ligneRes);
        }
        out.close();
    }
}
```

Sur disque dur,

- le fichier occupe un certain nombre de pistes,
- chaque piste est découpée en secteurs, pas nécessairement physiquement consécutifs,
- une piste contient *généralement* 512 caractères.

Système de répertoire pour retrouver un fichier

- un fichier possède un nom, et le répertoire permet de faire correspondre ce nom à l'emplacement physique sur le disque (cylindre-piste-secteur)
- en général, *16 secteurs de 512 octets/bloc*.

## Aujourd'hui,

- Les contrôleurs de disque masquent de plus en plus au système la structure physique du disque,
  - ▶ gestion optimisée des pistes et des cylindres.
- Le disque est présenté comme une séquence de blocs logiques repérés par leur numéro d'ordre.

## Système de fichiers

désigne

- le principe d'organisation des fichiers,
- les éléments logiciels qui réalisent ce principe,
- un ensemble de fichiers organisés selon ce principe.

On peut avoir plusieurs systèmes de fichiers sur un même disque,  $\rightsquigarrow$  disque partagé en partitions.



## Adressage

Un chemin consiste en une chaîne constituée de noms de fichiers (non vide et ne contenant pas /) séparés par des /.

- chemin absolu. Ex. */usr/bin/sh*
- chemin relatif. Ex. *sys/slides/chapitre2.tex*

Les éléments intermédiaires sont des répertoires ou des liens symboliques vers des répertoires.

- répertoire courant .
- répertoire parent ..

## Point de montage

Traditionnellement, sous Unix, l'arborescence est créée en **montant** (*mount*) des systèmes de fichiers en des répertoires de l'arborescence existante.

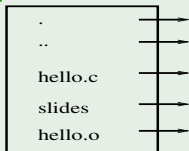
À l'initialisation du système, un système de fichiers racine est monté sur `/`, la racine absolue du système.

Il est possible de monter/démonter des systèmes de fichiers à *chaud*.

Les fichiers de l'arborescence peuvent donc se trouver sur des supports physiques différents (transparent pour l'utilisateur).

## Types de fichiers

- **Fichier régulier** Séquence d'octets quelconque, adressable, de longueur variable, modifiable à travers l'API Posix  
∃ une taille maximale
- **Répertoire** associe des noms à des fichiers, définit toujours . et ..



- **Lien symbolique** alias défini par un chemin (fichier contenant un chemin)

Il y a d'autres types de fichiers :

- **Tube nommé** mécanisme de communication entre processus  
(aucune donnée associée)
- **Fichier spécial** mécanisme d'E/S vers les périphériques  
(aucune donnée associée)
- **Socket** n'apparaît pas dans l'arborescence mais se manipule comme un fichier, mécanisme d'E/S réseau

## Super-bloc

Le super-bloc est une méta-donnée qui contient


- la taille du système de fichiers,
- le nombre de blocs libres,
- le début de la liste des blocs libres.

Reproduit en plusieurs copies à des emplacements précis (**vital !**).

## *i*-liste

La *i*-liste est une table d'index qui permet de retrouver les fichiers.

## constituée de *i*-nœuds

- un *i*-nœud contient
  - ▶ les pointeurs qui permettent de retrouver le fichier
  - ▶ des informations (attributs) du fichier :
    - les droits d'accès
    - l'identifiant numérique du propriétaire
    - sa taille,
    - la date du dernier accès au fichier,
    - sa date de dernière modification
    - ...
- chargé en mémoire seulement si le fichier associé est en cours d'utilisation 
- en Unix, on consulte un *i*-nœud avec la commande `ls`.

# Descripteur de fichier

Dans l'API Posix, les fichiers sont manipulés à travers des descripteurs.

Un descripteur de fichier est un entier indexant une table propre à chaque processus. Par convention :

- 0 entrée standard */dev/stdin*
- 1 sortie standard */dev/stdout*
- 2 sortie d'erreur */dev/stderr*

```
Ex. find / -name "*.c" -maxdepth 2 -print 2>/dev/null
```

Chaque entrée correspond à une structure de données contenant :

- des informations sur le fichier associé
- le mode d'ouverture (lecture, écriture)
- la position courante dans le fichier (offset)

Les droits d'accès sont vérifiés lors de la création du descripteur.

- Le disque est découpé en blocs `inode` et `data`.
- Un fichier = un `i-nœud`
- L'`i-nœud` pointe vers les données associées. Les répertoires associent noms et `i-nœuds`.
- La racine est un `i-nœud` fixé.

Le SE garantit :

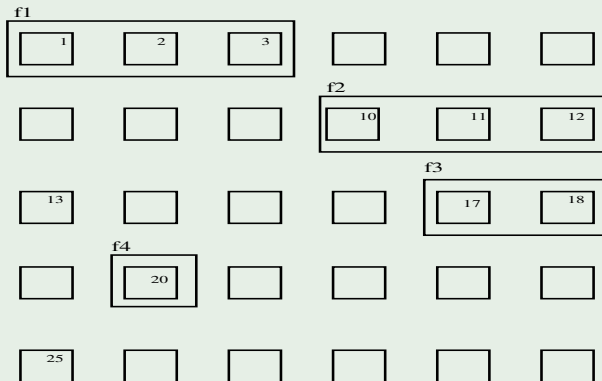
- pas plus d'un lien vers un répertoire (DAG)
- la gestion des blocs libres
- la récupération des incohérences

Il permet aussi l'accès concurrent et utilise des caches et la multiprogrammation pour optimiser les opérations.



## Allocation contiguë

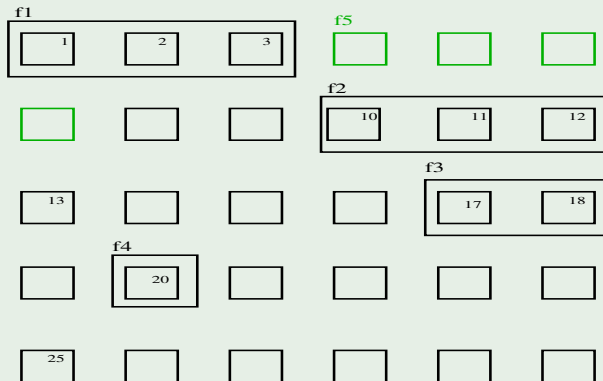
Un exemple : allocation du fichier *f5* d'une taille maximale évaluée de 4 blocs.



# Allocation contiguë

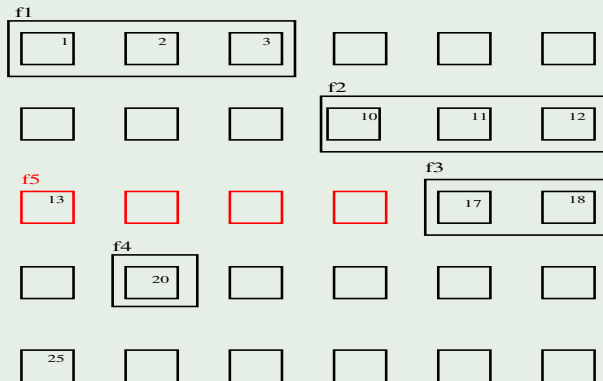
## First fit

blocs 4, 5, 6, 7.



## Best Fit

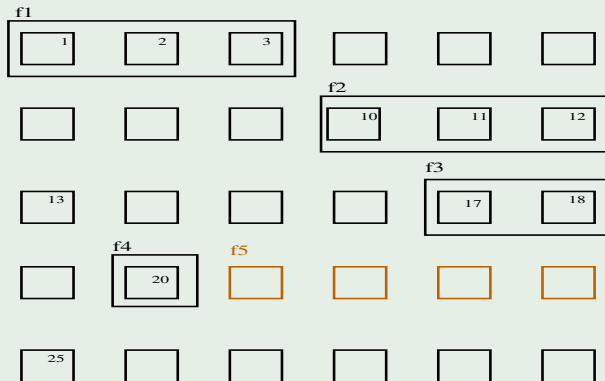
Allocation du fichier *f5* d'une taille maximale évaluée de 4 blocs :  
blocs 13, 14, 15, 16.






# Allocation contiguë

## Worst Fit

Allocation du fichier  $f5$  d'une taille maximale évaluée de 4 blocs :  
blocs 21, 22, 23, 24.



- Problèmes de fragmentation  $\rightsquigarrow$  déplacer les blocs du disque 
- Extension d'un fichier : si les blocs disques voisins ne sont pas libres ? 
- Contiguïté des blocs  $\Rightarrow$  bonnes performances pour l'accès aux différents blocs d'un même fichier 

## Allocation par zones

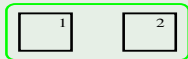
Un fichier peut être constitué de plusieurs zones physiques distinctes

- une zone est allouée dans un ensemble de blocs contigus
- 1<sup>ère</sup> zone allouée : zone primaire, les autres : zones secondaires
- la taille des zones est en général définie à la création du fichier
- le nombre des zones secondaires autorisées est limité.

# Allocation par zones

## Exemple

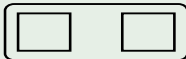
f1\_ZP



f1\_ZS



f5\_ZP



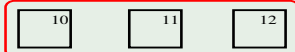
f1\_ZS



f3\_ZS



f2\_ZP



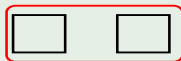
f3\_ZP



f4\_ZP





f2\_ZS



25




- Diminue les performances d'accès aux blocs d'un même fichier 
- Minimise les problèmes de fragmentation externe  mais ne les résout pas. . .





## Allocation par blocs chaînés


Un fichier est constitué d'une liste chaînée de blocs physiques, dispersés **n'importe où** sur le support de masse.

- Chaque bloc contient l'adresse du bloc suivant dans le fichier

- ▶ extension simple 

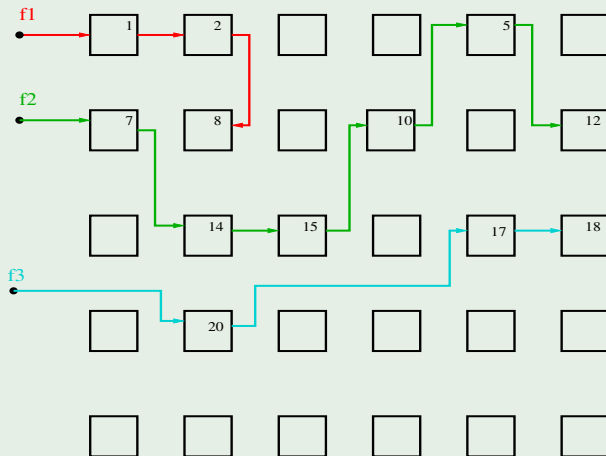
- ▶ pas de problèmes de fragmentation externe 

- ▶ le seul mode d'accès utilisable est le mode d'accès séquentiel 

- ▶ place occupée dans chaque bloc par le chaînage de la liste 

# Allocation par blocs chaînés

## Exemple

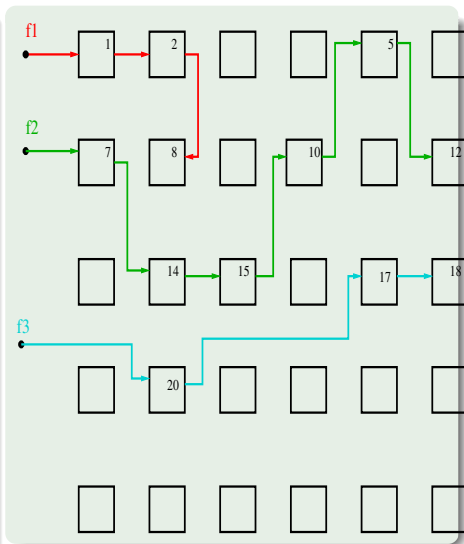


## Une variante (Windows)


- L'ensemble des chaînages des blocs d'un fichier est regroupé dans une table appelée *FAT (File Allocation Table)*
- chaque entrée de la FAT correspond à un bloc du disque
  - ▶ si bloc  $\in$  à un fichier, et si pas dernier bloc, l'entrée contient le numéro du bloc suivant du fichier,
  - ▶ si bloc  $\in$  à un fichier, et si dernier bloc, l'entrée vaut une valeur de fin de fichier,
  - ▶ si bloc  $\notin$  fichier, l'entrée contient une valeur de bloc libre.

# Allocation par blocs chaînés

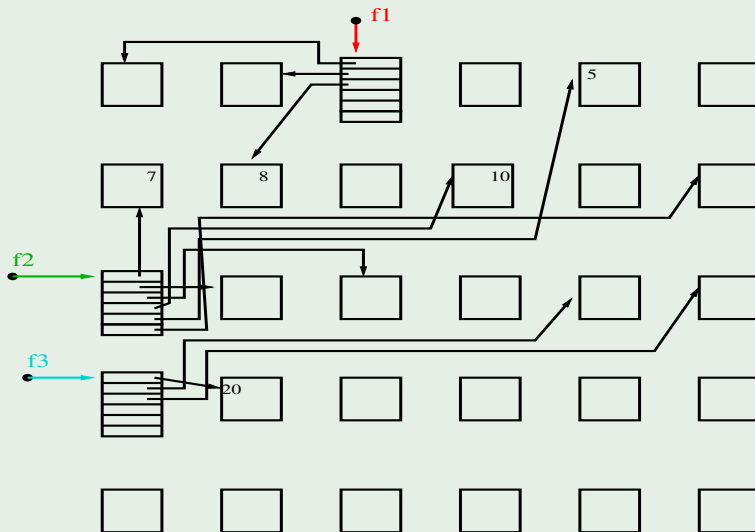
## Une variante





## Allocation indexée (Unix)

- Les adresses des blocs physiques d'un fichier sont rangées dans une table : **index**
- l'index est rangé dans un bloc du disque,
- accès directs aux blocs du fichier via l'index. 

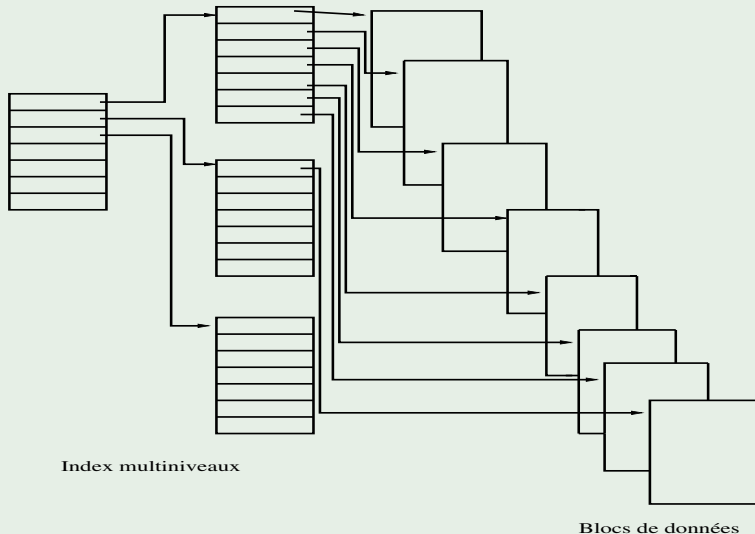
# Allocation indexée



- la taille de la table d'index est conditionnée par celle d'un bloc physique et par le nombre de blocs existants sur le disque,
  - ▶ si bloc grand, alors nombre d'entrées dans le bloc d'index faible ~→ fragmentation interne 
  - ▶ si bloc petit, le nombre d'entrées dans le bloc d'index peut être insuffisant ⇒ index à multiniveaux :  
le 1er bloc d'index contient des adresses de blocs d'index, les blocs d'index de second niveau contiennent les adresses de blocs de données.  
Si extension à 3 ou 4 niveaux, perte de performances 

# Allocation indexée

## Exemple





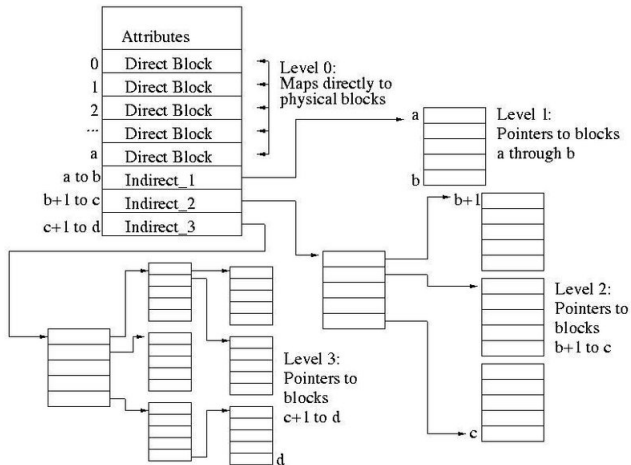
## Mise en œuvre de la $i$ -liste dans Unix

- 13 index
  - ▶ les 10 premières entrées contiennent les adresses des 10 premiers blocs de données du fichier,
  - ▶ la 11<sup>ème</sup> entrée pointe sur un bloc d'index qui contient les adresses des  $p$  blocs de données suivants du fichier

$$p = \frac{\text{taille bloc en octets}}{\text{taille en octets d'une adresse de bloc}}$$

- ▶ la 12<sup>ème</sup> entrée engendre un niveau d'indirection supplémentaire. Elle pointe sur un bloc d'index qui contient les adresses de  $p$  blocs d'index, dont les entrées pointent sur les  $p^2$  blocs de données du fichier
- ▶ la 13<sup>ème</sup> entrée ajoute encore un niveau d'index supplémentaire.

## Mise en œuvre dans Unix



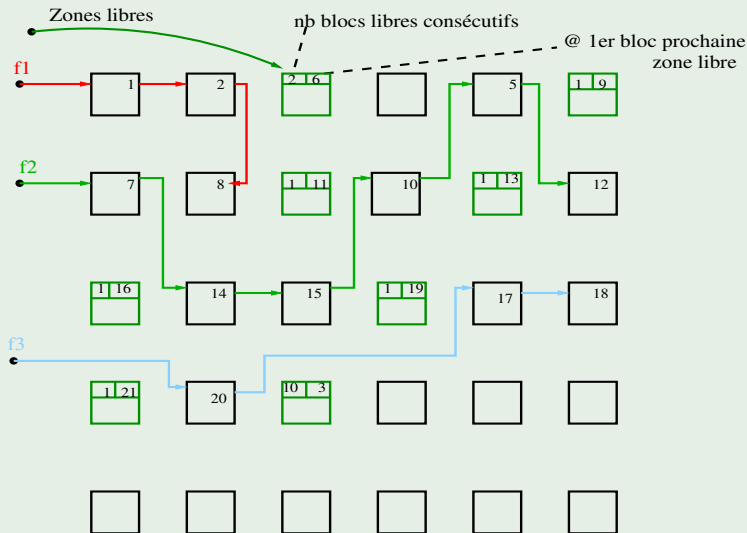
## Gestion de l'espace libre

Le SE maintient une liste des blocs disque libres. 3 représentations possibles de l'espace libre :

- ① gestion de l'espace libre par un vecteur de bits.  
L'espace libre sur le disque est représenté par un vecteur binaire dans lequel chaque bloc est figuré par un bit
  - ▶ longueur de la chaîne binaire = nb de blocs existants sur le disque
  - ▶ bit=0  $\iff$  bloc libre
- ② gestion de l'espace libre par liste chaînée.  
L'espace libre sur le disque est représenté par une liste chaînée de l'ensemble des blocs libres du disque
  - ▶ la recherche sur disque de  $n$  blocs consécutifs peut nécessiter le parcours d'une grande partie de la liste chaînée !

- une variante : chaque 1<sup>er</sup> bloc d'une zone libre indique
  - ▶ le nombre de blocs libres qui constitue la zone,
  - ▶ et l'adresse du 1<sup>er</sup> bloc de la zone libre suivante.

# Gestion de l'espace libre par liste chaînée




## Perte de données

Causes courantes :

- externes : feu, inondations, tremblements de terre...
- internes : fonctionnement défectueux du processeur, disque et bandes illisibles, bogues dans les programmes (le bug de l'an 2000 !)...
- erreurs humaines (les plus nombreuses !) : saisie de données erronées, utilisation d'un mauvais disque (ou bande), mauvaise exécution d'un programme, perte d'un CD...

**Solution universelle : la sauvegarde.**

- Périodiquement, le contenu de la mémoire secondaire est sauvegardé, généralement sur des bandes magnétiques, dans des endroits différents  $\rightsquigarrow$  la durée de sauvegarde peut être élevée 
- Technique de sauvegarde incrémentale : seuls sont sauvegardés les fichiers qui ont été modifiés depuis la sauvegarde précédente.
  - ▶ on ajoute un bit à chaque entrée de répertoires, initialisé à 0, positionné à 1 si modification.
- Généralement, la sauvegarde est associée à un compactage de la mémoire secondaire.

Linux, `rsync` permet la synchronisation à distance entre une machine locale et un autre hôte (distant).

## Contrôle d'accès aux fichiers

- la réalisation de la fonction d'accès est liée à la présentation d'un ou de plusieurs mots de passe ou,
- le contrôle d'accès concerne l'id de l'utilisateur qui veut accéder au fichier :
  - ▶ le créateur du fichier spécifie une liste d'utilisateurs, avec pour chacun une liste de modalités d'accès (lire, écrire, exécuter, etc...).
  - ▶ Les utilisateurs sont regroupés par catégories.  
Ex. dans Unix, 3 catégories : le créateur (propriétaire), le groupe et tous les autres utilisateurs.  
À chaque catégorie, sont associés 3 bits *r* (read), *w* (write), *x* (execute).  
111101000 indique ?