Cours 3: Systèmes de fichiers

Christophe Morvan

Université de Marne-la-Vallée

22 septembre 2015

Systèmes de fichiers ?

Observation

Les systèmes informatiques (ordinateurs/autres) ont besoin de données persistantes :

- au delà de la vie d'un processus
- en dehors du système
- au delà de la vie du système

Systèmes de fichiers ?

Observation

Les systèmes informatiques (ordinateurs/autres) ont besoin de données persistantes :

- au delà de la vie d'un processus
- en dehors du système
- au delà de la vie du système

Les disques physiques

Les disques (disques dur, disquettes, mémoires flash, ...) offrent une solution de persistance.

Systèmes de fichiers ?

Observation

Les systèmes informatiques (ordinateurs/autres) ont besoin de données persistantes :

- au delà de la vie d'un processus
- en dehors du système
- au delà de la vie du système

Les disques physiques

Les disques (disques dur, disquettes, mémoires flash, ...) offrent une solution de persistance.

Problèmes

- Très grandes capacités
- Moins grandes vitesses
- Plus grande exigeance de fiabilité

Performances

Comparaison

	Disque	MLC NAND Flash	DRAM
Écriture min.	secteur	secteur	octet
Écriture atomique	secteur	secteur	octet/mot
Accès aléatoire (l)	8 ms	75 μs	50 ns
Accès aléatoire (e)	8 ms	300 μs*	50 ns
Lecture séquentielle	100 Mo/s	400 Mo/s	$>$ 1 Go/s
Écriture séquentielle	100 Mo/s	250 Mo/s*	$>$ 1 Go/s
Coût (€ /Go)	0,04	0,6	8
Persistance	Non-volatile	Non-volatile	Volatile

* : les performances de la mémoire flash se dégradent avec le temps.

Plan

Progression

Les disques

Les secteurs

Sur le disque, physiquement les données sont stockés dans des secteurs.
C'est la plus petite donné qu'il est possible d'accéder ou d'adresser.
Aujourd'hui, 512 octets en général.

Les disques

Les secteurs

Sur le disque, physiquement les données sont stockés dans des secteurs. C'est la plus petite donné qu'il est possible d'accéder ou d'adresser. Aujourd'hui, 512 octets en général.

Écrire un bit ?

Lorsqu'on souhaite écrire 1 seul bit (ou toute donnée de taille inférieur à celle d'un secteur)

1) Lecture du secteur
2) Modification du bit concerné
3) Écriture du secteur

Les disques

Les secteurs

Sur le disque, physiquement les données sont stockés dans des secteurs. C'est la plus petite donné qu'il est possible d'accéder ou d'adresser. Aujourd'hui, 512 octets en général.

Écrire un bit ?

Lorsqu'on souhaite écrire 1 seul bit (ou toute donnée de taille inférieur à celle d'un secteur)

1) Lecture du secteur
2) Modification du bit concerné
3) Écriture du secteur

L'écriture d'un secteur est atomique : soit elle est faite totalement soit pas du tout. Même en cas de crash.

Les systèmes de fichiers

Les secteurs ne sont pas ergonomiques

Les OS proposent une abstraction des secteurs des disques sous la forme d'un système de fichiers.

Les systèmes de fichiers

Les secteurs ne sont pas ergonomiques

Les OS proposent une abstraction des secteurs des disques sous la forme d'un système de fichiers.

Système de fichiers : contrat

- Maintenir les données sur les périphériques de stockage (disques, disquettes, clefs USB,...)
- Structure le disque (fichiers, dossiers,...)
- Fournit une interface conviviale

Les systèmes de fichiers

Les secteurs ne sont pas ergonomiques

Les OS proposent une abstraction des secteurs des disques sous la forme d'un système de fichiers.

Système de fichiers : contrat

- Maintenir les données sur les périphériques de stockage (disques, disquettes, clefs USB,...)
- Structure le disque (fichiers, dossiers,...)
- Fournit une interface conviviale

Objectifs

- Optimiser l'utilisation de l'espace disponible
- Minimiser les temps d'accès
- Sécurité

Quelques systèmes de fichiers

Windows

- Fat 32
- ntfs
- Joliet

Unix/Linux

- ext2, ext3, ext4
- ReiserFS
- jfs, xfs
- iso 9660

MacOS

- HFS, HFS+

Les types de fichiers

Type généraux

- Fichiers ordinaires
- Répertoires
- Fichiers spéciaux de caractères (E/S)
- Fichiers spéciaux de bloc (disques)

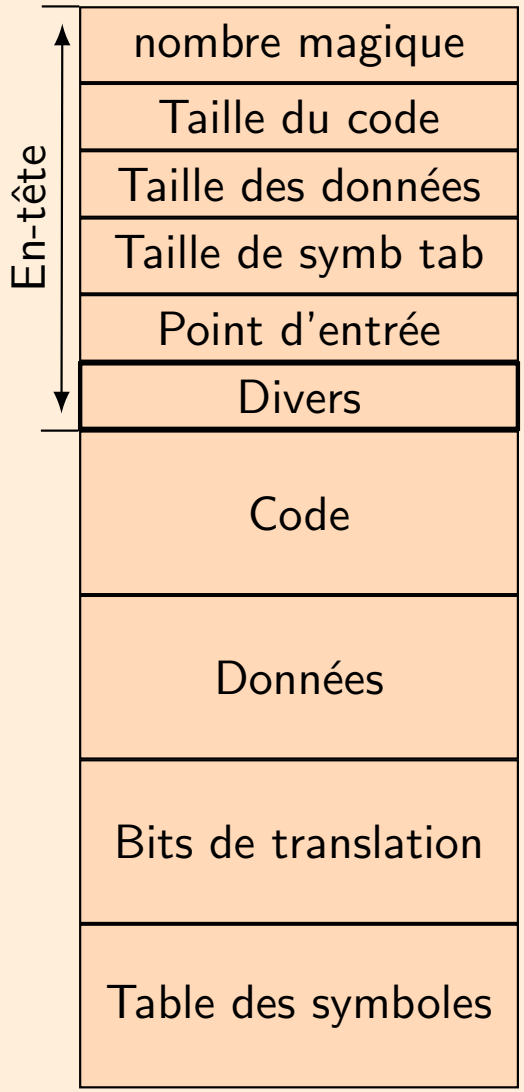
Les types de fichiers

Type généraux

- Fichiers ordinaires
- Répertoires
- Fichiers spéciaux de caractères (E/S)
- Fichiers spéciaux de bloc (disques)

Exemple

Un fichier exécutable Unix possède essentiellement la structure représenté sur la droite



Les noms et attributs de fichiers

Les caractéristiques des noms et attributs ont une grande variabilité suivant les OS

Noms

- Minimum 8 caractères ASCII
- Jusqu'à 255 caractères unicode
- (In)Sensible à la casse
- Utilisation d'une *extension* (symboles après un .)
- En Unix le nom est une convention

Les noms et attributs de fichiers

Les caractéristiques des noms et attributs ont une grande variabilité suivant les OS

Noms

- Minimum 8 caractères ASCII
- Jusqu'à 255 caractères unicode
- (In)Sensible à la casse
- Utilisation d'une *extension* (symboles après un .)
- En Unix le nom est une convention

Attributs

- Créateur, propriétaire
- Droits : lecture, écriture, exécution
- Caché
- Dates : création, accès, modification
- Taille
- Système

Opérations sur les fichiers

On verra en troisième partie les appels systèmes correspondant.

Quelques opérations

- Création/Destruction
- Ouverture/Fermeture
- Accès à une certaine position
- Lecture/Ecriture
- Connaître/changer les attributs
- Changement de nom

Opérations sur les fichiers

On verra en troisième partie les appels systèmes correspondant.

Quelques opérations

- Création/Destruction
- Ouverture/Fermeture
- Accès à une certaine position
- Lecture/Ecriture
- Connaître/changer les attributs
- Changement de nom

Voir cours 1 : `open`, `read`, `write`, `close`

Les dossiers : Historique

Étape 1 : un dossier unique pour le système

- Le dossier est a un point déterminé du disque
- Le dossier contient un index des fichiers
- Un nom de fichier = un fichier unique
- Utilisé dans beaucoup de systèmes antiques

Les dossiers : Historique

Étape 1 : un dossier unique pour le système

- Le dossier est a un point déterminé du disque
- Le dossier contient un index des fichiers
- Un nom de fichier = un fichier unique
- Utilisé dans beaucoup de systèmes antiques

Étape 2 : un dossier par utilisateur

- Finalement assez proche de l'étape précédente

Les dossiers : Historique

Étape 1 : un dossier unique pour le système

- Le dossier est a un point déterminé du disque
- Le dossier contient un index des fichiers
- Un nom de fichier = un fichier unique
- Utilisé dans beaucoup de systèmes antiques

Étape 2 : un dossier par utilisateur

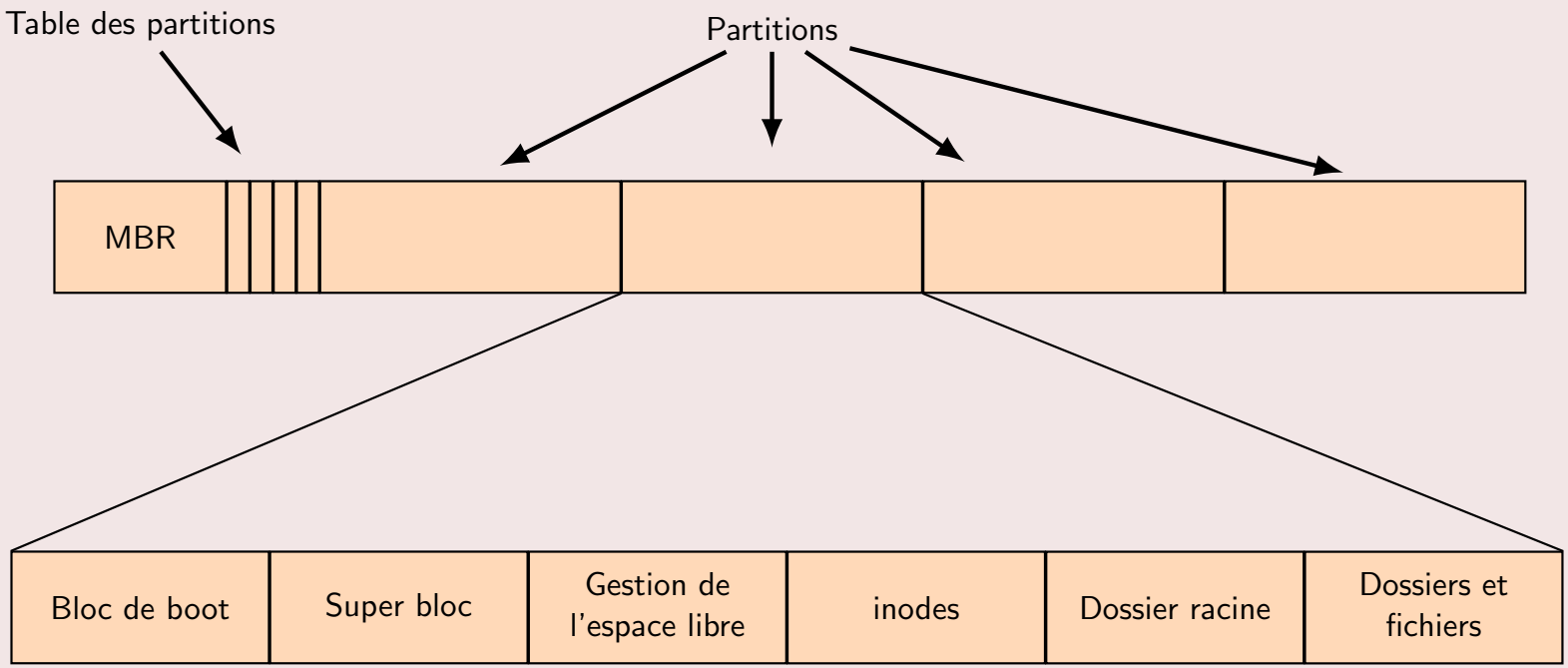
- Finalement assez proche de l'étape précédente

Étape 3 : organisation hiérarchique

- Les dossiers peuvent contenir d'autres dossiers
- La structure est un arbre (éventuellement un DAG)
- Valable en dehors des systèmes de fichiers (portées, dns,...)

Structure d'un système de fichiers

Schéma général (vision Unix)



Abstraction des secteurs : blocs

Les systèmes de fichiers n'utilisent pas les secteurs (ou les pistes) des disques directement. Les données sont groupés en blocs
Fondamentalement : compromis espace/temps
Plus les blocs sont petits plus on optimise l'espace disponible ; plus ils sont grand plus on gagne du temps en lecture

Abstraction des secteurs : blocs

Les systèmes de fichiers n'utilisent pas les secteurs (ou les pistes) des disques directement. Les données sont groupés en blocs
Fondamentalement : compromis espace/temps
Plus les blocs sont petits plus on optimise l'espace disponible ; plus ils sont grand plus on gagne du temps en lecture

Groupes de blocs

Pour des questions d'optimisation de déplacements de la tête de lecture les blocs sont souvent organisés en groupes

Abstraction des secteurs : blocs

Les systèmes de fichiers n'utilisent pas les secteurs (ou les pistes) des disques directement. Les données sont groupés en blocs
Fondamentalement : compromis espace/temps
Plus les blocs sont petits plus on optimise l'espace disponible ; plus ils sont grand plus on gagne du temps en lecture

Groupes de blocs

Pour des questions d'optimisation de déplacements de la tête de lecture les blocs sont souvent organisés en groupes

Gestion des blocs libres

Il est nécessaire d'effectuer une mémorisation des blocs disponibles (Sera vu plus loin)

Mise en œuvre

Pour définir un système de fichiers il faut déterminer un processus d'allocation des secteurs disque et une structure de donnée pour conserver ces informations

Problèmes

- La taille des fichiers évolue au fil du temps (imprévisible)
- Mémoriser les informations initiales et modifications

Mise en œuvre

Pour définir un système de fichiers il faut déterminer un processus d'allocation des secteurs disque et une structure de donnée pour conserver ces informations

Problèmes

- La taille des fichiers évolue au fil du temps (imprévisible)
- Mémoriser les informations initiales et modifications

Plusieurs approches d'allocation

- contiguë
- par liste chaînée
- par liste chaînée avec table en mémoire
- les inodes (ou nœud d'index)

Mise en œuvre Contiguë

Exemple

Considérons un disque possédant des secteurs de 512 o, un fichier de 50 Ko est placé dans 100 secteurs consécutifs

Mise en œuvre Contiguë

Exemple

Considérons un disque possédant des secteurs de 512 o, un fichier de 50 Ko est placé dans 100 secteurs consécutifs

Principe

Le système d'allocation, et la structure de donnée est simple :

- Trouver le premier espace libre de taille suffisante
- Mémoriser le nom et les attributs du fichier dans une table avec la taille du fichier et son secteur initial.

Mise en œuvre Contiguë

Exemple

Considérons un disque possédant des secteurs de 512 o, un fichier de 50 Ko est placé dans 100 secteurs consécutifs

Principe

Le système d'allocation, et la structure de donnée est simple :

- Trouver le premier espace libre de taille suffisante
- Mémoriser le nom et les attributs du fichier dans une table avec la taille du fichier et son secteur initial.

Avantages

- Rapide
- Simple
- Supports écriture seulement

Mise en œuvre Contiguë

Exemple

Considérons un disque possédant des secteurs de 512 o, un fichier de 50 Ko est placé dans 100 secteurs consécutifs

Principe

Le système d'allocation, et la structure de donnée est simple :

- Trouver le premier espace libre de taille suffisante
- Mémoriser le nom et les attributs du fichier dans une table avec la taille du fichier et son secteur initial.

Avantages

- Rapide
- Simple
- Supports écriture seulement

Inconvénients

- Fragmentation
- Anticiper la taille max d'un fichier

Mise en œuvre par liste chaînée

Principe

Dans chaque secteur le premier *mot* contient l'adresse du secteur suivant.

1) Identifier un secteur libre
2) Mémoriser le nom et les attributs du fichier dans une table avec son secteur initial
3) Trouver un secteur libre (`prochain`)
4) Stocker l'adresse de `prochain`, puis le début des données dans ce secteur
5) S'il reste des données retourner en 3)

Mise en œuvre par liste chaînée

Principe

Dans chaque secteur le premier *mot* contient l'adresse du secteur suivant.

1) Identifier un secteur libre
2) Mémoriser le nom et les attributs du fichier dans une table avec son secteur initial
3) Trouver un secteur libre (`prochain`)
4) Stocker l'adresse de `prochain`, puis le début des données dans ce secteur
5) S'il reste des données retourner en 3)

Avantages

- Pas de perte d'espace
- Accès séquentiel

Mise en œuvre par liste chaînée

Principe

Dans chaque secteur le premier *mot* contient l'adresse du secteur suivant.

1) Identifier un secteur libre
2) Mémoriser le nom et les attributs du fichier dans une table avec son secteur initial
3) Trouver un secteur libre (`prochain`)
4) Stocker l'adresse de `prochain`, puis le début des données dans ce secteur
5) S'il reste des données retourner en 3)

Avantages

- Pas de perte d'espace
- Accès séquentiel

Inconvénients

- Accès aléatoire
- $|\text{bloc}| < |\text{secteur}|$

Liste chaînée et table en mémoire

Pour résoudre les problèmes posés par le stockage des informations de secteur suivant dans chaque secteur il est possible de stocker la liste chaînée des secteurs en mémoire

Principe

- liste (tableau) des secteurs en mémoire
- numéro du premier secteur
- idem cas précédent avec la liste en mémoire

Liste chaînée et table en mémoire

Pour résoudre les problèmes posés par le stockage des informations de secteur suivant dans chaque secteur il est possible de stocker la liste chaînée des secteurs en mémoire

Principe

- liste (tableau) des secteurs en mémoire
- numéro du premier secteur
- idem cas précédent avec la liste en mémoire

Avantages

Accès aléatoire meilleur

$|\text{bloc}| = |\text{secteur}|$

Liste chaînée et table en mémoire

Pour résoudre les problèmes posés par le stockage des informations de secteur suivant dans chaque secteur il est possible de stocker la liste chaînée des secteurs en mémoire

Principe

- liste (tableau) des secteurs en mémoire
- numéro du premier secteur
- idem cas précédent avec la liste en mémoire

Avantages

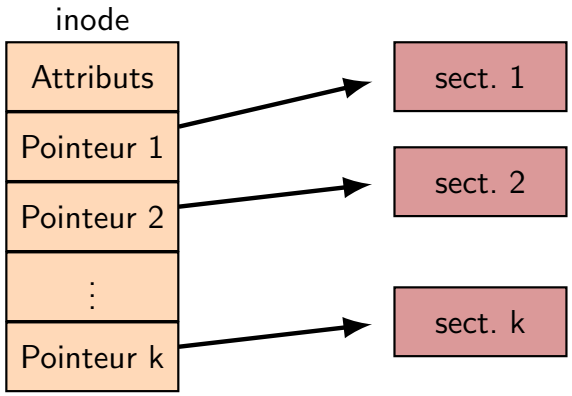
Accès aléatoire meilleur
$|\text{bloc}| = |\text{secteur}|$

Inconvénients

Taille de la liste prop. à la taille du disque
Exemple : disque de 1 To, secteurs de 512 o
$\rightarrow$ 2 milliards d'entrées $\rightarrow$ minimum 4 octets par entrée $\rightarrow$ table de 2 Go

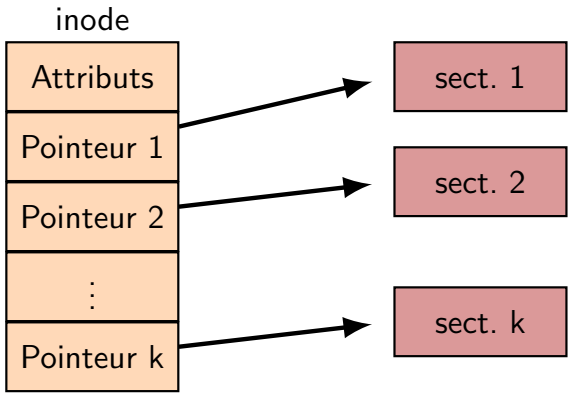
Mise en œuvre par inode

Pour chaque fichier le tronçon de la table des secteurs est stocké avec son nom, et ses attributs.



Mise en œuvre par inode

Pour chaque fichier le tronçon de la table des secteurs est stocké avec son nom, et ses attributs.

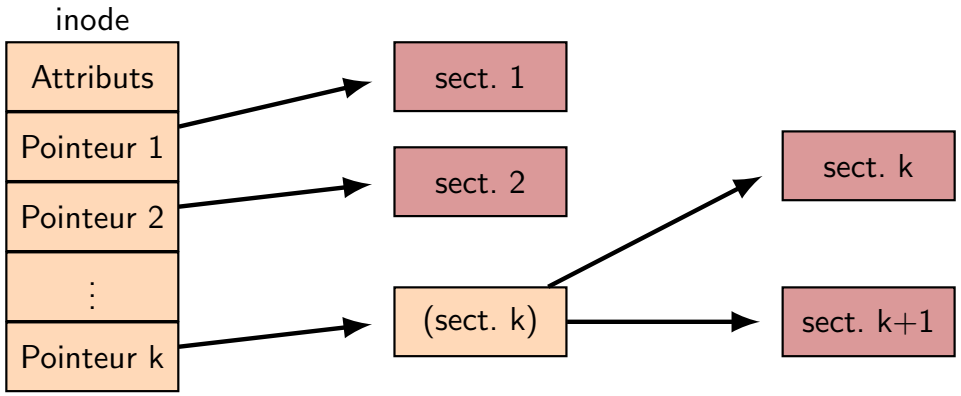


Problème

La taille de chaque enregistrement d'attribut est limité
Comment gérer des fichiers de taille arbitraire ?

Mise en œuvre par inode

Pour chaque fichier le tronçon de la table des secteurs est stocké avec son nom, et ses attributs.



Problème

La taille de chaque enregistrement d'attribut est limité
Comment gérer des fichiers de taille arbitraire ?

le dernier enregistrement contient l'adresse d'un secteur contenant des adresses supplémentaires (un peu comme un inode)

Ext 2

Ext 2

ext2 est le nom du système de fichiers *historique* sous Linux
Il a donné naissance à ext3 et ext4

Inodes – Attributs (méta-données)

- Permissions, propriétaire, groupe
- Taille, nombre de blocs, nombre de liens
- Dates : accès, modification, destruction
- Autres (ACL, réservés, ...)

Ext2 – 2

Inodes – données

Quinze pointeurs :

- Les douze premiers pointent vers des blocs de données
- Le treizième définit un niveau de redirection
- Le quatorzième un double niveau de redirection
- Le quinzième un triple niveau de redirection

Ext2 – 2

Inodes – données

Quinze pointeurs :

- Les douze premiers pointent vers des blocs de données
- Le treizième définit un niveau de redirection
- Le quatorzième un double niveau de redirection
- Le quinzième un triple niveau de redirection

Tailles maximales selon celle du bloc

	Taille			
Bloc	1 Ko	2 Ko	4 Ko	8 Ko*
Fichier	16 Go	256 Go	2 To	2 To
Sys. Fichiers**	4 To	8 To	16 To	32 To

(∗) : uniquement systèmes spécifiques

(∗∗) : Limite de $2 To$ sur les noyaux $< 2.6.17$

Mise en œuvre des répertoires

Les répertoires permettent de structurer les informations des fichiers

Principe général

- Liste des fichiers
- Associer des droits à chaque fichier
- Noms : deux option
  - taille (du nom) déterminée au départ, nom enregistré après les attributs
  - nom placé dans un tas, dans une zone dédiée. Simple pointeur dans l'enregistrement

Mise en œuvre des répertoires

Les répertoires permettent de structurer les informations des fichiers

Principe général

- Liste des fichiers
- Associer des droits à chaque fichier
- Noms : deux option
  - taille (du nom) déterminée au départ, nom enregistré après les attributs
  - nom placé dans un tas, dans une zone dédiée. Simple pointeur dans l'enregistrement

Approche Unix

- Les dossiers sont des fichiers ordinaires
- Les attributs des fichiers sont stockés dans les inodes

Mise en œuvre des liens *durs* (Unix)

Un lien permet à plusieurs chemins vers un fichier

Mise en œuvre des liens *durs* (Unix)

Un lien permet à plusieurs chemins vers un fichier

Principes

- Plusieurs entrées de dossiers pointent sur le même inode
- La modification d'un inode met à jour les informations dans tous les liens (car le inode est partagé).
- L'opération d'effacement (`rm`) ne supprime qu'un lien. Les données ne sont supprimées que lors de la suppression du dernier lien
- Lors de la suppression, il y a récupération du inode et mise à jour de la structure de l'espace libre

Mise en œuvre des liens *durs* (Unix)

Un lien permet à plusieurs chemins vers un fichier

Principes

- Plusieurs entrées de dossiers pointent sur le même inode
- La modification d'un inode met à jour les informations dans tous les liens (car le inode est partagé).
- L'opération d'effacement (`rm`) ne supprime qu'un lien. Les données ne sont supprimées que lors de la suppression du dernier lien
- Lors de la suppression, il y a récupération du inode et mise à jour de la structure de l'espace libre

Comment assurer la récupération de l'espace disque lors de la suppression du dernier lien ?

Mise en œuvre des liens *durs* (Unix)

Un lien permet à plusieurs chemins vers un fichier

Principes

- Plusieurs entrées de dossiers pointent sur le même inode
- La modification d'un inode met à jour les informations dans tous les liens (car le inode est partagé).
- L'opération d'effacement (`rm`) ne supprime qu'un lien. Les données ne sont supprimées que lors de la suppression du dernier lien
- Lors de la suppression, il y a récupération du inode et mise à jour de la structure de l'espace libre

Comment assurer la récupération de l'espace disque lors de la suppression du dernier lien ?
Les fichiers possèdent un attribut *nombre de liens* qui est modifié à chaque création/destruction de lien

Mise en œuvre dans ext2++

Quelques inodes utilisés

Num	typ	droit	lnk	blk1	···
58	d	0755	1	45	
77	-	0666	2	6	
145	d	0755	1	133	
221	-	0600	3	16	

Bloc 6
Premières données du fichier ...

Bloc 16
Premières données de l'autre fichier ...

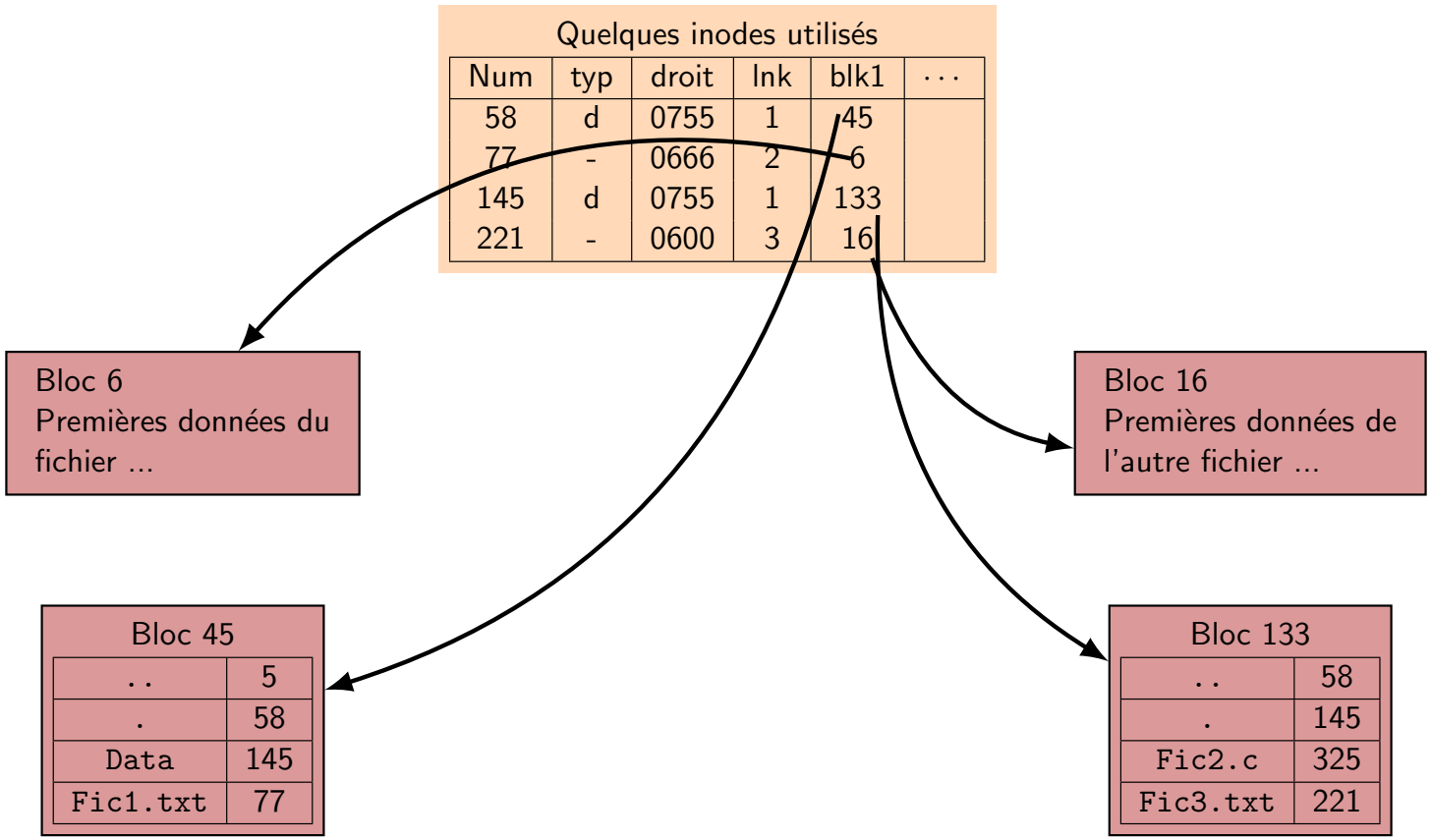
Bloc 45

..	5
.	58
Data	145
Fic1.txt	77

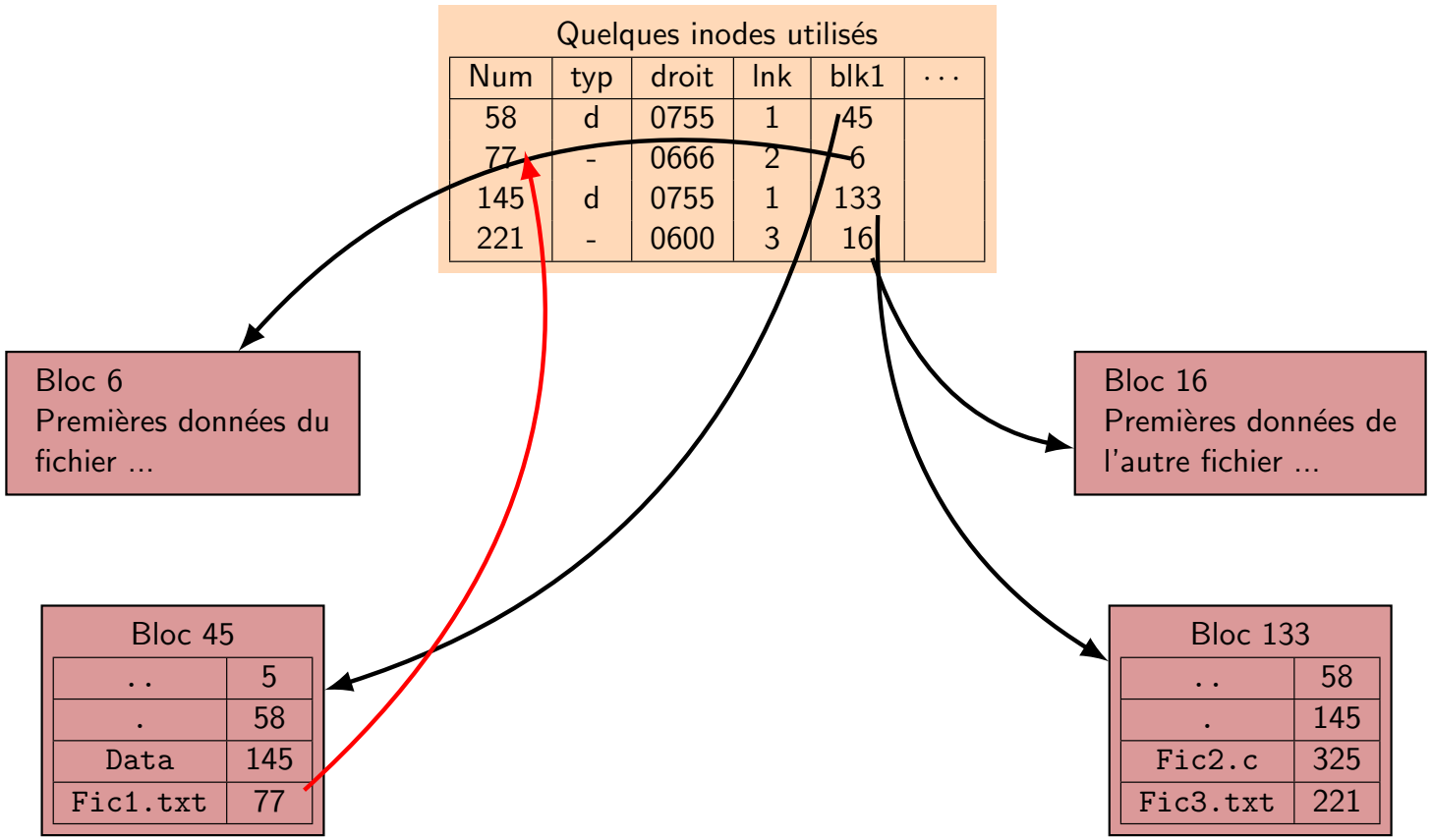
Bloc 133

..	58
.	145
Fic2.c	325
Fic3.txt	221

Mise en œuvre dans ext2++



Mise en œuvre dans ext2++



Systèmes de fichiers journalisés

Les dégâts occasionnés au système de fichiers lors d'un plantage d'un système peuvent être considérables

Systèmes de fichiers journalisés

Les dégâts occasionnés au système de fichiers lors d'un plantage d'un système peuvent être considérables

Exemple : suppression d'un fichier

1) Suppression dans le dossier
2) Libération de l'inode, placement dans le groupe des inodes libres
3) Libération des secteurs, les placer dans les secteurs libres

Problème : plantage entre 1) et 2), ou entre 2) et 3)

Systèmes de fichiers journalisés

Les dégâts occasionnés au système de fichiers lors d'un plantage d'un système peuvent être considérables

Exemple : suppression d'un fichier

1) Suppression dans le dossier
2) Libération de l'inode, placement dans le groupe des inodes libres
3) Libération des secteurs, les placer dans les secteurs libres

Problème : plantage entre 1) et 2), ou entre 2) et 3)

Solution

Le principe de la journalisation consiste à ajouter un espace de journal contient les informations des opérations à effectuer

Systèmes de fichiers journalisés

Les dégâts occasionnés au système de fichiers lors d'un plantage d'un système peuvent être considérables

Exemple : suppression d'un fichier

1) Suppression dans le dossier
2) Libération de l'inode, placement dans le groupe des inodes libres
3) Libération des secteurs, les placer dans les secteurs libres

Problème : plantage entre 1) et 2), ou entre 2) et 3)

Solution

Le principe de la journalisation consiste à ajouter un espace de journal contient les informations des opérations à effectuer
Attention : les actions de journalisation doivent être *idempotente*, i.e., qu'on puisse les répéter sans danger

Gérer l'espace libre

Gestion des blocs libres (deux options)

- Liste chaînée des blocs libres
- Table de bits (chaque bloc est représenté par 1 bit)

Gérer l'espace libre

Gestion des blocs libres (deux options)

- Liste chaînée des blocs libres
- Table de bits (chaque bloc est représenté par 1 bit)

La plupart des systèmes de fichiers Unix emploie la première option
Ils effectuent également une gestion des inodes libres similaire

Gérer l'espace libre

Gestion des blocs libres (deux options)

- Liste chaînée des blocs libres
- Table de bits (chaque bloc est représenté par 1 bit)

La plupart des systèmes de fichiers Unix emploie la première option
Ils effectuent également une gestion des inodes libres similaire

Optimisations (liste chaînée)

- Gestion dans l'espace libre lui-même
- Chaque maillon longueur + espace libre suivant
- k Listes de tailles déterminées

Gestion du cache

Les performances disques étant moins bonnes que celles de la mémoire, il est essentiel de mettre ne place un système de cache pour les fichiers disque.

Gestion du cache

Les performances disques étant moins bonnes que celles de la mémoire, il est essentiel de mettre ne place un système de cache pour les fichiers disque.

Principe général

Les blocs présents dans le cache sont accédés à la place des blocs sur le disque (en lecture ou en écriture)
Le système maintient une table de hashage des blocs présents dans le cache

Gestion du cache

Les performances disques étant moins bonnes que celles de la mémoire, il est essentiel de mettre ne place un système de cache pour les fichiers disque.

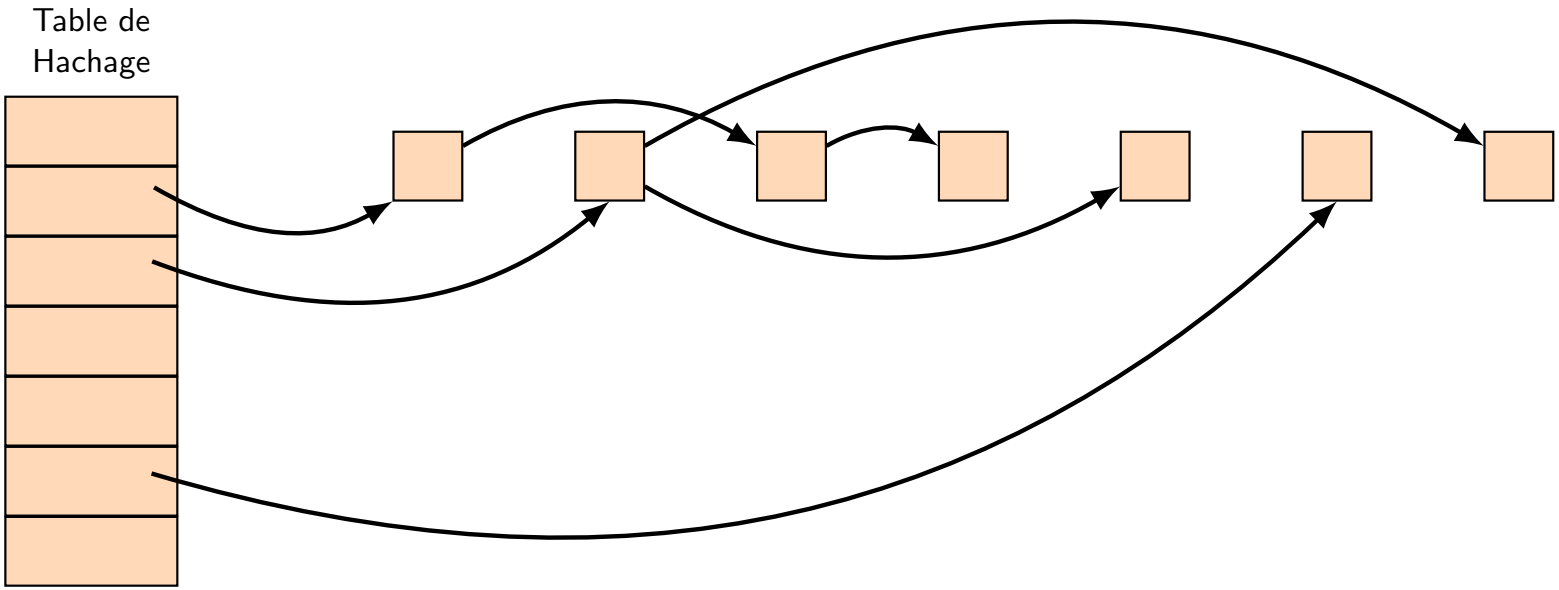
Principe général

Les blocs présents dans le cache sont accédés à la place des blocs sur le disque (en lecture ou en écriture)
Le système maintient une table de hashage des blocs présents dans le cache

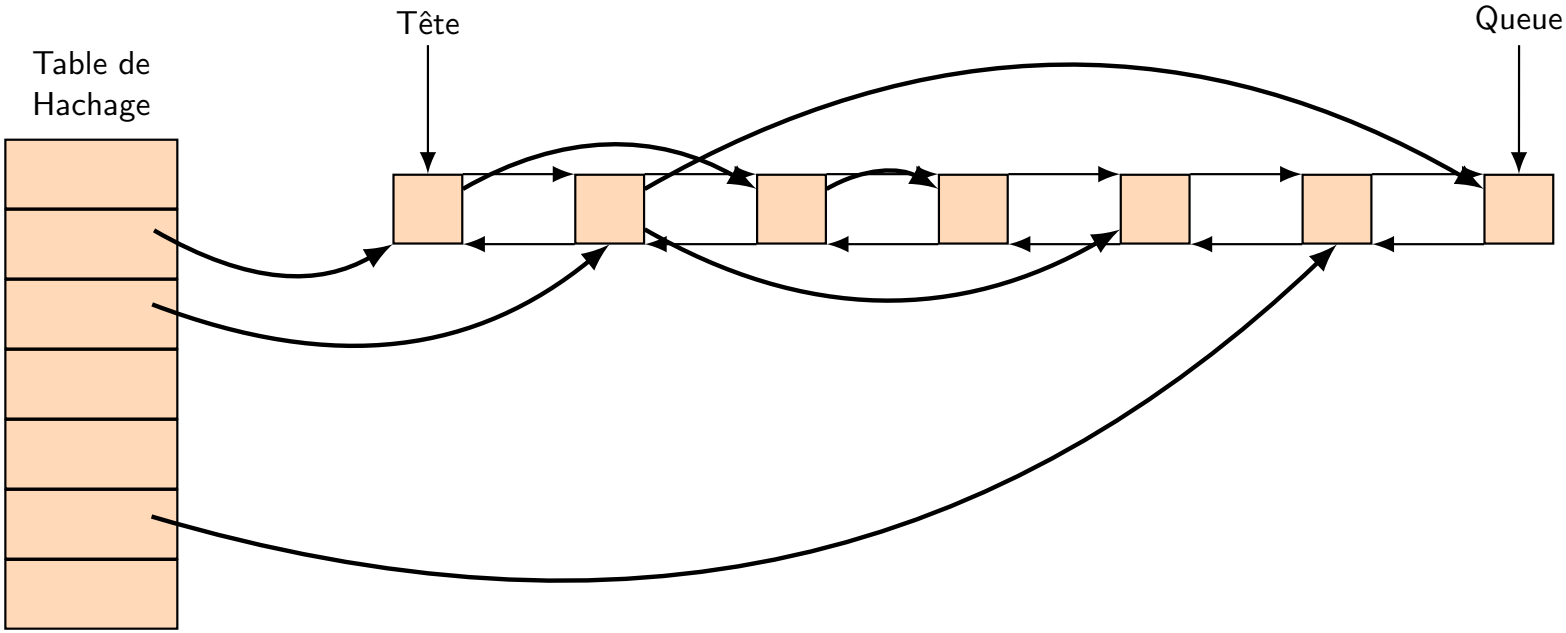
Problèmes

Il est nécessaire de déterminer quand procéder à l'écriture effective des données
De déterminer quelle page supprimer du cache lorsqu'une nouvelle page doit être chargée

Buffer Cache Unix



Buffer Cache Unix



Liste de priorité

Les pages sont également stockées dans une liste de priorités pour déterminer qui sort du cache en premier

Les pages modifiées sont écrites sur disque à la sortie du cache

Sortie du cache

Problème (Inodes)

Si des inodes sont dans le cache à la survenue d'un plantage le système de fichiers peut être fortement corrompu.
Les inodes restent dans le cache mais sont réécrit dès qu'ils sont modifiés

Sortie du cache

Problème (Inodes)

Si des inodes sont dans le cache à la survenue d'un plantage le système de fichiers peut être fortement corrompu.
Les inodes restent dans le cache mais sont réécrit dès qu'ils sont modifiés

Problème (Hyper utilisation)

Si un page est utilisée "constament" par l'utilisateur du système elle n'est jamais écrite sur le disque
Un mécanisme doit assurer l'écriture des pages modifiées sur le disque, à interval régulier (Linux appelle `sync` toutes les 30 secondes)

Progression

Fondamentaux

Principes

Donner accès aux répertoires et aux fichiers aux utilisateurs du systèmes, et aux programmeurs

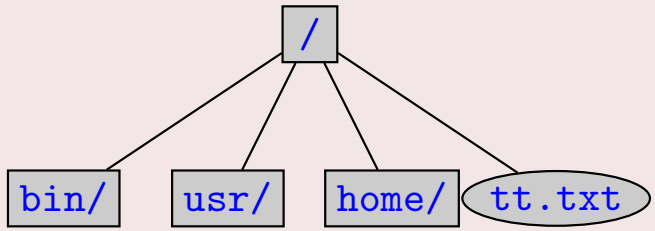
Rappels (Unix)

- Montage
- Organisation
- Liens
- Bibliothèque standard C

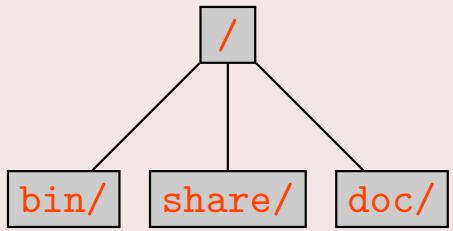
Montage

Opération de montage

Disque 1

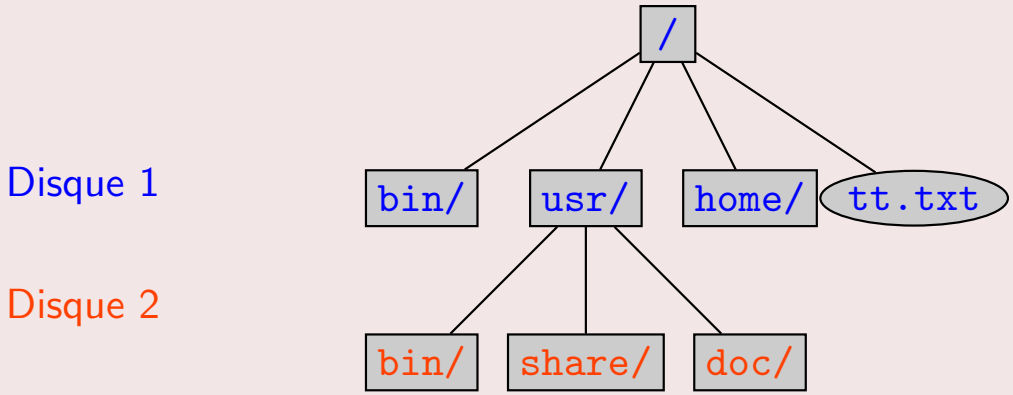


Disque 2



Montage

Opération de montage



Shell

Instruction : `mount -t ext3 /dev/sda4 /mnt/extraData`

Réciproque : `umount /mnt/extraData`

Il y a aussi un appel système

Arborescence Unix

`/` : La racine

`/bin`, `/sbin` : exécutables

`/lib` : bibliothèques

`/etc` : Fichiers de config.

`/boot` : Fichiers de démarrage

`/var` : Données système

`/dev` : Périphériques

`/proc` : Les processus

`/sys` : informations générales

`/mnt`, `/media` : Disques amovibles

`/home` : Dossiers utilisateurs

`/usr` : Les applications

`/usr/bin` : Les exécutables

`/usr/lib` : Les bibliothèques

`/usr/share` : Données (applis)

`/usr/X11R6` : Serveur X (Xorg)

Liens

Liens durs (*hard links*)

Liens vu précédemment
Indistinguable du fichier original
Interne à un unique système de fichiers
Ne peut pas porter sur un dossier

Liens

Liens durs (*hard links*)

Liens vu précédemment
Indistinguable du fichier original
Interne à un unique système de fichiers
Ne peut pas porter sur un dossier

Liens symboliques (*soft links*)

Fichiers spéciaux
Contiennent le chemin (relatif ou absolu) vers un autre fichier ou dossier
Tout a fait distinct du fichier pointé

Progression

Déjà vu – open, close, read, write, lseek

open/close

```
int open(char *path, int flags, /*mode*/...);
```

Fournit un descripteur (entier) sur un fichier

```
int close (int fd);
```

Détruit le descripteur, libère le fichier

Déjà vu – open, close, read, write, lseek

open/close

```
int open(char *path, int flags, /*mode*/...);
```

Fournit un descripteur (entier) sur un fichier

```
int close (int fd);
```

Détruit le descripteur, libère le fichier

read/write/seek

- `int read (int fd, void *buf, int nbytes);`
- `int write (int fd, void *buf, int nbytes);`
- `off_t lseek (int fd, off_t pos, int whence);`

Exemple : copie

Programme exemple

```
void copyfile (char *source, char *cible)
{
  int d_so, d_ci, nread;
  char buf[1024];
  d_so = open (source, O_RDONLY);
  d_ci = open (cible, O_WRONLY|O_CREAT,0644);
  if (d_so == -1||d_ci== -1){
      perror (cible);
      return; }
  while ((nread = read (d_so, buf, sizeof (buf))) > 0)
    write (d_ci, buf, nread);

  close (d_so);close(d_ci);
}
```

Informations : appel stat

stat : trois variantes — unistd.h sys/stat.h sys/types.h

- `int stat(const char *path, struct stat *buf);`
- `int lstat(const char *path, struct stat *buf);`
- `int fstat(int fd, struct stat *buf);`

La structure stat

```
struct stat {
  dev_t     st_dev;     /* ID of device containing file */
  ino_t     st_ino;     /* inode number */
  mode_t    st_mode;    /* protection */
  nlink_t   st_nlink;   /* number of hard links */
  uid_t     st_uid;     /* user ID of owner */
  gid_t     st_gid;     /* group ID of owner */
  dev_t     st_rdev;    /* device ID (if special file) */
  off_t     st_size;    /* total size, in bytes */
  blksize_t st_blksize; /* blocksize for file system I/O */
  blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
  time_t    st_atime, st_mtime,st_ctime;   /* time of acc./mod/chnge */
};
```

Droits : appel chmod, chown

chmod : deux variantes — sys/stat.h

- `int chmod(const char *path, mode_t mode);`
- `int fchmod(int fd, mode_t mode);`

Droits : appel chmod, chown

chmod : deux variantes — sys/stat.h

- `int chmod(const char *path, mode_t mode);`
- `int fchmod(int fd, mode_t mode);`

chown : trois variantes — unistd.h

- `int chown(const char *path, uid_t owner, gid_t group);`
- `int lchown(const char *path, uid_t owner, gid_t group);`
- `int fchown(int fd, uid_t owner, gid_t group);`

Droits : appel chmod, chown

chmod : deux variantes — sys/stat.h

- `int chmod(const char *path, mode_t mode);`
- `int fchmod(int fd, mode_t mode);`

chown : trois variantes — unistd.h

- `int chown(const char *path, uid_t owner, gid_t group);`
- `int lchown(const char *path, uid_t owner, gid_t group);`
- `int fchown(int fd, uid_t owner, gid_t group);`

chown vs lchown

`lchown` modifie un lien symbolique lui-même et pas le fichier pointé
Similaire pour `stat` et `lstat`