3η Εργαστηριακή Άσκηση Λειτουργικών Συστημάτων

Νικόλαος Οικονόμου: 03120014
Πέτρος Αυγερίνος: 03115074

1 Εισαγωγή

Αντικείμενο της παρούσας αναφοράς είναι η εξοικείωση με τα συστήματα αρχείων που χρησιμοποιούνται για την οργάνωση δεδομένων σε συσκευές block. Στο πρώτο μέρος της άσκησης λάβαμε τρεις εικόνες δίσκων και μας ζητήθηκε να αντιμετωπίσουμε κάποιες προκλήσεις με χρήση CLI tools και με χρήση ενός hex editor, από την μερία δηλαδή του χρήστη.

Στο δεύτερο μέρος θα ασχοληθούμε με τα συστήματα αρχείων από την μερία του πυρήνα του Linux, υλοποιώντας κάποιες συναρτήσεις σε ένα νέο σύστημα αρχείων ext2-lite, βασισμένο στο ext2 και ελέγχοντας τη σωστή του λειτουργία.

2 FS Disk 1

Θα ορίσουμε αρχικά κάποιες μεταβλητές για καλύτερη χρήση του hexedit:

1. boot = boot sector offset in bytes = 1024
2. blocksize = blocksize in bytes = 1024
3. blocks per group = bpg = 8192
4. superblock = superblock in bytes = 1024
5. block group descriptor = bgd = bgd in bytes = 1024

2.1 Ερώτημα 1ο

Στο αρχείο utopia.sh προσθέτουμε την εντολή "-drive format=raw,file=fsdisk1.img,if=virtio" στο executable του qemu. Με την εντολή μέσα στο VM "lsblk -f" προκύπτει ότι η συσκευή με label fsdisk1.img είναι μία Virtio Block Device, με σύστημα αρχείων ext2.

2.2 Ερώτημα 2ο

1. **tool:** Με χρήση της εντολής lsblk μπορούμε να δούμε ότι το μέγεθος της συσκευής vdb είναι 50MB.
2. **hexedit:** Με χρήση hexedit ανοίγουμε το /dev/vdb και στην οθόνη βλέπουμε συνολικό μέγεθος 0x3200000 = 50MB.

2.3 Ερώτημα 3ο

1. **tool:** Όπως είπαμε προηγουμένως το σύστημα αρχείων είναι το ext2.

2. **hexedit:** Στα offset 56-57 του superblock μπορούμε να δούμε την υπογραφή του ext2, 0xef53.

2.4 Ερώτημα 4ο

1. **tool:** Με την χρήση της εντολής dumpe2fs -h /dev/vdb μπορούμε να δούμε ότι το σύστημα δημιουργήθηκε την Τρίτη 12 Δεκεμβρίου του 2023 στις 17:23:16.
2. **hexedit:** Δεν μπορούμε να το δούμε μέσω hexedit για το ext2 filesystem.

2.5 Ερώτημα 5ο

1. **tool:** Με την ίδια εντολή το σύστημα αρχείων προσαρτήθηκε την ίδια ώρα και μέρα ακριβώς.
2. **hexedit:** Στα offset 44-47 βλέπουμε τον αριθμό 0x65787AE4 = 1702394596 που είναι σε POSIX time, άρα με την μετατροπή προκύπτει η ίδια ημερομηνία και ώρα με πάνω.

2.6 Ερώτημα 6ο

1. **tool:** Όμοια εντολή με πριν.Στο μονοπάτι /cslab-bunker.

2. **hexedit:** Αρχικά θα πρέπει να ελέγξουμε αν έχουμε extended superblock fields μέσω του offset 76-79. Πράγματι έχουμε αφού είναι διαφορετικό του μηδενός και μπορούμε να ελέγξουμε τα bytes στο offset 136-199. Το μονοπάτι είναι /cslab-bunker.

2.7 Ερώτημα 7ο

tool: Με την ίδια εντολή το σύστημα αρχείων γράφτηκε τελευταία ένα δευτερόλεπτο μετά την γέννησή του.

hexedit: Στα offset 48-51 βλέπουμε τον αριθμό 0x65787AE5 = 1702394597 που είναι σε POSIX time, άρα με την μετατροπή προκύπτει η ίδια ημερομηνία και ώρα με πάνω.

2.8 Ερώτημα 8ο

Σε ένα σύστημα αρχείων, το block είναι η μεγαλύτερη συνεχόμενη ποσότητα χώρου στο δίσκο που μπορεί να προσδοθεί σε ένα αρχείο και επίσης η μεγαλύτερη ποσότητα δεδομένων που μπορεί να μεταφερθεί σε μία μοναδική διαδικασία I/O. Δεν αντιστοιχίζεται στη φυσική διαρρύθμιση της μνήμης αλλά είναι virtual.

2.9 Ερώτημα 9ο

1. **tool:** Με χρήση της εντολής dumpe2fs /dev/vdb βλέπουμε ότι το μέγεθος του block είναι 1024.

2. **hexedit:** Πηγαίνοντας στη θέση 0x418 που είναι το 24 byte μέσα στο superblock βλέπουμε ότι είναι ίση με το μηδέν. Γνωρίζουμε όμως ότι για αύτη την θέση ισχύει $log_2(\text{blocksize}) - 10$ = τιμή στις θέσεις [0x418:0x421] = $0 \Rightarrow$ blocksize = 1024

2.10 Ερώτημα 10ο

Είναι ένα data structure το οποίο εκφράζει ένα αντικείμενο μέσα στο σύστημα αρχείων. Αποθηκεύει τα χαρακτηριστικά και τις τοποθεσίες των δεδομένων του αντικειμένου μέσα στα blocks. Δεν περιέχει τα ίδια τα δεδομένα, αλλά μία σύνδεση στο block το οποίο τα περιέχει.

2.11 Ερώτημα 11ο

1. **tool:** Με χρήση της εντολής dumpe2fs -h /dev/vbd βλέπουμε ότι το μέγεθος του inode είναι 128 bytes.

2. **hexedit:** Στα offset 88-89 έχουμε την πληροφορία του μεγέθους του inode που είναι ίση με 0x0080 = 128 bytes.

2.12 Ερώτημα 12ο

tool: Τα διαθέσιμα blocks με την ίδια εντολή είναι 49552 ενώ τα διαθέσιμα inodes είναι 12810.

hexedit: Στα offset 12-15 μπορούμε να δούμε τον αριθμό διαθέσιμων blocks τα οποία είναι 0xC190=49552 και στα offset 16-19 μπορούμε να δούμε τον αριθμό διαθέσιμων inodes τα οποία είναι 0x320A=12810.

2.13 Ερώτημα 13ο

Το superblock είναι ένας κατάλογος των χαρακτηριστικών ενός συστήματος αρχείων και περιέχει πληροφορίες σχετικά με το βασικό μέγεθος και σχήμα του συστήματος αρχείων.

2.14 Ερώτημα 14ο

Το superblock βρίσκεται στην αρχή του συστήματος αρχείων με offset 1024 bytes από την αρχή του όγκου (μετά το bootsector).

2.15 Ερώτημα 15ο

Υπό κανονική λειτουργία χρησιμοποιείται μόνο το αρχικό superblock, αλλά σε περίπτωση που προκύψει corruption ή overwrite στο superblock χρησιμοποιούνται τα backups.

2.16 Ερώτημα 16ο

1. **tool:** Με χρήση dumpe2fs /dev/vdb βλέπουμε ότι τα εφεδρικά superblocks βρίσκονται στο Group 1 στο block 8193, στο Group 2 στο block 16385, στο Group 3 στο block 24577, στο Group 4 στο 32769, στο Group 5 στο block 40961, στο Group 6 στο block 49153.
2. **hexedit:** superblock position$_i$ = hex(boot + $(i-1) \cdot$ bpg $\cdot$ blocksize)
   (a) superblock position$_1$ = $0x400$
   (b) superblock position(εφ.)$_2$ = $0x800400$
   (c) superblock position(εφ.)$_3$ = $0x1000400$
   (d) superblock position(εφ.)$_4$ = $0x1800400$
   (e) superblock position(εφ.)$_5$ = $0x2000400$
   (f) superblock position(εφ.)$_6$ = $0x2800400$
   (g) superblock position(εφ.)$_7$ = $0x3000400$

2.17 Ερώτημα 17ο

Πολλά blocks μαζεύονται για να αποτελέσουν ένα block group. Δεδομένα για ένα αρχείο τοποθετούνται σε ένα μόνο block group για να μειωθούν οι αναζητήσεις για I/O λειτουργίες.

2.18 Ερώτημα 18ο

Ο συνολικός αριθμός των block group εξαρτάται από το μέγεθος του partition και το μέγεθος του block. Κατανέμονται σειριακά σε ένα partition του δίσκου ύστερα από το superblock.

2.19 Ερώτημα 19ο

1. **tool:** Θα το βρούμε με χρήση της εντολής dumpe2fs -g /dev/vdb. Τα groups είναι ίσα με 7.
2. **hexedit:** Αν στη συνάρτηση που ορίσαμε στο ερώτημα 16 βάλουμε σαν όρισμα το 8, πάμε στη θέση 0x3800400 και προκύπτει error για invalid position στο hexedit, επομένως δεν υπάρχει όγδοο group, άρα σύνολο 7.

2.20 Ερώτημα 20ο

Ο block descriptor είναι ένα data structure το οποίο περιγράφει ένα block group. Περιέχει τα bitmaps των blocks και των inodes, τον πίνακα των inodes και τον αριθμό ελεύθερων blocks, ελεύθερων inodes και χρησιμοποιημένων directories.

2.21 Ερώτημα 21ο

Σε περίπτωση που αποτύχει το κύριο block descriptor.

2.22 Ερώτημα 22ο

Αποθηκεύονται όμοια με τα εφεδρικά superblocks.

1. **tool:** Με dumpe2fs /dev/vdb βλέπουμε ότι για το Group 1 το BGD βρίσκεται στο block 8194, για το Group 2 στο block 16386, για το Group 3 στο block 24578, για το Group 4 στο block 32770, για το Group 5 στο στο 40962, για το Group 6 στο 49154.
2. **hexedit:** BGD position$_i$ = hex(boot + $(i-1) \cdot$ bpg $\cdot$ blocksize + superblock)
   (a) BGD position$_1$ = $0x800$
   (b) BGD position(εφ.)$_2$ = $0x800800$
   (c) BGD position(εφ.)$_3$ = $0x1000800$
   (d) BGD position(εφ.)$_4$ = $0x1800800$
   (e) BGD position(εφ.)$_5$ = $0x2000800$
   (f) BGD position(εφ.)$_6$ = $0x2800800$
   (g) BGD position(εφ.)$_7$ = $0x3000800$

2.23 Ερώτημα 23ο

Για κάθε group στο σύστημα έχουμε ένα block bitmap το οποίο διατηρεί την πληροφορία για το ποια blocks χρησιμοποιούνται και ποια είναι ελεύθερα.

1. **tool:** Με dumpe2fs /dev/vdb βλέπουμε ότι για το Group 0 το block bitmap βρίσκεται στο block 3, για το Group 1 στο block 8195, για το Group 2 στο block 16387, για το Group 3 στο block 24579, για το Group 4 στο block 32771, για το Group 5 στο στο 40963, για το Group 6 στο 49155.
2. **hexedit:** block bitmap position$_i$ = hex(boot + $(i-1) \cdot$ bpg $\cdot$ blocksize + superblock + bgd)
   (a) block bitmap position$_1$ = $0xC00$
   (b) block bitmap position$_2$ = $0x800C00$
   (c) block bitmap position$_3$ = $0x1000C00$
   (d) block bitmap position$_4$ = $0x1800C00$
   (e) block bitmap position$_5$ = $0x2000C00$
   (f) block bitmap position$_6$ = $0x2800C00$
   (g) block bitmap position$_7$ = $0x3000C00$

Για κάθε group στο σύστημα έχουμε επίσης ένα inode bitmap το οποίο διατηρεί την πληροφορία για το ποια inodes χρησιμοποιούνται και ποια είναι ελεύθερα.

1. **tool:** Με dumpe2fs /dev/vdb βλέπουμε ότι για το Group 0 το inode bitmap βρίσκεται στο block 4, για το Group 1 στο block 8196, για το Group 2 στο block 16388, για το Group 3 στο block 24580, για το Group 4 στο block 32772, για το Group 5 στο στο 40964, για το Group 6 στο 49156.
2. **hexedit:** inode bitmap position$_i$ = hex(boot + $(i-1) \cdot$ bpg $\cdot$ blocksize + superblock + bgd + block bitmap)
   (a) inode bitmap position$_1$ = $0x1000$
   (b) inode bitmap position$_2$ = $0x801000$
   (c) inode bitmap position$_3$ = $0x1001000$
   (d) inode bitmap position$_4$ = $0x1801000$
   (e) inode bitmap position$_5$ = $0x2001000$
   (f) inode bitmap position$_6$ = $0x2801000$
   (g) inode bitmap position$_7$ = $0x3001000$

2.24 Ερώτημα 24ο

Είναι ένας πίνακας από inodes, ένα data structure το οποίο περιέχει πληροφορίες για όλα τα αρχεία μέσα στο σύστημα, την τοποθεσία τους, το μέγεθός τους, τον τύπο τους και τα δικαιώματα χρήσης. Το όνομα του αρχείου δεν περιέχεται εδώ καθώς χρησιμοποιούνται οι αριθμοί των inodes για προσπέλαση των δεδομένων αυτών. Περιέχονται σε κάθε block group μετά τα inode bitmaps.

1. **tool:** Με dumpe2fs /dev/vdb βλέπουμε ότι για το Group 0 το inode table βρίσκεται στα blocks 5-233, για το Group 1 στα blocks 8197-8425, για το Group 2 στα blocks 16389-16617, για το Group 3 στα blocks 24581-24809, για το Group 4 στα blocks 32773-33001, για το Group 5 στα 40965-41193, για το Group 6 στα 49157-49385.
2. **hexedit:** inode bitmap position$_i$ = $hex(\text{boot} + (i-1) \cdot \text{bpg} \cdot \text{blocksize} + \text{superblock} + \text{bgd} + \text{block bitmap} + \text{inode bitmap})$
   (a) inode table position$_1$ = $0x1400$
   (b) inode table position$_2$ = $0x801400$
   (c) inode table position$_3$ = $0x1001400$
   (d) inode table position$_4$ = $0x1801400$
   (e) inode table position$_5$ = $0x2001400$
   (f) inode table position$_6$ = $0x2801400$
   (g) inode table position$_7$ = $0x3001400$

2.25 Ερώτημα 25ο

Το inode περιέχει πληροφορίες σχετικά με το αρχείο, πιο συγκεκριμένα με τις άδειες, τον τύπο, το user id, το μέγεθος του αρχείου, χρόνους δημιουργίας, τελευταίας αλλαγής, τελευταίας προσπέλασης, διαγραφής, το group id, τον αριθμό των hardlinks, των αριθμό των sectors που χρησιμοποιεί το inode, flags, operation system specific value, και δείκτες σε blocks. Τα inodes αποθηκεύονται μέσα στο inode table.

2.26 Ερώτημα 26ο

1. **tool:** Με χρήση της εντολής dumpe2fs -h /dev/vdb έχουμε 8192 blocks/group και 1832 inodes/group.

2. **hexedit:** Στα offset του superblock 0-3 έχουμε τον συνολικό αριθμό inode στο σύστημα αρχείων ίσο με 0x3218 = 12824, και γνωρίζουμε ότι έχουμε 7 groups άρα 1832 inodes/group. Όμοια στα offset 4-7 έχουμε τον συνολικό αριθμό των blocks ο οποίος είναι 0xC800 = 51200, για 7 groups προκύπτει λοιπόν 8192 blocks/group. Επίσης η πληροφορία υπάρχει στα offset 32-35 για τα blocks ανά group και στα offset 40-43 για τα inodes ανά group.

2.27 Ερώτημα 27ο

1. **tool:** Με μερικές εντολές:

   ```
   mkdir /tmp/disk1
   mount /dev/vdb /tmp/disk1/
   cd /tmp/disk1/dir2/
   stat helloworld
   ```

   Προκύπτει ότι το inode του αρχείου είναι 9162.

2. **hexedit:** Θα ξεκινήσουμε από το Block Group Descriptor το οποίο περιέχει για κάθε block group τις θέσεις για τις τοποθεσίες των block bitmaps, των inode bitmaps, των inode tables και άλλα. Αρχικά το Block Group Descriptor βρίσκεται στη θέση 0x800, αφού πρώτα είναι ο bootsector και ύστερα το superblock. Στο offset 8-11 του Block Group Descriptor υπάρχει η πληροφορία για το αρχικό block του inode table, το οποίο είναι το 5, άρα πάμε στη θέση $5 \cdot$ blocksize του συστήματος αρχείων. Εδώ μπορούμε να δούμε όλη τη δομή ενός inode, και τα 128 bytes. Εμείς γνωρίζουμε ότι το inode 2 είναι το root directory άρα αυτό θα μελετήσουμε για να βρούμε το dir2 και τελικά το αρχείο helloworld. Άρα κοιτάμε το inode στη θέση

0x1480. Στα offset 40-43 μπορούμε να δούμε το block στο οποίο δείχνεί το root directory μας, το οποίο είναι το 234, άρα πάμε στη θέση $234 \cdot 1024$ η οποία είναι η 0x3A800. Στη θέση αυτή μπορούμε να διακρίνουμε κάποια directory entry blocks. Το directory entry block που μας ενδιαφέρει είναι αυτό του dir2 το οποίο έχει inode ίσο με 0x23C9 το οποίο είναι ίσο με 9161, αλλά γνωρίζουμε ότι κάθε BG περιέχει 1832 inodes, άρα το συγκεκριμένο inode είναι το πρώτο inode στο BG5. Άρα πρέπει να πάμε στη θέση 0x2801400 για να βρούμε το inode του dir2. Το πρώτο inode μας παραπέμπει στο block 0x7F, το οποίο είναι στη θέση 0x3dc00 και εδώ βρίσκουμε ένα νέο directory entry block με όνομα helloworld και inode 0x23CA το οποίο είναι το δεύτερο inode στο BG5 με αριθμό 9162. Το δεύτερο directory entry βρίσκεται στο block 0x401, άρα στη θέση 0x100400. Πηγαίνοντας στη θέση αυτή μπορούμε να δούμε το κείμενο "Welcome to the Mighty World of Filesystems". Το αρχείο αναγράφεται σε 42 bytes.

2.28 Ερώτημα 28ο

Για να βρούμε το block group ενός inode αρκεί η πράξη (inode − 1)/inodes per group, επομένως για το συγκεκριμένο inode το οποίο είναι ίσο με 9162, το block group του είναι το 5.

2.29 Ερώτημα 29ο

1. **tool:** Με χρήση dumpe2fs /dev/vdb για το group 5, το inode table ξεκινάει στο block 40965.
2. **hexedit:** Στο ερώτημα 27 αποδείξαμε ότι το inode table του inode 9162 είναι το 40965.

2.30 Ερώτημα 30ο

tool:

```
root@utopia:/tmp/disc1/dir2# stat /tmp/disc1/dir2/helloworld
  File: /tmp/disc1/dir2/helloworld
  Size: 42              Blocks: 2          IO Block: 1024   regular file
Device: fe10h/65040d    Inode: 9162        Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2023-12-12 17:23:16.000000000 +0200
Modify: 2023-12-12 17:23:16.000000000 +0200
Change: 2023-12-12 17:23:16.000000000 +0200
 Birth: -
root@utopia:/tmp/disc1/dir2#
```

hexedit:

```
02801480   A4 81 00 00  2A 00 00 00  E4 7A 78 65  E4 7A 78 65  ....*....zxe.zxe
02801490   E4 7A 78 65  00 00 00 00  00 00 01 00  02 00 00 00  .zxe............
028014A0   00 00 00 00  01 00 00 00  01 04 00 00  00 00 00 00  ................
028014B0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
028014C0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
028014D0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
028014E0   00 00 00 00  BC F3 45 A3  00 00 00 00  00 00 00 00  ......E.........
028014F0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

2.31 Ερώτημα 31ο

Τα δεδομένα του inode αυτού αποδείξαμε στο ερώτημα 27 ότι είναι αποθηκευμένα στο block 0x401 δηλαδή στο 1025.

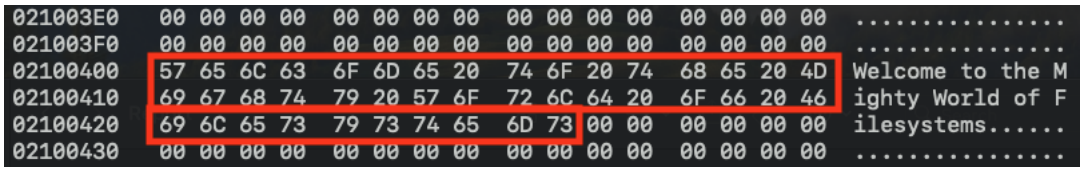
2.32 Ερώτημα 32ο

1. **tool:** Με χρήση της εντολής stat helloworld βλέπουμε ότι το μέγεθος του αρχείου είναι 42 bytes.
2. **hexedit:** Στο ερώτημα 27 αποδείξαμε ότι το μέγεθος του αρχείου είναι 42 bytes.

2.33 Ερώτημα 33ο

1. **tool:** Με χρήση της εντολής cat helloworld το περιεχόμενο του αρχείου είναι "Welcome to the Mighty World of Filesystems".

2. **hexedit:** Στο ερώτημα 27 αποδείξαμε ότι το περιεχόμενο του αρχείου είναι το "Welcome to the Mighty World of Filesystems".



3 FS Disk 2

3.1 Ερώτημα 1ο

Με την εντολή "mount /dev/vdb /mnt" προσαρτήσαμε το δίσκο στον κατάλογο mnt.

3.2 Ερώτημα 2ο

Με την εντολή "touch /mnt/file1" δημιουργούμε ένα νέο αρχείο.

3.3 Ερώτημα 3ο

Η εντολή απέτυχα καθώς δεν υπάρχει χώρος στη συσκευή. Με την εντολή "dumpe2fs /dev/vdb" παρατηρούμε ότι δεν υπάρχουν διαθέσιμα inodes στο παρόν σύστημα αρχείων.

3.4 Ερώτημα 4ο

Η κλήση συστήματος που απέτυχε ήταν η open() με κωδικό -ENOSPC.

```
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=3369792, ...}) = 0
mmap(NULL, 3369792, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f004d849000
close(3)                                = 0
openat(AT_FDCWD, "file1", O_WRONLY|O_CREAT|O_NOCTTY|O_NONBLOCK, 0666) = -1 ENOSP
C (No space left on device)
utimensat(AT_FDCWD, "file1", NULL, 0)   = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/locale.alias", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=2996, ...}) = 0
read(3, "# Locale name alias data base.\n#"..., 4096) = 2996
read(3, "", 4096)                       = 0
close(3)                                = 0
openat(AT_FDCWD, "/usr/share/locale/en_US/LC_MESSAGES/coreutils.mo", O_RDONLY) =
 -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1
 ENOENT (No such file or directory)
write(2, "touch: ", 7touch: )                   = 7
write(2, "cannot touch 'file1'", 20cannot touch 'file1')    = 20
openat(AT_FDCWD, "/usr/share/locale/en_US/LC_MESSAGES/libc.mo", O_RDONLY) = -1 E
NOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOE
NT (No such file or directory)
write(2, ": No space left on device", 25: No space left on device) = 25
```

3.5 Ερώτημα 5ο

1. **tool:** Με χρήση της εντολής dumpe2fs /dev/vdb μπορούμε να δούμε ότι τα συνολικά directories μέσα στο σύστημα αρχείων είναι 84+92+83=259. Αλλιώς μπορούμε με την εντολή "find . -type d — wc -l" να δούμε το ίδιο αποτέλεσμα. Σχετικά με τα αρχεία, η εντολή "find . -type f — wc -l" θα μας δείξει ότι έχουμε συνολικά 4868. Να σημειώσουμε ότι συνολικά τα inodes είναι 5136, αλλά εμείς έχουμε χρησιμοποιήσει για αρχεία και directories τα 5127(λαμβάνοντας υπόψην τα root,lost+found directories). Τα υπόλοιπα 9 είναι δεσμευμένα από το ΣΑ.

2. **hexedit:** Με hexedit, θα οδηγηθούμε στη θέση 0x800, για να δούμε το Block Group Descriptor. Για κάθε block group, θα πάμε στο offset 16-17, για να βρούμε τον αριθμό των directories μέσα στο block group.

   (a) Για το Group 0, έχουμε 0x54 = 84 directories.

   (b) Για το Group 1, έχουμε 0x53 = 83 directories.

   (c) Για το Group 2, έχουμε 0x5C = 92 directories.

Στο offset 0-3 στο superblock μπορούμε να βρούμε τον συνολικό αριθμό inodes ο οποίος είναι ίσος με 0x1410 = 5136. Στο offset 16-19 στο superblock πάλι μπορούμε να βρούμε τον συνολικό αριθμό ελεύθερων inodes, ο οποίος είναι ίσος με μηδέν. Επομένως, γνωρίζουμε ότι χρησιμοποιούνται όλα τα inodes του ΣΑ, άρα ο αριθμός των αρχείων θα είναι ίσος με: 5136 - δεσμευμένα από ΣΑ - αριθμός directories + 2 = 4868 αρχεία.

3.6 Ερώτημα 6ο

1. **tool:** Με χρήση της εντολής df -h /dev/vdb βλέπουμε ότι τα δεδομένα καταλαμβάνουν χώρο ίσο με 270Kbytes από τα 20 διαθέσιμα MB.

2. **hexedit:** Στο offset 4-7 του superblock υπάρχει η πληροφορία της συνολικής ποσότητας των block μέσα στο fs, ενώ στο offset του superblock 12-15 υπάρχουν τα διαθέσιμα blocks, επομένως συνολικά καταλαμβάνονται 925 blocks μέσα σε αυτό το σύστημα αρχείων, σαφώς όχι πλήρως, αλλά ένα μέρος των bit τους μόνο, το οποίο δεν είναι εφικτό να βρούμε με χρήση της πληροφορίας που μας δίνει η δομή ενός ext2.

3.7 Ερώτημα 7ο

1. **tool:** Είδαμε παραπάνω με τη χρήση της εντολής df ότι συνολικά το αρχείο είναι 20MB.

2. **hexedit:** Με χρήση του hexedit βλέπουμε κάτω αριστερά ότι το μέγεθος του αρχείου είναι ίσο με 0x1400000 = 20MB.

3.8 Ερώτημα 8ο

1. **tool:** Με χρήση dumpe2fs βλέπουμε ότι τα διαθέσιμα blocks είναι 19555. Με την εντολή df βλέπουμε ότι ο mounted δίσκος έχει διαθέσιμα 19555 blocks το οποίο είναι το 98% του συνολικού χώρου.

2. **hexedit:** Με hexedit, μπορούμε στα offset 12-15 του superblock να δούμε τον συνολικό αριθμό των διάθεσιμων blocks ο οποίος είναι ίσος με 0x4C63 = 19555.

3.9 Ερώτημα 9ο

1. **tool:** Με χρήση dumpe2fs βλέπουμε ότι δεν υπάρχουν διαθέσιμα inodes.

2. **hexedit:** Με χρήση hexedit μπορούμε να δούμε ότι στο offset 16-19 τα διαθέσιμα inodes είναι μηδέν.

4 FS Disk 3

4.1 Ερώτημα 1ο

Το εργαλείο που χρησιμοποιούμε είναι το fsck για επιδιορθώσεις στο σύστημα αρχείων.

4.2 Ερώτημα 2ο

Λόγοι για τους οποίος μπορεί να αποτύχει ένα σύστημα αρχείων είναι οι εξής:

1. Αποτυχία του hardware.
2. Αποτυχία του λογισμικού.
3. Λάθος shutdowns του συστήματος ή απώλεια ενέργειας κατά την διάρκεια ενός I/O operation.
4. Επιθέσεις από ιούς και λογισμικά.
5. Περιπτώσεις όπου το σύστημα αρχείων είναι γεμάτο, το οποίο μπορεί να προκαλέσει αποτυχία στην προσπέλαση δεδομένων.
6. Ο τρόπος χρήσης του συστήματος αρχείων.
7. Ανθρώπινα σφάλματα.
8. Ασύμβατοι οδηγοί συσκευών.

Πιθανές αλλοιώσεις ενός συστήματος αρχείων είναι οι εξής:

1. Φυσική αλλοίωση του δίσκου.
2. Αλλαγή σε bootsector.
3. Αλλαγή σε superblock.
4. Αλλαγή σε block group descriptor.
5. Αλλαγή σε block bitmap.
6. Αλλαγή σε inode bitmap.
7. Αλλαγή σε inode table.
8. Αλλαγή σε data block.
9. Αλλαγή σε δεσμευμένο directory entry.
10. Διαγραφή data block.
11. Αργή απόδοση.
12. Αλλοίωση των μεταδεδομένων.
13. Πολλαπλά αρχεία μοιράζονται το ίδιο block.

4.3 Ερώτημα 3ο

Εξαντλητικά όλα τα σφάλματα που εντόπισε το εργαλείο είναι τα εξής:

1. First entry "BOO" (inode=1717) in directory inode 1717 (/dir-2) should be "."
2. Inode 3425 ref count is 1, should be 2.
3. Block bitmap differences: +34
4. Free blocks count wrong for group #0 (7960, counted=7961)
5. Free blocks count wrong (19800, counted=19801)

4.4 Ερώτημα 4ο

1. Το πρώτο πρόβλημα που προκύπτει είναι ότι το πρώτο entry του /dir-2 είναι το "BOO" ενώ θα έπρεπε να είναι το "." δηλαδή το ίδιο το directory. Αρχικά πρέπει να βρούμε σε ποιο block group βρίσκεται αυτό το inode, το οποίο είναι το 1 και το index του, το οποίο είναι το 5. Ο navigator μας δίνει την θέση του inode table στο πρώτο group. Είναι στη θέση 0x801400. Το πέμπτο inode βρίσκεται στη block 0x20DC, δηλαδή στο 8412. Η θέση αυτή είναι η 0x837000. Πλέον μπορούμε να δούμε τα directory entries και το πρώτο είναι το "BOO" αλλά θα έπρεπε να το ".". Άρα πρέπει να αλλάξουμε τα bytes 0x837008-0x83700A από 42 4F 4F σε 2E 00 00. Πρέπει επίσης να αλλάξουμε το μέγεθος του ονόματος από τρία bytes σε ένα byte στη θέση 0x837006.

2. Το inode 3425 είναι το πρώτο inode στο group 2. Στο offset 26-27 έχουμε τον αριθμό των hardlinks ο οποίος πρέπει να είναι ίσος με 2 αλλά είναι ίσος με 1. Επομένως αλλάζουμε την τιμή αυτή στη σωστή τιμή.

3. Στο πρώτο block bitmap στη θέση 0xC04 πρέπει να κάνουμε allocate to block 34, επομένως αλλάζουμε το D με F.

4. Με την διόρθωση του block bitmap διορθώθηκε και αυτό το πρόβλημα αφού πλέον είναι allocated ένα παραπάνω block.

5. Με την διόρθωση του block bitmap διορθώθηκε και αυτό το πρόβλημα αφού πλέον είναι allocated ένα παραπάνω block.

4.5 Ερώτημα 5ο

Στο dry run το πρόγραμμα τρέχει χωρίς κανένα σφάλμα.

5 Μέρος 2ο

5.1 Υλοποίηση των init_ext2_fs και exit_ext2_fs στο αρχείο super.c

```
static int __init init_ext2_fs(void)
{
        int err = init_inodecache();
        if (err)
                return err;
        /* Register ext2-lite filesystem in the kernel */
        /* If an error occurs remember
    to call destroy_inodecache() */
        /* ? */
    err = register_filesystem(&ext2_fs_type);
    if (err) {
        destroy_inodecache();
        return err;
    }
    return 0;
}

static void __exit exit_ext2_fs(void)
{
        /* Unregister ext2-lite filesystem from the kernel */
        /* ? */

    unregister_filesystem(&ext2_fs_type);
        destroy_inodecache();
}
```

5.2 Υλοποίηση της ext2_find_entry στο αρχείο dir.c

```
ext2_dirent *ext2_find_entry(struct inode *dir, const struct qstr *child,
                             struct page **res_page)
{
        const char *name = child->name;
        int namelen = child->len;
    // Directory record length
        unsigned reclen = EXT2_DIR_REC_LEN(namelen);
        unsigned long npages = dir_pages(dir);
        unsigned long i;
        struct page *page = NULL;
        ext2_dirent *de;
        char *kaddr;

        if (npages == 0)
                return ERR_PTR(-ENOENT);

        *res_page = NULL;

        /* Scan all the pages of the directory to find the requested name. */
        for (i=0; i < npages; i++) {
                /* ? */
        // Read pages of a catalog
        page = ext2_get_page(dir,i,0);

        // If reading fails
        if (IS_ERR(page))
            return ERR_CAST(page);
```

```
        // Return address of address space for the given page
        kaddr = page_address(page);

        // Return last byte of page
        de = (ext2_dirent *) kaddr;
        kaddr += ext2_last_byte(dir, i) - reclen;
        while((char *) de <= kaddr) {
            // Zero length of directory entry
            if(de->rec_len == 0) {
                ext2_error(dir->i_sb,__func__, "zero-length directory entry");
                ext2_put_page(page);
                return ERR_PTR(-ENOENT);
            }

            // If i find it
            if (ext2_match(namelen, name, de)) {
                // Release page
                // ext2_put_page(page);
                *res_page = page;
                return de;
            }

            // Next entry
            de = ext2_next_entry(de);
        }

        // Release page
        ext2_put_page(page);
        }
        return ERR_PTR(-ENOENT);
}
```

5.3 Υλοποίηση της ext2_get_inode στο αρχείο inode.c

```
static struct ext2_inode *ext2_get_inode(struct super_block *sb, ino_t ino,
                                         struct buffer_head **p)
{
        struct buffer_head *bh;
        unsigned long block_group;
        unsigned long block;
        unsigned long offset;
        struct ext2_group_desc *gdp;
        unsigned long inodes_pg = EXT2_INODES_PER_GROUP(sb);
        int inode_sz = EXT2_INODE_SIZE(sb);
        unsigned long blocksize = sb->s_blocksize;

        *p = NULL;
        /* Check the validity of the given inode number. */
        if ((ino != EXT2_ROOT_INO && ino < EXT2_FIRST_INO(sb)) ||
            ino > le32_to_cpu(EXT2_SB(sb)->s_es->s_inodes_count))
                goto einval;

        /* Figure out in which block is the inode we are looking for and get
         * its group block descriptor. */
        /* ? */
    block_group = (ino -1)/inodes_pg;
    //                  group descriptor             block group
    gdp = ext2_get_group_desc(sb,block_group,NULL);
    //
    if (!gdp)
```

```
        goto eio;

        /* Figure out the offset within the block group inode table */
        /* ? */
    offset = ((ino - 1)%inodes_pg)*inode_sz;
    block = le32_to_cpu(gdp->bg_inode_table) + (offset >> EXT2_BLOCK_SIZE_BITS(sb));
        if (!(bh = sb_read(sb, block)))
                goto eio;

        /* Return the pointer to the appropriate ext2_inode */
        /* ? */
        *p = bh;
        offset &= (blocksize - 1);
        return (struct ext2_inode *) (bh->b_data + offset);

einval:
        ext2_error(sb, __func__, "bad inode number: %lu", (unsigned long)ino);
        return ERR_PTR(-EINVAL);
eio:
        ext2_error(sb, __func__, "unable to read inode block - inode=%lu, block=%lu",
                   (unsigned long)ino, block);
        return ERR_PTR(-EIO);
}
```

5.4 Υλοποίηση της ext2_iget στο αρχειο inode.c

```
struct inode *ext2_iget(struct super_block *sb, unsigned long ino)
{
        struct ext2_inode_info *ei;
        struct buffer_head *bh = NULL;
        struct ext2_inode *raw_inode;
        struct inode *inode;
        long ret = -EIO;
        int n;

        ext2_debug("request to get ino: %lu\n", ino);

        /*
         * Allocate the VFS node.
         * We know that the returned inode is part of a bigger ext2_inode_info
         * inode since iget_locked() calls our ext2_sops->alloc_inode() function
         * to perform the allocation of the inode.
         */
        inode = iget_locked(sb, ino);
        if (!inode)
                return ERR_PTR(-ENOMEM);
        if (!(inode->i_state & I_NEW))
                return inode;

        /*
         * Read the EXT2 inode *from disk*
         */
        raw_inode = ext2_get_inode(inode->i_sb, ino, &bh);
        if (IS_ERR(raw_inode)) {
                ret = PTR_ERR(raw_inode);
                brelse(bh);
                iget_failed(inode);
                return ERR_PTR(ret);
        }
```

```
/*
 * Fill the necessary fields of the VFS inode structure.
 */
inode->i_mode = le16_to_cpu(raw_inode->i_mode);
i_uid_write(inode, (uid_t)le16_to_cpu(raw_inode->i_uid));
i_gid_write(inode, (gid_t)le16_to_cpu(raw_inode->i_gid));
set_nlink(inode, le16_to_cpu(raw_inode->i_links_count));
inode->i_atime.tv_sec = (signed)le32_to_cpu(raw_inode->i_atime);
inode->i_ctime.tv_sec = (signed)le32_to_cpu(raw_inode->i_ctime);
inode->i_mtime.tv_sec = (signed)le32_to_cpu(raw_inode->i_mtime);
inode->i_atime.tv_nsec = 0;
inode->i_mtime.tv_nsec = 0;
inode->i_ctime.tv_nsec = 0;
inode->i_blocks = le32_to_cpu(raw_inode->i_blocks);
inode->i_size = le32_to_cpu(raw_inode->i_size);
if (i_size_read(inode) < 0) {
        ret = -EUCLEAN;
        brelse(bh);
        iget_failed(inode);
        return ERR_PTR(ret);
}
//> Setup the {inode,file}_operations structures depending on the type.
if (S_ISREG(inode->i_mode)) {
        /* ? */
inode->i_op = &ext2_file_inode_operations;
inode->i_fop = &ext2_file_operations;
inode->i_mapping->a_ops = &ext2_aops;

} else if (S_ISDIR(inode->i_mode)) {
        /* ? */
inode->i_op = &ext2_dir_inode_operations;
inode->i_fop = &ext2_dir_operations;
inode->i_mapping->a_ops = &ext2_aops;

} else if (S_ISLNK(inode->i_mode)) {
        if (ext2_inode_is_fast_symlink(inode)) {
                inode->i_op = &simple_symlink_inode_operations;
                inode->i_link = (char *)ei->i_data;
                nd_terminate_link(ei->i_data, inode->i_size,
                    sizeof(ei->i_data) - 1);
        } else {
                inode->i_op = &page_symlink_inode_operations;
                inode_nohighmem(inode);
                inode->i_mapping->a_ops = &ext2_aops;
        }
} else {
        inode->i_op = &ext2_special_inode_operations;
        if (raw_inode->i_block[0])
                init_special_inode(inode, inode->i_mode,
                   old_decode_dev(le32_to_cpu(raw_inode->i_block[0])));
        else
                init_special_inode(inode, inode->i_mode,
                   new_decode_dev(le32_to_cpu(raw_inode->i_block[1])));
}

/*
 * Fill the necessary fields of the ext2_inode_info structure.
 */
ei = EXT2_I(inode);
ei->i_dtime = le32_to_cpu(raw_inode->i_dtime);
ei->i_flags = le32_to_cpu(raw_inode->i_flags);
```

```
        ext2_set_inode_flags(inode);
        ei->i_dtime = 0;
        ei->i_state = 0;
        ei->i_block_group = (ino - 1) / EXT2_INODES_PER_GROUP(inode->i_sb);
        //> NOTE! The in-memory inode i_data array is in little-endian order
        //> even on big-endian machines: we do NOT byteswap the block numbers!
        for (n = 0; n < EXT2_N_BLOCKS; n++)
                ei->i_data[n] = raw_inode->i_block[n];

        brelse(bh);
        unlock_new_inode(inode);
        return inode;
}
```

5.5 Υλοποίηση της ext2_allocate_in_bg στο αρχείο balloc.c

```
static int ext2_allocate_in_bg(struct super_block *sb, int group,
                               struct buffer_head *bitmap_bh, unsigned long *count)
{
        ext2_fsblk_t group_first_block = ext2_group_first_block_no(sb, group);
        ext2_fsblk_t group_last_block = ext2_group_last_block_no(sb, group);
        ext2_grpblk_t nblocks = group_last_block - group_first_block + 1;
        ext2_grpblk_t first_free_bit;
        unsigned long num;

        /* ? */
    first_free_bit = find_next_zero_bit_le(bitmap_bh->b_data,
                            nblocks, group_first_block);
    if (first_free_bit >= nblocks) {
        return -1;
    }

    num = 0;

    for (; num < * count && first_free_bit < nblocks; first_free_bit++) {
        // If I don't allocate
        if (ext2_set_bit_atomic(sb_bgl_lock(EXT2_SB(sb), group),
                    first_free_bit, bitmap_bh->b_data)) {
            // Allocated nothing
            if (num == 0)
                continue;

            // Space filled
            break;
        }
        // Else +1 to the number of allocations
        num++;
    }

    if (num == 0)
        return -1;

    *count = num;
    return first_free_bit - num;
}
```

5.6 Υλοποίηση Testing

Για την υλοποίηση του testing, αρχικά είδαμε τα kernel logs για να αντιληφθούμε τυχόντα σφάλματα στον κώδικα. Όταν όλα τα σφάλματα διορθώθηκαν υλοποιήσαμε το εξής script:

```
#!/bin/bash

cd /mnt

for i in {1..5}
do
mkdir dir$i
        cd dir$i
        for j in {1..3}
        do
                touch file$j
                echo "1" > file$j
                cat file$j
        done
        rm file3
        cd ..
done
rm -r dir1
```

Το filesystem λειτουργεί σωστά με όλες τις παραπάνω εντολές.

navigator

February 4, 2024

1 FS DISK 1: Constants

[18]:
```
blocksize = 1024
blocks_per_group = 8192
inodes_per_group = 1832
inode_size = 128
bootsector = 1024
superblock_size = 1024
block_group_descriptor_size = 1024
block_bitmap_size = 1024
inode_bitmap_size = 1024
number_of_groups = 7
inode_table_size = inode_size*inodes_per_group
```

2 Positions of Blocks of Interest

[5]:
```
def superblock_positions(i):
    return hex(bootsector
               +(i-1)*blocks_per_group*blocksize)

def blockGroupDescriptor_positions(i):
    return hex(bootsector
               +(i-1)*blocks_per_group*blocksize
               +superblock_size)

def blockBitMap_positions(i):
    return hex(bootsector
               +(i-1)*blocks_per_group*blocksize
               +superblock_size
               +block_group_descriptor_size)

def inodeBitMap_positions(i):
    return hex(bootsector
               +(i-1)*blocks_per_group*blocksize
               +superblock_size
               +block_group_descriptor_size
               + block_bitmap_size)
```

```
def inodeTable_positions(i):
    return hex(bootsector
               +(i-1)*blocks_per_group*blocksize
               +superblock_size
               +block_group_descriptor_size
               +block_bitmap_size
               +inode_bitmap_size)

def datablock_positions(i):
    return hex(bootsector
               +(i-1)*blocks_per_group*blocksize
               +superblock_size
               +block_group_descriptor_size
               +block_bitmap_size
               +inode_bitmap_size
               +inode_table_size)
```

3 Functions for inode searching

[16]:
```
def block_group_of_inode(inode):
    return (inode-1)//inodes_per_group

def index_of_inode(inode):
    return (inode-1)%inodes_per_group

def containing_block_of_inode(inode):
    return (index_of_inode(inode)*inode_size)//blocksize
```

3.1 Finding all blocks of interest

[8]:
```
supers = [superblock_positions(i+1) for i in range(number_of_groups)]
bgds = [blockGroupDescriptor_positions(i+1) for i in range(number_of_groups)]
bbms = [blockBitMap_positions(i+1) for i in range(number_of_groups)]
ibms = [inodeBitMap_positions(i+1) for i in range(number_of_groups)]
its= [inodeTable_positions(i+1) for i in range(number_of_groups)]
dbs = [datablock_positions(i+1) for i in range(number_of_groups)]
```

3.2 Printing blocks of interest for easy navigation

[85]:
```
for index,superblock in enumerate(supers):
    print(f'Group {index}')
    print(f'Superblock in position {superblock}')
    print(f'Block Group Descriptor in position {bgds[index]}')
    print(f'Block Bitmap in position {bbms[index]}')
    print(f'Inode Bitmap in position {ibms[index]}')
```

```
    print(f'Inode Table in position {its[index]}')
    print(f'Data Blocks in position {bds[index]}')
    print('====================================')
```

```
Group 0
Superblock in position 0x400
Block Group Descriptor in position 0x800
Block Bitmap in position 0xc00
Inode Bitmap in position 0x1000
Inode Table in position 0x1400
Data Blocks in position 0x3a800
====================================
Group 1
Superblock in position 0x800400
Block Group Descriptor in position 0x800800
Block Bitmap in position 0x800c00
Inode Bitmap in position 0x801000
Inode Table in position 0x801400
Data Blocks in position 0x83a800
====================================
Group 2
Superblock in position 0x1000400
Block Group Descriptor in position 0x1000800
Block Bitmap in position 0x1000c00
Inode Bitmap in position 0x1001000
Inode Table in position 0x1001400
Data Blocks in position 0x103a800
====================================
Group 3
Superblock in position 0x1800400
Block Group Descriptor in position 0x1800800
Block Bitmap in position 0x1800c00
Inode Bitmap in position 0x1801000
Inode Table in position 0x1801400
Data Blocks in position 0x183a800
====================================
Group 4
Superblock in position 0x2000400
Block Group Descriptor in position 0x2000800
Block Bitmap in position 0x2000c00
Inode Bitmap in position 0x2001000
Inode Table in position 0x2001400
Data Blocks in position 0x203a800
====================================
Group 5
Superblock in position 0x2800400
Block Group Descriptor in position 0x2800800
Block Bitmap in position 0x2800c00
```

```
Inode Bitmap in position 0x2801000
Inode Table in position 0x2801400
Data Blocks in position 0x283a800
====================================
Group 6
Superblock in position 0x3000400
Block Group Descriptor in position 0x3000800
Block Bitmap in position 0x3000c00
Inode Bitmap in position 0x3001000
Inode Table in position 0x3001400
Data Blocks in position 0x303a800
====================================
```

4 FS DISK 3: Constants:

[9]:
```
blocksize = 1024
blocks_per_group = 8192
inodes_per_group = 1712
inode_size = 128
bootsector = 1024
superblock_size = 1024
block_group_descriptor_size = 1024
block_bitmap_size = 1024
inode_bitmap_size = 1024
number_of_groups = 3
inode_table_size = inode_size*inodes_per_group
```

4.1 Finding all blocks of interest

[10]:
```
supers3 = [superblock_positions(i+1) for i in range(number_of_groups)]
bgds3 = [blockGroupDescriptor_positions(i+1) for i in range(number_of_groups)]
bbms3 = [blockBitMap_positions(i+1) for i in range(number_of_groups)]
ibms3 = [inodeBitMap_positions(i+1) for i in range(number_of_groups)]
its3 = [inodeTable_positions(i+1) for i in range(number_of_groups)]
dbs3 = [datablock_positions(i+1) for i in range(number_of_groups)]
```

4.2 Printing blocks of interest for easy navigation

[13]:
```
for index,superblock in enumerate(supers3):
    print(f'Group {index}')
    print(f'Superblock in position {superblock}')
    print(f'Block Group Descriptor in position {bgds3[index]}')
    print(f'Block Bitmap in position {bbms3[index]}')
    print(f'Inode Bitmap in position {ibms3[index]}')
    print(f'Inode Table in position {its3[index]}')
    print(f'Data Blocks in position {dbs3[index]}')
    print('====================================')
```

```
Group 0
Superblock in position 0x400
Block Group Descriptor in position 0x800
Block Bitmap in position 0xc00
Inode Bitmap in position 0x1000
Inode Table in position 0x1400
Data Blocks in position 0x36c00
====================================
Group 1
Superblock in position 0x800400
Block Group Descriptor in position 0x800800
Block Bitmap in position 0x800c00
Inode Bitmap in position 0x801000
Inode Table in position 0x801400
Data Blocks in position 0x836c00
====================================
Group 2
Superblock in position 0x1000400
Block Group Descriptor in position 0x1000800
Block Bitmap in position 0x1000c00
Inode Bitmap in position 0x1001000
Inode Table in position 0x1001400
Data Blocks in position 0x1036c00
====================================
```