Chapitre 2

Système de Gestion de Fichiers

Le système de gestion de fichiers est un outil de manipulation des fichiers et de la structure d'arborescence des fichiers sur disque et a aussi le rôle sous UNIX de conserver toutes les informations dont la pérennité est importante pour le système (et pour les utilisateurs biensur). Ainsi tous les objets importants du système sont référencés dans le système de fichiers (mémoire, terminaux, périphériques variés, etc).

Le système de gestion de fichier permet une manipulation simple des fichiers et gère de façon transparente les différents problèmes d'accès aux supports de masse :

- partage : utilisation d'un même fichier/disque par plusieurs utilisateurs
- efficacité : utilisation de cache, uniformisation des accès
- droits : protection des éléments important du système et protection interutilisateurs
- alignement : transtypage entre la mémoire et les supports magnétiques

2.1 Le concept de fichier

L'unité logique de base de l'interface du Système de Gestion de Fichiers : `le fichier`.

Un fichier Unix est une suite finie de bytes (octets) Matérialisée par des blocs disques, et une inode qui contient les propriétés du fichier (mais pas son nom).
Le contenu est entièrement défini par le créateur, la gestion de l'allocation des ressources nécessaires est a la seule responsabilité du système.

Sur Unix les fichiers ne sont pas typés du point de vue utilisateur, le concept de fichier permet de proposer un type générique (polymorphe) aux programmeurs le système gérant la multiplicité des formats effectifs (différenttes marques et conceptions de disques dur par exemple).

L'inode définit le fichier, soit principalement les informations :

- la localisation sur disque,
- le propriétaire et le groupe propriétaire,
- les droits d'accès des différents utilisateurs,
- la taille,
- la date de création.

On trouvera sur d'autre systèmes d'autres structures d'information pour décrire les fichiers, par exemple NT utilise des "objets files records".

Un nom est lié à un fichier (une référence indique un fichier) mais un fichier n'est pas lié à une référence, un fichier peut exister sans avoir de nom dans l'arborescence.

2.2 Fichiers ordinaires / Fichiers spéciaux.

Le système est un utilisateur du système de gestion de fichier et en temps que créateur il définit quelques contenus structurés ces fichiers auront de ce fait des accès règlementés.

Pour le système les fichiers sont donc organisés en deux grandes familles :

les fichiers standards que sont par exemple les fichiers texte, les exécutables, etc. C'est-à-dire tout ce qui est manipulé et structuré par les utilisateurs.

Les fichiers spéciaux périphériques, mémoire, et autre fichiers "physiques" ou logique. Ces fichiers ont une structure interne définie (par les développeurs du système) qui doit être respecté c'est pourquoi leur manipulation n'est possible que par par l'intermédiaire du système (encore un bon exemple d'encapsulation).

Les catalogues sont des fichiers spéciaux, il faut pour les manipuler physiquement faire appel au système ce qui en protège la structure[1].

Les fichiers physiques dans le répertoire /dev (dev comme devices dispositifs matériels, les périphériques et quelques dispositifs logiques)

- Character devices (périphériques ou la communication ce fait octets par octets)
  les terminaux (claviers, écrans)
  les imprimantes
  la mémoire
  etc
- Block devices (périphériques ou la communication ce fait par groupe d'octet appelés blocs)
  les disques
  les bandes magnétiques
  etc

Les fichiers à usages logiques et non physiques

- liens symboliques
- pseudo-terminaux
- sockets
- tubes nommés

  Ce dernier type de fichiers spéciaux est utilisé pour servir d'interface entre disques, entre machines et simuler : des terminaux, des lignes de communication, etc.

Cette distinction entre fichier ordinaire et spéciaux et tout simplement le fait que le système est un utilisateur comme les autres des fichiers. Pour certains fichier le système utilise une structure interne spéciale (d'ou le nom) qui ne doit pas être modifier sous peine de comportement indéfini. Pour se protéger le système ne permet pas l'accès direct aux informations c'est lui qui fait toutes les entrées sortie sur les fichiers spéciaux de façon a en assurer l'intégrité. Ceci est indépendant du système de droits d'accès, la structure du code du noyau ne permet pas d'autres accès que les accès "spéciaux" [2].

2.3 Organisation utilisateur des Disques

Comment permettre aux utilisateurs d'identifier les données sur les supports de masse ?
Le système le plus répendu aujourd'hui est un système arborescent avec des fichiers utilisés comme

[1] les répertoires sont resté accessible longtemps en lecture comme des fichiers ordinaires mais l'accès en écriture était contraint, pour assurer la structure arborescente acyclique. Aujourd'hui tout les accès au répertoires ont contraint et on a un ensemble d'appels système spécifiques pour réaliser des entrés sortie dans les repertoires. `opendir(3), closedir(3), dirfd(3), readdir(3), rewinddir(3), scandir(3),seekdir(3), telldir(3)` approche qui permet d'être effectivement plus indépendant sur la structure interne des répertoires, avec des système plus efficaces que les listes utilisées dans les première implémentations. Voire Reiser fs par exemple.

[2] Pour plsu d'information sur le sujet aller voire dans les sources les structures de sgf et d'inode TODO : nom de fichiers concernés .

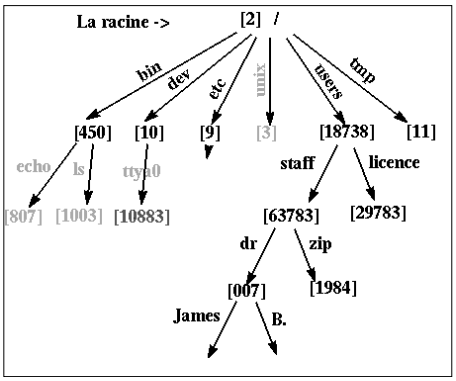

FIG. 2.1 – l'arborescence MULTICS

noeud de l'arbre qui permette de lister les fichiers et les sous arbres qu'il contienti, d'autres organisations "plates" existe ou l'on organise les fichiers en utilisans des types et des extentions de nom de fichier pour "organiser".

Les arborescences de fichiers et de catalogues, organisées comme un graphe acyclique [3], apparaissent avec le projet MULTICS.
Cette organisation logique du disque a les avantages suivants :

Une racine, un accès absolu aisé (à la différence de certains système qui ont de nombreuses "racines").
Une structure dynamique.
Une grande puissance d'expression.
Un graphe acyclique.

L'organisation est arborescente avec quelques connections supplémentaires (liens multiples sur un même fichier) qui en font un graphe. Mais ce graphe doit rester acyclique, pour les raisons suivantes :

L'ensemble des algorithmes simples utilisables sur des graphe acycliques comme le parcours, la vérification des fichiers libres, etc. deviennent beaucoup plus difficiles à écrire pour des graphes admettant des cycles.

Des algorithmes de ramasse-miettes doivent être utilisés pour savoir si certains objets sont utilisés on non et pour récuperer les inodes ou blocs perdus après un crash.

Tous les algorithmes de détection dans un graphe quelconque ont une complexité beaucoup plus grande que ceux qui peuvent profiter de l'acyclicité du graphe.

Sous Unix nous sommes assurés que le graphe est acyclique car il est interdit d'avoir plusieurs références pour un même catalogue (sauf la référence spéciale " . . ").

Sous UNIX c'est un graphe acyclique !

2.4 Les inodes

L'inode est le passage obligé de tous les échanges entre le système de fichier et la mémoire. L'inode est la structure qui contient toutes les informations sur un fichier donné à l'exception de

[3] Ce n'est pas un arbre car un fichier peut avoir plusieurs références

sa référence dans l'arborescence (son nom), l'arborescence n'étant qu'un outil de référencement des fichiers.

Les informations stockées dans une inode disque sont :
- utilisateur propriétaire
- groupe propriétaire
- type de fichier
  - fichiers ordinaires
  d répertoire (dyrectory)
  b mode bloc
  c mode caractère
  l lien symbolique
  p pour une fifo (named pipe)
  s pour une socket
- droits d'accès (ugo*rwx)
- date de dernier accès
- date de dernière modification
- date de dernière modification de l'inode
- taille du fichier
- adresses des blocs-disque contenant le fichier.

Dans une inode en mémoire (fichier en cours d'utilisation par un processus) on trouve d'autres informations supplémentaires :

le statut de l'inode

{ locked,
waiting P
inode à écrire,
fichier à écrire,
le fichier est un point de montage
}

Et deux valeurs qui permettent de localiser l'inode sur un des disques logiques :

Numéro du disque logique
Numéro de l'inode dans le disque

cette information est inutile sur le disque (on a une bijection entre la position de l'inode sur disque et le numéro d'inode).
On trouve aussi d'autres types d'informations comme l'accès à la table des verrous ou bien des informations sur les disques à distance dans les points de montage.

2.5 Organisation des disques System V

L'organisation disque décrite sur la figure 2.2 est la plus simple que l'on peut trouver de nos jours sous UNIX, il en existe d'autres où l'on peut en particulier placer un même disque logique sur plusieurs disques physiques (dangereux), certaines où les blocs sont fragmentables, etc.

Boot bloc utilisé au chargement du système.

Super Bloc il contient toutes les informations générales sur le disque logique.

Inode list Table des inodes.

blocs les blocs de données chainés à la création du disque (`mkfs`).

Les blocs de données ne sont pas fragmentables sous Système V.

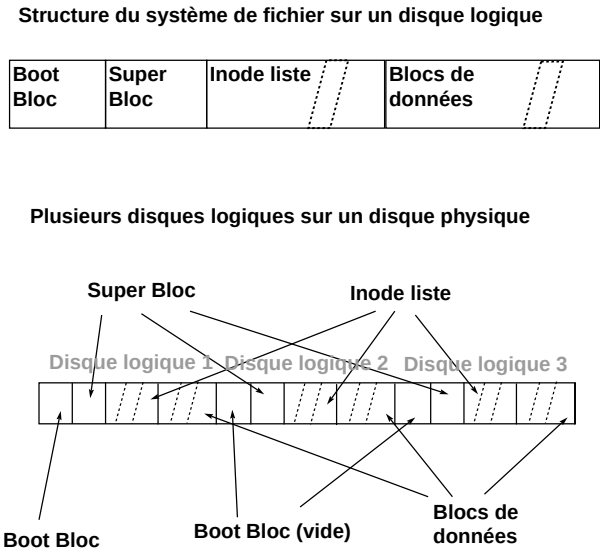

FIG. 2.2 – Organisation des blocs et des inodes (SYS V)

2.6 Adressage des blocs dans les inodes

Le système d'adressage des blocs dans les inodes (système V) consiste en 13 adresses de blocs. Les dix premières adresses sont des adresses qui pointent directement sur les blocs de données du fichier. Les autres sont des adresses indirectes vers des blocs de données contenant des adresses. La figure 2.3 nous montre les trois niveaux d'indirection. L'intérêt de cette représentation est d'économiser sur la taille des inodes tout en permettant un accès rapide au petits fichiers (la majorité des fichiers sont petits). Mais en laissant la possibilité de créer de très gros fichiers :

$$10 + 256 + (256 \times 256) + (256 \times 256 \times 256)$$

blocs disques.

2.7 Allocation des inodes d'un disque

L'allocation des inodes est réalisée en recherchant dans la zone des inodes du disque une inode libre. Pour accélérer cette recherche : un tampon d'inodes libres est géré dans le SuperBloc, de plus l'indice de la première inode libre est gardé en référence dans le SuperBloc afin de redémarrer la recherche qu'à partir de la première inode réellement libre.

Mais ce système a une faille qu'il faut prévoir dans l'écriture dans l'algorithme ialloc d'allocation d'inode, cette faille est décrite dans la Figure 2.10

2.8 Allocation des blocs-disque

L'algorithme utilisé pour gérer l'allocation des inodes s'appuie sur le fait que l'on peut tester si une inode est libre ou non en regardant son contenu. Ceci n'est plus vrai pour les blocs. La solution est de chaîner les blocs. Ce chaînage est réalisé par blocs d'adresses pour accélérer les accès et profiter au maximum du buffer cache. Il existe donc un bloc d'adresses dans le super bloc qui sert de zone de travail pour l'allocateur de blocs. L'utilisation de ce bloc et le mécanisme d'allocation sont décrits dans les Figures 2.11 à 2.16

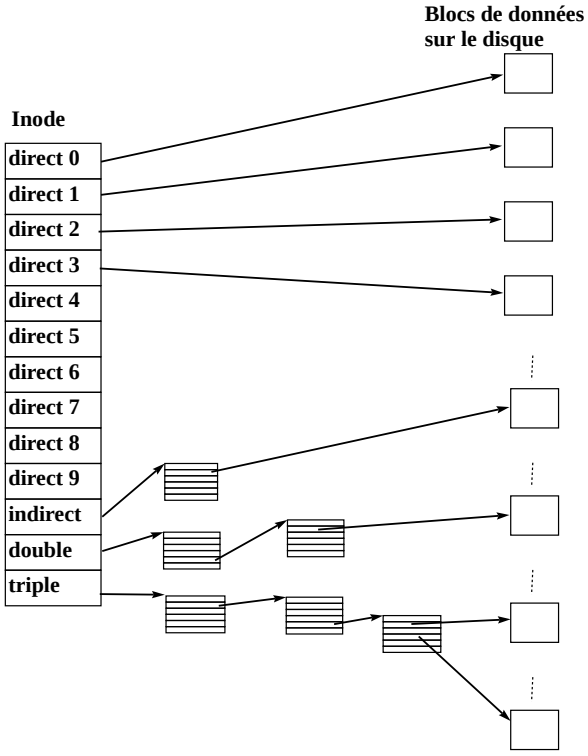

FIG. 2.3 – Adressage direct et indirect des inode UNIX

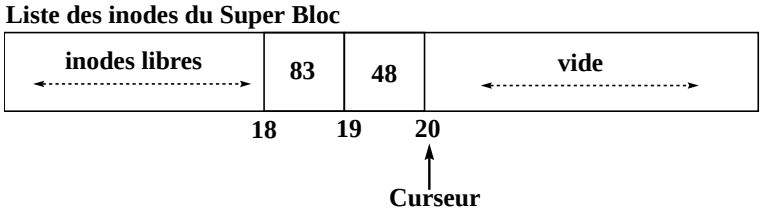

FIG. 2.4 – Inodes libres dans le SuperBloc.

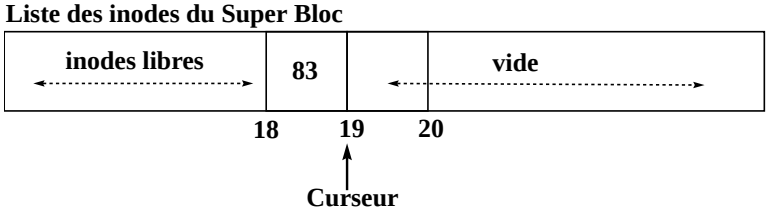

FIG. 2.5 – Allocation d'une inode.

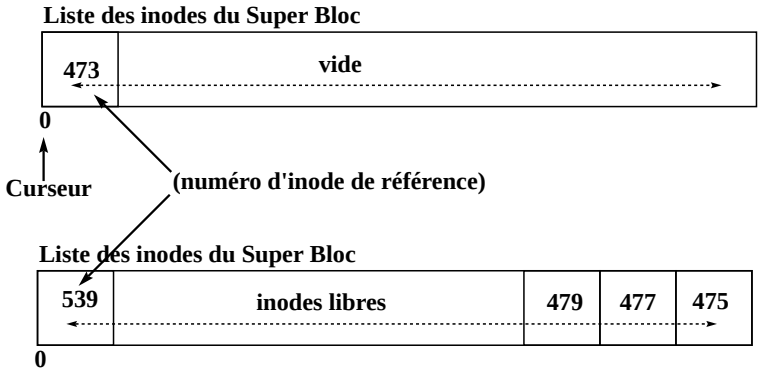

FIG. 2.6 – Si le SuperBloc est vide.

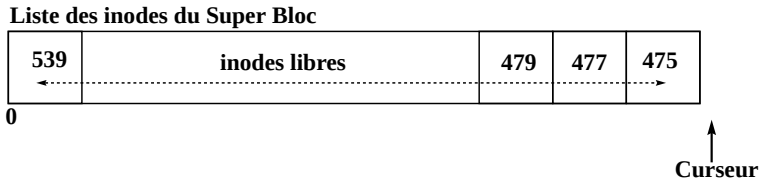

FIG. 2.7 – Libération d'une inode avec le SuperBloc plein.

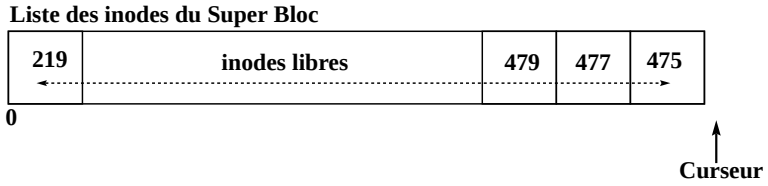

FIG. 2.8 – Le numéro d'inode inférieur au numéro de référence.

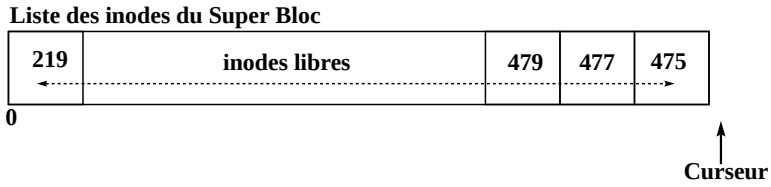

FIG. 2.9 – Le numéro d'inode supérieur au numéro de référence.

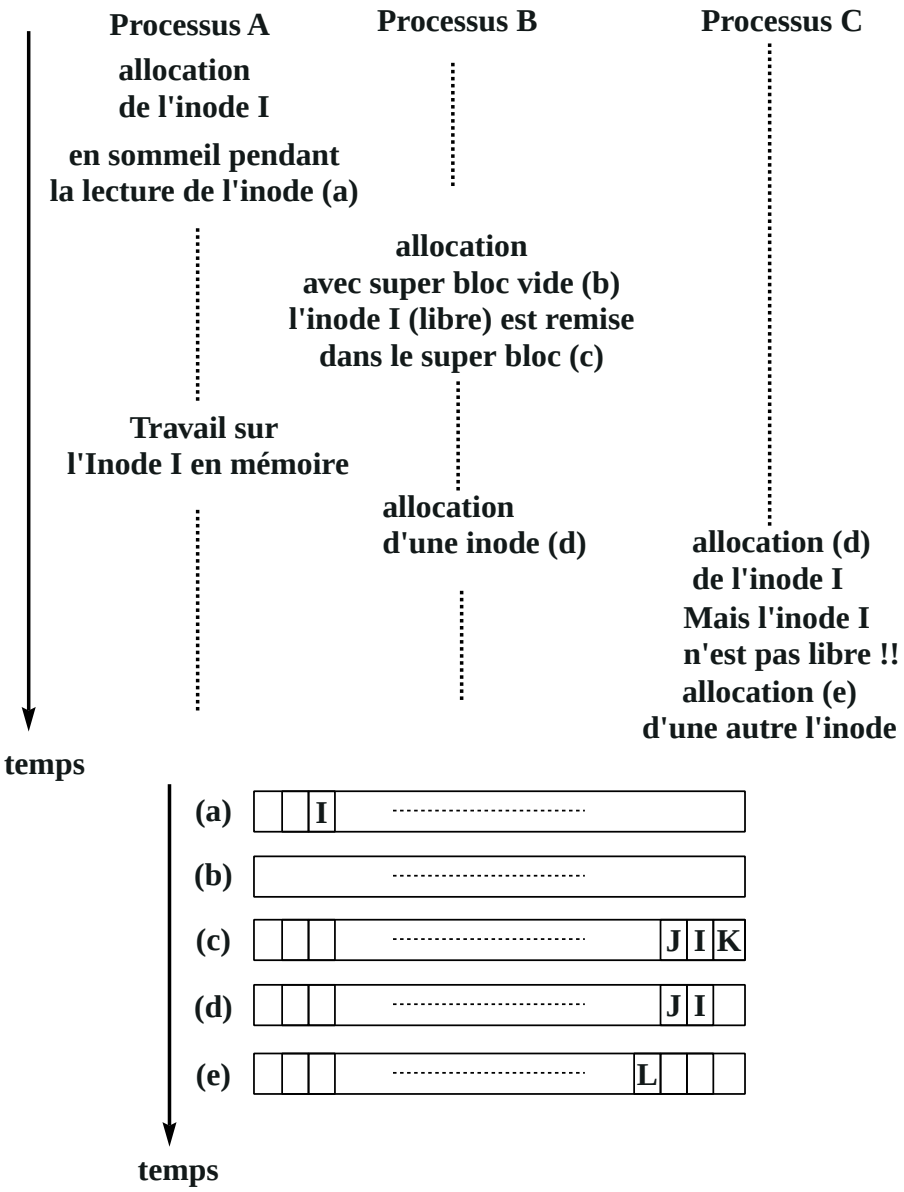

FIG. 2.10 – Faille de l'algorithme d'allocation.

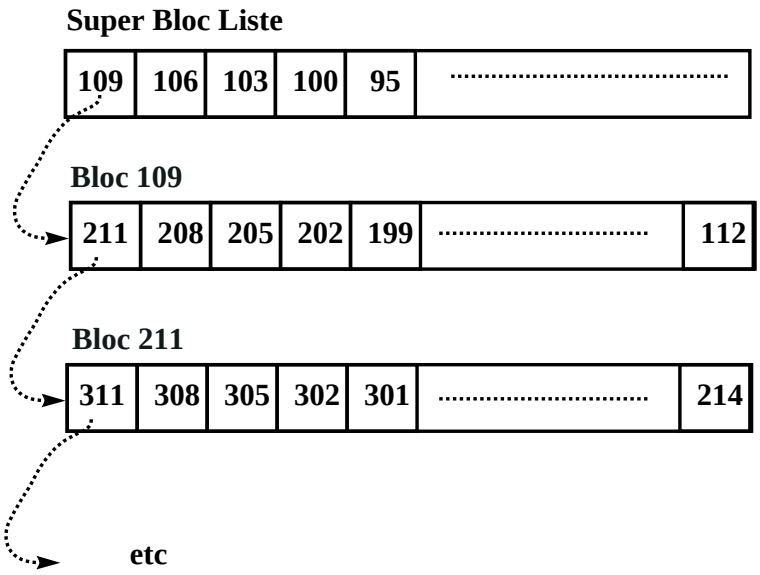

FIG. 2.11 – Liste chainée de blocs.

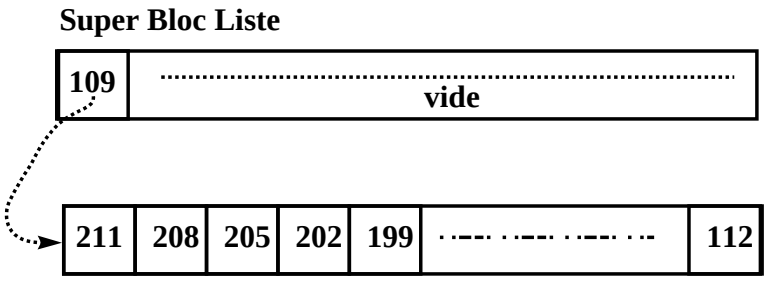

FIG. 2.12 – Etat initial du SuperBloc.

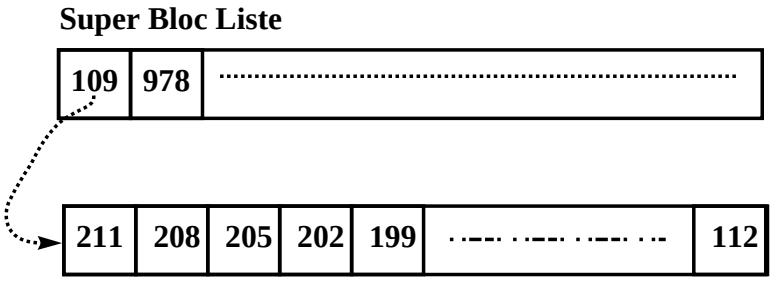

FIG. 2.13 – Libération du bloc 978.

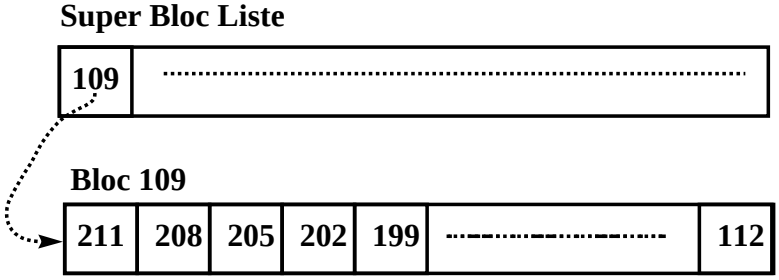

FIG. 2.14 – Allocation du bloc 978.

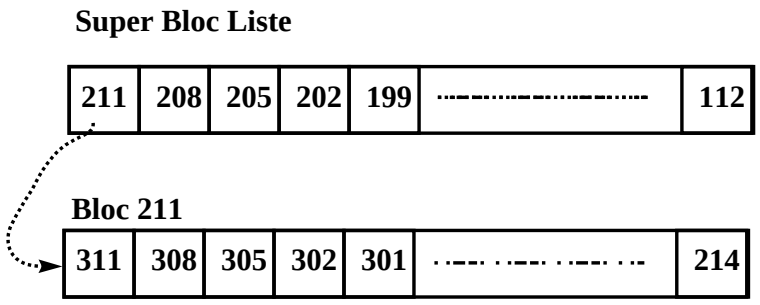

FIG. 2.15 – Allocation du bloc 109.

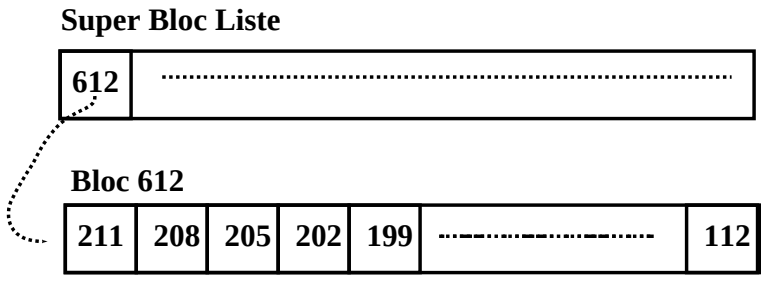

FIG. 2.16 – Libération du bloc 612.