HE2B ESI

# Le système de fichiers ext4

Costil Iuliana

Mai 2023

# Contents

# Chapter 1
# Introduction

Ext4 (quatrième système de fichiers étendu) est un système de fichiers journalisé pour Linux, développé pour succéder à ext3. Ext4 intègre amélioration de l'évolutivité et de la fiabilité pour la prise en charge de systèmes de fichiers volumineux (64 bits) en accord avec l'augmentation des capacités de disque et l'état de l'art exigences de fonctionnalité.

## 1.1 L'évolution des systèmes de fichiers ext

1. ext : le premier système de fichiers ext a été introduit en 1992 dans le cadre du noyau Linux. Il prenait en charge les noms de fichiers jusqu'à 255 caractères et avait une taille de fichier maximale de 2 Go. Il a également introduit des fonctionnalités telles que la prise en charge des liens symboliques et des autorisations de fichiers.

2. ext2 : En 1993, ext2 a été publié, ajoutant la prise en charge de fichiers de plus grande taille jusqu'à 2 To et améliorant les performances grâce à des fonctionnalités telles que les descripteurs de groupes de blocs, qui ont amélioré les temps d'accès au disque. Il a aussi introduit la journalisation, qui a permis une récupération plus rapide en cas de panne du système ou de panne de courant.

3. ext3 : en 2001, ext3 a été publié en tant qu'extension d'ext2, ajoutant la prise en charge de la journalisation par défaut. Cela a rendu le système de fichiers plus fiable et réduit le risque de perte de données. Il a de plus introduit la possibilité de convertir un système de fichiers ext2 en ext3 sans perte de données.

4. ext4 : en 2008, ext4 a été publié avec des améliorations significatives en termes de performances et d'évolutivité. Il a ajouté la prise en charge de systèmes de fichiers plus volumineux. Il a également introduit des fonctionnalités telles que l'allocation différée, qui a amélioré les performances d'écriture, et les étendues, qui ont amélioré l'efficacité du système de fichiers en réduisant la fragmentation.

## 1.2 Principales améliorations d'ext4 par rapport à ses prédécesseurs

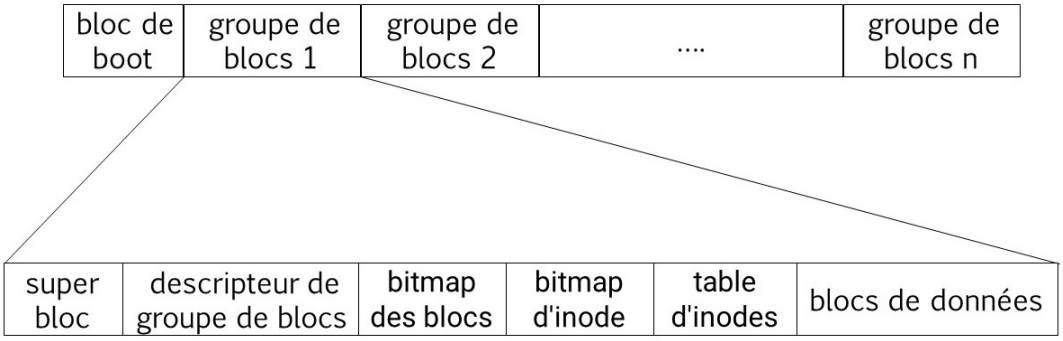
Ext4 a été introduit pour répondre à certaines des limitations d'ext3 et pour fournir de meilleures performances, fiabilité et évolutivité. Voici quelques-unes des principales améliorations d'ext4 par rapport à ses prédécesseurs :
**Augmentation de la taille des fichiers** : Ext4 permet des tailles de fichiers plus importantes que ext3. La taille de fichier maximale dans ext4 est de 16 téraoctets. Dans ext3, la taille de bloc maximale est de 4 Ko. Avec une taille de bloc de 4 Ko, la taille de fichier maximale en ext3 est de 2 To.
**Performances améliorées** : Ext4 utilise une allocation plus efficace de l'espace

disque, ce qui peut améliorer les performances lors de la création, de la suppression et du redimensionnement des fichiers. Il utilise également l'allocation différée, ce qui améliore les performances en réduisant la fragmentation du disque.
**Meilleure fiabilité** : Ext4 inclut la prise en charge de la somme de contrôle du journal, ce qui permet de détecter et de corriger les erreurs dans le système de fichiers. Il comprend également une fonctionnalité appelée "fast commit", qui réduit le risque de perte de données en cas de panne de courant. Évolutivité améliorée : Ext4 peut gérer plus de fichiers et de répertoires qu'ext3, et il peut prendre en charge des disques plus grands avec une allocation de blocs plus efficace.
**Rétrocompatibilité** : Ext4 est conçu pour être rétrocompatible avec ext3, ce qui signifie qu'il peut être monté en tant que système de fichiers ext3 si nécessaire. Dans l'ensemble, ext4 représente une amélioration significative par rapport à ext3 en termes de performances, de fiabilité et d'évolutivité. Il s'agit d'un système de fichiers largement utilisé pour Linux et pris en charge par la plupart des distributions Linux modernes.

# Chapter 2
# Architecture

## 2.1 La structure

Le système de fichiers Ext4 est constitué d'un bloc de démarrage ainsi que de plusieurs groupes de blocs. Chacun de ces groupes contient plusieurs éléments clés tels que le superbloc, la table de descripteurs de groupe de blocs, le bitmap de blocs, le bitmap d'inodes, la table d'inodes et le bloc de données. Chaque élément remplit une fonction spécifique dans la gestion du système de fichiers et de son contenu. De plus, la redondance est intégrée dans le système pour assurer une fiabilité accrue.




Bloc de boot : le bloc de boot (de démarrage) est le premier secteur du système de fichiers et contient le code du chargeur de démarrage. Il est utilisé pour démarrer le système d'exploitation, il est optionnel.

Superbloc : le superbloc est une structure de données qui contient des informations sur le système de fichiers, telles que sa taille, la taille du bloc, le nombre d'inodes et le nombre de blocs libres. Il est situé à une position fixe dans le système de fichiers et est répliqué plusieurs fois dans tout le système de fichiers pour la redondance.

Table de descripteur de groupe de blocs : la table de descripteur de groupe de blocs contient des informations sur chaque groupe de blocs du système de fichiers, telles que le bitmap de bloc, le bitmap d'inode et la table d'inode. Il est situé après le superbloc et est répliqué pour la redondance.

Bitmap des blocs : le bitmap de bloc est une carte des blocs de données utilisés et libres dans un groupe de blocs. Il est utilisé pour allouer et désallouer des blocs de données.

Bitmap d'inode: le bitmap d'inodes est une carte des inodes utilisés et libres dans un groupe de blocs. Il est utilisé pour allouer et désallouer des inodes.

Table d'inodes : la table d'inodes contient les métadonnées de chaque fichier du système de fichiers, telles que le type de fichier, la propriété, les autorisations et les horodatages. Il est utilisé pour localiser les blocs de données qui stockent le contenu réel du fichier.

Blocs des données : le bloc de données est l'endroit où le contenu réel du fichier est stocké. Ext4 prend en charge différentes tailles de bloc allant de 1 Ko à 64 Ko, ce qui permet de meilleures performances et une utilisation efficace du stockage.

## 2.2 Groupes de blocs

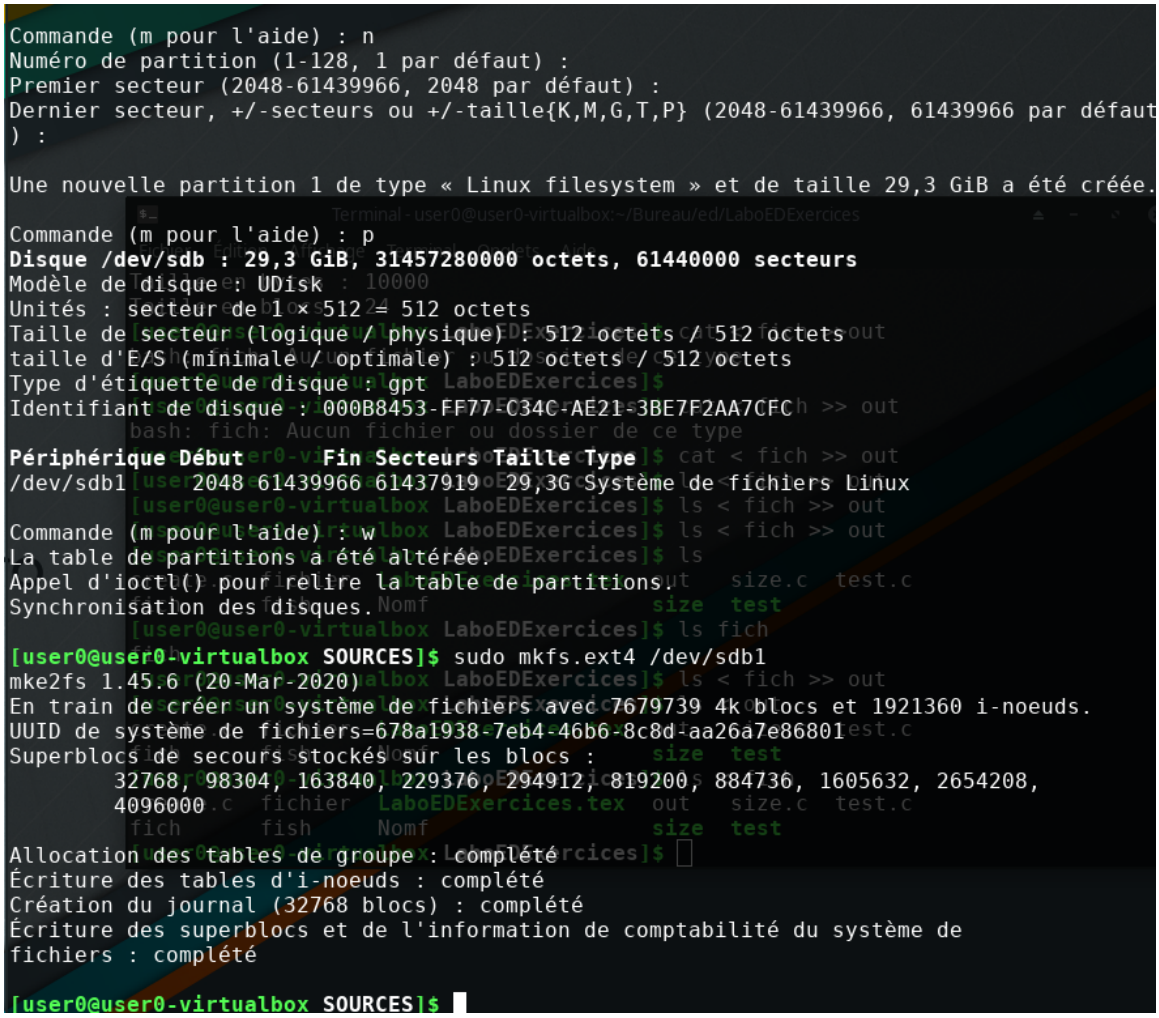
### 2.2.1 Superbloc

Le superbloc est une structure de données qui stocke des informations sur le système de fichiers dans son ensemble, telles que sa taille, son emplacement et son état. Le superbloc est dupliqué au cas où une copie serait endommagée. Il est fixé en place sur le disque et contient les informations suivantes:

| Offset | Taille | Nom | Description |
|---|---|---|---|
| 0x0 | __le32 | s_inodes_count | Total inode count. |
| 0x4 | __le32 | s_blocks_count_lo | Total block count. |
| 0x8 | __le32 | s_r_blocks_count_lo | This number of blocks can only be allocated by the super-user. |
| 0xC | __le32 | s_free_blocks_count_lo | Free block count. |
| 0x10 | __le32 | s_free_inodes_count | Free inode count. |
| 0x14 | __le32 | s_first_data_block | First data block. This must be at least 1 for 1k-block filesystems and is typically 0 for all other block sizes. |
| 0x18 | __le32 | s_log_block_size | Block size is 2 ^ (10 + s_log_block_size). |
| 0x1C | __le32 | s_log_cluster_size | Cluster size is 2 ^ (10 + s_log_cluster_size) blocks if bigalloc is enabled. Otherwise s_log_cluster_size must equal s_log_block_size. |
| 0x20 | __le32 | s_blocks_per_group | Blocks per group. |
| 0x24 | __le32 | s_clusters_per_group | Clusters per group, if bigalloc is enabled. Otherwise s_clusters_per_group must equal s_blocks_per_group. |
| 0x28 | __le32 | s_inodes_per_group | Inodes per group. |
| 0x2C | __le32 | s_mtime | Mount time, in seconds since the epoch. |
| 0x30 | __le32 | s_wtime | Write time, in seconds since the epoch. |
| 0x34 | __le16 | s_mnt_count | Number of mounts since the last fsck. |
| 0x36 | __le16 | s_max_mnt_count | Number of mounts beyond which a fsck is needed. |
| 0x38 | __le16 | s_magic | Magic signature, 0xEF53 |
| ..... | ..... | ..... | ..... |
| 0x27C | __le16 | s_encoding | Filename charset encoding. |
| 0x27E | __le16 | s_encoding_flags | Filename charset encoding flags. |
| 0x280 | __le32 | s_orphan_file_inum | Orphan file inode number. |
| 0x284 | __le32 | s_reserved[94] | Padding to the end of the block. |
| 0x3FC | __le32 | s_checksum | Superblock checksum. |

La taille ”le32” signifie ”little endian 32 bit integer”

Créons un système de fichiers ext4 pour explorer la structure.




Les informations que nous voyons ici sont :

- Un système de fichiers est créé composé de 7679739 blocs.
- La taille de bloc de 4096B est la valeur automatiquement sélectionnée, nous pouvons également créer avec une autre taille de bloc (-b blocksize). Cette taille de bloc n'est pas la taille de bloc logique (de LBA), c'est la taille de bloc du système de fichiers, et les blocs d'environ 4 M mentionnés ci-dessus sont les blocs du système de fichiers et non les blocs logiques.
- Il y a 1921360 inodes.
- L'UUID du système de fichiers est l'UUID de la partition dans GPT.
- Il y a quelque chose qui s'appelle Superblock avec 10 sauvegardes. Les groupes de blocs 0, 1 et les puissances de 3, 5 et 7 auront des superblocs de secours (par exemple : 343, 243, 125, 81, etc.)
- Il y a des choses appelées tables de groupe, tables d'inodes et journal.

Nous pouvons voir le contenu du superbloc avec dumpe2fs et l'option -h n'imprime que le Superblock. LectSuperBloc.c contient un code pour faire ça manuellement.


Le concept de disposition de stockage le plus basique d'ext4 est Block. ext4 alloue le stockage dans une série de blocs. La taille de bloc dans Superblock est de 4096B = 4KB. Le nombre de blocs dans ce système de fichiers est 7679739.
Le deuxième concept de disposition de stockage de base d'ext4 est le groupe de blocs. Les blocs contigus sont organisés en groupes de blocs, et dans notre système de fichiers, c'est 32768 = 32K.

La taille de bloc est de 4 KB. 8 * 4096 = 32768 blocs alloués par groupe.
Puisque nous avons 7679739 blocs, en divisant cela par 32K : 234,36 Nous avons donc 235 groupes de blocs contenant chacun 32K blocs (sauf le dernier).

### 2.2.2 Descripteur de groupe de blocs

| Offset | Taille | Nom | Description |
|---|---|---|---|
| 0x0 | __le32 | bg_block_bitmap_lo | Lower 32-bits of location of block bitmap. |
| 0x4 | __le32 | bg_inode_bitmap_lo | Lower 32-bits of location of inode bitmap. |
| 0x8 | __le32 | bg_inode_table_lo | Lower 32-bits of location of inode table. |
| 0xC | __le16 | bg_free_blocks_count_lo | Lower 16-bits of free block count. |
| 0xE | __le16 | bg_free_inodes_count_lo | Lower 16-bits of free inode count. |
| 0x10 | __le16 | bg_used_dirs_count_lo | Lower 16-bits of directory count. |
| 0x12 | __le16 | bg_flags | Block group flags. |
| 0x14 | __le32 | bg_exclude_bitmap_lo | Lower 32-bits of location of snapshot exclusion bitmap. |
| 0x18 | __le16 | bg_block_bitmap_csum_lo | Lower 16-bits of the block bitmap checksum. |
| 0x1A | __le16 | bg_inode_bitmap_csum_lo | Lower 16-bits of the inode bitmap checksum. |
| 0x1C | __le16 | bg_itable_unused_lo | Lower 16-bits of unused inode count. |
| 0x1E | __le16 | bg_checksum | Group descriptor checksum; |
| 0x20 | __le32 | bg_block_bitmap_hi | Upper 32-bits of location of block bitmap. |
| 0x24 | __le32 | bg_inode_bitmap_hi | Upper 32-bits of location of inodes bitmap. |
| 0x28 | __le32 | bg_inode_table_hi | Upper 32-bits of location of inodes table. |
| 0x2C | __le16 | bg_free_blocks_count_hi | Upper 16-bits of free block count. |
| 0x2E | __le16 | bg_free_inodes_count_hi | Upper 16-bits of free inode count. |
| 0x30 | __le16 | bg_used_dirs_count_hi | Upper 16-bits of directory count. |
| 0x32 | __le16 | bg_itable_unused_hi | Upper 16-bits of unused inode count. |
| 0x34 | __le32 | bg_exclude_bitmap_hi | Upper 32-bits of location of snapshot exclusion bitmap. |
| 0x38 | __le16 | bg_block_bitmap_csum_hi | Upper 16-bits of the block bitmap checksum. |

Chaque groupe de blocs sur le système de fichiers est associé à l'un de ces descripteurs. Les descripteurs de groupe sont le deuxième élément du groupe de blocs.
Remarquez comment le descripteur de groupe enregistre l'emplacement des deux bitmaps et de la table d'inodes (c'est à dire qu'ils peuvent flotter). Cela signifie qu'au sein d'un groupe de blocs, les seules structures de données avec des emplacements fixes sont le superbloc et la table de descripteurs de groupe. Le mécanisme flex_bg utilise cette propriété pour regrouper plusieurs groupes de blocs dans un groupe flexible et disposer tous les bitmaps et tables d'inodes des groupes en un seul long terme dans le premier groupe du groupe flexible.

### 2.2.3 Table d'inodes

Le système de fichiers ext4 prend en charge à la fois l'allocation basée sur l'inode et l'étendue. Dans la méthode d'allocation traditionnelle basée sur l'inode, l'inode contient des pointeurs vers des blocs de données individuels sur le disque. Dans la méthode d'allocation basée sur l'étendue, un seul inode peut représenter plusieurs blocs de données contigus, appelés extents. La structure d'un inode ext4 avec des extensions ou des pointeurs vers des blocs de données est similaire, avec quelques différences clés.

| Offset | Size | Name | Description |
|---|---|---|---|
| 0x0 | __le16 | i_mode | File mode. |
| 0x2 | __le16 | i_uid | Lower 16-bits of Owner UID. |
| 0x4 | __le32 | i_size_lo | Lower 32-bits of size in bytes. |
| 0x8 | __le32 | i_atime | Last access time. |
| 0xC | __le32 | i_ctime | Last inode change time. |
| 0x10 | __le32 | i_mtime | Last data modification time. |
| 0x14 | __le32 | i_dtime | Deletion Time, in seconds since the epoch. |
| 0x18 | __le16 | i_gid | Lower 16-bits of GID. |
| 0x1A | __le16 | i_links_count | Hard link count. |
| 0x1C | __le32 | i_blocks_lo | Lower 32-bits of “block” count. If huge_file is set and EXT4_HUGE_FILE_FL IS set in inode.i_flags, then this file consumes (i_blocks_lo + i_blocks_hi << 32) filesystem blocks on disk. |
| 0x20 | __le32 | i_flags | Inode flags. |
| 0x24 | 4 bytes | i_osd1 | Has multiple meanings depending on the filesystem creator. |
| 0x28 | 60 bytes | i_block[15] | Block map or extent tree. |
| 0x64 | __le32 | i_generation | File version (for NFS). |
| 0x68 | __le32 | i_file_acl_lo | Lower 32-bits of extended attribute block. ACLs are of course one of many possible extended attributes. |
| 0x6C | __le32 | i_size_high / i_dir_acl | Upper 32-bits of file/directory size. |
| 0x70 | __le32 | i_obso_faddr | (Obsolete) fragment address. |
| 0x74 | 12 bytes | i_osd2 | Has multiple meanings depending on the filesystem creator |
| 0x80 | __le16 | i_extra_isize | Size of this inode - 128. |
| 0x82 | __le16 | i_checksum_hi | Upper 16-bits of the inode checksum. |
| 0x84 | __le32 | i_ctime_extra | Extra change time bits. This provides sub-second precision. See Inode Timestamps section. |
| 0x88 | __le32 | i_mtime_extra | Extra modification time bits. This provides sub-second precision. |
| 0x8C | __le32 | i_atime_extra | Extra access time bits. This provides sub-second precision. |
| 0x90 | __le32 | i_crtime | File creation time. |
| 0x94 | __le32 | i_crtime_extra | Extra file creation time bits. This provides sub-second precision. |

Lorsque vous utilisez des extensions, le tableau i_block ne contient que les extensions elles-mêmes, alors qu'avec les blocs indirects, le tableau i_block contient des pointeurs vers des blocs indirects, qui à leur tour contiennent des pointeurs vers des blocs de données. L'en-tête d'inode est également légèrement différent lors de l'utilisation d'étendues, car le champ i_flags aura le bit EXT4_EXTENTS_FL défini.

| **Value** | **Description** |
|---|---|
| 0x1 | This file requires secure deletion (EXT4_SECRM_FL). (not implemented) |
| 0x2 | This file should be preserved, should undeletion be desired (EXT4_UNRM_FL). (not implemented) |
| 0x4 | File is compressed (EXT4_COMPR_FL). (not really implemented) |
| 0x8 | All writes to the file must be synchronous (EXT4_SYNC_FL). |
| 0x10 | File is immutable (EXT4_IMMUTABLE_FL). |
| 0x20 | File can only be appended (EXT4_APPEND_FL). |
| 0x40 | The dump(1) utility should not dump this file (EXT4_NODUMP_FL). |
| 0x80 | Do not update access time (EXT4_NOATIME_FL). |
| 0x100 | Dirty compressed file (EXT4_DIRTY_FL). (not used) |
| 0x200 | File has one or more compressed clusters (EXT4_COMPRBLK_FL). (not used) |
| 0x400 | Do not compress file (EXT4_NOCOMPR_FL). (not used) |
| 0x800 | Encrypted inode (EXT4_ENCRYPT_FL). This bit value previously was EXT4_ECOMPR_FL (compression error), which was never used. |
| 0x1000 | Directory has hashed indexes (EXT4_INDEX_FL). |
| 0x2000 | AFS magic directory (EXT4_IMAGIC_FL). |
| 0x4000 | File data must always be written through the journal (EXT4_JOURNAL_DATA_FL). |
| 0x8000 | File tail should not be merged (EXT4_NOTAIL_FL). (not used by ext4) |
| 0x10000 | All directory entry data should be written synchronously (see dirsync) (EXT4_DIRSYNC_FL). |
| 0x20000 | Top of directory hierarchy (EXT4_TOPDIR_FL). |
| 0x40000 | This is a huge file (EXT4_HUGE_FILE_FL). |
| 0x80000 | Inode uses extents (EXT4_EXTENTS_FL). |
| ..... | ..... |

# Chapter 3
# Les fonctionnalités d'ext4

Ext4 est connu pour ses excellentes performances et son évolutivité, ce qui en fait un choix populaire pour de nombreux cas d'utilisation. Voici quelques fonctionnalités clés liées aux performances d'Ext4 :
Les extents : Ext4 utilise un système de stockage de fichiers basé sur l'étendue, qui est plus efficace que le stockage traditionnel basé sur les blocs utilisé par les anciens systèmes de fichiers comme Ext2 et Ext3. Les étendues permettent des blocs de fichiers plus grands et plus efficaces, ce qui peut améliorer les performances de lecture et d'écriture.
Allocation différée : Ext4 utilise l'allocation différée pour optimiser l'utilisation de l'espace disque. Au lieu d'allouer des blocs immédiatement lorsqu'un fichier est écrit, Ext4 retarde l'allocation jusqu'à ce qu'elle soit nécessaire. Cela permet à Ext4 d'allouer des blocs contigus, ce qui améliore les performances et réduit la fragmentation.
Allocation multibloc : Ext4 propose une allocation multibloc, qui permet au système de fichiers d'allouer plusieurs blocs contigus à la fois. Cela peut améliorer considérablement les performances en réduisant le nombre de recherches de disque nécessaires pour lire et écrire des données.
Journalisation : Ext4 propose la journalisation, qui fournit une couche supplémentaire de protection des données en enregistrant les modifications du système de fichiers avant qu'elles ne soient enregistrées sur le disque. Cela peut améliorer les performances en réduisant le besoin de vérifications de cohérence intensives sur le disque après une panne du système.

## 3.1 Extents

Les étendues (*extents*) sont l'une des caractéristiques les plus importantes du système de fichiers *ext4*. Dans *ext3*, les données de fichier sont stockées dans une liste chaînée de blocs, alors que dans *ext4*, les données de fichier sont stockées dans des étendues, permettant de réduire l'utilisation des blocs de métadonnées et d'améliorer l'efficacité du disque.

Sous l'ancien schéma, l'attribution d'une série contiguë de 1 000 blocs nécessite un bloc indirect pour mapper les 1 000 entrées ; avec les extensions, le mappage est réduit à une seule structure ext4_extent avec ee_len = 1000. Les étendues sont disposées sous forme d'arborescence. Chaque nœud de l'arbre commence par une structure ext4_extent_header. Si le nœud est un nœud intérieur, l'en-tête est suivi d'instances eh.eh_entries de la structure ext4_extent_idx; chacune de ces entrées d'index pointe vers un bloc contenant plus de nœuds dans l'arborescence d'étendue. Si le nœud est un nœud feuille eh.eh_depth == 0, alors l'en-tête est suivi d'instances eh.eh_entries de la structure ext4_extent; ces instances pointent vers les blocs de données du fichier. Le nœud racine de l'arborescence des étendues est stocké dans inode.i_block, ce qui permet d'enregistrer les quatre premières étendues sans utiliser de blocs de métadonnées supplémentaires.

Le programme C suivant illustre l'utilisation des extensions dans ext4 :

```
Terminal - user0@user0-virtualbox:/home/user0/Bureau/codes/creation_mount/partext4/LaboED0202/SO
Fichier  Édition  Affichage  Terminal  Onglets  Aide
Le programme FichCreux.dat a crée un fichier FichCreux.dat
qui contient Hello world au début et
Bye world en position 100000

Ce fichier a une taille de 100009 bytes
et occupe sur le disque 8,0K

la taille d'un bloc sur le disque est de 4,0K

Les extents
Filesystem type is: ef53
File size of FichCreux.dat is 100009 (25 blocks of 4096 bytes)
 ext:     logical_offset:        physical_offset: length:   expected: flags:
   0:        0..       0:          0..         0:      1:             unknown_lo
c,delalloc
   1:       24..      24:          0..         0:      1:         24: last,unkno
wn_loc,delalloc,eof
FichCreux.dat: 2 extents found
----------------------------------
Le fichier FichCreux.dat est maintenant effacé pour éviter
de le copier sur un usb formaté en FAT
ou de l'ouvrir avec un traitement de texte
             --> Enter pour continuer
```

2 extents ont été utilisé pour créer le fichier FichCreux.dat.

```
[user0-virtualbox SOURCES]# sudo debugfs -R "blocks <1046543>" /dev/sdb1
debugfs 1.45.6 (20-Mar-2020)
33824 33848
[user0-virtualbox SOURCES]# sudo dd if=/dev/sdb1 bs=4096 skip=33824 count=1 | od -tx1
0000000 48 65 6c 6c 6f 20 77 6f 72 6c 64 00 00 00 00 00
0000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
1+0 enregistrements lus
1+0 enregistrements écrits
4096 octets (4,1 kB, 4,0 KiB) copiés, 0,00159869 s, 2,6 MB/s
0010000
[user0-virtualbox SOURCES]# sudo dd if=/dev/sdb1 bs=4096 skip=33825 count=1 | od -tx1
0000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
1+0 enregistrements lus
1+0 enregistrements écrits
4096 octets (4,1 kB, 4,0 KiB) copiés, 0,000322175 s, 12,7 MB/s
0010000
[user0-virtualbox SOURCES]# sudo dd if=/dev/sdb1 bs=4096 skip=33847 count=1 | od -tx1
0000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
1+0 enregistrements lus
1+0 enregistrements écrits
4096 octets (4,1 kB, 4,0 KiB) copiés, 0,000325512 s, 12,6 MB/s
0010000
[user0-virtualbox SOURCES]# sudo dd if=/dev/sdb1 bs=4096 skip=33848 count=1 | od -tx1
0000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0003240 42 79 65 20 77 6f 72 6c 64 00 00 00 00 00 00 00
0003260 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
1+0 enregistrements lus
1+0 enregistrements écrits
4096 octets (4,1 kB, 4,0 KiB) copiés, 0,000393172 s, 10,4 MB/s
0010000
```

Avec ls -i on a observé que l'inode du fichier est: 1046543. debugfs -R "blocks ¡1046543¿" /dev/sdb1 nous montre les adresses des blocks utilisés pour accueillir le contenu du fichier. On peut voir que les blocks sont contigus. La commande dd nous permet de voir ce qui se trouve dans ces blocks. On peut voir que dans le premier block il y a le texte "Hello world" et dans le dernier il y a "Bye world". Entre ces deux blocks se trouvent des blocks vides, non alloués.