Implémentation du système de fichiers ext2 dans Linux

Rémy Card
E-Mail: card@masi.ibp.fr

Laboratoire MASI — Institut Blaise Pascal
Université Pierre et Marie Curie
4, place Jussieu — 75252 Paris Cedex 05

Abstract

Linux is an operating system with a Unix-like interface which runs on PC 386. Initially, Linux included only a limited Minix-compatible filesystem. This paper describes the two extended filesystems that we have designed for Linux.

An indirection layer has been added to Linux to access files. This layer, called the virtual filesystem, calls physical filesystems via function pointers. The two extended filsystems use common strategies: directory trees, i-nodes, indirect data blocks. The first extended filesystem uses a linked list to keep track of the free blocks and i-nodes. It manages directories as variable length entries. The second extended filesystem is an extension of the first one. It provides additional features and optimizations used to improve performances. It has been designed to allow the easy addition of new extensions like access control lists.

Résumé

Linux est un système d'exploitation à interface de type Unix fonctionnant sur PC 386. À l'origine, ce système ne supportait qu'un système de fichiers limité compatible avec Minix. Ce papier présente les deux systèmes de fichiers étendus que nous avons conçus pour Linux.

Une couche d'indirection a été ajoutée à Linux pour les accès aux fichiers. Cette couche, le système de fichiers virtuel, communique avec les systèmes de fichiers physiques par le biais de pointeurs sur fonctions. Les deux systèmes de fichiers étendus utilisent des stratégies communes : répertoires sous forme arborescente, i-noeuds, accès aux blocs de données par indirection. Le premier système de fichiers utilise, de plus, une liste chaînée pour garder la trace des blocs et i-noeuds libres. Il gère les répertoires sous formes d'entrées de tailles variables. Le deuxième système de fichiers est une extension du premier. Il offre des fonctionnalités supplémentaires ainsi que des optimisations permettant d'améliorer les performances. Enfin, il a été conçu de façon à inclure facilement de nouvelles extensions comme les listes de contrôle d'accès.

Introduction

Linux est un système d'exploitation offrant une compatibilité source avec Unix pour micro-ordinateurs de types PC 386. Il a été développé initialement comme une extension du système Minix[1] [Tanenbaum 1987] et ne comprenait, dans ses premières versions qu'un système de fichiers compatible avec celui de Minix.

Ce système de fichiers souffre de deux limitations importantes :

1. les adresses de blocs sont codées sur deux octets, limitant ainsi la taille maximale d'un système de fichiers à 64 méga-octets ;
2. les répertoires sont gérés comme des tableaux et chaque entrée de répertoire permet de stocker un nom de fichier de 14 caractères maximum.

Nous avons successivement développé deux nouveaux systèmes de fichiers, appelés "*Extended File System*" (ext fs) et "*Second Extended File System*" (ext2 fs) afin de lever ces limitations et d'ajouter des extensions. Ces deux systèmes de fichiers ont été inclus dans la version officielle du noyau et sont maintenant utilisables de manière transparente.

Nous détaillons tout d'abord comment les fichiers sont gérés de manière interne dans Linux. Ensuite, nous décrivons le premier système de fichiers "étendu". Puis, le second système de fichiers "étendu" est présenté. Enfin, nous concluons sur l'état actuel du système de fichiers et sur les développements futurs.

1 Gestion de fichiers dans Linux

1.1 Système de Fichiers Virtuel

Linux contient une couche d'indirection utilisée lors de la réalisation des appels système sur fichiers. Cette couche, appelée "Système de Fichiers Virtuel" (SFV), est chargée de réaliser tous les appels indépendants du format physique des fichiers et d'appeler les fonctions nécessaires lors d'un accès à la structure physique des fichiers. Ce mécanisme de couche d'indirection est fréquemment utilisé dans les systèmes compatibles Unix afin de permettre l'intégration et la coexistence de plusieurs types de systèmes de fichiers [Kleiman 1986, Seltzer *et al.* 1993].

La figure 1 représente la structure des modules de gestion de fichiers sous Linux où plusieurs systèmes de fichiers différents (sur la figure, les systèmes de fichiers compatibles Minix et MS/DOS ainsi que les deux systèmes de fichiers "étendus") peuvent coexister et être utilisés de manière transparente. Les appels système concernant les fichiers sont dirigés vers le SFV qui les transmet au système de fichiers physique concerné. Ce module implémente les opérations physiques sur les fichiers en utilisant les fonctions d'entrées-sorties fournis par la couche de gestion des tampons mémoire (*Buffer Cache*). Cette dernière couche utilise les sous-programmes de lecture et d'écriture de blocs offerts par le pilote du périphérique sur lequel se trouve le système de fichiers.

Linux constitue un noyau monolithique dans lequel toutes les fonctions de services du système sont rassemblées. Les différents modules partagent le même espace d'adressage. Le SFV peut donc accéder

[1] Linux signifiait à l'origine Linus's version of Minix, du prénom de son auteur.

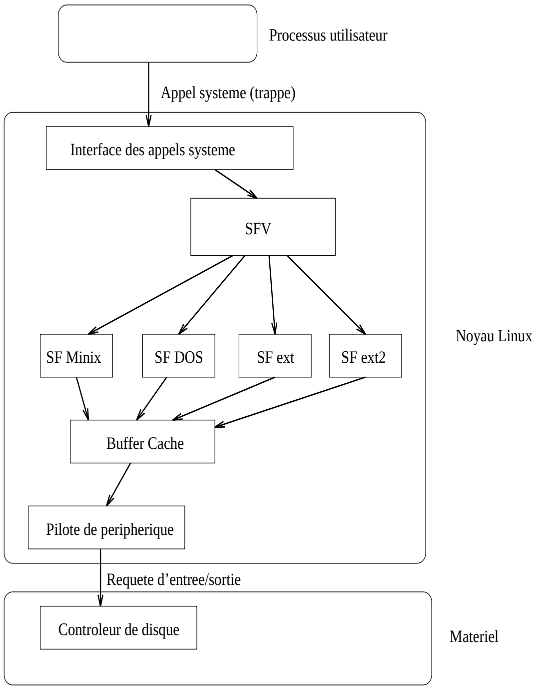

Figure 1: Structure des couches de gestion de fichiers dans Linux

directement aux données et aux sous-programmes des autres modules de gestion de fichiers.

Afin de permettre la coexistence et l'appel transparent à plusieurs systèmes de fichiers différents, le SFV définit une interface que tout système de fichiers doit respecter pour être inclus dans Linux. Cette interface, que nous avons contribué à définir lors de l'inclusion de notre premier système de fichiers[2], est composée d'un ensemble d'opérations qu'un système de fichiers doit offrir. Ces opérations peuvent être associées à trois types d'objets :

- le système de fichiers lui-même ;
- un i-nœud, i.e. un descripteur de fichier sur disque ;
- un descripteur de fichier ouvert.

1.2 Description d'un système de fichiers

Le SFV a juste besoin de connaître les systèmes de fichiers supportés par le noyau. Pour cela, une table définit, pour chaque système de fichiers inclus dans Linux :

- une chaîne de caractêres définissant son type ;

[2] Avant cette inclusion, Linux ne supportait que le système de fichiers Minix.

- un pointeur sur la fonction appelée lorsqu'un système de fichiers de ce type est monté. Cette fonction est chargée d'initialiser les données en mémoire correspondant au système de fichiers et d'indiquer au SFV l'ensemble des opérations disponibles ;
- un booléen indiquant si le système de fichiers nécessite une partition disque ou non.

La fonction appelée lors du montage du système dans l'arborescence des répertoires est chargée de lire la description du système de fichiers physique depuis le disque, d'initialiser ses données internes et de renvoyer au SFV un descripteur de système de fichiers monté.

Un descripteur de système de fichiers contient :

- des informations communes à tous les systèmes de fichiers physiques :
  - le périphérique correspondant au système de fichiers ;
  - la taille des blocs logiques supportés par le système de fichiers ;
  - les paramètres de montage ;
- la liste des opérations définies sur le système de fichiers. Ces opérations sont rendus accessibles au SFV en fournissant à ce dernier des pointeurs sur les fonctions suivantes :
  - **`read_inode`** : lecture en mémoire d'un i-nœud ;
  - **`notify_change`** : mise à jour du contenu d'un i-nœud ;
  - **`write_inode`** : ré-écriture sur disque du contenu d'un i-nœud ;
  - **`put_inode`** : désallocation d'un descripteur de fichier en mémoire ;
  - **`put_super`** : désallocation du descripteur de système de fichiers, appelée lorsqu'il est démonté ;
  - **`write_super`** : ré-écriture sur disque des copies mémoire des données ;
  - **`statfs`** : statistiques d'utilisation du système de fichiers ;
  - **`remount`** : opérations à réaliser lorsqu'un système de fichiers est "remonté" ;
- les données spécifiques au système de fichiers qu'il est nécessaire de maintenir en mémoire pendant qu'il est monté.

Une fois qu'un système de fichiers est monté, le SFV connait ses opérations de base grâce au descripteur renvoyé par la fonction d'initialisation. Il dialogue ensuite avec le module de gestion du système de fichiers physique en utilisant deux types de descripteurs :

- les i-nœuds (*inodes*) représentant les fichiers accédés par des opérations ponctuelles ne nécessitant pas obligatoirement une ouverture préalable ;
- les fichiers (*files*) représentant les fichiers ayant été ouverts par un appel à la fonction `open`. À chaque fichier ouvert correspond un i-nœud.

1.3 Descripteur d'i-nœuds

Le descripteur d'un i-nœud contient les informations suivantes :

- des informations communes à tous les systèmes de fichiers :
  - le périphérique sur lequel est stocké le fichier ;
  - le numéro de fichier dans le système de fichiers ;
  - le type et les droits d'accès associés au fichier ;
  - le nombre de noms associés au fichier ;
  - l'identificateur de l'utilisateur propriétaire du fichier ;
  - la taille du fichier en octets ;
  - les dates de création, de modification et de dernier accès au fichier ;
  - la taille et le nombre des blocs logiques constituant le fichier ;
- des informations utilisés pour la gestion de l'i-nœud en mémoire :
  - indicateur de modification ;
  - pointeurs sur i-nœuds précédent et suivant ;
  - indicateur de verrouillage en mémoire ;
- la description des opérations possibles sur le fichier. Chaque opération est rendue accessible par un pointeur sur la fonction qui l'implémente dans le module de gestion du système de fichiers physique. Ces opérations sont les suivantes :
  - `create` : création d'un fichier ;
  - `lookup` : recherche du nom d'un fichier dans un répertoire ;
  - `link` : création d'un lien sur un fichier ;
  - `unlink` : suppression d'un lien sur un fichier avec suppression du fichier lui-même lorsque le dernier lien est détruit ;
  - `symlink` : création d'un lien symbolique ;
  - `mkdir` : création d'un répertoire ;
  - `rmdir` : suppression d'un répertoire ;
  - `mknod` : création d'un fichier spécial ;
  - `rename` : changement du nom d'un fichier ;
  - `read_link` : lecture du nom d'un fichier pointé par un lien symbolique ;
  - `follow_link` : renvoi de l'i-nœud d'un fichier pointé par un lien symbolique ;
  - `bmap` : renvoi du numéro de bloc physique correspondant à un bloc logique contenu dans un fichier ;
  - `truncate` : modification de la taille d'un fichier avec désallocation de blocs si nécessaire ;
  - `permission` : vérification de droits d'accès sur un fichier ;
- les informations liées au système de fichiers physique.

1.4 Descripteur de fichier ouvert

Un descripteur de fichiers ouvert contient les informations suivantes :

- des informations communes à tous les systèmes de fichiers :
  - le mode d'ouverture ;
  - la position courante dans le fichier ;
  - un pointeur sur le descripteur d'i-nœud associé ;
- des pointeurs sur fonctions représentant les opérations disponibles sur le fichier :
  - `lseek` : positionnement dans un fichier ;
  - `read` : lecture depuis un fichier ;
  - `write` : écriture dans un fichier ;
  - `readdir` : lecture de répertoire ;
  - `select` : attente de données à lire dans un fichier ;
  - `ioctl` : opération de contrôle ;
  - `mmap` : initialisation du fichier pour rendre son contenu accessible en mémoire virtuelle ;
  - `open` : ouverture d'un fichier ;
  - `release` : fonction appelée lorsque le fichier est fermé par tous les processus l'ayant précédemment ouvert ;
  - `fsync` : ré-écriture de tous les blocs modifiés du fichier sur disque.

2 Représentation des fichiers sur disque

Tous les systèmes de fichiers inclus dans Linux utilisent des méthodes communes de stockage des fichiers. Nous détaillons ici ces points communs avant d'exposer les particularités des systèmes de fichiers ext et ext2.

Les répertoires dans les systèmes de fichiers respectent une structure arborescente. Un répertoire est un catalogue de noms de fichiers et de répertoires. Un fichier est caractérisé par un nom unique constitué des répertoires à parcourir depuis la racine de l'arborescence pour accéder au fichier.

Afin de maintenir les informations liées aux fichiers, les systèmes de fichiers utilisent un descripteur de fichier. Ce descripteur, appelé i-nœud, contient les informations caractérisant le fichier. Les trois systèmes de fichiers permettent d'allouer des blocs non contigus aux fichiers. Chaque i-nœud contient des pointeurs sur les blocs alloués aux fichiers. Comme dans Unix [Bach 1986], ces pointeurs peuvent être :

- des pointeurs directs contenant le numéro d'un bloc de données ;
- un pointeur indirect contenant le numéro d'un bloc contenant lui-même les numéros des blocs de données ;
- un pointeur doublement indirect contenant le numéro d'un bloc contenant lui-même des numéros de blocs qui contiennent les numéros des blocs de données ;

- un pointeur triplement indirect (seulement dans le cas des deux systèmes de fichiers “étendus”) contenant le numéro d'un blocs qui contient des numéros de blocs contenant eux-mêmes des numéros de blocs qui contiennent les numéros des blocs de données.

La figure 2 schématise cette indexation des blocs alloués en se limitant à deux niveaux d'indirection.

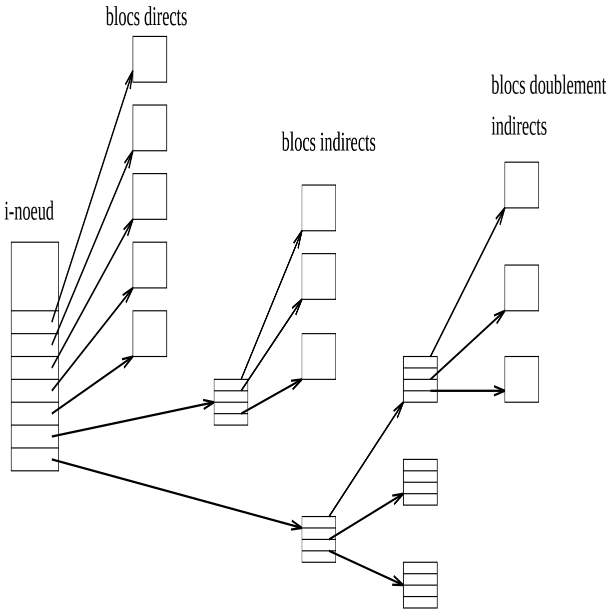

Figure 2: Indexation des blocs dans les systèmes de fichiers

3 Le Premier Système de Fichiers Étendu

Nous avons développé le système de fichiers “étendu” afin de supprimer les deux limitations les plus génantes du système de fichiers Minix (noms de fichiers sur 14 caractères, 64 méga-octets de taille maximale pour les systèmes de fichiers).

Le premier système de fichiers “étendu” utilise de nombreuses stratégies du système de fichiers Minix dont il est issu. Il inclut, de plus, certaines améliorations que nous détaillons dans les sections suivantes.

3.1 Gestion des blocs libres

La première innovation fondamentale est la façon dont il gère les blocs et i-nœuds libres. Alors que le système de fichiers Minix gère un vecteur de bits pour maintenir la liste des blocs et des i-nœuds disponibles, nous avons choisi d'utiliser des listes chaînées : le super bloc contient le numéro du premier bloc disponible qui contient lui-même les numéros de 254 autres blocs libres et le numéro du bloc suivant dans la liste. De la même façon, un i-nœud disponible contient les numéros de 14 autres i-nœuds non alloués et le numéro de l'i-nœud suivant dans la liste.

La figure 3 représente la liste chaînée de blocs libres.

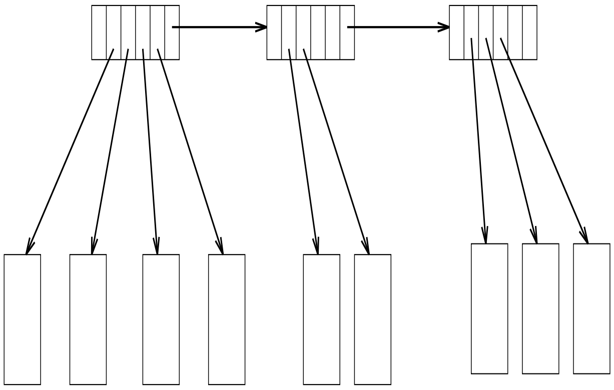

Figure 3: Liste chaînée des blocs libres dans le système de fichiers “étendu”

Lors d'une allocation de bloc (resp. i-nœud), le premier bloc (resp. i-nœud) de la liste est utilisé. Lorsqu'un bloc (resp. i-nœud) est désalloué, il est ajouté en tête de liste. Cette gestion de liste chaînée permet d'économiser la place utilisée par les vecteurs de bits dans le système de fichiers Minix (1 kilo-octets par 8192 blocs ou i-nœuds).

3.2 Structure des répertoires

La deuxième innovation importante du système de fichiers concerne la structure des répertoires : dans ce système de fichiers, les entrées de répertoire sont de taille variable afin de permettre des noms de fichiers de taille plus importante en utilisant l'espace disque de manière optimale. Chaque entrée de répertoire contient :

- un numéro d'i-nœud codé sur 4 octets ;
- la longueur de l'entrée de répertoire, codée sur 2 octets ;
- la taille du nom de fichier contenu dans cette entrée, codée sur 2 octets ;
- le nom de fichier lui-même utilisant un nombre d'octets multiple de 4.

La structure d'un répertoire contenant les fichiers `f1`, `nom_plus_long` et `f2` est représentée sur la figure 4.

i3	12	02	`f1`	i4	24	13	`nom_plus_long`	i5	12	02	`f2`

Figure 4: Structure d'un répertoire dans le système de fichiers “étendu”

Un répertoire peut être considéré comme une liste chaînée d'entrées : à partir d'une entrée, il est possible d'accéder à la suivante en ajoutant la taille de l'entrée courante à la position actuelle dans le répertoire.

3.3 Structure physique du système de fichiers

Un système de fichiers “étendu” est composé :

- d’un bloc de boot contenant le code lancé lors du démarrage du système ;
- d’un super bloc contenant les caractéristiques du système de fichiers :
  - nombre de blocs ;
  - nombre d’i-nœuds ;
  - adresse du premier bloc libre ;
  - nombre de blocs libres ;
  - numéro du premier i-nœud libre ;
  - nombre d’i-nœuds libres ;
  - taille des blocs logiques ;
  - taille maximale d’un fichier ;
  - signature ;
- d’une table des i-nœuds contenant les renseignements sur les fichiers existants. Chaque i-nœud caractérisant un fichier contient :
  - le type et les droits d’accès associés au fichier ;
  - la taille du fichier en octets ;
  - la date de dernière modification ;
  - l’identificateur de l’utilisateur propriétaire du fichier ;
  - le nombre de noms (i.e. liens) associés au fichier ;
  - 12 adresses de blocs (9 adresses de blocs directs, l’adresse d’un bloc indirect, l’adresse d’un bloc doublement indirect et l’adresse d’un bloc triplement indirect) ;
- des blocs de données utilisés pour stocker le contenu des fichiers et des répertoires.

La structure du système de fichiers sur disque est représentée sur la figure 5.

Secteur de boot	Super Bloc	Table des i-nœuds	Blocs de données

Figure 5: Structure physique du système de fichiers “étendu”

4 Le Second Système de Fichiers Étendu

4.1 Objectifs

Nous avons conçu et développé le second système de fichiers "étendu" afin de lever les limitations héritées du système de fichiers Minix ou introduites dans la première version.

Ces limitations sont les suivantes :

- De même que le système de fichiers Minix, le système de fichiers "étendu" ne conserve que la date de dernière modification d'un fichier sans sauvegarder la date de création ni la date de dernier accès.
- Le nombre de blocs alloués à chaque fichier n'est pas sauvegardé dans les i-nœuds.
- L'utilisation de listes chaînées de blocs et d'i-nœuds disponibles permet d'économiser de la place disque mais entraine une dégradation des performances. En effet, au fur et à mesure que des fichiers sont créés et supprimés, des blocs sont alloués et désalloués dans un ordre quelconque. La liste des blocs libres cesse vite d'être triée et le système alloue des blocs répartis sur tout le système de fichiers. L'accès à un fichier constitué de blocs distants provoque des déplacements importants des têtes de lecture-écriture du disque qui augmentent les temps d'entrées-sorties.

De nouvelles fonctionnalités ont également été ajoutées au système de fichiers :

- la sauvegarde temporaire de fichiers supprimés permettant d'implémenter un programme utilitaire de restauration des fichiers détruits ;
- la réservation d'un certain pourcentage de blocs pour le "super-utilisateur" ;
- la gestion de versions pour les i-nœuds ;
- la duplication des informations cruciales du système de fichiers permettant leur restauration si elles sont endommagées ;
- la gestion de listes de contrôle d'accès permettant une protection des fichiers plus fine que le mécanisme traditionnellement utilisé sous Unix ;
- la gestion de blocs de taille variable : alors que les autres systèmes de fichiers ne supportent que des blocs de 1 kilo-octets, le second système de fichiers "étendu" supporte des blocs de plus grande taille ;
- un indicateur de validité du système de fichiers utilisé par `fsck` pour ne pas vérifier la cohérence du système de fichiers si ce n'est pas nécessaire.

Ce système de fichiers offre une structure de répertoires identique à celle de son prédécesseur permettant ainsi de stocker des noms de fichiers de taille variable en optimisant la place utilisée dans chaque entrée de répertoire.

4.2 Structure physique du système de fichiers

Un système de fichiers est composé de plusieurs ensembles logiques de blocs comme représenté sur la figure 6.

La décomposition en ensembles, qui peuvent être considérés comme des “sous systèmes de fichiers”, permet d'offrir :

- un plus grand degré de résistance aux pannes, puisque les informations cruciales sont dupliqués dans plusieurs ensembles ;
- des stratégies d'allocation optimisées présentées plus loin.

Secteur de boot	Ensemble de blocs 1	Ensemble de blocs 2	...	Ensemble de blocs N

Figure 6: Structure physique du second système de fichiers “étendu”

4.3 Structure physique d'un ensemble de blocs

Chacun des ensembles logiques contenus dans le système de fichiers contient :

- une copie du super bloc contenant :
  - le nombre de blocs ;
  - le nombre de blocs réservés au “super-utilisateur” ;
  - le nombre d'i-nœuds ;
  - le nombre de blocs libres ;
  - le nombre d'i-nœuds libres ;
  - les tailles d'un bloc logique et d'un fragment logique ;
  - le nombre de blocs par ensemble ;
  - le nombre de fragments par ensemble ;
  - le nombre d'i-nœuds par ensemble ;
  - la date de dernier montage du système de fichiers ;
  - la date de dernière écriture du super bloc ;
  - la signature ;
  - un indicateur de validité mis à 0 lorsque le système de fichiers est monté et remis à 1 lorsqu'il est démonté ;
- une copie des descripteurs des ensembles composant le système de fichiers. Chaque descripteur contient :
  - l'adresse du bloc contenant le vecteur de bits indiquant les blocs alloués ;
  - l'adresse du bloc contenant le vecteur de bits indiquant les i-nœuds alloués ;
  - l'adresse du premier bloc contenant la partie de la table des i-nœuds ;
  - le nombre de blocs libres dans le groupe ;

  - le nombre d'i-nœuds libres dans le groupe ;
  - le nombre de répertoires contenus dans le groupe ;

- un vecteur de bits indiquant les blocs alloués et disponibles ;
- un vecteur de bits indiquant les i-nœuds alloués et disponibles ;
- une partie de la table des i-nœuds. Chaque i-nœud contient :
  - le type et les droits d'accès associés au fichier ;
  - la taille du fichier en octets ;
  - les dates de création, de dernier accès et de dernière modification ;
  - la date de suppression du fichier ;
  - l'identificateur de l'utilisateur propriétaire du fichier ;
  - le nombre de noms (i.e. liens) associés au fichier ;
  - le numéro de version du fichier ;
  - le nombre de blocs alloués au fichier ;
  - des pointeurs sur les listes de droits d'accès associés au fichier ;
  - 15 adresses de blocs (12 adresses de blocs directs, l'adresse d'un bloc indirect, l'adresse d'un bloc doublement indirect et l'adresse d'un bloc triplement indirect) ;
  - divers champs réservés pour des extensions futures ;
- des blocs de données utilisés pour stocker le contenu des fichiers et des répertoires.

La structure d'un ensemble de blocs est représentée sur la figure 7.

Super Bloc	Descrip-teurs	Bitmap blocs	Bitmap i-nœuds	Table des i-nœuds	Blocs de données

Figure 7: Structure d'un ensemble de blocs dans le second système de fichiers "étendu"

4.4 Allocations optimisées

Le second système de fichiers utilise de nombreuses optimisations lors d'allocation de blocs et d'i-nœuds :

- Les i-nœuds de fichiers sont alloués, autant que possible, dans le même ensemble que l'i-nœud de leur répertoire père. De la sorte, lors de l'analyse de noms de fichiers, il est possible, dans de nombreux cas de limiter la recherche à un seul ensemble.
- Les i-nœuds alloués à des répertoires sont répartis sur tous les ensembles composant le système de fichiers de manière à maintenir une moyenne de répertoires équitable dans les ensembles de blocs.

- Les blocs d'un fichier sont alloués, autant que possible, dans le même ensemble que l'i-nœud correspondant. De la sorte, lors de l'accès au contenu d'un fichier, la lecture des blocs après la lecture de l'i-nœud donne lieu à un déplacement minimal de la tête de lecture-écriture. De plus, comme la majorité des blocs constituant un fichier sont localisés dans le même groupe, la lecture du fichier dans son intégralité donne lieu également à des déplacements minimaux.
- Lorsque des écritures de taille importante sont effectuées sur un fichier et que plusieurs blocs sont alloués, le système de fichiers essaie d'allouer des blocs consécutifs afin de permettre des performances optimales lors des lectures ultérieures de ces blocs.

Certaines de ces optimisations se rapprochent des stratégies implémentées dans le système de fichiers présent dans la version BSD d'Unix [McKusick *et al.* 1984] mais nous n'avons pas jugé judicieux de prendre en compte les caractéristiques physiques des disques sous-jacents comme le fait ce système de fichiers. La prise en compte de ces caractéristiques entrainerait des algorithmes beaucoup plus complexes. De plus, de nombreux disques modernes présentent au système d'exploitation des caractéristiques "logiques" différentes des caractéristiques physiques réelles. Il est donc vain d'essayer d'optimiser l'allocation des blocs en tenant compte de ces caractéristiques puisque de telles optimisations ne correspondraient pas à la répartition physique des blocs sur le disque.

4.5 Utilisation de blocs de tailles variables

Une autre innovation importante de ce système de fichiers consiste à permettre l'utilisation de blocs de taille variable. Alors que les autres systèmes de fichiers ne gèrent que des blocs de un kilo-octets, le second système de fichiers "étendu" permet de spécifier une taille de blocs logique plus importante. L'accroissement de la taille des blocs permet d'obtenir de meilleures performances lors des entrées-sorties. En effet, si on multiplie la taille des blocs par un facteur N, N fois moins de requêtes d'entrées-sorties sont nécessaires pour accéder à la même quantité de données, réduisant ainsi le temps de déplacement des têtes de lecture-écriture. De plus, en multipliant la taille des blocs logiques, de nombreux fichiers peuvent être exprimés sans bloc d'indirection, ce qui réduit ainsi le nombre d'entrées-sorties nécessaires.

L'accroissement de la taille des blocs logiques permet donc d'obtenir une amélioration importante des performances mais elle entraine également une mauvaise utilisation de l'espace disque due à la fragmentation interne. En effet, statistiquement la moitié seulement du dernier bloc alloué à un fichier est utilisée. Plus la taille des blocs est importante, plus la perte d'espace disque est considérable. Des études [Leffler *et al.* 1989] ont montré que l'utilisation de blocs de 4 kilo-octets provoque une perte d'environ 45% de l'espace disque.

Afin de limiter la fragmentation interne dans les fichiers, nous avons minimisé la taille du dernier bloc alloué à un fichier. En effet, comme la perte d'espace disque est proportionnelle à la taille du dernier bloc, la réduction de cette taille minimise cette perte.

Afin de réduire la taille du dernier bloc, nous avons considéré des unités d'allocation plus petites qu'un bloc. Dans le second système de fichiers "étendu", un fichier est composé de plusieurs blocs complets et de quelques unités d'allocation consécutives. Considérons un fichier contenant 14334 octets situé sur un système de fichiers composé de blocs de 4 kilo-octets et d'unités d'allocation de 1 kilo-octets. Un tel fichier sera constitué de 3 blocs complets, soit 12288 octets, et de deux unités d'allocation consécutives, soit 2048 octets. 14336 octets sont donc alloués physiquement au fichier alors que 16384 octets seraient nécessaires si le système de fichiers ne gérait que des blocs de 4 kilo-octets. Il est ainsi possible de réduire la place disque inutilisée de 45% à 11% environ [McKusick *et al.* 1984, Leffler *et al.* 1989].

Afin de gérer des unités d'allocation plus petites que les blocs, le vecteur de bits présent dans chaque ensemble contient un bit par unité d'allocation. Lors de l'allocation d'espace disque, le système recherche autant d'unités d'allocation consécutives qu'il est nécessaire.

5 Conclusion

Le système de fichiers ext2 a été, depuis quelques mois, inclus dans le noyau "officiel" de Linux. Il est le système de fichiers recommandé pour installer les distributions binaires de Linux.

Grâce à l'utilisation de ce système de fichiers par une grande partie de la communauté Linux et grâce aux comptes-rendus d'utilisation[3] il a été possible de stabiliser le code et de l'améliorer jusqu'à l'état actuel.

Le système de fichiers ext2 repose sur certaines idées qui ne sont pas toutes nouvelles[4] mais est basé sur une nouvelle implémentation relativement simple et évolutive.

Bien que le système de fichiers ext2 soit actuellement stable et robuste, son développement continue. Certaines fonctionnalités prévues, comme les listes de contrôle d'accès selon les sémantiques Posix[IEEE 1992] ou les fragments, sont en cours d'implémentation et de nouvelles fonctionnalités, comme la compression transparente de fichiers, vont être ajoutées en tirant parti des possibilités d'extension incluses dans le système de fichiers lors de sa conception.

De plus, ce système de fichiers développé sous Linux est un composant de base d'un nouveau système d'exploitation, Masix[Card 1993, Card *et al.* 1993], basé sur le micro-noyau Mach[Accetta *et al.* 1986].

Références

[Accetta *et al.* 1986] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, et M. Young. Mach: A New Kernel Foundation For UNIX Development. In *Proceedings of the USENIX 1986 Summer Conference*, June 1986.

[Bach 1986] M. Bach. *The Design of the UNIX Operating System.* Prentice Hall, 1986.

[Card *et al.* 1993] R. Card, E. Commelin, S. Dayras, et F. Mével. The MASIX Multi-Server Operating System. In *OSF Workshop on Microkernel Technology for Distributed Systems*, June 1993.

[Card 1993] R. Card. *MASIX - Un Système d'Exploitation Multi-Environnements utilisant le micro-noyau MACH - Conception et Réalisation.* PhD thesis, Laboratoire MASI, Université Pierre et Marie Curie, May 1993.

[IEEE 1992] *SECURITY INTERFACE for the Portable Operating System Interface for Computer Environments - Draft 13.* Institute of Electrical and Electronics Engineers, Inc, 1992.

[Kleiman 1986] S. Kleiman. Vnodes: An Architecture for Multiple File System Types in Sun UNIX. In *Proceedings of the Summer USENIX Conference*, pages 260–269, June 1986.

[Leffler *et al.* 1989] S. Leffler, M. McKusick, M. Karels, et J. Quaterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison Wesley, 1989.

[3] Concernant principalement les "bugs" !

[4] Par exemple, certaines stratégies contenues dans BSD FFS ont inspiré le développement.

[McKusick *et al.* 1984] M. McKusick, W. Joy, S. Leffler, et R. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[Seltzer *et al.* 1993] M. Seltzer, K. Bostic, M. McKusick, et C. Staelin. An Implementation of a Log-Structured File System for UNIX. In *Proceedings of the USENIX Winter Conference*, January 1993.

[Tanenbaum 1987] A. Tanenbaum. *Operating Systems: Design and Implementation.* Prentice Hall, 1987.