Systèmes d'exploitation

Les fichiers (suite)

Gérard Padiou

Département Informatique et Mathématiques appliquées
ENSEEIHT

7 octobre 2010

plan

Plan

Principes de conception

Stuart E. Madnick *dans* Design Strategies for File Systems, TR-78, MIT, 1970

Abstraction	Niveau	Fonction
Répertoires	6	Désignation (chemins via les répertoires)
	5	Localisation des descripteurs
Fichiers	4	Contrôle d'accès (listes ou capacités)
	3	Accès au contenu (lire, écrire)
Blocs	2	Allocation et caches
	1	Gestion des échanges

Principes de conception

Abstraction Répertoire

Rôle

- Désignation des fichiers dans une structure d'arbre ;
- Résolution des chemins d'accès relatifs ou absolus ;
- Passage d'un chemin d'accès à un nom interne (i-node) ;
- Suite de couples (nom symbolique, nom interne) ;
- Représentation : type de fichier.

Principes de conception
Abstraction Fichier

Rôles

- Contrôle d'accès lors de la connexion (open) ;
- Table de descripteurs de fichiers ;
- Accès aux informations du descripteur (i-node) ;
- Mise en œuvre d'une méthode d'accès ;
- Accès au contenu (read/write) ;
- Représentation : carte d'implantation.

Principes de conception
Abstraction Blocs (Clusters)

Rôles

- Bloc = n secteurs consécutifs (logiquement) : $n = 2, 4, 8$;
- Allocation dynamique à la demande ;
- Possibilité d'optimisation des échanges via des caches ;
- Problème : gestion de la cohérence entre copie en cache et copie sur disque ;
- Plusieurs stratégies de représentation des cartes.

Principes de conception

Les types de fichiers

Typage par leur contenu

- les fichiers ordinaires ;
- les fichiers répertoires ;
- les fichiers spéciaux : permettent de désigner et utiliser les ressources physiques comme des fichiers.

Typage par leur visibilité

- les fichiers « cachés » : n'apparaissent dans les fenêtres ou commandes ls ou dir ;

Plan

La protection des fichiers

Le contrôle d'accès aux fichiers

Objectif

Mécanismes permettant la mise en œuvre de politiques de sécurité

Qui, quoi, quand contrôler ?

- Côté utilisateur : un processus s'exécutant pour un usager ;
- Côté fichier : un usager propriétaire (créateur) du fichier ;
- Contrôle d'accès : l'usager pour lequel s'exécute le processus appelant a-t-il le droit d'exécuter la primitive d'accès appelée ?
- Moment du contrôle : phase de connexion (ouverture).

La protection des fichiers

Le contrôle d'accès aux fichiers

Modélisation générique

Approche matricielle : $Usager \times Objet \longrightarrow Droits$;

$$\begin{array}{cccc} & \ldots & f_k & \ldots \\ u_1 & & & \\ \ldots & & \downarrow & \\ u_i & \longrightarrow & \boxed{d_{i,k}} & \\ \ldots & & & \end{array}$$

- Usagers regroupés par classes ;
- Objets de différents types $\Rightarrow$ Types de droits différents.

La protection des fichiers

Le contrôle d'accès aux fichiers

Approche par ligne

- Chaque processus possède une liste des objets qu'il peut accéder ;

  ☞ Notion de capacité

- Problème de base : contrôler la distribution des capacités de façon à garantir un politique de sécurité ?

Approche par colonne

- Chaque objet possède une liste des usagers ayant un droit d'usage ;
- Même problème de base : contrôler la définition de ces listes de façon à garantir un politique de sécurité ?

La protection des fichiers
L'exemple Unix

- Approche par colonne : les droits d'accès sont codés dans le descripteur de fichier ;
- Usagers répartis en 3 classes : propriétaire, membre du groupe, autre ;
- Trois types d'accès : lire, écrire, exécuter.
- Codage des droits par 3×3 bits indicateurs.

Exemple

Fichier de données : r w - r - - r - -
Fichier binaire exécutable : r w x r - x r - -

La protection des fichiers

L'exemple Unix

Le contrôle d'accès lors de l'ouverture

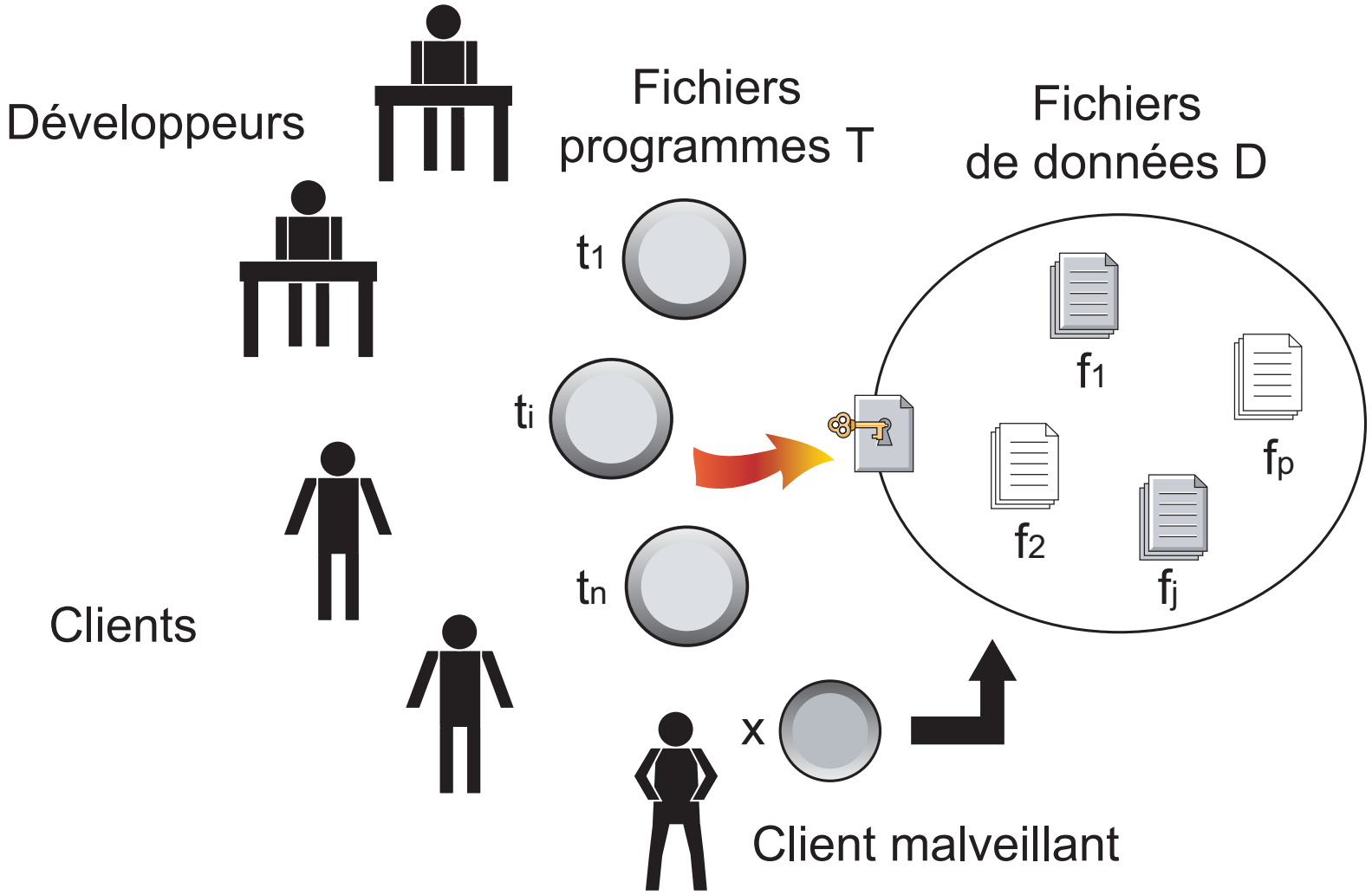
- Un processus « roule »pour un usager : (UID_p, GID_p) ;
- L'ouverture demande des droits parmi : $dem \in \{r, w, x\}$;
- Le fichier possède :
  - Une identification du propriétaire : (UID_f, GID_f) ;
  - un codage des droits d'accès : $df =$ rwxrwxrwx
- $\exists$ un *super-utilisateur* : $(UID_{super}, GID_{super})$

Algorithme de contrôle : accès autorisé ssi

- soit $UID_P = UID_{super}$
- soit $UID_p = UID_f \wedge dem \subset df$
- soit $UID_p \neq UID_f \wedge GID_f = GID_p \wedge dem \subseteq df$
- soit $UID_p \neq UID_f \wedge GID_f \neq GID_p \wedge dem \subseteq df$

La protection des fichiers

Partage de fichiers



La protection des fichiers

Partage de fichiers

Propriétés à garantir

(1) Comment protéger ces données contre les accès directs par des programmes applicatifs ?

(2) Comment autoriser que des programmes applicatifs spécifiques accèdent aux données ?

(3) Tout usager doit pouvoir utiliser les programmes applicatifs spécifiques.

Exemple

- Autoriser l'accès au groupe et autres :
  assure (2) et (3) mais viole (1)
- Autoriser l'accès aux usagers développeurs (à leur groupe) :
  assure (1) et (2) mais viole (3)

La protection des fichiers

Partage des fichiers

Principe

☞ Un processus change momentanément d'identité

- Un processus « roule » pour un usager de base (initial) ;
- Mais il peut en changer pour exécuter un programme ;
- Il prend l'identité du propriétaire du fichier programme ;
- $\Rightarrow$ notion d'usager effectif : (UID_{eff}, GID_{eff}).

Mécanisme

☞ 2 bits indicateurs supplémentaires :

- Un bit pour autoriser le changement d'usager ;
- Un bit pour autoriser le changement de groupe.

Extension : les ACL's (Access Control Lists)

Bonne documentation : http ://www.freebsd.org/doc/en/books/handbook/fs-acl.html

Objectif

- Mise en place de politiques de sécurité plus sophistiquées ;
- Possibilité de distinguer des droits distincts pour une même catégorie usagers ou groupes ;
- Suite d'Access Control Entries (ACE) ;
- Contenu d'une ACE = entrée : [uid/gid] : permissions ;
- Deux commandes :
  - Lecture : getfacl *fichier*...
  - Mise-à-jour : setfacl -x|-m|-s [ACL] *fichier*... ;
- ☞ Politique de sécurité : Seul le propriétaire ou l'usager root peuvent modifier l'ACL d'un fichier ;
- Mise en œuvre délicate : suite de longueur variable.

Extension : les ACL's (Access Control Lists)

Types d'entrées

Permissions classiques en ACL

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

Permissions étendus en ACL

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

☞ Rôle du masque : limiter les permissions accordées aux usagers et groupes nommés.

Extension : les ACL's (Access Control Lists)

Types d'entrées (suite)

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*.

Extension : les ACL's (Access Control Lists)

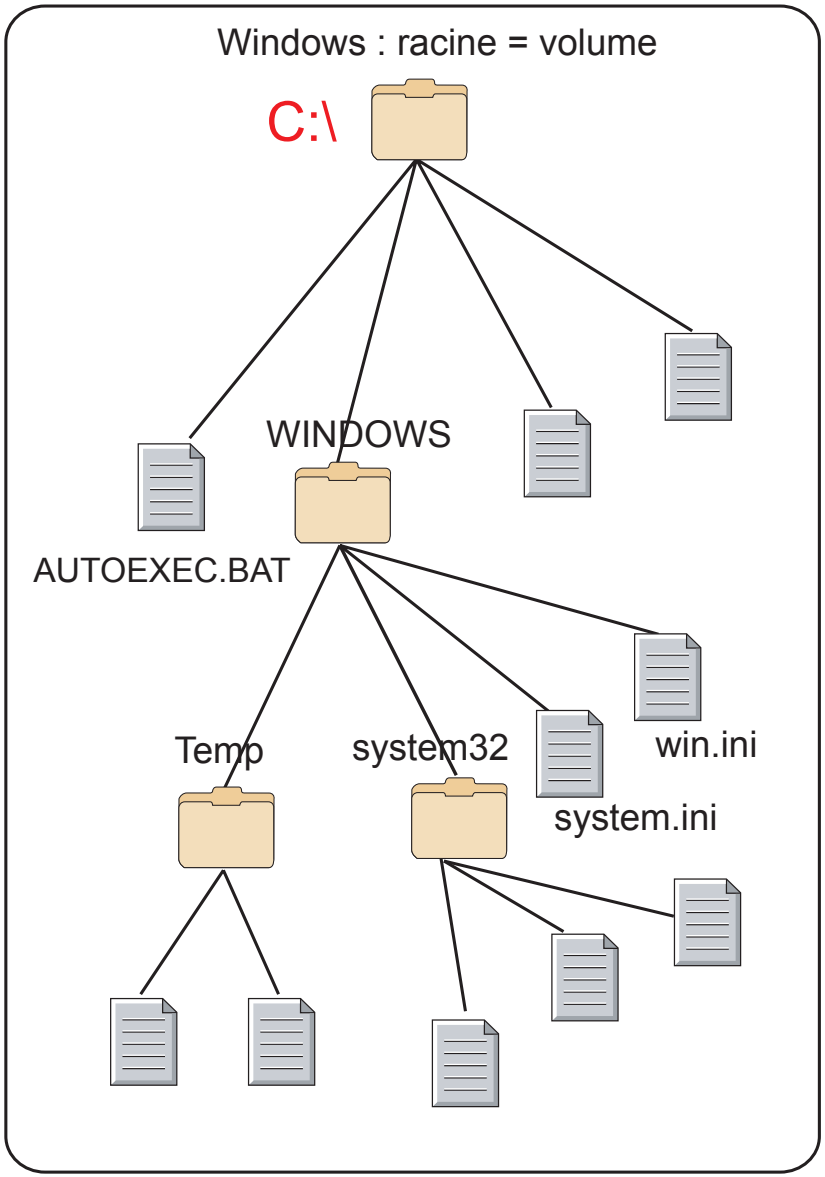
Types d'entrées (suite)

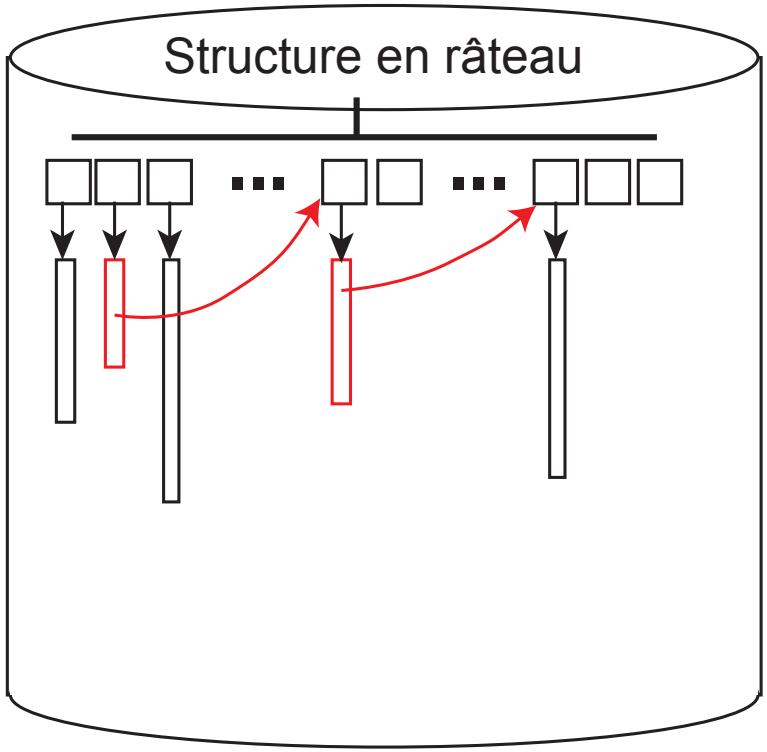
Exemple

```
>getfacl NOAM
# file : NOAM
# owner : padiou
# group : in
user : :rwx
group : :--
other : :--
```

Plan

La représentation des fichiers sur un volume





La structure d'un système de fichiers sur un volume

- Zone bootstrap-descripteur de volume ;
- Zone des descripteurs de fichiers ;
- Zone de données ;
- Zone de réplication.

Le mécanisme de montage/démontage de volume

Propriétés

- Unicité de l'arbre
- Transparence des volumes

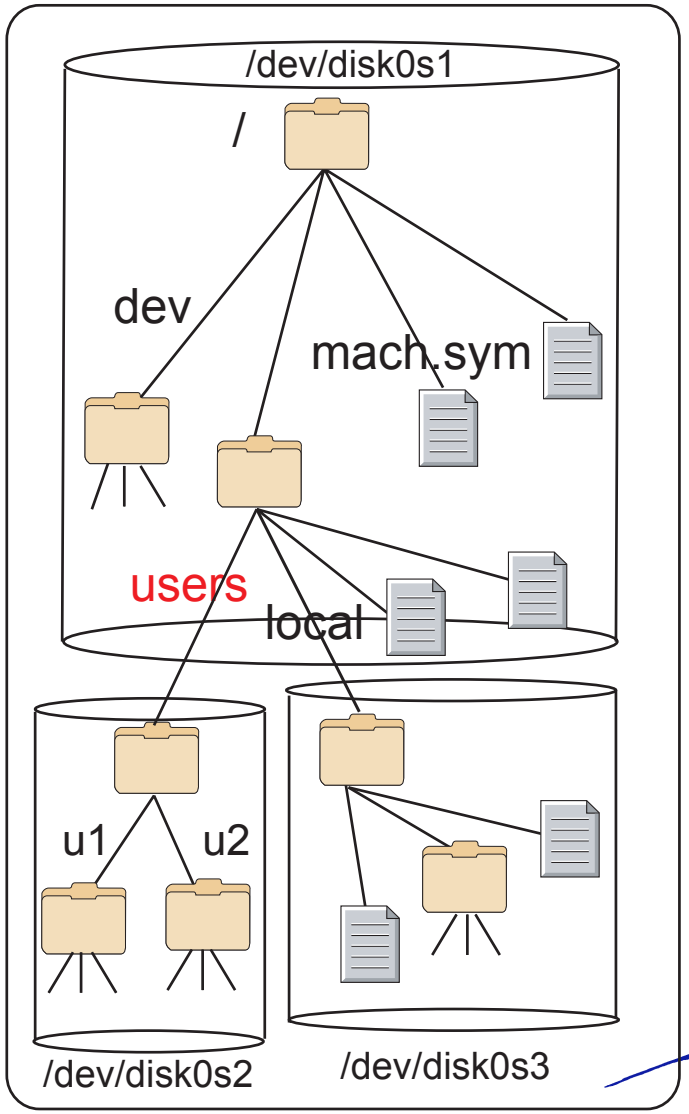
Commandes

- Montage :

  mount /dev/disk0s2 users

- Démontage :

  umount /dev/disk0s2



Descripteur de fichier (i-node)

Exemple Unix

Compteur de référence
Longueur du fichier (en octets)
Propriétaire créateur (groupe,id)
Date de créaction
Date de dernière modification
Date de dernière lecture
Type de fichier
Protection : indicateurs par exemple
Carte d'implantation

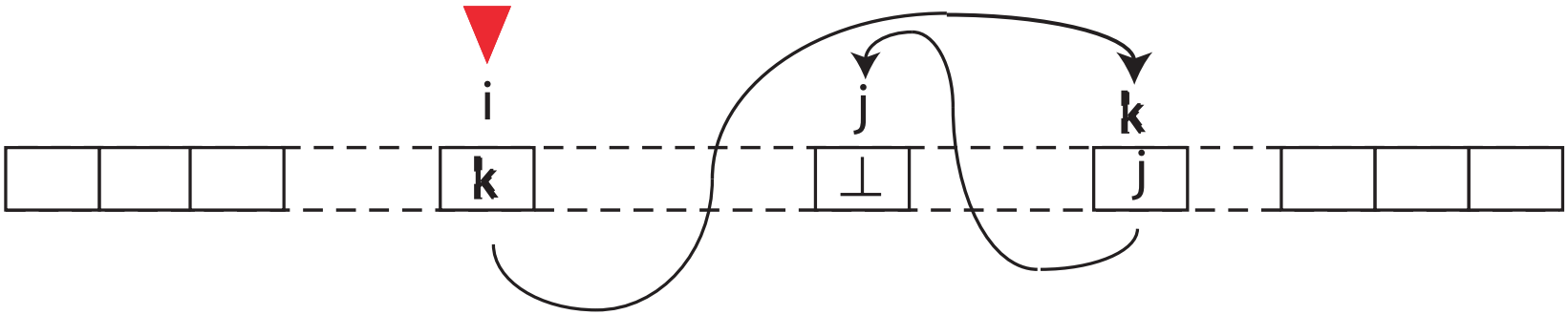
La carte d'un fichier

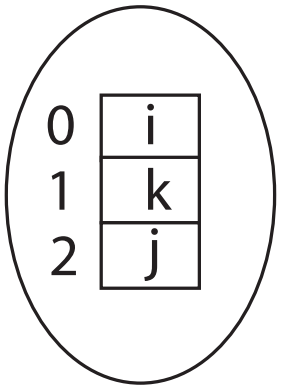
- Les blocs sont alloués dynamiquement ;
- Problème :

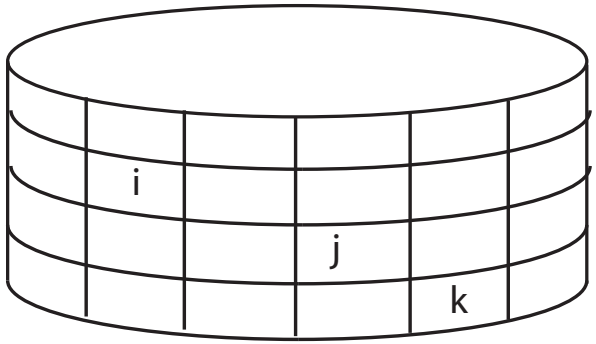
  comment déterminer la suite ordonnée de blocs alloués ?
- Deux possibilités (exemples) :
  - une liste chaînée (Windows) ;
  - tables imbriquées (Unix).

La carte d'un fichier

Version Windows (fat)



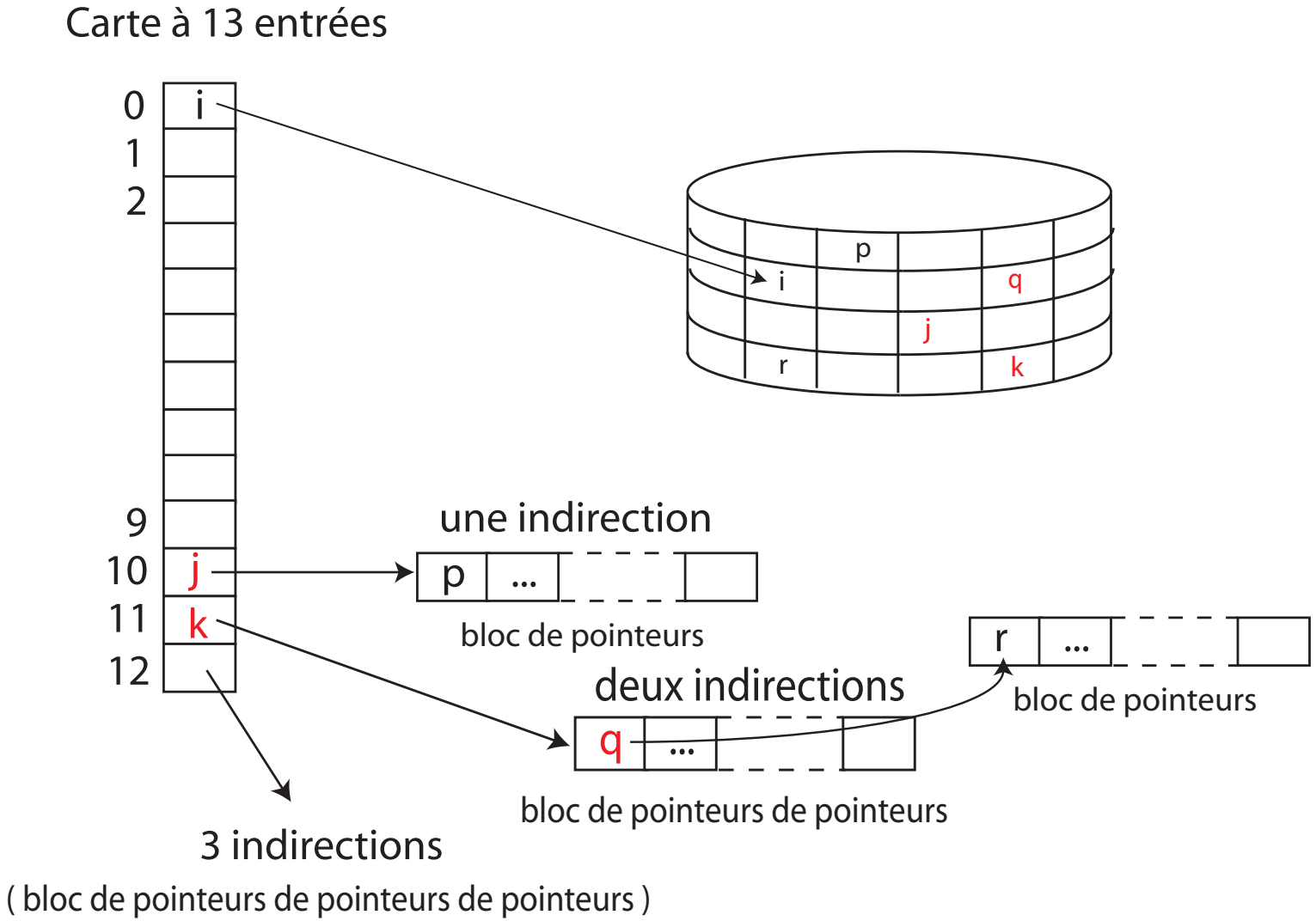




Carte représentée

La carte d'un fichier

Version Unix



La gestion des blocs libres

Version Unix