# ext3Viewer
# Rapport de projet

Laurent Sebag Nathan Periana

Tuteur:M. Konstantin Verchinine

2006-2007

# Table des matières

# Chapitre 1

# Introduction

## 1.1 Introduction

L'idée générale que l'on se fait souvent d'un système de fichier est celle d'une organisation permettant la gestion de deux composants : les fichiers, et les dossiers, qui les contiennent. Les systèmes de fichiers sont utilisés sur les mémoires secondaires : disques durs, clés USB, cd-rom ...

Ce projet tutoré porte sur le système de fichier ext3, très utilisé dans le monde de Linux. Ce système de fichier est dit "journalisé", c'est à dire que l'on garde trace des actions effectuées sur le disque[1]. D'un point de vue technique, les objets habituels sont tous representés par un *inode*. L'*inode* est une structure de données qui va permettre de représenter tous les objets que l'utilisateur manipule : les fichier, les dossiers, les périphériques, les liens, etc.

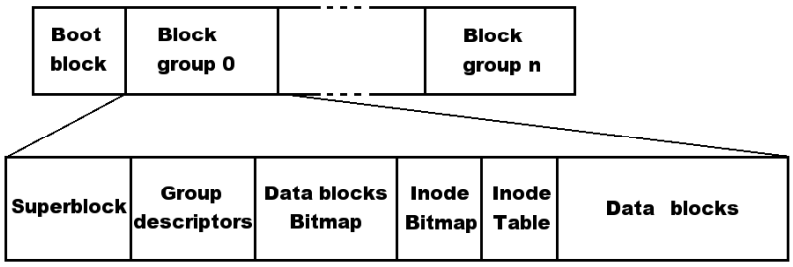
## 1.2 Description

Le but du projet ext3Viewer est de permettre l'exploration du système de fichier à un très bas niveau.
Beaucoup d'informations administratives sont contenues dans le système de fichier. Ces données internes, qui permettent l'organisation et la préservation de l'intégrité des fichiers, sont cachées à l'utilisateur normal. Ext3Viewer est destiné aux étudiants en informatique ayant des connaissances sur ext3. Il leur permet d'accéder à toutes les structure cachées de leur partition. Ainsi, ils auront une meilleur compréhension de la façon dont sont stockées physiquement les données sur un disque dur utilisant ext3.
Le projet ext3viewer a été proposé par M. Konstantin Verchinine ; il est destiné à remplacer le projet e2view, qui sert pendant le cours d'Architecture, Système et Réseaux sur les systèmes de fichiers.

[1] Pendant le moment où les données sont en cache et pas encore écrites sur le disque, pour garder l'intégrité du système de fichier en cas de panne matérielle.

## 1.3 Historique et fonctionnement de ext3

Le système de fichier ext3 est un système de fichiers journalisé, majoritairement utilisé par le système d'exploitation Linux ; mais il est également compatible avec BSD, et dans une moindre mesure, avec Windows. Il a été introduit en Novembre 2001, dans le kernel 2.4.15, pour remplacer ext2. L'une des raisons de son succès est la rétrocompatibilité avec ce dernier : une partition ext2 peut être convertie vers ext3, et vice versa ; et un système de fichier ext3 peut être monté (utilisé) en ext2, moyennant la perte de certaines fonctionnalités.
ext3 est organisé de la façon suivante[2] :




Le 'boot block' peut ne pas être utilisé, par exemple dans le cas d'un fichier image. Chaque 'block group' contient une copie du superblock et des group descriptors ; mais seuls ceux du premier block group (block group 0) sont utilisés par le noyau, les autres étant uniquement des copies de sauvegarde. L'inode table n'est pas une structure, elle contient simplement l'ensemble des inodes d'un groupe.

## 1.4 Choix préliminaires réalisés

Le projet est réalisé en anglais, comme conseillé par M. Verchinine. Ceci peut s'expliquer de plusieurs façons : l'anglais est le langage de l'informatique ; le système de fichiers ext3 a donc été développé en anglais, et il n'existe malheureusement pas de traductions françaises officielles pour certaines de ses fonctionnalités.
Il a été décidé de ne pas se baser sur le projet e2view,pour plusieurs raisons. D'abord, ce programme était pour sa plus grande partie en langue française, et les interfaces consoles et graphiques n'étaient pas suffisament intuitives et ergonomiques ; il aurait fallu réécrire une grande partie du code pour les adapter à nos besoins. Ensuite, le projet ext3viewer se voulait un moyen de découvrir le fonctionnement de ext3 ; il était donc préférable, d'un point de vue didactique, de ne pas reprendre un programme déjà existant.

[2]Le vocabulaire technique est expliqué dans la partie glossaire.

# Chapitre 2

# Cahier des charges

## 2.1 Rappel du cahier des charges

Le programme devait être divisé en deux parties : une partie graphique, et une partie console.

### Fonctionnalités utilisateur

#### Visualisation des structures ext3

Le programme devait permettre de visualiser les différentes structures du système de fichier ext3 ainsi que les valeurs prises par chacun des champs, ceux-ci étant explicités.

#### Navigation à travers le système de fichier

L'interface graphique permettrait de naviguer à travers le système de fichier tandis que le mode console, réservé aux utilisateurs experimentés, devait permettre de visualiser les structures. Grâce à l'interface graphique, l'utilisateur devait voir ses fichiers et dossiers habituels et pouvoir afficher la structure de chaque inode, les informations concernant le *boot sector*, les *group blocks*, le *superblock*, les *bitmaps*. Le journal du système de fichier devait être affiché. Une explication sur les structures, les utilisations du système de fichier et leurs conséquences avait été prévue pour une partie aide.

#### Correspondance nom/inode et inversement

Le logiciel pourrait traduire un nom de fichier vers le numéro d'inode correspondant, chercher un fichier, à partir de son numéro d'inode, et afficher à quel inode est rattaché un block en tapant son numéro de block.

#### Recherche de structures libres

Le programme permettrait de trouver le prochain inode libre sur le disque, ou le prochain block libre. On pourrait de plus tester si un inode ou un block est utilisé, et le cas échéant se rendre au bon endroit dans l'arborescence pour afficher la structure du fichier ou dossier.

#### Plate-formes d'utilisation

Le programme serait compatible Linux, et peut être Windows et MacOS.

## 2.2 Evolution du cahier des charges

Comme expliqué précédemment, le projet a été réalisé en anglais ; les modules d'internationalisation n'ont donc pas été implémentés. Le découpage mode console/mode graphique a été conservé. Par contre, certaines fonctionnalités n'ont pas été développées, car elles n'étaient pas indispensables à la compréhension générale de ext3. Certaines structures, comme les ACLs, n'ont pas été affichées, ou bien affichées d'une autre façon après une étude plus détaillée du système de fichier.
La compatibilité avec Windows a été provisoirement abandonnée, du fait de l'absence de nombreuses fonctions basiques (open, read, lseek...), mais un port vers cette platforme est théoriquement possible, en créant ses propres fonctions, en utilisant les fonctions de compatibilité de la Glib (bibliothèque de Gnome, multiplateforme), ou en utilisant Cygwin...
La compatibilité avec MacOS n'a pas encore été testée, mais est envisageable (MacOS X est compatible POSIX).

### Résumé des fonctionnalités prévues/créées

#### Visualisation des structures ext3

OK : Les structures principales sont affichées, avec des explications supplémentaires sur certains de leurs champs.
Structures affichées : *Superblocks, Inodes, Block Bitmap, Inode bitmap, Group Descriptors, Journal Superblock Header, Journal Header, Direntries, Extended Attributes,Blocks*
Structures non affichées : *Access Control Lists.*

#### Navigation à travers le système de fichier

OK :La navigation est assurée par un 'explorateur' de fichiers en mode graphique, et par l'option -ls, plus le numéro d'inode (l'inode de la racine étant l'inode 2).

#### Correspondance nom/inode et inversement

OK : En mode graphique, l'utilisateur peut y accéder grace à une barre de recherche visible sur l'écran principal ; en mode console, il peut y accéder grace aux options -find (nom, et numéro d'inode du dossier parent vers numéro d'inode) et -iname (numéro d'inode vers nom).

#### Recherche de structures libres

OK :Le mode graphique possède dans le menu 'Tools' des boutons pour l'affichage des bitmaps (*inode bitmap*, *block bitmap*), et la recherche du premier inode ou block libre, pour chaque *block group.*
Les fonctions pour déterminer si un inode ou un block sont utilisées n'ont pas

été développées, car l'utilisateur peut aller voir dans l'inode ou le block bitmap si la structure est occupée ou non.

### Plate-formes d'utilisation

Le programme ext3viewer a été testé avec Gentoo, SuSe 10.1, Slackware 10.1,11.0 en architecture little-endian. La compatibilité Windows n'est pas assurée, pour la version 1.0. La compatibilité Mac OS n'a pas été testée. Bien que des fonctions du type *le32_to_cpu()* soient utilisées dans le code, pour assurer la compatibilité avec les machines possèdant une architecture différente, il faudrait pouvoir la vérifier directement sur celles-ci.

# Chapitre 3

# Développement

## 3.1 Outils employés

Au cours de notre projet, nous avons employé de nombreux outils ; certains très connus, d'autres moins. Les voici, résumés et décrits.

- GCC : GCC est le compilateur standard sous Linux.
- Langage C : Le langage C est l'un des langages les plus utilisés dans le code source et dans le monde de Linux. Ext3viewer se positionne donc en continuité avec cette philosophie.
- Le C permet également un accès bas niveau au disque et à la mémoire, particulièrement utiles pour le projet.
- eFence : Bibliothèque Electric Fence, utile pour la vérification de la gestion dynamique de la mémoire. Elle a été utilisée lors de la phase de tests, pour détecter les erreurs possibles de segmentation fault.
- groff : groff est un outil de formattage, dédié à la création de pages manuel.
- GTK :Gtk est une bibliothèque graphique, qui a été implémentée pour de nombreux langages (C, C++, Java, Python...), et de nombreuses plateformes (Linux,Windows...). Elle a été choisie pour son efficacité, et sa simplicité d'utilisation, ainsi que pour sa disponibilité sur les machines de l'IUT.
- LaTeX : LaTeX (prononcer 'Latek') est un langage de balises, utilisé pour la production des documents, avec une qualité professionnelle (génération automatique d'index, mise en page, ...).

## 3.2 Planning et execution

### 3.2.1 Planning prévu

**Septembre- fin Octobre :**

Planification générale du projet ; création du planning détaillé, rédaction d'un cahier des charges. Durée : 1 mois.

**Octobre - mi-décembre :**

Codage des accès au disque, des fonctions de lecture et des affichages en mode console. Durée : 1+1/2 mois

**Fin décembre-fin février :**

Codage de l'interface graphique. Rédaction d'une partie de l'aide utilisateur. Durée : 2+1/2 mois.

**Fin février - mi-mars :**

Rédaction du reste de l'aide utilisateur. Rédaction des documentations utilisateur, développeur, et de la documentation sur ext3. Création des fichiers d'internationalisation.

### 3.2.2 Planning effectué et réajustements

Le planning subi des réajustements, en fonction des difficultés rencontrées : certaines parties du projet se sont révélées plus complexes que prévues, et certaines ont pris moins de temps. Certaines fonctionnalités secondaires ont été abandonnées, pour pouvoir se concentrer sur d'autres plus importantes.
En particulier, l'interface console a nécessité un travail préliminaire d'étude de ext3 ; la signification précise des champs des structures et les méthodes d'accès à celles-ci ont dues être retrouvées. Au vu de la nouvelle estimation de la charge de travail, le délai fixé pour le développement du mode console a donc été rallongé.
L'interface graphique, par contre, a nécessité moins de temps que prévu, la bibliothèque GTK étant relativement simple à apprendre et à utiliser.

**Septembre-Octobre**

Choix et étude préalable du sujet, analyse des besoins, analyse de l'existant, définition des fonctionnalités nécessaires, organisation du travail. Rédaction du cahier des charges. Durée : 1 mois.

**Novembre - mi-décembre**

Codage d'une partie du mode console : recherche d'un fichier, correspondance inode/chemin, lecture et affichage du superblock, lecture et affichage des group descriptors, lecture et affichage des direntries (dossiers), lecture et affichage du contenu d'un fichier, lecture et affichage du contenu des blocks, lecture

et affichage du contenu d'un inode . Recherche de documentation détaillée sur ext3, apprentissage du fonctionnement et de l'organisation de ext3 sur le disque.

**Fin décembre-fin février**

Codage de l'interface console : Lecture d'un block en donnant son numéro, affichage de l'arbre (simplifié) de l'allocation des blocks pour un inode, affichage du contenu d'un lien symbolique, lecture et affichage des copies du superblock, affichage des blocks indirect comme pointeurs de blocks, lecture et affichage du block bitmap et de l'inode bitmap, affichage du premier block ou inode libre, Création de la page manuel, et de l'option d'aide -help (mode console). Début de la rédaction des documentations.

**Fin février - mi-mars**

Codage de l'interface console : Lecture et affichage du journal. Affichage des extended user attributes. Nettoyage et réorganisation du code. Codage de l'interface graphique. Rédaction des documentations utilisateur et développeur. Mise à jour de la page manuel pour refléter les changements apportés au projet. Tests du programme. Rédaction du rapport.

### 3.2.3 Répartition du travail et méthodes d'organisation

**Répartition du travail**

Laurent Sebag

- Création de l'interface graphique
- recherche d'un fichier
- correspondance inode/chemin
- lecture du superblock
- lecture et affichage des group descriptors
- lecture et affichage des direntries (dossiers)
- lecture et affichage du contenu d'un fichier
- lecture et affichage du contenu des blocks
- lecture du contenu d'un inode
- Lecture d'un block en donnant son numéro
- Lecture et affichage du journal
- conversion d'un nom de fichier vers un numéro d'inode
- affichage de l'arbre (simplifié) de l'allocation des blocks pour un inode
- affichage du contenu d'un lien symbolique
- lecture et affichage des copies du superblock
- affichage des blocks indirect comme pointeurs de blocks
- Recherche de documentation détaillée sur ext3
- Rédaction du cahier des charges
- Rédaction de la documentation utilisateur
- Rédaction de la documentation développeur
- Rédaction d'une partie du rapport
- Rédaction de la page manuel
- Tests du programme
- Coordination du travail

Nathan Periana

- affichage et explication du superblock
- affichage et explication du contenu d'un inode
- Recherche de documentation détaillée sur ext3
- lecture et affichage du block bitmap
- lecture et affichage de l'inode bitmap
- affichage du premier block libre
- affichage du premier inode libre
- Lecture et affichage du journal
- Lecture et affichage des extended user attributes
- Rédaction du cahier des charges
- Rédaction d'une partie de la documentation utilisateur
- Rédaction d'une partie de la documentation développeur
- Correction de la page manuel
- Tests du programme
- Rédaction du rapport

**Méthodes d'organisation**

Les fonctionnalités à implémenter étaient regroupées dans une 'TODO list', qui évoluait au cours du projet ; elles étaient attribuées à un développeur, et marquées comme 'à faire', 'effectuées' ou 'abandonnées'.
Du fait de la petite taille du projet (2 personnes), le système de CVS n'a pas été utilisé. Plusieurs techniques manuelles ont tout de même été utilisées pour le remplacer. Chaque développeur travaillait sur ses fichier en local, en essayant d'éviter de modifier simultanément le même code source. Dans les cas où les deux sources étaient modifiées en même temps, l'un des développeurs se chargeait de fusionner les codes sources, avec l'aide de l'utilitaire tkdiff (une interface graphique à l'utilitaire diff, qui compare deux fichiers ligne par ligne). Le transfert des fichiers se faisait par mail, et la communication par messagerie instantanée.

## 3.3 Difficultés rencontrées

L'une des difficultés majeures rencontrées dans ce projet est le manque de documentation technique sur certaines structures. En effet, les documents trouvés sur internet concernent majoritairement ext2 ; bien que les deux systèmes de fichiers partagent de nombreuses structures, certaines possèdent des différences importantes, ou sont totalement spécifiques à ext3.
Deux structures ont posé particulièrement problème : le journal, et les ACL.
Deux structures ont posé particulièrement problème : le journal, et les ACL. Le journal est géré par une couche applicative créée pour être distincte de ext3 : le JBD (Journaling Block Device). La documentation sur le fonctionnement de l'API du JBD existe ; par contre, son implémentation sur le disque n'est que partiellement visible, dans les 'includes' de développement. Les différents champs des structures sont indiqués, mais peu commentés, et la position des structures elles-mêmes sur le disque a du être 'devinée', par des tests et par la consultation des sources de l'utilitaire 'debugfs', qui affiche quelques informations sur le journal.
La documentation concernant les Access Control Lists pose un problème similaire, il renseigne sur le fonctionnement général de celles ci, mais pas sur leur organisation sur le disque ; de plus, les ACL changent selon les versions du

noyau. En effet, les ACL faisaient l'objet d'une spécification POSIX, mais elle a été abandonée ; les ACLs sont implémentées différement entre les versions 2.4.28 et 2.6.10+.
La programmation de l'accès au disque a également généré des complications. En effet, l'API disponible avec ext3 s'est révélée inadaptée aux besoins du projet ; elle ne pouvait fonctionner que sur un système monté. Le projet, pour des raisons de cohérence des données lues et de simplicité d'utilisation, devait être capable de travailler avec des images ou des disques démontés ; il a donc été nécessaire de créer toutes les fonctions d'accès au disque.

# Chapitre 4

# Bilans

### Laurent Sebag

Ce projet m'a permis d'apprendre à répartir les taches d'un projet entre plusieurs personnes. Pour cela j'ai utilisé le principe des todo lists : à période régulière, je fixait des objectif à atteindre.

J'ai appris à utiliser une API graphique permettant la creation de fenêtres : gtk+-. Cependant cela m'a fait réfléchir sur l'utilisation du c++ ou d'un autre language orienté objet. Ce serait plus aproprié pour réaliser une interface graphique.

Gtk simule le concept d'objet, ce qui rend le codage plutot lourd car utilise des macros pour cela. C'est pourquoi ce projet m'a donné envie d'essayer gtkmm, la version de gtk en C++.

Durant le projet j'ai aussi appris à utiliser vim qui est une formidable alternative à emacs : je la conseille car elle comporte des fonctionalité pratiques pour le codage. L'utilisation de LaTeX est très utiles aussi, le fait de donner l' organisation d'un texte et qu'il produise en retour un document bien présenté est agréable.

L'études de parties de codes du noyau et de debugfs m'ont aussi fait réfléchir sur l'organisation des fichiers sources dans un projet. L'utilisation de couches d'abstractions permet de rendre un logiciel plus modulaire et lors des prochains projets j'essairai d'applique ce principe dès le début de la programmation.

Pour conclure ce projet a été une bonne expérience de travail a deux. Le résultat me donne satisfaction car pour une première fois, nous avons crée un logiciel qui sera normalement utilisé (pour les tp d' ASR).

### Nathan Periana

Ce projet m'a beaucoup appris au niveau technique. J'ai découvert le fonctionnement bas niveau de ext3, à travers la documentation, le code source du noyau, et la méthode du 'trial and error' ; cela m'a aidé à mieux comprendre certaines fonctionnalités présentes sur mon système d'exploitation, comme la réservation d'un certain nombre de blocks pour le superutilisateur, l'utilisation des Access Control Lists, et le stockage des données sur le disque, lors de la création des fichiers, et lors de leur suppression.
J'ai également gagné de l'experience dans le développement d'un long projet en

équipe, les seuls projets réalisés jusqu'à présent ayant été relativement courts, et moins bien coordonnés. J'ai le sentiment que le groupe a bien fonctionné, et je suis plutôt satisfait du résultat.
J'ai pu apprendre les bases de GTK, et je me suis également initié à LaTeX, avec l'aide de Laurent ; j'ai trouvé ces deux technologies très utiles, et je les conseillerais vivement à tout programmeur souhaitant réaliser un projet de ce type.
J'espère que le travail que nous avons effectué pourra être utile, tant pour les étudiants cherchant à connaître mieux ext3, que pour les développeurs en recherche de documentation.

# Chapitre 5

# Conclusion

Le développement d'une application n'est pas un processus figé. Le projet ext3viewer pourrait être modifié et amélioré, c'est pourquoi nous avons voulu le placer, sauf objection de la part de l'IUT de Fontainebleau, sous la licence GNU GPL.
Un nouveau système de fichier, nommé ext4, a d'ailleurs déja été développé pour succéder à ext3. Il est disponible depuis le noyau 2.6.19 de linux. Notre programme pourrait donc être modifié, pour être capable de fonctionner avec celui-ci ; ext4 étant rétrocompatible avec ext3, il suffirait de rajouter les fonctionnalités spécifiques à ext4.

# Chapitre 6

# Annexes

## 6.1 Glossaire

Voici quelques définitions [1] techniques :

**Inode** : Structure des systèmes de fichiers linux qui sert à garder les informations administratives sur un fichier (taille, nom...)

**Block** : zone délimitée sur le disque contenant un certain nombre d'octets (sous ext3, 1024, 2048 ou 4096).

**Data block** : block destiné spécifiquement à contenir les données de l'utilisateur (fichiers, ACLs...), plutôt que les structures du système de fichiers.

**Block group** : Groupe de blocks, contenant les structures du système de fichier (superblock, inodes, data blocks...). Leur nombre est fonction de la taille des blocks et de la taille de la partition.

**Superblock** :structure des systèmes de fichiers ext2/3, qui conserve les informations sur le système de fichiers lui-même (taille d'un block, fonctionnalités présentes...)

**Group descriptor** :structure des systèmes de fichiers ext2/3, qui conserve les informations sur le groupe (position de l'inode bitmap, nombre de dossiers...)

**Inode table** : ensemble de tous les inodes d'un block group.

**Bitmap** : 'carte', qui fait correspondre à un bit l'état booléen d'une donnée. Dans le cas de l'inode et du block bitmap, il s'agit de savoir si le block est libre (0) ou occupé (1).

**Journal** : 'partie' d'un système de fichier qui aide à préserver l'intégrité de celui-ci en enregistrant ses modifications.

**POSIX** : Portable Operating System Interface Ensemble de standards, créés pour rendre compatibles entre elles différentes variantes de UNIX (Solaris, Cygwin, Mac OS X, Linux (partiellement)).

**ACL** : Access Control List. Listes de permissions dédiées à la gestion plus 'fine' des droits utilisateurs (lecture, écriture, execution...), et disponibles sur plusieurs systèmes de fichiers (ext3, NTFS...)

[1] Le vocabulaire technique est utilisé en anglais, par souci de cohérence avec le code.

**API** :Application Programming Interface. Bibliothèque contenant des fonctions pouvant être appelées depuis un programme.

## 6.2 Remerciements

Nous tenons à remercier les personnes suivantes :
- M. Konstantin Verchinine, et M. Andrei Paskevich, pour leur aide dans le développement de ce programme.
- Julien Poitrat et son équipe, pour leur projet e2view.

## 6.3 Liens

Sélection de documents interessants qui nous ont servi lors de notre projet.
`http ://kerneltrap.org/node/6741` :
Informations sur le journal (JBD).
`http ://www.suse.de/~ agruen/acl/linux-acls/online/` :
Documentation technique concernant les ACL.
`http ://www.lea-linux.org/cached/index/Gestion_des_ACL.html` :
Comment créer et utiliser des ACL.
`http ://lxr.linux.no/` :
Sources du noyau pour plusieurs versions, avec des possibilités de comparaison.
`http ://www.nongnu.org/ext2-doc/ext2.html` :
Documentation technique concernant ext2. Un bon nombre des informations restent valable pour ext3.
`http ://www.oreilly.com/catalog/linuxkernel2/chapter/ch17.pdf` :
Documentation technique concernant surtout ext2, mais contenant également une partie sur ext3.