Cours « système d'exploitation »
2ème année
IUT de Caen, Département d'Informatique
Année 2005 - 2006
(François Bourdon)

Plan

1. **Système de Gestion des Fichiers : Concepts avancés**
3. **Création et Ordonnancement de Processus**
4. **Synchronisation de Processus**
5. **Communication entre Processus : les Signaux**
6. **Echange de données entre Processus :**

   les Tubes et les Verrous
6. **Communication entre Processus : les IPC**
   - **segments de mémoire partagée**
   - **files de messages**
   - **sémaphores**
7. **Communication sous UNIX - TCP/IP : les sockets**
8. **Gestion de la mémoire**
9. **Les systèmes distribués**
10. **Les systèmes distribués à objets (CORBA)**

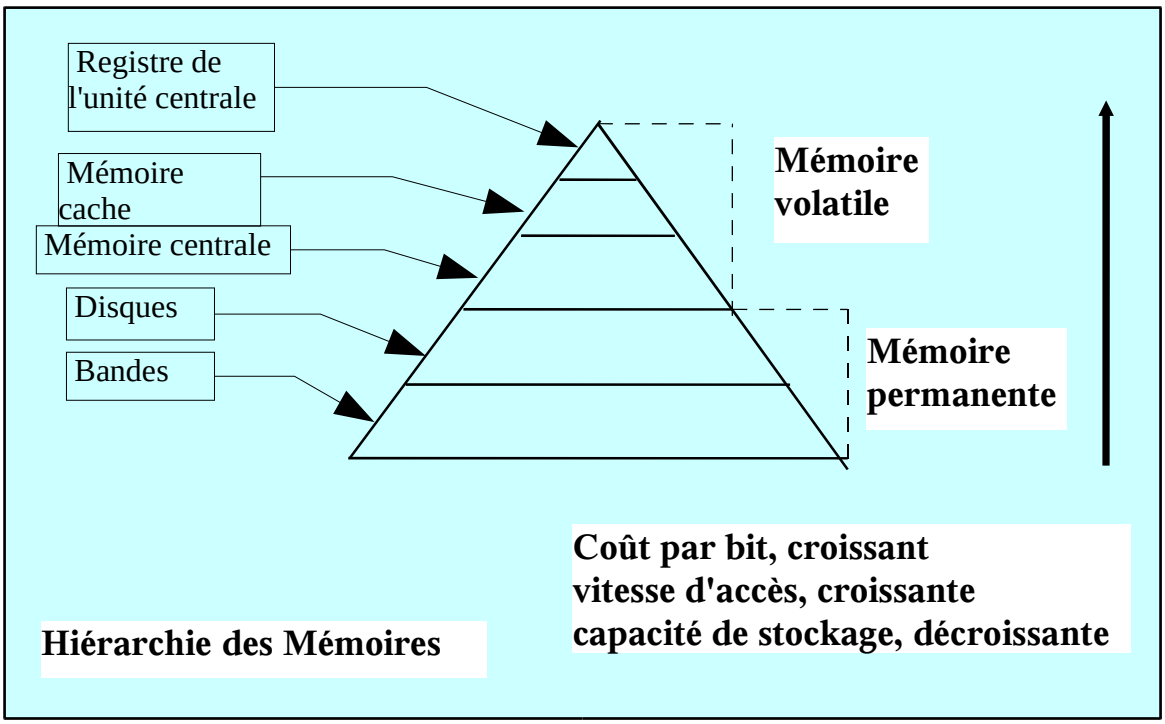
Chapitre 1
Système de Gestion des Fichiers : Concepts Avancés (partie 1)

Plan

1. **Système de Gestion des Fichiers : Concepts avancés**
    a. **Représentation interne du SGF**
    b. **Les E/S et le SGF**
    c. **E/S tamponnées : le Buffer Cache**
    d. **Le système de fichiers virtuel (SFV)**
    e. **Appels système et SGF**
    f. **Cohérence d'un SGF**
    g. **Le système de fichiers « /proc »**
    h. **Monter un SGF**
    i. **SGF et caractéristiques physiques d'un disque**
    j. **Organisation classique d'un SGF**
2. **Création et Ordonnancement de Processus**
3. **Synchronisation de Processus**
4. **...**

1.1 Représentation interne du SGF

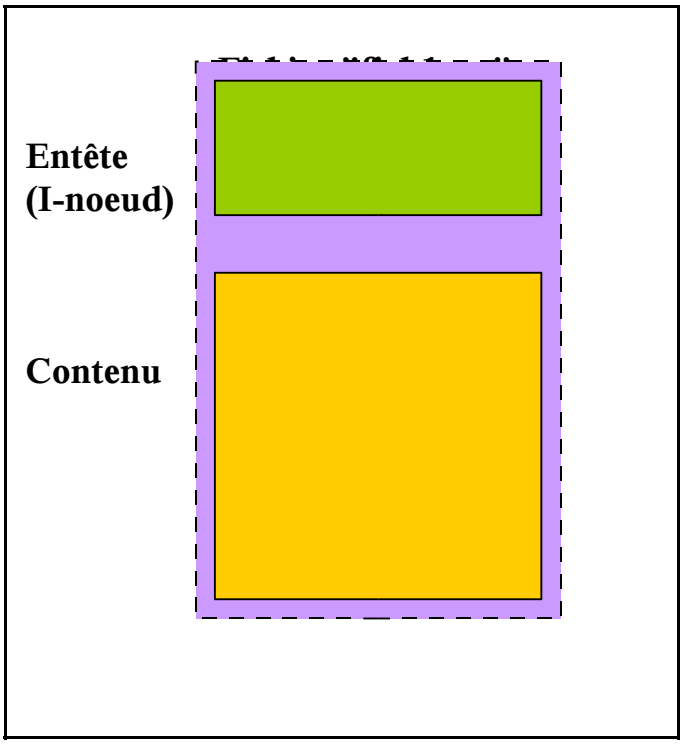
Le système d'exploitation doit pouvoir stocker à long terme de grandes quantités d'informations : ses propres modules, les programmes d'application, les bibliothèques de fonctions, les programmes et les données des utilisateurs.



Le support physique (mémoire secondaire) doit être de grande capacité et à un faible coût. Par contre il possède des temps d'accès beaucoup plus long que les supports volatiles (mémoire centrale).

Le système d'exploitation doit assurer la correspondance entre la représentation physique des informations et la représentation logique qu'en a l'utilisateur.

Le fichier physique est la représentation physique de l'information tandis que le fichier logique est un identificateur, qui désigne à la fois le fichier physique (i-noeud) et la représentation logique (nom), c'est-à-dire l'ensemble d'informations tel qu'il est vu par l'utilisateur.

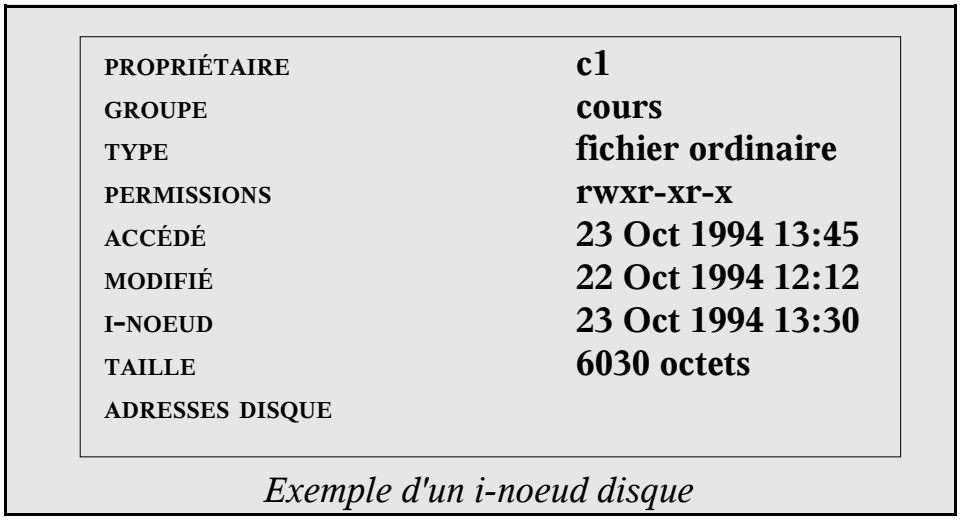


LES PROCESSUS ACCÈDENT AUX FICHIERS PAR UN ENSEMBLE D'APPELS SYSTÈMES EN SPÉCIFIANT UN FICHIER PAR UNE CHAÎNE DE CARACTÈRES QUI CONSTITUE LE CHEMIN D'ACCÈS.

Chaque chemin d'accès spécifie de manière unique un fichier, et c'est le noyau qui convertit le chemin d'accès en l'i-noeud du fichier.

Le noyau lit les i-noeuds sur disque et les place en mémoire dans des i-noeuds mémoire pour pouvoir les gérer.

L'I-NOEUD CONTIENT LES INFORMATIONS NÉCESSAIRES À UN PROCESSUS POUR ACCÉDER À UN FICHIER, TELS QUE LE PROPRIÉTAIRE, LES DROITS D'ACCÈS, LA TAILLE ET LA LOCALISATION DES DONNÉES DANS LE SYSTÈME DE FICHIERS.



Exemple d'un i-noeud disque

Les dates d'accès au fichier donnent les dates de la dernière modification du fichier, du dernier accès au fichier et de la dernière modification de l'i-noeud.

La table des adresses disque correspond aux numéros des blocs du disque qui contiennent les données du fichier. Bien que les utilisateurs traitent les données d'un fichier comme une suite logique d'octets, le noyau sauvegarde les données dans des blocs du disque non contigus.

L'i-noeud ne spécifie pas le chemin d'accès au fichier.

Changer le contenu d'un fichier implique automatiquement un changement dans l'i-noeud, mais pas l'inverse.

QUESTION : UN I-NOEUD CONTIENT-IL LE NOM DU FICHIER CORRESPONDANT ?

Rappel : Tout fichier a un i-noeud (index node), mais peut avoir plusieurs noms, chaque nom étant en relation avec l'i-noeud.

Chaque nom est appelé un **lien**. Quand un processus se réfère à un fichier par un nom, le noyau analyse le nom du fichier, une composante à la fois, vérifie que le processus a la permission d'inspecter les répertoires du chemin, et finalement extrait l'i-noeud du fichier.

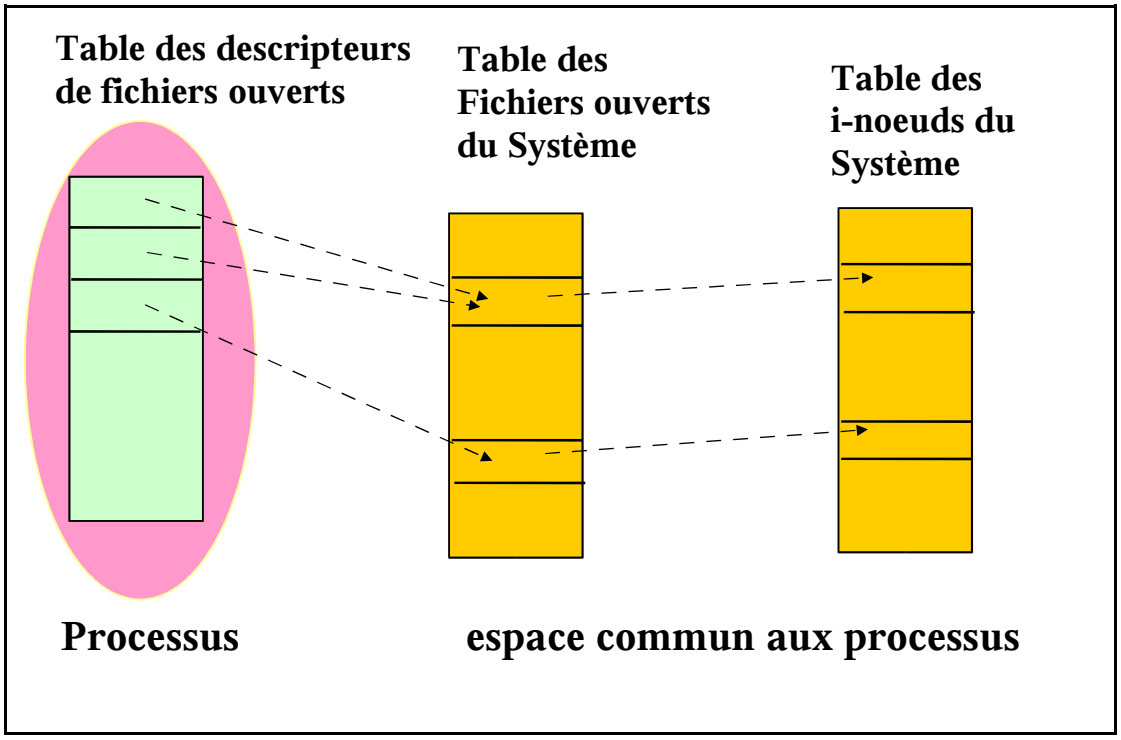
Par exemple, si un processus appelle

```
open (« /fs2/mjb/rje/fichiersource »,1);

le noyau extrait l'i-noeud de (« /fs2/mjb/rje/fichiersource »).
```

Quand un processus crée un nouveau fichier, le noyau lui assigne un i-noeud inutilisé. Les i-noeuds sont rangés dans le système de fichiers, mais le noyau les lit et les mémorise dans une table des i-noeuds lorsqu'il manipule les fichiers.

Le noyau contient deux autres structures de données, *la table des fichiers ouverts* et la *table des descripteurs de fichier utilisateur*.



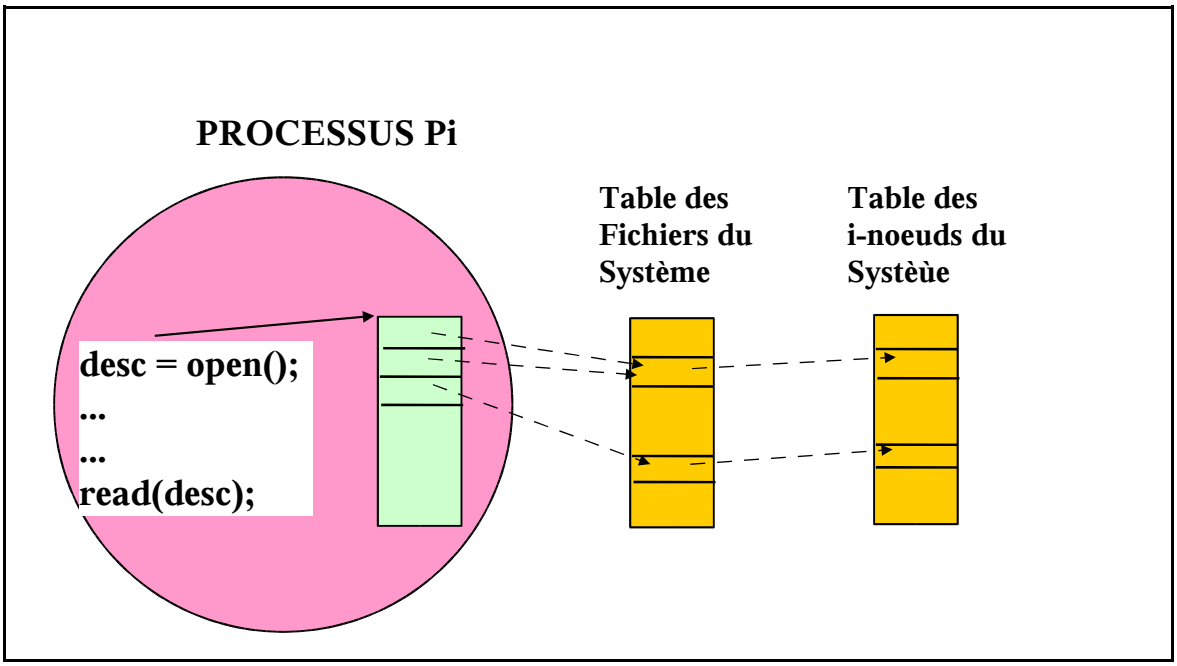
La table des fichiers ouverts est une structure globale du noyau, mais la table des descripteurs de fichier utilisateur est allouée pour un processus.

Quand un processus ouvre (*open*) ou crée (*creat*) un fichier, le noyau alloue un élément dans chaque table, correspondant à l'i-noeud du fichier.

CES TROIS TABLES, CONSTRUITES EN MÉMOIRE VOLATILE, ENTRETIENNENT L'ÉTAT DU FICHIER ET SON ACCÈS PAR LES UTILISATEURS. ELLES PERMETTENT À DIVERS DEGRÉ DE PARTAGER L'ACCÈS À UN FICHIER.

La table des fichiers préserve le déplacement octet dans le fichier d'où le prochain *read* ou *write* utiliseur débutera, ainsi que les droits d'accès permis au processus qui fait l'*open*.

La table des descripteurs de fichiers utilisateur identifie tous les fichiers ouverts par un processus.



Le noyau retourne un descripteur de fichier sur les appels système *open* et *creat*, qui est un indice dans la table des descripteurs de fichier utilisateur. En exécutant les appels système *read* et *write*, le noyau utilise le descripteur de fichier pour accéder à la table des i-noeuds, et, de l'i-noeud, retrouve les données du fichier.

Le système UNIX maintient les fichiers ordinaires et les répertoires sur des périphériques blocs tels que les bandes ou les disques.

Une installation peut avoir plusieurs unités de disques physiques, chacune contenant un ou plusieurs systèmes de fichiers. On peut partager des unités de disques à distance (mount), l'accès étant réalisé par le réseau.

En partitionnant un disque en plusieurs systèmes de fichiers, on facilite, aux administrateurs, la gestion des données qui y sont stockées. Le noyau traite, à un niveau logique, avec les systèmes de fichiers plutôt qu'avec les disques, en considérant chacun d'eux comme un périphérique logique identifié par un numéro de périphérique logique.

La conversion des adresses des périphériques logiques (systèmes de fichiers) en adresses de périphériques physiques (disque) est faite par le contrôleur du disque.

Un système de fichier consiste en une suite de blocs logiques, chacun contenant 512, 1024, 2048, ou n'importe quel multiple de 512 octets, selon la version du système. La taille d'un bloc logique est homogène (fixe) dans un système de fichiers.

Les fichiers de données sur les disques se répartissent dans des blocs de taille fixe correspondant à des unités d'entrées-sorties du contrôleur de ce périphérique. La lecture ou l'écriture d'un élément d'un fichier impliquera donc le transfert du bloc entier qui contient cet élément.

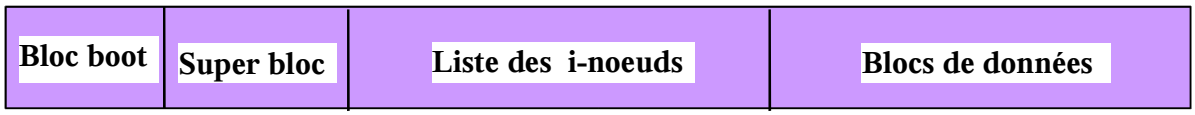
En formattant les disques, donc utilisables par le système d'exploitation, avec une taille particulière de bloc, on joue sur le compromis entre la vitesse d'accès aux éléments des fichiers et l'espace perdu sur le disque.

Lors d'un transfert de données d'un disque vers l'espace d'adressage d'un processus, le temps de lecture ou d'écriture sur le disque est négligeable devant le temps d'accès au bloc, et ceci quelque soit la taille du bloc.

Pour un accès rapide, on aura intérêt à prendre des blocs de grande taille.

Cependant, les fichiers, y compris les fichiers de 1 octet, ont une taille minimale de 1 bloc. Si un disque comprend beaucoup de fichiers de petite taille et si les blocs sont de grande dimension, l'espace gaspillé sera alors considérable.

La structure sur disque logique d'un système de fichier est la suivante :



- Le ***bloc root*** occupe le début du système de fichiers, généralement le premier secteur, et peut contenir le code du *bootstrap* qui est lu et mis dans la machine pour amorcer (*boot*) le système (chargement d'UNIX en mémoire).

- Le ***super bloc*** décrit l'état du système de fichiers, son importance, le nombre de fichiers qu'il peut stocker, les endroits (blocs libres) où trouver de l'espace libre sur le système de fichiers, la taille des blocs (généralement 1k pour UNIX) et d'autres informations.

- La ***liste des i-noeuds*** : contient la définition (caractéristiques) des fichiers. Une telle définition est un i-noeud et son n° dans la table (i-liste) est le i-nombre (inode). L'administrateur spécifie la taille de la *liste des i-noeuds* en configurant un système de fichiers. Le noyau fait référence aux i-noeuds selon un indice dans la liste des i-noeuds. L'un d'entre eux (2) est l'i-noeud racine du système de fichier ; c'est l'i-noeud par lequel la structure en répertoires du système de fichiers est accessible (montée) après l'exécution de l'appel système *mount*.

- Les ***blocs de données*** contiennent les données des fichiers c'est-à-dire les blocs logiques qui les constituent. Un bloc de données alloué peut appartenir à un et à un seul fichier du système de fichiers. Cette région contient aussi des blocs libres qui seront alloués puis libérés en fonction des besoins. Ces blocs libres sont répertoriés par le super bloc.

- Enfin une partie du disque (***swap***) est réservée pour le transfert des processus et pour la mémoire virtuelle.

La structure du super bloc comprend :

- **la taille du système de fichiers,**
- **le nombre de blocs libres,**
- **une liste des blocs libres,**
- **l'indice du premier bloc libre dans la liste des blocs libres,**
- **la taille de la liste des i-noeuds,**
- **le nombre de i-noeuds libres,**
- **une liste des i-noeuds libres,**
- **l'indice du premier i-noeud libre dans la liste des i-noeuds libres,**
- **des champs de verrous pour l'accès aux listes d'i-noeuds et de blocs libres,**
- **un drapeau indiquant si le super bloc a été modifié.**

Le super bloc est une structure très importante pour le maintien de la cohérence du système de fichiers. C'est pour cela que seul le noyau a le droit de le manipuler et il en fait des copies fréquentes.

Il faut remarquer que compte tenu des nombreux accès au super bloc une image mémoire de celui-ci est présente en permanence.

D'autre part, l'accès au super bloc doit être exclusif pour éviter toute incohérence sur celui-ci (mise en place de verrous d'accès).

L'I-NOEUD MÉMOIRE, COPIE MÉMOIRE DANS LA TABLE DES I-NOEUDS DE L'I-NOEUD DISQUE, CONTIENT DES CHAMPS SUPPLÉMENTAIRES À CEUX DE L'I-NOEUD DISQUE :

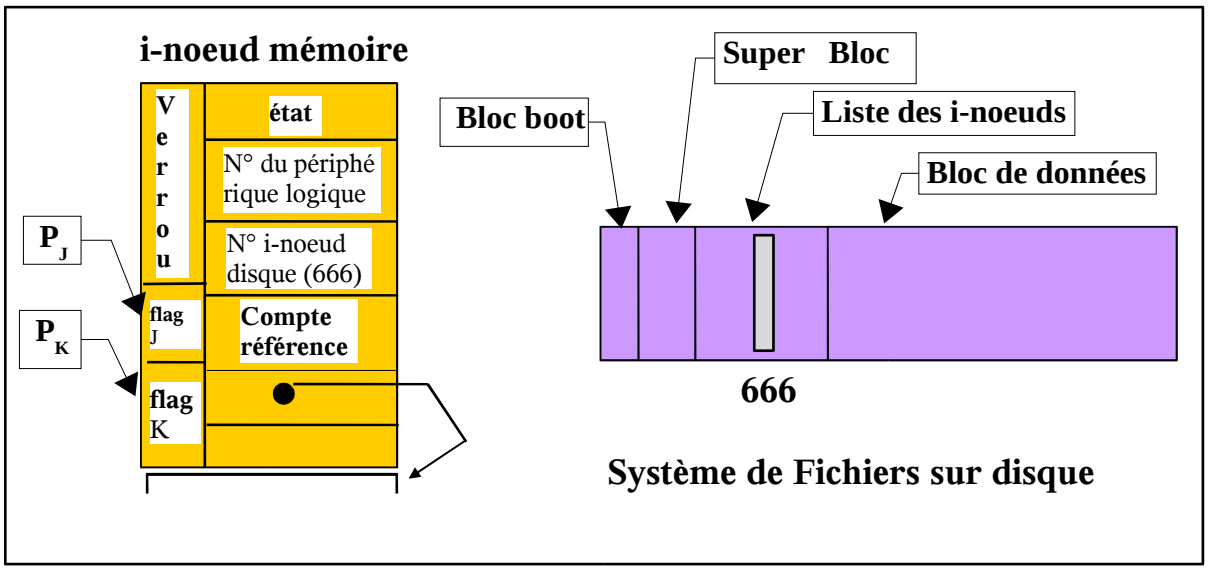
- **L'état de l'i-noeud** (l'i-noeud est verrouillé, un processus attend que l'i-noeud soit déverrouillé, ...).

- **Le numéro du périphérique logique** du système de fichiers qui contient le fichier.

- **Le numéro de l'i-noeud**. Puisque les i-noeuds sont rangés dans un tableau linéaire sur le disque, le noyau identifie le numéro d'un i-noeud disque par sa position dans le tableau. L'i-noeud disque n'a pas besoin de ce champ.

- **Des pointeurs** sur d'autres i-noeuds mémoire. Le noyau chaîne les i-noeuds dans une liste d'i-noeuds libres et dans des files à adressage calculé.

- **Un compte référence**, indiquant le nombre d'instances du fichier qui sont actives (par exemple, ouvert par open).



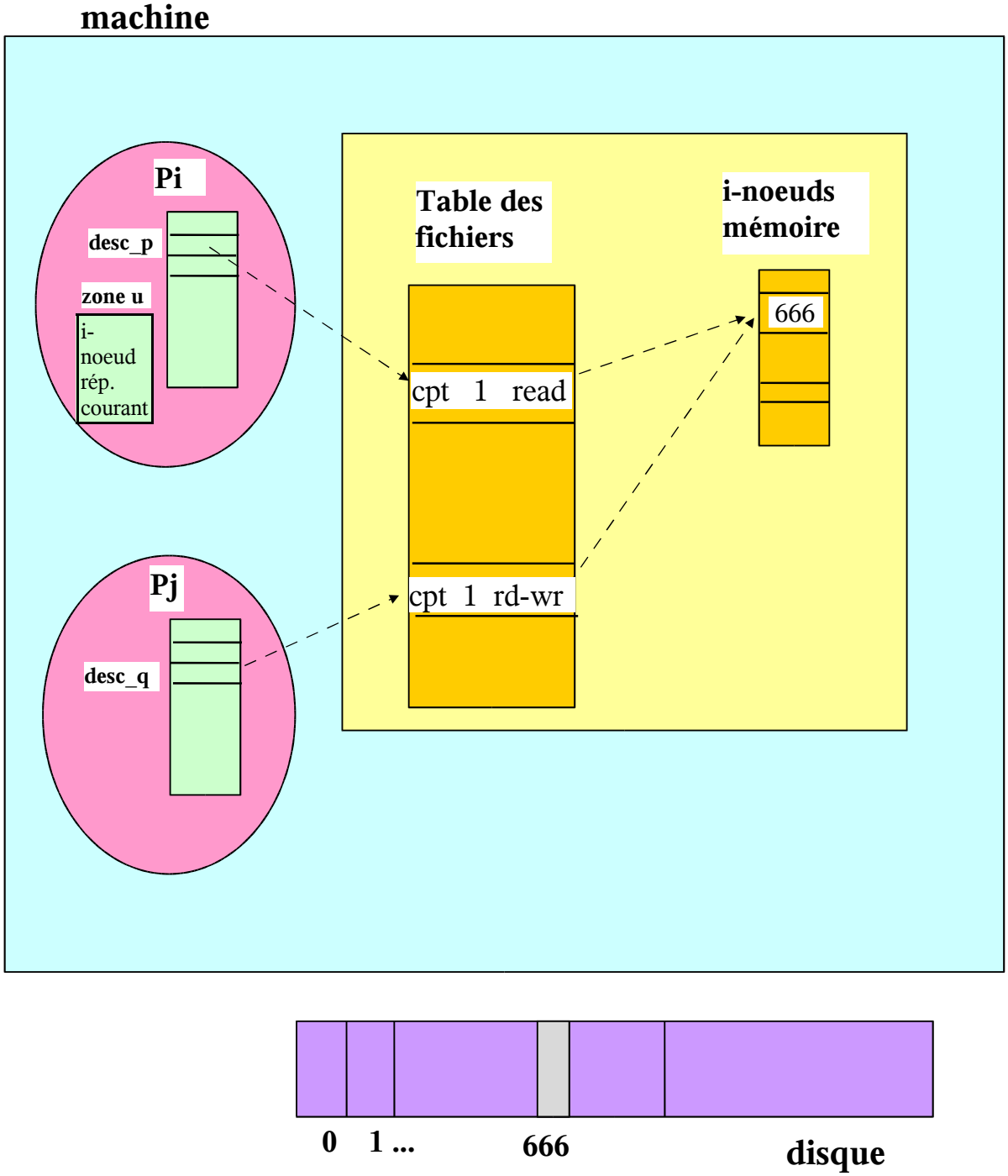
Le verrou d'un i-noeud, quand il est positionné, empêche les autres processus d'accéder à l'i-noeud ; les autres processus positionnent un drapeau (*flag*) dans l'i-noeud lorsqu'ils essayent de l'accéder pour indiquer qu'ils devront être réveillés quand le verrou sera libéré.

Le noyau positionne d'autres drapeaux pour indiquer une différence entre l'i-noeud disque et sa copie mémoire. Quand le noyau a besoin d'enregistrer le changement du fichier ou de l'i-noeud, il écrit la copie mémoire de l'i-noeud sur le disque après avoir examiné ces drapeaux.

Un i-noeud est actif quand un processus se l'attribue, comme lorsqu'il ouvre le fichier. Un i-noeud est dans la liste des i-noeuds libres seulement si son compte référence vaut 0, ce qui signifie que le noyau peut le réallouer à un autre i-noeud disque.

Le noyau peut contenir au plus une copie mémoire de l'i-noeud disque, mais un i-noeud mémoire peut être à la fois dans une file et dans la liste des i-noeuds libres.

La liste des i-noeuds libres sert ainsi de cache d'i-noeuds inactifs : si un processus tente d'accéder à un fichier dont l'i-noeud n'est pas dans la réserve des i-noeuds mémoire, le noyau réalloue un i-noeud mémoire depuis la liste des i-noeuds libres pour l'utiliser.


machine
Pi
desc_p
zone u
i-
noeud
rép.
courant
Table des
fichiers
i-noeuds
mémoire
666
cpt 1 read
Pj
desc_q
cpt 1 rd-wr
0 1 ... 666
disque

Le noyau utilise des algorithmes efficaces de recherche des i-noeuds en s'appuyant sur les files à adressage calculé.

Voici un exemple d'algorithme d'attribution d'un « i-noeud mémoire » (iget) :

```
algorithme iget
  entrée : numéro d'i-noeud dans un système de fichiers
  sortie : i-noeud verrouillé
 {
  while (non effectué)
  {
        if (i-noeud présent dans le cache des i-noeuds)
        {
          if (i-noeud verrouillé)
          {
            sleep(évènement: i-noeud est dévérouillé);
            continue;
          }
          /* traitement spécial pour les points de montage */
          if (i-noeud dans la liste des i-noeuds libres)
             extraire l'i-noeud de la liste des i-noeuds libres;
          incrémenter le compte référence i-noeud;
          return (i-noeud);
        }
        else
        {
          /* i-noeud n'est pas dans le cache des i-noeuds */
          if (pas d'i-noeud dans la liste des i-noeuds libres)
            return (erreur);
          extraire un nouvel i-noeud de la liste des i-noeuds libres;
          repositionner numéros d'i-noeud et de système de fichiers;
          extraire l'i-noeud de l'ancienne file, le placer dans la
                                                          nouvelle;
          lire l'i-noeud depuis le disque (algorithme bread);
          initialiser l'i-noeud (compte référence à 1);
          return (i-noeud);
        }
 } }
```

Il existe d'autres algorithmes comme par exemple celui de la libération d'un i-noeud.

En particulier **si le compte de référence de l'i-noeud mémoire devient nul, le noyau écrit l'i-noeud sur disque si cette copie mémoire diffère de la copie disque**. L'i-noeud mémoire est alors placé dans la liste des i-noeuds libres (idem avec les blocs de données).

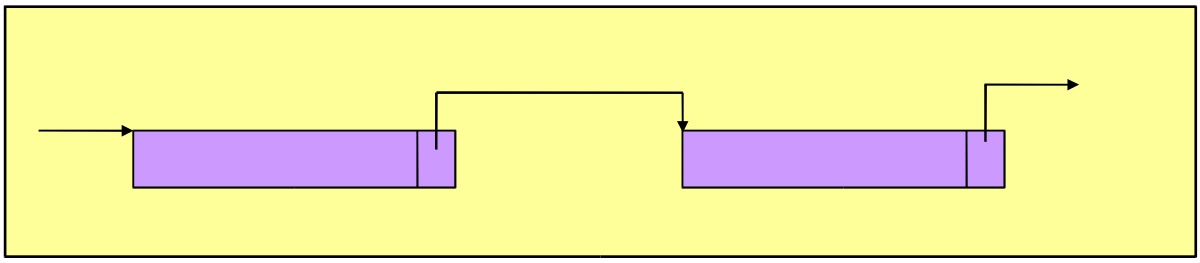
Répartition physique des fichiers en blocs

A chaque fichier correspond une liste de blocs contenant ses données. L'allocation est en générale non contigüe et les blocs sont donc répartis quasi-aléatoirement sur le disque. Les fichiers conservent l'ensemble de leurs blocs suivant deux méthodes :

- la **liste chaînée** et
- la **table d'index**.

La liste chaînée

L'ensemble des blocs d'un fichier peut être chaînée sous la forme d'une liste. Chaque bloc contiendra des données ainsi que l'adresse du bloc suivant.



Cette méthode rend l'accès aléatoire aux éléments d'un fichier particulièrement inefficace lorsqu'elle est utilisée telle quelle. En effet **pour atteindre un élément sur le bloc *n* d'un fichier, le système devra parcourir les *n-1* blocs précédents**.

Le système MS-DOS utilise des listes chaînées. Il conserve le premier bloc de chacun des fichiers dans son répertoire. Il optimise ensuite l'accès des blocs suivants en gardant leurs références dans une **Table d'Allocation de Fichiers (FAT)**.

Chaque disque dispose d'une FAT et cette dernière possède autant d'entrées qu'il y a de blocs sur le disque. Chaque entrée de FAT contient le numéro du bloc suivant.

Voici un exemple de table (FAT) :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
X	X	EOF	13	2	9	8	L	4	12	3	L	EOF	EOF	L	BE	...

Où :

« XX » -> la taille du disque,
« L » -> un bloc libre et
« BE » -> un bloc endommagé.

Le fichier commençant au bloc 6, sera constitué des blocs :

6 - 8 - 4 - 2.

Le parcours de la FAT est nettement plus rapide que la chaîne des blocs. Cependant si elle n'est pas constamment, tout entière en mémoire, elle ne permet pas d'éviter les entrées-sorties du disque.

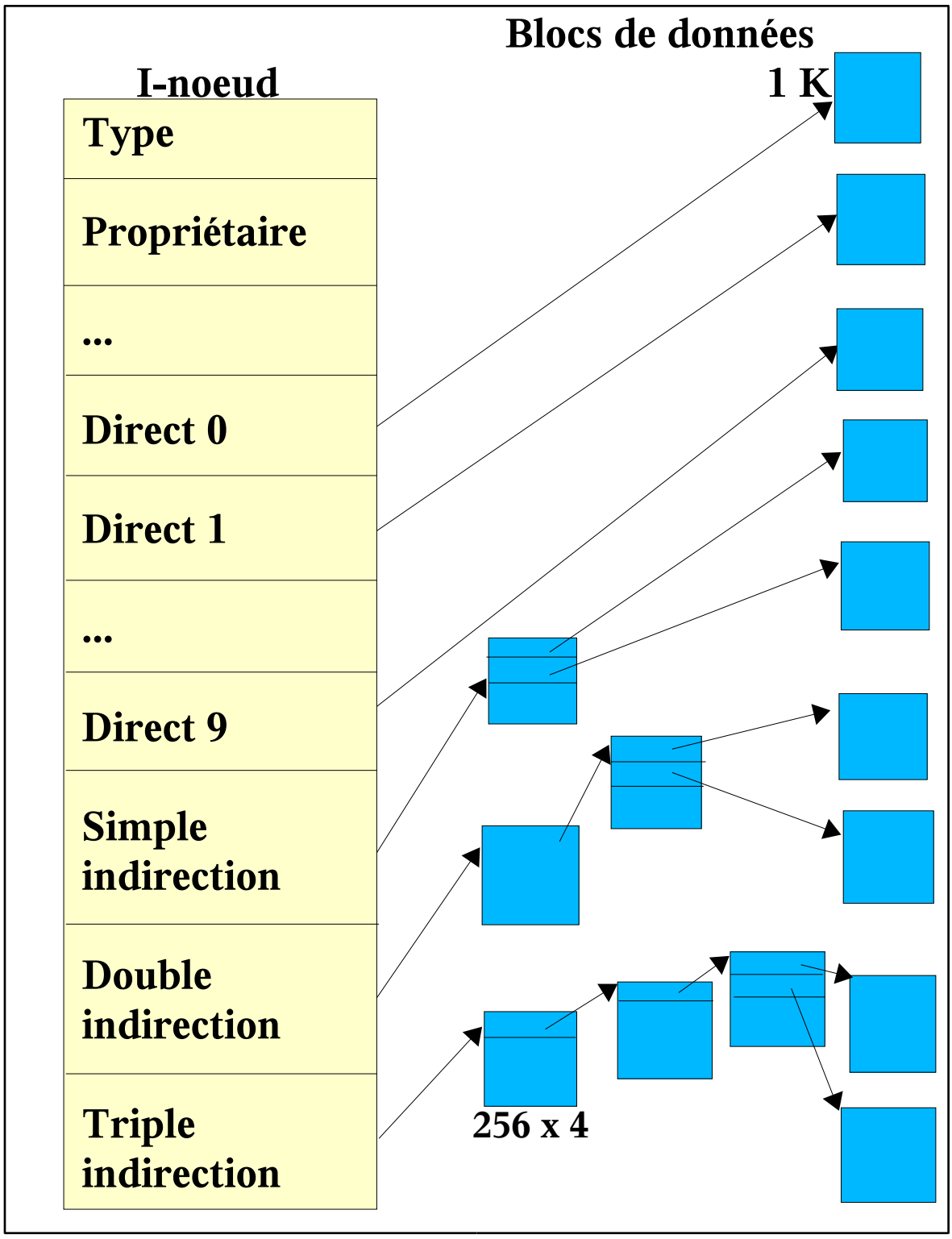
La table d'index

Tout comme pour les listes chaînées, dans le cas de l'utilisation d'une table d'index (système UNIX) le noyau attribue de l'espace aux fichiers un bloc à la fois et permet aux données d'être disséminées dans tout le système de fichiers.

Nous avons vus précédemment que l'**i-noeud possède un « champ » correspondant aux adresses sur le disque des blocs de données du fichier. Il s'agit en fait d'une table**.

Pour maintenir une petite structure d'i-noeuds qui permette néanmoins des fichiers volumineux, la table des adresses des blocs disque se compose de 13 champs. Cette table est indépendante de la taille des fichiers.

Les 10 premiers champs (direct) correspondent aux adresses des 10 premiers blocs de données sur disque. Pour les fichiers de plus de 10 blocs on a recours à des indirections. Le bloc n° 11 contient le numéro d'un bloc composé lui-même d'adresses de blocs de données. Un bloc (11) de 1024 octets pourra désigner jusqu'à 256 blocs de données, sachant que chacun d'eux est numéroté sur 4 octets. Le bloc n°12 contient une adresse à double indirection et le bloc n° 13 en contient une à triple indirection.



La taille maximale théorique d'un fichier décrit par un i-noeud est de :

$(10x1k) + (256x1k) + (256^2x1k) + (256^3x1k) > 16$ Go

Mais comme le champ taille du fichier dans un i-noeud est codé sur 32 bits, la taille maximale effective pour un fichier est de 4 Go (2^{32}).

Un processus accède aux données d'un fichier à l'aide d'un déplacement d'octet.

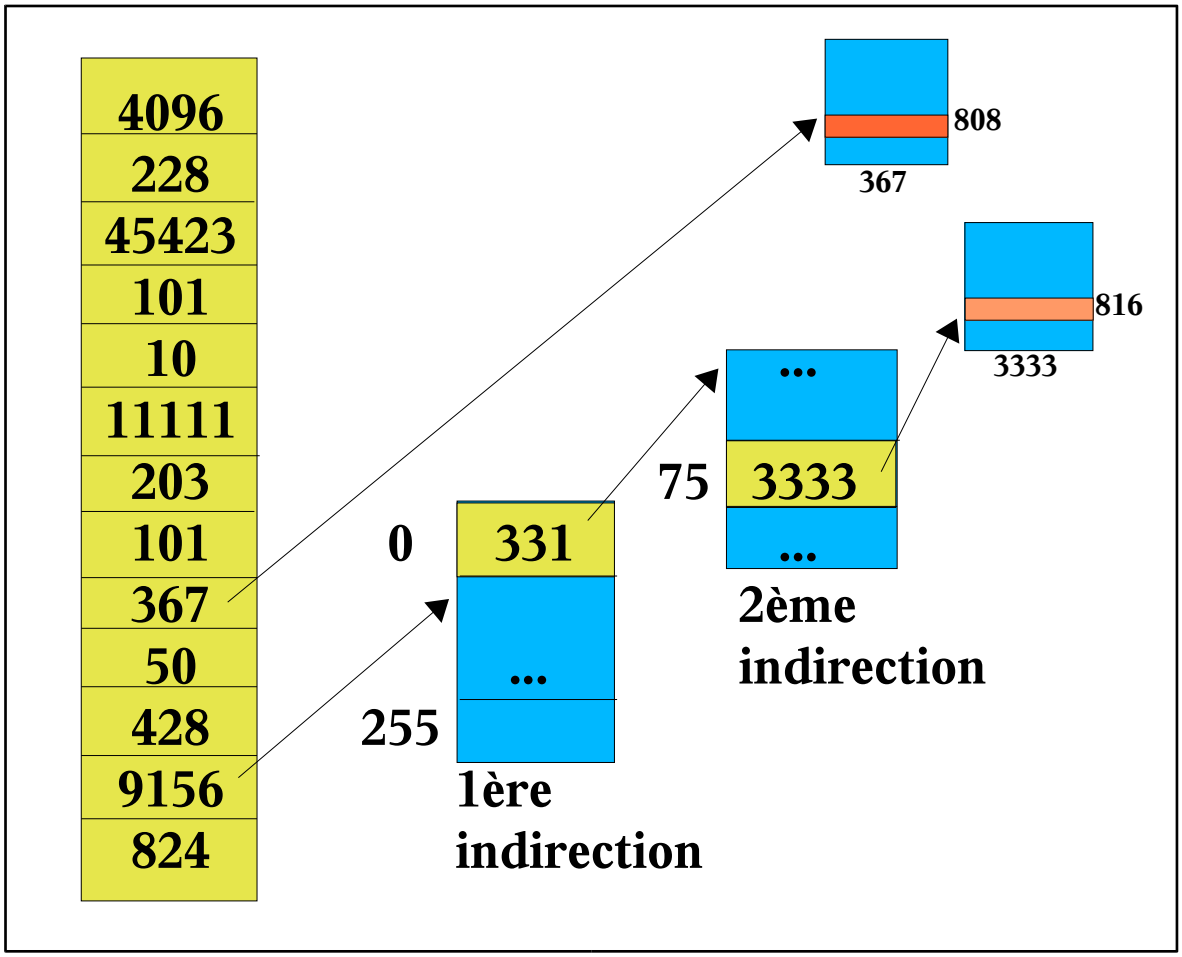
Il travaille en terme de compte d'octets et voit le fichier comme un flot d'octets débutant à l'adresse octet 0 et allant jusqu'à la taille du fichier. Le noyau convertit la vue octets d'un utilisateur en une vue blocs : le fichier commence au bloc logique 0 et continue jusqu'au numéro de bloc logique correspondant à la taille du fichier.

Le noyau accède à l'i-noeud et convertit le bloc logique du fichier en bloc disque approprié (algorithme *bmap*). Cet algorithme convertit un déplacement octets dans un fichier (vue utilisateur) en un bloc disque physique.

Pb-1 : Accéder au déplacement octet 9000 !

Pb-2 : Accéder au déplacement octet 350 000 !

Les solutions



- **Explication pb-1 : 9000 / 1024 = 8 (reste 808). L'octet « 9000 » est donc dans le bloc (de numéro 367) d'adresse Direct-8 du I-noeud. Il correspond au 808ème octet du fichier.**

- **Explication pb-2 : 350000 / 1024 = 256 + 10 (reste 77616). Le noyau doit accéder à un bloc double indirection, celui de numéro 9156 sur le schéma. Puisqu'un bloc d'indirection peut contenir 256 n° de blocs, le premier octet accédé via le bloc double indirection est l'octet n° 272384 = 256K (simple indirection) + 10K (accés directs) ; l'octet n° 350000 dans le fichier est donc l'octet n° 77616 = 350000 - 272384 du bloc double indirection. 77616 / 1024 = 75 (reste 816). Finalement, l'octet 350000 dans le fichier est l'octet 816 du bloc 3333.**

Les répertoires

Les répertoires donnent au système de fichiers sa structure arborescente et jouent un rôle important dans la conversion d'un nom de fichier en numéro d'i-noeud.

Un répertoire est un fichier dont les données sont une suite d'éléments qui comprennent chacun un numéro d'i-noeud et le nom d'un fichier contenu dans ce répertoire.

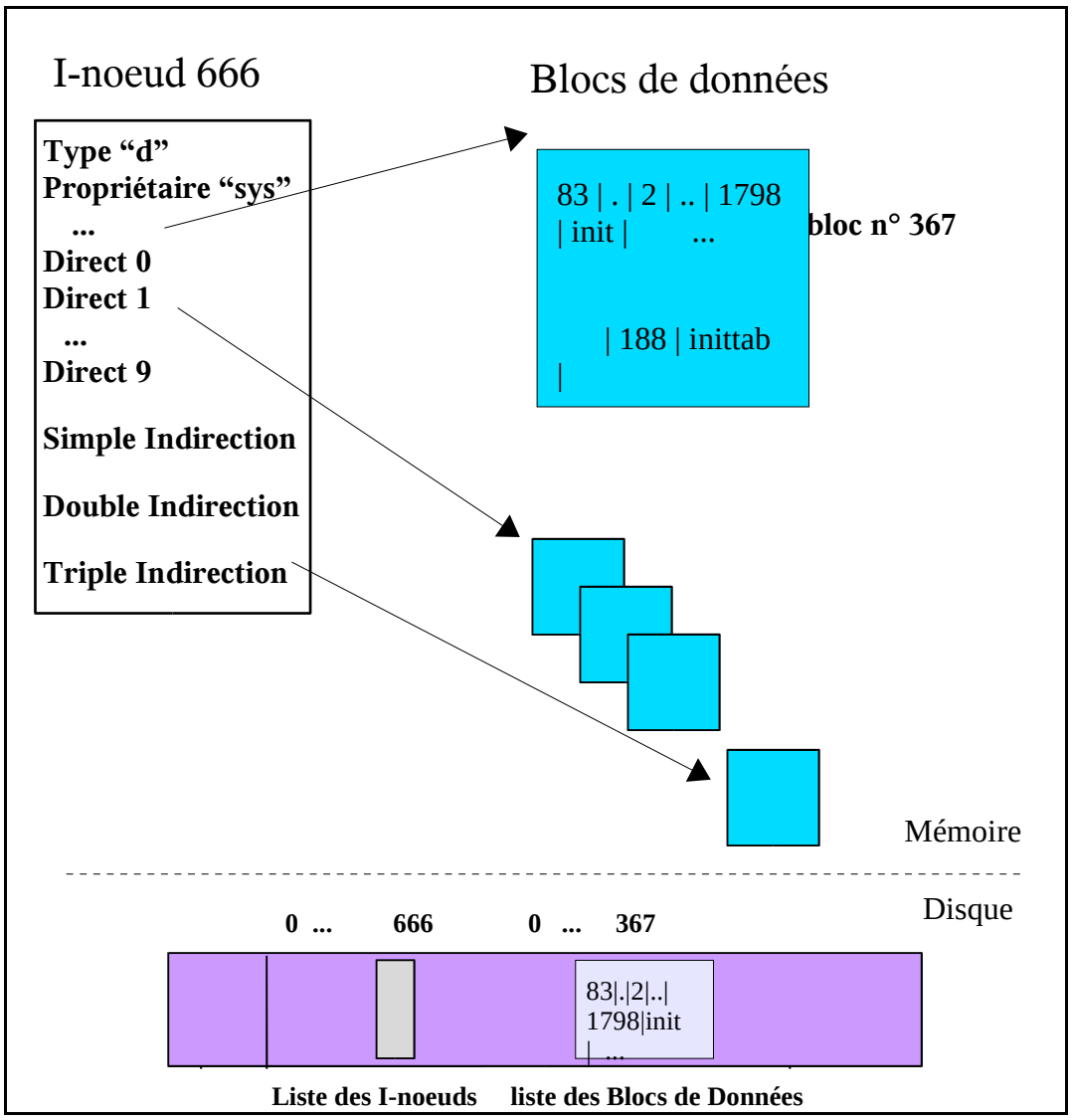
Déplacement octet dans le répertoire	Numéro d'i-noeud (2 octets)	Noms de fichiers
0	83	.
16	2	..
32	1798	init
48	1276	fsck
64	85	clri
80	1268	motd
96	1799	mount
112	88	mknod
128	2114	passwd
144	1717	umount
160	1851	checklist
176	92	fsdblb
192	84	config
208	1432	getty
224	0	crash
240	95	mkfs
256	188	inittab

UNIX Système V restreint les noms des composants à 14 caractères ; le numéro d'i-noeud étant contenu dans deux octets, la taille d'un élément d'un répertoire est de 16 octets.

C'est la structure **direct** définie dans le fichier **/usr/include/sys/dir.h** qui implémente cet enregistrement :

```
# define DIRSIZ 14
struct direct {
        ushort      d_ino;
        char        d_name [DIRSIZ];
};
```

Cette structure correspond au contenu des catalogues. En effet, quand un i-noeud référence un catalogue et non un fichier ordinaire, les blocs de données contiennent des enregistrements **direct** qui références les fichiers présents dans le répertoire. Pour chaque fichier présent on stocke son numéro d'i-noeud et son nom dans la structure **direct**.



Le noyau se réserve le droit exclusif d'écriture dans les répertoires, assurant ainsi une structure correcte.

Le noyau fait une recherche linéaire dans le fichier du répertoire associé à l'i-noeud de travail en essayant de faire coïncider la composante d'un chemin d'accès à un nom d'un élément du répertoire.

Les éléments d'un répertoire peuvent être vides, le numéro de l'i-noeud vaut 0.

Structure interne dans les versions BSD 4.[2-3]

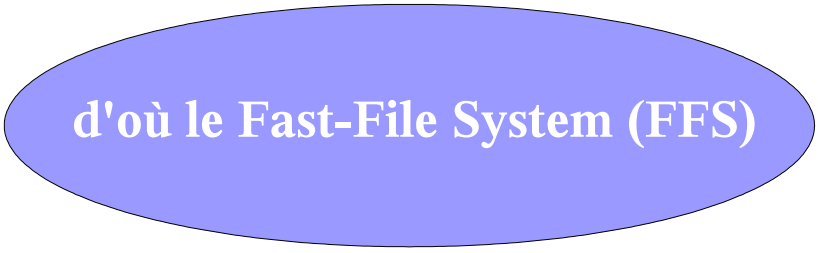
Modifier les versions System-V pour corriger leurs faiblesses :

> La liste des blocs libres, initialement ordonnée pour des accès optimaux, devient rapidement désordonnée lorsque les fichiers sont créés puis effacés.

Cela peut conduire à :

- **un positionnement du bras de lecture du disque entre chaque accès aux blocs d'un même fichier,**
- **des déplacement du bras de cylindre en cylindre (données et i-noeuds séparés).**

Même un simple « ls » peut nécessiter la consultation de blocs non consécutifs sur le disque.



Principe

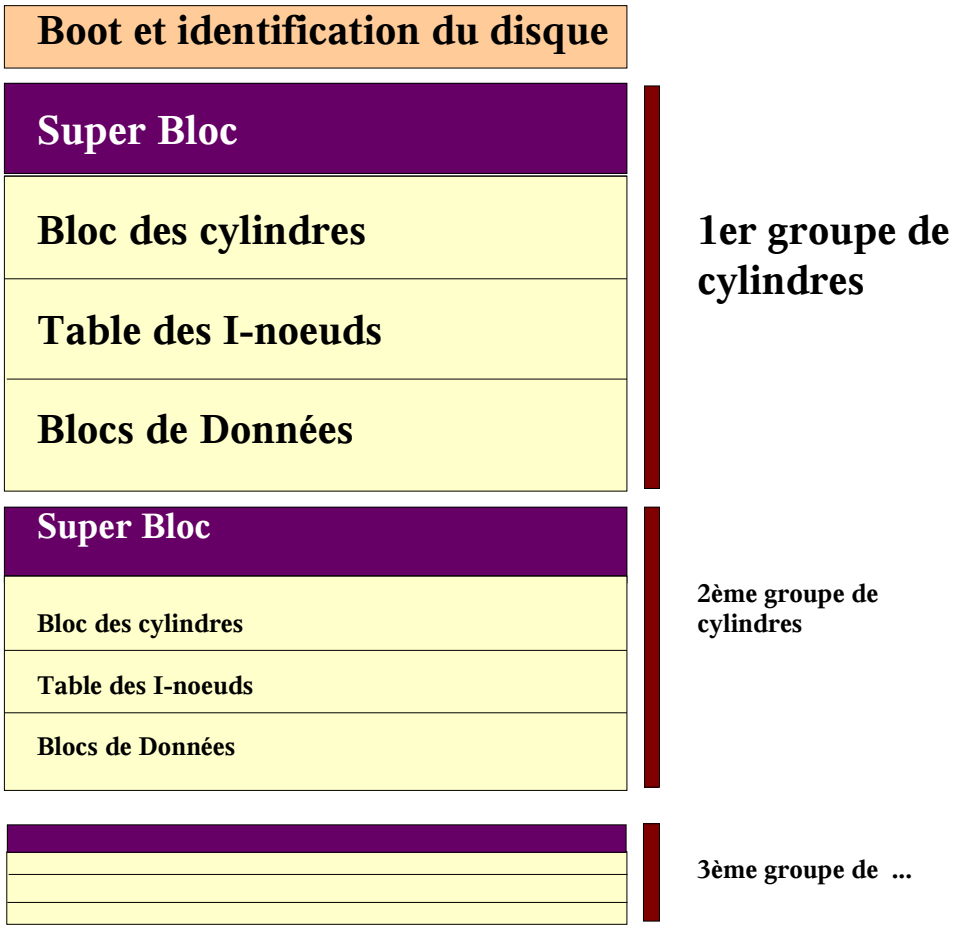
- **Duplication du « Super-Bloc »**
  => + de sûreté.

- **Des blocs plus gros (4K ou 8K)**
  => + performant.

- **Des blocs divisés en fragments (au minimum le 1/8ème de la taille d'un bloc)**
  => bonne gestion des petits fichiers.

Pour cela on introduit le « Bloc des cylindres », qui regroupe l'ensemble des paramètres dynamiques d'un groupe de cylindres particulier (bit-map des blocs libres, celle des i-noeuds libres ...).

Cela permet de ne laisser que des informations statiques (donc identiques) dans le « Super Bloc » (taille totale du SGF, taille des blocs et des fragments ...).

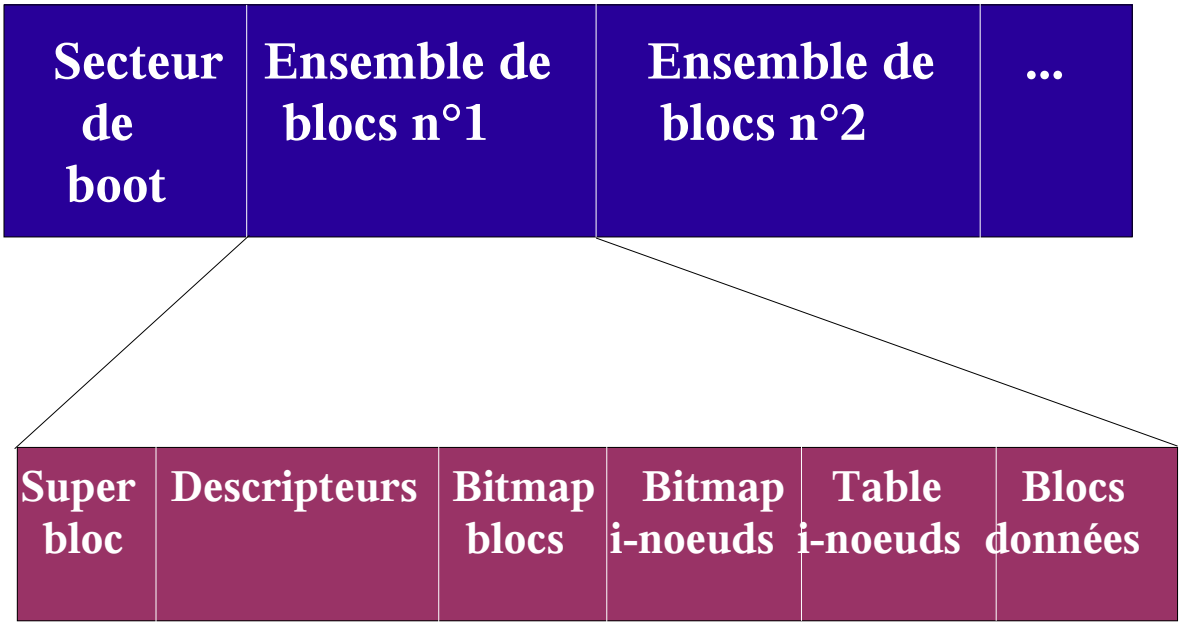


Les caractéristiques en chiffres ...

- L'augmentation de la taille des blocs => suppression de la triple indirection (lente).

- Un bloc de 8K et une adresse sur 4 octets =>
  un adressage directe jusqu'à 80K de données
  un adressage par simple indirection -> 2 Mo
  un adressage par double indirection -> 4 Go

Structure Physique de Ext2 (Linux)



- **une copie du « Super Bloc »** (informations de contrôle)
- **une table de descripteurs** :

  @ des blocs de bitmap et table des i-noeuds de tous les ensembles
- **un bloc de bitmap** : l'image des blocs de données (un bit/bloc à 1 si alloué, à 0 sinon)
- **un bloc de bitmap** : image des i-noeuds (un bit/i-noeud à 1 si alloué, à 0 sinon)
- **une table des i-noeuds** (concernés par cette ensemble)
- **des blocs de données** (fichiers et répertoires concernés)

Les deux premières zones sont dupliquées dans chaque ensemble de blocs.

La taille standard d'un ensemble : 8192 blocs de données et 2048 i-noeuds.