Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών ΕΜΠ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Εργαστήριο Λειτουργικών Συστημάτων 2024 - 2025
3η εργαστηριακή άσκηση : Filesystems

Ομάδα : 12
Ονοματεπώνυμο : Φέζος Κωνσταντίνος / Α.Μ : 03118076
Οναματεπώνυμο : Τσουκνίδας Οδυσσέας Αρθούρος Ρήγας / Α.Μ : 03120043

Μέρος 1ο

Η εικόνα fsdisk1.img

1. Στο utopia.sh προσθέτουμε την γραμμή:
-drive file=fsdisk1.img,format=raw,if=virtio
Με αυτόν τον τρόπο προσθέσαμε μία νέα συσκευή στο virtual machine μας, η οποία είναι εικονική συσκευή αποθήκευσης που το QEMU την βλέπει ως άλλο ένα drive. Με την παράμετρο if ορίζουμε ότι το είδος διεπαφής είναι virtio.

2. Με την χρήση της εντολής **lsblk** παίρνουμε το παρακάτω output:

```
fd0     2:0    1    4K  0 disk
sr0    11:0    1 1024M  0 rom
vda   254:0    0   11G  0 disk
└─vda1 254:1    0   11G  0 part /
vdb   254:16   0   50M  0 disk
```

Οπότε καταλαβαίνουμε ότι ο δίσκος έχει μέγεθος 50 M.

Εναλλακτική:
Με την εντολή **hexdump -C -s 1024 -n 64 /dev/vdb** μπορούμε να δουμε την αρχή του 1ου block που βρίσκεται το superblock:

```
00000400  18 32 00 00 00 c8 00 00  00 0a 00 00 90 c1 00 00  |.2..............|
00000410  0a 32 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |.2..............|
00000420  00 20 00 00 00 20 00 00  28 07 00 00 1f c3 72 67  |. ... ..(.....rg|
00000430  1f c3 72 67 02 00 ff ff  53 ef 00 00 01 00 00 00  |..rg....S.......|
00000440
```

Στα byte 4 με 7, βρίσκουμε το block count είναι 0000c800 (είναι Little Endian, και μαρκαρισμένα με bold), δηλαδή 51.200, ενώ στα byte 24-27 βρίσκουμε πως το log2(block_size) - 10 = 0, άρα το block size είναι 2^10 = 1024. Έτσι μπορούμε να υπολογίσουμε το μέγεθος του δίσκου που είναι 1024*51200 = 52.428.800 bytes = 50 MB.

3. Με την χρήση της εντολής **blkid** μπορώ παίρνουμε το output:

```
/dev/vdb: LABEL="fsdisk1.img" UUID="c63028e5-711b-410d-a263-e7ca2b15a8d3" BLOCK_SIZE="1024" TYPE="ext2"
```

Και έτσι καταλαβαίνουμε ότι το σύστημα αρχείων που περιέχει η εικόνα είναι ext2.

4. Με χρήση της **dumpe2fs /dev/vdb** βρίσκουμε την ημερομηνία δημιουργίας του filesystem:
Filesystem created: Tue Dec 12 17:23:16 2023

Με μία αναζήτηση στο διαδίκτυο
(https://unix.stackexchange.com/questions/731531/unix-ext2-superblock-file-system-

creation-date) βρίσκουμε πως στα byte 264-7 (που είναι unused κανονικά) βρίσκεται σε POSIX time το timestamp για την δημιουργία του filesystem.
Με την εντολή **hexdump -C -s$((blocksize+ 264)) -n 4 /dev/vdb**, βρίσκουμε την τιμή e4 7a 78 65 (Little Endian), που αντιστοιχεί στον δεκαδικό: 1702394596.

Τέλος με την **date -d @1702394596** παίρνουμε: Τρι 12 Δεκ 2023 05:23:16 μμ EET
το ίδιο δηλαδή με αυτό που βρήκα παραπάνω.

5. Αντίστοιχα με το 4, η **dumpe2fs /dev/vdb** μας δίνει την ημερομηνία τελευταίας προσάρτησης του συστήματος αρχείων:
Last mount time: Mon Dec 30 17:58:23 2024

Γνωρίζουμε ακόμα πως το timestamp αυτό σε POSIX time βρίσκεται στα byte 44 με 47 του superblock, οπότε από το output που βρήκαμε στην ερώτηση 2, έχουμε: 1f c3 72 67, που σε δεκαδικό σύστημα είναι ο αριθμός 1735574303.
Και τρέχοντας λοιπόν την εντολή: **date -d @1735574303** βρίσκουμε πως αυτό αντιστοιχεί σε Δευ 30 Δεκ 2024 05:58:23 μμ EET, ότι είχαμε βρει δηλαδή και παραπάνω.

6. Με την χρήση της **dumpe2f** έχουμε:
Last mounted on: /cslab-bunker

Ακόμα, με μία αναζήτηση στο διαδίκτυο βρήκαμε την δομή του superblock του ext2 (https://www.nongnu.org/ext2-doc/ext2.html#superblock) και έτσι βρήκαμε πως στο superblock στα byte 136 υπάρχει το πεδίο s_last_mounted το οποίο έχει μήκος 64 bytes.
Έτσι εκτελούμε την εντολή **hexdump -C -s$((blocksize+136)) -n 64 /dev/vdb** και έτσι βρίσκουμε πως στο πεδίο με τους ASCII χαρακτήρες πως το σύστημα αρχείων προσαρτήθηκε τελευταία φορά στο /cslab-bunker.

7. Με dumpe2fs:
Last write time: Mon Dec 30 17:58:23 2024

Ενώ με hexdump -C -s$((blocksize+48)) -n 4 /dev/vdb, παίρνουμε την τιμή 1f c3 72 67, η οποία αν μετατραπεί με τον τρόπο που χρησιμοποιήσαμε και στις 2 παραπάνω ερωτήσεις σε timestamp είναι: Δευ 30 Δεκ 2024 05:58:23 μμ EET

8. Block σε ένα σύστημα αρχείων είναι ένα συνεχόμενο κομμάτι του δίσκου σταθερού και προκαθορισμένου μεγέθους, και αποτελεί την μικρότερη σε μέγεθος διαμέριση του φυσικού χώρου αποθήκευσης.

9. Με dumpe2fs βρίσκουμε:
Block size: 1024
ενώ με hexdump το έχουμε ήδη υπολογίσει στην ερώτηση 2.

10. Το inode είναι μία δομή στο σύστημα αρχείων που υπάρχει για κάθε αρχείο και αποθηκεύει metadata για το αρχείο (όπως πχ permissions, owner κλπ) αλλά και pointers στα blocks που είναι αποθηκευμένα τα data του αρχείου.

11. Με dumpe2fs βρίσκουμε:
Inode size: 128
ενώ αφού βρούμε στην παραπάνω σελίδα με την δομή του superblock πως το inode size βρίσκεται στα bytes 88-9 εκτελούμε την εντολή hexdump -s$((blocksize+ 88)) -n 2 -C /dev/vdb και παίρνουμε απάντηση 8000 δηλαδή 0x0080 = 128

12. Με dumpe2fs βρίσκουμε:

```
Free blocks:          49552
Free inodes:          12810
```

ενώ με **hexdump -s$((blocksize+ 12)) -n 8 -C /dev/vdb:**

```
90 c1 00 00 0a 32 00 00
```

δηλαδή 0x0000c190=0d49552 διαθέσιμα blocks, και 0x0000320a = 0d12810 διαθέσιμα inodes.

13. Το superblock στο σύστημα αρχείων ext2 είναι μία δομή που διατηρεί τα σημαντικότερα metadata για το σύστημα αρχείων.

14. Το superblock αποθηκεύεται στην αρχή κάθε block group, αν και πρέπει να σημειωθεί πως το έγκυρο superblock είναι αυτό του block group #0, τα άλλα είναι απλά αντίγραφα και μπορεί να είναι outdated.

15 Το superblock είναι δομή απαραίτητη για την χρήση του συστήματος αρχείων και συνεπώς είναι σημαντικό να διατηρούνται αντίγραφα του σε πολλά σημεία.

16 Από το output της dumpe2fs βλέπουμε μεταξύ άλλων και τα παρακάτω:

```
Group 0: (Blocks 1-8192)
  Primary superblock at 1, Group descriptors at 2-2
Group 1: (Blocks 8193-16384)
  Backup superblock at 8193, Group descriptors at 8194-8194
Group 2: (Blocks 16385-24576)
  Backup superblock at 16385, Group descriptors at 16386-16386
Group 3: (Blocks 24577-32768)
  Backup superblock at 24577, Group descriptors at 24578-24578
Group 4: (Blocks 32769-40960)
  Backup superblock at 32769, Group descriptors at 32770-32770
Group 5: (Blocks 40961-49152)
  Backup superblock at 40961, Group descriptors at 40962-40962
Group 6: (Blocks 49153-51199)
  Backup superblock at 49153, Group descriptors at 49154-49154
```

17. Στο ext2 ένα block group είναι μία σειρά από συνεχόμενα blocks που ομαδοποιούνται.

18. Ο αριθμός των block groups εξαρτάται από το μέγεθος του συστήματος αρχείων και το block size ή τα blocks per group.

19. Όπως είδαμε στην ερώτηση 16, αυτό το σύστημα αρχείων έχει 7 block groups.

20. Το block group descriptor είναι δομή η οποία περιέχει πληροφορίες για το μέρος που είναι αποθηκευμένες σημαντικές δομές δεδομένων για το block group, όπως πχ το inode bitmap και το inode table.
(https://wiki.osdev.org/Ext2#Block_Group_Descriptor_Table)

21. Τα block group descriptors είναι απαραίτητα για να προσπελάσουμε τα δεδομένα του block group, και συνεπώς είναι σημαντικό να υπάρχουν εφεδρικά αντίγραφα τους, ώστε να μην κινδυνεύουμε να χάσουμε ουσιαστικά τα δεδομένα μας.

22. Όπως μπορούμε να παρατηρήσουμε και από το output της dumpe2fs στην ερώτηση 16, τα block group descriptors είναι αποθηκευμένα σε κάθε block group, και μάλιστα στο 2ο block του κάθε group.

23. Τα block και inode bitmaps είναι bitmaps που αποθηκεύονται σε κάθε block group μετά τους block group descriptors, όπου με 0 σηματοδοτούνται τα ελεύθερα blocks και inodes, ενώ με 1 τα δεσμευμένα. Η ακριβής τους θέση προσδίδεται από τον descriptor του εκάστοτε block.

24. Τα inodes tables είναι ένας πίνακας από inodes, που αποθηκεύεται σε κάθε block group μετά τους block group descriptors και τα παραπάνω bitmaps, με την ακριβή θέση του να δίνεται από τον αντίστοιχο descriptor.

25. Τα inodes περιέχουν πεδία για το είδος και τα permissions του αρχείου, ένα User ID, πληροφορίες για το μέγεθος, σημαντικά timestamp (όπως για το creation time), το Group ID, τον αριθμό των hard links, σημαίες, pointers σε block δεδομένων και άλλα. Όλα αυτά αποθηκεύονται στα entries του inode table, το μέγεθος των οποίων καθορίζεται κατά την δημιουργία του συστήματος αρχείων.

26. Με την χρήση του **dumpe2fs** βρίσκουμε:
Blocks per group: 8192
Inodes per group: 1832

Και επειδή γνωρίζουμε πως τα blocks per group βρίσκονται στα bytes 32 με 35 στο superblock, τρέχοντας την εντολή: **hexdump -s$((blocksize + 32)) -n 4 -C /dev/vdb** παίρνουμε απάντηση 00 20 00 00, δηλαδή 0x2000=0d8192.

Αντίστοιχα για τα inodes per group: **hexdump -s$((blocksize + 40)) -n 4 -C /dev/vdb**, όπου βρίσκουμε 28 07 00 00, δηλαδή 0x728=0d1832.

27.
Αφού τρέξουμε την εντολή **mount /dev/vdb /mnt**, μπορούμε να τρέξουμε την εντολή **stat /mnt/dir2/helloworld**
και να δούμε τα παρακάτω:

```
  File: /mnt/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: -
```

Από δω καταλαβαίνουμε πως το αρχείο αυτό έχει inode 9162.

Το να βρούμε την ίδια πληροφορία με hexdump χρειάζεται παραπάνω δουλειά:

Καταρχάς γνωρίζουμε το inode του root directory το οποίο είναι πάντα #2, και αφού έχουμε 1832 inodes σε κάθε block group, καταλαβαίνουμε πως θα είναι στο BG #0 το inode του root.

Έπειτα θα ορίσουμε κατάλληλες environmental variables:

```
export blocksize=1024
export GDT_OFFSET=$((2*blocksize))
export GD_SIZE=32
```

με GDT να σηματοδοτεί το Group Descriptor table.

Θα πάμε λοιπόν να ψάξουμε σε αυτό το για να βρούμε το block στο οποίο είναι το inode table του BG #0:

```
root@utopia:~# hexdump -s$((GDT_OFFSET)) -n$((GD_SIZE)) -C /dev/vdb
00000800  03 00 00 00 04 00 00 00  05 00 00 00 08 1f 1d 07  |................|
00000810  02 00 04 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Βλέπωντας (https://www.kernel.org/doc/html/latest/filesystems/ext4/group_descr.html) πως το inode table βρίσκεται στα bytes 8-11, βλέπουμε πως το inode table του BG #0 βρίσκεται στο block #5.

Έπειτα πρέπει να πάμε να διαβάσουμε το παραπάνω inode:

```
root@utopia:~# export INODE_SIZE=128
root@utopia:~# hexdump -s$((5*blocksize+ INODE_SIZE)) -n$((INODE_SIZE)) -C /dev/vdb
00001480  ed 41 00 00 00 04 00 00  e4 7a 78 65 e4 7a 78 65  |.A.......zxe.zxe|
00001490  e4 7a 78 65 00 00 00 00  00 00 05 00 02 00 00 00  |.zxe............|
000014a0  00 00 00 00 02 00 00 00  ea 00 00 00 00 00 00 00  |................|
000014b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```
*
00001500
```

Βλέποντας ακόμα πως από το byte #40 ξεκινάει το block map, καταλαβαίνουμε πως το directory entry του root βρίσκεται στο block 0xea=0d234.

Έτσι τρέχουμε το παρακάτω:
root@utopia:~# hexdump -s$((234*blocksize)) -n$((128)) -C /dev/vdb

```
0003a800  02 00 00 00 0c 00 01 00  2e 00 00 00 02 00 00 00  |................|
0003a810  0c 00 02 00 2e 2e 00 00  0b 00 00 00 14 00 0a 00  |................|
0003a820  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 29 07 00 00  |lost+found..)...|
0003a830  0c 00 04 00 64 69 72 31  c9 23 00 00 c8 03 04 00  |....dir1.#......|
0003a840  64 69 72 32 00 00 00 00  00 00 00 00 00 00 00 00  |dir2............|
0003a850  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0003a880
```

Και ακολουθώντας την δομή του directory entry
(https://www.kernel.org/doc/html/latest/filesystems/ext4/directory.html)
δηλαδή ότι για κάθε entry έχουμε 8 bytes και μετά τους ASCII χαρακτήρες,
καταλαβαίνουμε πως για το dir2 αναφέρονται οι εξής τιμές:

```
c9 23 00 00 c8 03 04 00 64 69 72 32
```

δηλαδή το inode του directory είναι 0x23c9 = 0d9161.

Επειδή γνωρίζουμε πως το κάθε block group έχει 1831 inodes, καταλαβαίνουμε πως το inode του dir2 είναι το πρώτο inode του BG #5, του οποίου το inode table πρέπει τώρα να βρούμε με την παρακάτω εντολή:

```
root@utopia:~# hexdump -s$((GDT_OFFSET + 5*GD_SIZE)) -n$((GD_SIZE)) -C /dev/vdb
000008a0  03 a0 00 00 04 a0 00 00  05 a0 00 00 17 1f 26 07  |..............&.|
000008b0  01 00 04 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000008c0
```

Και αντίστοιχα με παραπάνω βρίσκουμε πως το inode table του BG #5 βρίσκεται στο block 0xa005 = 0d40965.

```
root@utopia:~# hexdump -s$((40965*blocksize)) -n$((INODE_SIZE)) -C /dev/vdb
02801400  ed 41 00 00 00 04 00 00  e4 7a 78 65 e4 7a 78 65  |.A.......zxe.zxe|
02801410  e4 7a 78 65 00 00 00 00  00 00 02 00 02 00 00 00  |.zxe............|
02801420  00 00 00 00 02 00 00 00  f7 00 00 00 00 00 00 00  |................|
02801430  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
02801460  00 00 00 00 c5 b2 72 92  00 00 00 00 00 00 00 00  |......r.........|
02801470  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
02801480
```

Έτσι αντίστοιχα με πάνω βρίσκουμε πως το directory entry του dir2 βρίσκεται στο block 0xf7= 0d247, και έτσι εκτελούμε την παρακάτω εντολή:

```
root@utopia:~# hexdump -s$((247*blocksize)) -n$((128)) -C /dev/vdb
```

```
0003dc00  c9 23 00 00 0c 00 01 00  2e 00 00 00 02 00 00 00  |.#..............|
0003dc10  0c 00 02 00 2e 2e 00 00  ca 23 00 00 e8 03 0a 00  |.........#......|
0003dc20  68 65 6c 6c 6f 77 6f 72  6c 64 00 00 00 00 00 00  |helloworld......|
0003dc30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0003dc80
```

Και έτσι μπορούμε επιτέλους να βρούμε το inode του hellowolrd το οποίο είναι 0x000023ca = 0d9162, όπως ακριβώς βρήκαμε και με την χρήση του stat.

28. Όπως είδαμε ήδη στο προηγούμενο ερώτημα, το inode αυτό αντιστοιχεί στο BG #5.

29. Με **dumpe2fs** μπορούμε να βρούμε τις παρακάτω πληροφορίες για το BG #5:

```
Group 5: (Blocks 40961-49152)
  Backup superblock at 40961, Group descriptors at 40962-40962
  Block bitmap at 40963 (+2)
  Inode bitmap at 40964 (+3)
  Inode table at 40965-41193 (+4)
  7959 free blocks, 1830 free inodes, 1 directories
  Free blocks: 41194-49152
  Free inodes: 9163-10992
```

Έτσι καταλαβαίνουμε ότι το inode table του βρίσκεται στα blocks 40965 με 41193.

Όσον αφορά το hexdump, την πληροφορία αυτή την έχουμε ήδη βρει στο 27 ερώτημα, όπου βρήκαμε πως το inode table του BG#5 ξεκινάει από το 0d40965, και γνωρίζοντας πως inodes per group = 1832, block size=1024 και inode = 128, μπορούμε να βρούμε πως χρειάζονται 229 blocks για το inode table, άρα αυτό θα βρίσκεται εώς και το block 40965+229-1 = 41193, όπως ακριβώς βρήκαμε και με το higher level εργαλείο.

30.Για να δούμε κάποια από τα πεδία του inode μπορούμε να χρησιμοποιήσουμε το εργαλείο stat:

```
root@utopia:~# stat /mnt/dir2/helloworld
  File: /mnt/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: -
```

Παρόλα αυτά εάν τα θέλουμε όλα τα πεδία του inode χρειάζεται να καταφύγουμε στο hexdump.

Το inode βρήκαμε ότι είναι το 9162, άρα είναι το δεύτερο inode του BG#5, και γνωρίζουμε ήδη ότι το inode table του BG#5 βρίσκεται στο block 40965. Οπότε τρέχουμε την παρακάτω εντολή:

```
root@utopia:~# hexdump -s$((40965*blocksize +INODE_SIZE)) -n$((INODE_SIZE)) -C /dev/vdb
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  |................|
*
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  |................|
02801500
```

Την ακριβής αντιστοίχιση bytes με fields μπορούμε να την κάνουμε εύκολα βασιζόμενοι στην δομή του inode στο ext2, την οποία μπορούμε να βρούμε στην παρακάτω τοποθεσία: https://www.nongnu.org/ext2-doc/ext2.html#inode-table

31. Παρατηρώντας τα bytes 40-43 στο παραπάνω inode καταλαβαίνουμε πως το block που είναι αποθηκευμένα τα δεδομένα του αρχείου είναι το block 0x401=0d1025.

32. Ακόμα, από τα bytes 4 με 6 στο inode, βρίσκουμε πως το μέγεθος του αρχείου είναι 0x21=0d42 bytes, κάτι που μπορούμε να βρούμε εύκολα και με την εντολή **stat /mnt/dir2/helloworld**, η οποία μας δίνει Size: 42.

33. Η high level επιλογή είναι η εντολή **cat**:

```
root@utopia:~# cat /mnt/dir2/helloworld
Welcome to the Mighty World of Filesystemsroot@utopia:~#
```

Και η low level είναι να πάμε να δούμε τα δεδομένα στο block 1025, που βρήκαμε στην ερώτηση 31. Τρέχουμε λοιπόν το παρακάτω:

```
root@utopia:~# hexdump -s$((1025*blocksize)) -n$((42)) -C /dev/vdb
00100400  57 65 6c 63 6f 6d 65 20  74 6f 20 74 68 65 20 4d  |Welcome to the M|
00100410  69 67 68 74 79 20 57 6f  72 6c 64 20 6f 66 20 46  |ighty World of F|
00100420  69 6c 65 73 79 73 74 65  6d 73                    |ilesystems|
0010042a
```

Η εικόνα fsdisk2.img

1. Προσθέτουμε δηλαδή στο script την γραμμή:
-drive file=fsdisk2.img,format=raw,if=virtio
και τρέχουμε την εντολή **lsblk -f** για να βρούμε που είναι το filesystem και να τσεκάρουμε ότι έχει συνδεθεί σωστά, και τέλος την εντολή: **mount /dev/vdc /mnt**

2. Προσπαθούμε να τρέξω την εντολή touch /mnt/file1, ώστε να δημιουργήσουμε το νέο αρχείο στο νέο σύστημα αρχείων και όχι σε αυτό που βρίσκεται το root directory.

3. Όχι δεν πέτυχε η εντολή.

```
root@utopia:~# touch /mnt/file1
touch: cannot touch '/mnt/file1': No space left on device
```

4. Με την χρήση της εντολής strace μπορούμε να βρούμε ότι το πρόβλημα υπήρξε σε openat κλήσεις συστήματος. Συγκεκριμένα:

```
openat(AT_FDCWD, "/mnt/file1", O_WRONLY|O_CREAT|O_NOCTTY|O_NONBLOCK, 0666) =
-1 ENOSPC (No space left on device)
```

Άρα ο κωδικός αποτυχίας είναι ENOSPC.

5. Με την χρήση της εντολής **find** μπορούμε να βρούμε τον αριθμό των αρχείων (4868) και τον αριθμό των directories (259).

```
root@utopia:~# find /mnt/ -type f | wc -l
4868
root@utopia:~# find /mnt/ -type d | wc -l
259
```

Για να βρούμε τις παρακάτω πληροφορίες και με hexdump θα πρέπει αρχικά να βρούμε πόσα block groups έχει το σύστημα αρχείων και μετά να βρούμε από το GDT (global descriptor table) τον αριθμό των directories σε κάθε ένα από αυτά.

Με την παρακάτω εντολή βρίσκουμε από το superblock των αριθμό των block per group:

```
root@utopia:~# hexdump -s$((blocksize+32)) -n 4 -C /dev/vdc
00000420  00 20 00 00                                       |. ..|
00000424
```

Άρα έχουμε 0x2000=0d8192 groups per group.

Έπειτα βρίσκουμε τον συνολικό αριθμό blocks, που είναι 0x5000=0d20480

```
root@utopia:~# hexdump -s$((blocksize+4)) -n 4 -C /dev/vdc
00000404  00 50 00 00                                       |.P..|
00000408
```

Πλέον μπορούμε να καταλάβουμε συνεπώς πως στο σύστημα αρχείων έχουμε 20480/8192 = 2.5, άρα συνολικά 3 block groups.

Στην συνέχεια συμβουλευόμαστε την παρακάτω σελίδα για την δομή των group descriptors: https://www.kernel.org/doc/html/latest/filesystems/ext4/group_descr.html

Έτσι βρίσκουμε τον αριθμό των directories διαδοχικά για τα block groups 0,1 και 2 διαδοχικά με τις παρακάτω εντολές και τα προσθέτουμε:

```
root@utopia:~# hexdump -s$((GDT_OFFSET+16)) -n 2 -C /dev/vdc
00000810  54 00                                             |T.|
00000812
root@utopia:~# hexdump -s$((GDT_OFFSET+GD_SIZE +16)) -n 2 -C /dev/vdc
00000830  53 00                                             |S.|
00000832
root@utopia:~# hexdump -s$((GDT_OFFSET+2*GD_SIZE +16)) -n 2 -C /dev/vdc
00000850  5c 00                                             |\.|
00000852
```

Άρα έχουμε συνολικά 0x54+0x53+0x5c = 0d84+0d83 + 0d92 = 0d259 directories, όσα ακριβώς είχαμε βρει και με την find δηλαδή.

Τώρα θέλουμε να βρούμε τον αριθμό των συνολικών inodes, και των ελεύθερων, ώστε να βρούμε πόσα συνολικά αρχεία (κάθε μορφής υπάρχουν). Τρέχουμε τις παρακάτω εντολές για να βρουμε τις πληροφορίες στο superblock:

```
root@utopia:~# hexdump -s$((blocksize)) -n 4 -C /dev/vdc
00000400  10 14 00 00                                       |....|
00000404
root@utopia:~# hexdump -s$((blocksize+16)) -n 4 -C /dev/vdc
00000410  00 00 00 00                                       |....|
00000414
```

Έχουμε συνεπώς χρησιμοποιούμενα inodes = 0x1410 - 0x0 = 0d5136.

Από αυτά γνωρίζουμε ήδη πως 259 είναι directories. Κάνοντας την αφαίρεση βρίσκουμε πως υπάρχουν 4877 αρχεία. Προφανώς υπάρχουν και 9 systems reserved αρχεία που αν τα αφαιρέσουμε και αυτά θα βρω 4868 συνολικά αρχεία που είναι και ο αριθμός που περιμένουμε από την find. Παρόλα αυτά δεν βρήκαμε τρόπο να βρούμε με hexdump την παραπάνω πληροφορία.

6. Με την χρήση της df μπορούμε να δούμε πως από τα 20MB του συστήματος, χρησιμοποιούνται μόνο 270KB.

```
root@utopia:~# df -h /dev/vdc
Filesystem      Size  Used Avail Use% Mounted on
/dev/vdc         20M  270K   20M   2% /mnt
```

Για τα metadata γνωρίζουμε ήδη ότι έχουμε 3 blocks για το superblock και τα αντίγραφα του, καθώς και 3 blocks για το GDT, που αποθηκεύεται σε κάθε Block group.

Μας μένει να βρούμε τα blocks που δαπανώνται για την αποθήκευση των block και inode bitmaps, καθώς και των inode tables, που μπορώ να τα βρούμε στο GDT. Πχ για το block

group #0 έχουμε:

```
root@utopia:~# hexdump -s$((GDT_OFFSET)) -n$((GD_SIZE)) -C /dev/vdc
00000800  03 00 00 00 04 00 00 00  05 00 00 00 6b 1e 00 00  |............k...|
00000810  54 00 04 00 00 00 00 00  00 00 00 00 00 00 00 00  |T...............|
00000820
```

και συμβουλευόμενοι την δομή του (https://www.nongnu.org/ext2-doc/ext2.html#block-group-descriptor-table) μπορούμε να βρούμε ότι χρησιμοποιούνται 2 blocks για τα 2 bitmaps, καθώς και 214 blocks για το inode table. Συνολικά 216.

Αντίστοιχα βρίσκουμε για το BG #1 και #2: 2 + 214 = 216 blocks.

Συνολικά δηλαδή 3+3+3*216 = 654 blocks, δηλαδή 654*1024 = 654 KB.

Για τα data:
Γνωρίζουμε ήδη τον συνολικό αριθμό blocks 20480, και άρα μας μένει να βρούμε τον αριθμό των ελεύθερων blocks:

```
root@utopia:~# hexdump -s$((blocksize + 12)) -n 4 -C /dev/vdc
0000040c  63 4c 00 00                                       |cL..|
00000410
```

Άρα used blocks = 0d20480 - 0x4c63 - 1(boot sector) - #metadata_blocks = 0d20480 - 0d19555 -0d1 -0d654 = 0d270 blocks. Και συνεπώς 270 KB είναι τα δεδομένα.

7. Όπως μπορούμε να δούμε και από το output της df, το σύστημα μας έχει μέγεθος 20MB.

Με hexdump μπορούμε να βρούμε τον αριθμό των blocks και το block size (εδώ 1024), από το superblock (που μπορούμε να το βρούμε σε offset 1024 στο file system, ανεξαρτήτως block size) να τα πολλαπλασιάσουμε:

Με την παρακάτω εντολή βρίσκουμε πως ο συνολικός αριθμός blocks είναι 0x5000 = 0d20480

```
root@utopia:~# hexdump -s$((1024 + 4)) -n 4 -C /dev/vdc
00000404  00 50 00 00                                       |.P..|
00000408
```

Άρα συνεπώς έχουμε συνολικό μέγεθος: 20480*1024 = 20971520 bytes = 20 MB, όπως βρήκαμε και προηγουμένος.

8. Με dumpe2fs βρίσκουμε:
Free blocks: 19555

Με hexdump βρίσκουμε επίσης 0x4c63 =0d19555

```
root@utopia:~# hexdump -s$((blocksize+12)) -n 4 -C /dev/vdc
0000040c  63 4c 00 00                                       |cL..|
00000410
```

9. Με την dumpe2fs βλέπουμε ακόμα:
Free inodes: 0

Αντίστοιχο αποτέλεσμα βρίσκουμε με hexdump:

```
root@utopia:~# hexdump -s$((blocksize+16)) -n 4 -C /dev/vdc
00000410  00 00 00 00                                       |....|
00000414
```

Καταλαβαίνουμε δηλαδή ότι το πρόβλημα δεν είναι ότι δεν υπάρχει χώρος αλλά ότι δεν υπάρχουν ελεύθερα inodes.

Η εικόνα fsdisk3.img

1. Το εργαλείο του Linux που αναλαμβάνει τον έλεγχο ενός συστήματος αρχείων ext2 για αλλοιώσεις είναι το e2fsck.

2. Ορισμένοι παράγοντες που μπορούν να προκαλέσουν αλλοιώσεις στο σύστημα αρχείων είναι οι παρακάτω:

1. Φυσικές φθορές στον δίσκο.
2. Κακόβουλο λογισμικό.
3. Κακός κώδικας του χρήστη που εισάγεται στον πυρήνα.
4. Λάθη στον driver του δίσκου.
5. Προβλήματα στην φυσική διεπαφή δίσκου - υπολογιστή.
6. Διακοπή παροχής ρεύματος κατά την διάρκεια σημαντικών ενεργειών.
7. Race conditions και άλλες προβληματικές καταστάσεις που δεν διαχειρίζονται σωστά, σε κώδικα που ανοίγει αρχεία στο συγκεκριμένο file system.
8. Άλλα προβλήματα που έχουν να κάνουν με το hardware (πχ με την RAM του υπολογιστή που κάνει αλλαγές στο σύστημα).
9. Bugs στον κώδικα του συστήματος αρχείων.
10. Σφάλματα κατά την ανανέωση του λογισμικού/λειτουργικού.

3. Κάνοντας χρήση του παραπάνω εργαλείου και επιλέγοντας την διόρθωση των σφαλμάτων παίρνουμε:

```
root@utopia:~# e2fsck /dev/vdd
e2fsck 1.46.2 (28-Feb-2021)
fsdisk3.img contains a file system with errors, check forced.
```

```
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
First entry 'BOO' (inode=1717) in directory inode 1717 (/dir-2) should be '.'
Fix<y>? yes
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Inode 3425 ref count is 1, should be 2.  Fix<y>? yes
Pass 5: Checking group summary information
Block bitmap differences:  +34
Fix<y>? yes
Free blocks count wrong (926431538, counted=19800).
Fix<y>? yes
fsdisk3.img: *****  FILE SYSTEM WAS MODIFIED *****
fsdisk3.img: 23/5136 files (0.0% non-contiguous), 680/20480 blocks
```

Το πρώτο θέμα που βρήκε και διόρθωσε το e2fsck είναι ότι το πρώτο entry σε κάθε directory πρέπει να είναι '.', και στο directory /dir-2 ήταν 'BOO'.

Ακόμα βρήκε πως αν και το inode 3425 είναι referenced 2 φορές, το ref count του ήταν 1.

Βρήκε ακόμα ότι ενώ το block 34 είναι used, στο block bitmap ήταν σημειωμένο ως free.

Τέλος, βρήκε λανθασμένη τιμή για τα ελεύθερα blocks, σε σχέση με αυτά που μέτρησε.

4 και 5. Επαναφέρουμε την αρχική εικόνα μέσω του backup μας και χρησιμοποιούμε το εργαλείο hexedit για να βρούμε τα παραπάνω προβλήματα και να τα επιδιορθώσουμε

a. Βρίσκουμε ότι έχουμε 1712 inodes/block group στην εικόνα αυτή, άρα το inode 1717 είναι το 5ο inode του BG#1. Βρίσκουμε πως το inode table του BG#1 βρίσκεται στο block 8197, ενώ block size=1024 και inode size =128. Άρα για να βρουμε το inode πρέπει να πάμε με το hexedit σε offset 8197*1024 + 4*128 + 40 (επειδή σε αυτό το offset μέσα στο inode θα βρούμε το block στο οποίο υπάρχει το directory entry) =8394280. Πηγαίνοντας εκεί και διαβάζοντας τα πρώτα 4 bytes βλέπουμε πως ο αριθμός του block που χρειαζόμαστε είναι 0x20DC=0d8412. Έτσι πηγαίνουμε σε offset 1024*8412=8613888, και βλέπουμε τα παρακάτω:

```
B5 06 00 00  0C 00 03 00  42 4F 4F 00
        ........BOO.
```

Τα πρώτα 4 bytes (σύμφωνα με την δομή του directory entry https://www.nongnu.org/ext2-doc/ext2.html#linked-directories) είναι το inode (0x6B5==0d1717), τα επόμενα 2 (0xC=0d12) είναι το offset μετά από το entry που ξεκινάει το επόμενο entry. Έπειτα υπάρχει 1 byte που καθορίζει το length του ονόματος (εδώ 3 για τους 3 χαρακτήρες στο 'BOO'), 1 byte για το είδος του αρχείου (πάντα 0 από ότι βλέπουμε στο documentation μετά από revision) και τέλος έχουμε τους ASCII χαρακτήρες

του ‘BOO’ και ένα αχρησιμοποίητο byte. Πρέπει συνεπώς να αλλάξουμε το όνομα από ‘BOO’ σε ‘.’ αλλά και το name length από 3 σε 1.

Στο τέλος θα βλέπουμε το παρακάτω δηλαδή:

```
B5 06 00 00  0C 00 01 00  2E 00 00 00
          ..........
```

Μπορούμε να τρέξουμε και την εντολή **fsck** στο utopia VM και πράγματι βλέπουμε ότι το πρώτο από τα προηγούμενα προβλήματα δεν υπάρχει πλέον:

```
root@utopia:~# fsck -n /dev/vdd
fsck from util-linux 2.36.1
e2fsck 1.46.2 (28-Feb-2021)
fsdisk3.img contains a file system with errors, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
```

b. Το επόμενο πρόβλημα είναι ότι το inode 3425 (το οποίο καταλαβαίνουμε ότι είναι το 1ο inode του BG#2) έχει reference count 1, ενώ το πραγματικό είναι 2. Βρίσκουμε ακόμα πως το inode table του group αυτού ξεκινάει στο block 16389. Έτσι συνεπώς πρέπει με το hexedit να πάμε σε offset 16389*1024 + 26, αφού το 26 είναι το offset μέσα στο inode του links_count. Κάνοντας το βλέπουμε τα 2 bytes του πεδίου αυτού που είναι: 01 και τα διορθώνουμε σε 02.

Τρέχοντας και την fsck μετά βλέπω ότι πλέον ούτε το δεύτερο πρόβλημα δεν υπάρχει:

```
Pass 4: Checking reference counts
```

c. Το επόμενο πρόβλημα είναι ότι το block 34 εμφανίζεται ως free, αν και είναι used. Πρέπει δηλαδή να πάμε στο block bitmap του block group του (το οποίο προφανώς είναι το group #0, αφού τα blocks per group ειναι 8192) και να αλλάξουμε το 35ο bit (το 2ο LSBit του 5ου byte) από 0 σε 1.

Βρίσκουμε από το GDT ότι το block bitmap του group #0 βρίσκεται στο block 3 και έτσι το offset που πρέπει να βάλουμε στο hexedit είναι 3*1024 +4 = 3076. Παρατηρούμε ότι το byte έχει την τιμή 0xFD και για να αλλάξουμε το 2ο LSBit από 0 σε 1, γράφουμε στο byte την τιμή 0xFF.

Τρέχοντας και την fsck μετά βλέπουμε ότι πλέον ούτε το τρίτο πρόβλημα υπάρχει.

d. Τέλος πρέπει να αλλάξουμε τον αριθμό των free blocks στο superblock. Από την δομή του βρίσκουμε ότι το πεδίο free blocks count βρίσκεται στα byte 12-15 του superblock (που βρίσκεται στο block #1). Συνεπώς έχουμε offset για hexedit 1024+12=1036. Στο σημείο εκείνο βλέπουμε τα επόμενα 4 byte (που έχουν τιμή: 32 39 38 37, δηλαδή

αναπαριστούν τον αριθμό 0x37383932=0d926431538) και τα αλλάζουμε σε τιμή 0x4D58 (δηλαδή γράφω 58 4D 00 00).

Τέλος τρέχουμε και μία τελευταία φορά τα fsck και e2fsck για να βεβαιωθούμε ότι όλα τα θέματα έχουν λυθεί:

```
root@utopia:~# fsck -n /dev/vdd
fsck from util-linux 2.36.1
e2fsck 1.46.2 (28-Feb-2021)
fsdisk3.img: clean, 23/5136 files, 680/20480 blocks
```

Μέρος 2ο

Στο δεύτερο μέρος της άσκησης θα ασχοληθούμε από την μεριά του πυρήνα με ένα σύστημα αρχείων, το ext2-lite, που είναι ένα υποσύνολο του γνωστού συστήματος αρχείων του Linux ext2, και του οποίου ο κώδικας του μας παρέχεται από το εργαστήριο. Συγκεκριμένα μας δίνεται κώδικας σε C, με κενά σε κάποιες συγκεκριμένες συναρτήσεις, τις οποίες καλούμαστε να συμπληρώσουμε.

Παρακάτω παρουσιάζονται οι ολοκληρωμένες συναρτήσεις με συνοπτική περιγραφή της λειτουργικότητας τους.

init_ext2_fs και exit_ext2_fs (από το αρχείο super.c)

Η συνάρτηση **init_ext2_fs** καλείται για την καταχώρηση του του συστήματος αρχείων στον πυρήνα. Καταρχάς αρχικοποιεί την cache για τα inodes, και εφόσον δεν προκύψει κάποιο σφάλμα καλεί την **register_filesystem** του VFS δίνοντας του ως παράμετρο την δομή **ext2_fs_type** που ορίζεται παραπάνω στο αρχείο **super.c**. Η συνάρτηση επιστρέφει 0 σε περίπτωση επιτυχίας.

```
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);  //ext2-lite is also defined as
ext2_fs_type
    if(err)
```

```
        goto out;
    return 0;

out:
    destroy_inodecache();
    return err;

}
```

Η συνάρτηση **exit_ext2_fs** καλείται αντίστοιχα για διαγραφή του συστήματος αρχείων από τον πυρήνα του Linux, και η δουλειά της είναι να καλεί την συνάρτηση **unregister_filesystem** του VFS, αλλά και τη **destroy_inodecache()** για την απελευθέρωση των πόρων που δεσμεύτηκαν για την inode cache.

```
static void __exit exit_ext2_fs(void)
{
    /* Unregister ext2-lite filesystem from the kernel */
    unregister_filesystem(&ext2_fs_type);
    destroy_inodecache();
}
```

ext2_find_entry (από το αρχείο dir.c)

Η δουλειά της **ext2_find_entry** είναι να ψάχνει μέσα σε ένα directory για ένα entry με συγκεκριμένο όνομα.

Αρχικά εξάγει από τις παραμέτρους το όνομα που πρέπει να ψάξει και το μήκος του, καθώς και τον αριθμό των σελίδων στην μνήμη που αντιστοιχούν στο directory αυτό και έναν δείκτη προς την δομή **ext2_inode_info** για το directory αυτό.

Στην συνέχεια αφού βεβαιωθεί ότι δεν υπάρχει θέμα με τον αριθμό των σελίδων, βρίσκει μέσω της δομής **ext2_inode_info** το starting point και μετά "σκανάρει" μία μία τις σελίδες, ψάχνοντας διαδοχικά τα directory entries κάθε σελίδας και αν βρει directory entry με το όνομα αυτό (με την χρήση της συνάρτησης **ext2_match**), τότε επιστρέφει δείκτη σε αυτό το entry, ενώ αλλιώς επιστρέφει κατάλληλη τιμή error.

```
ext2_dirent *ext2_find_entry(struct inode *dir, const struct qstr *child,
                             struct folio **foliop)
{
    const char *name = child->name;
    int namelen = child->len;
```

```
unsigned reclen = EXT2_DIR_REC_LEN(namelen);
unsigned long npages = dir_pages(dir);
unsigned long i, start;
struct ext2_inode_info *ei = EXT2_I(dir);
ext2_dirent *de;
char *kaddr;

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

/* Scan all the pages of the directory to find the requested name. */
start = ei->i_dir_start_lookup;
if (start >= npages)
    start = 0;
for (i=start; i < npages; i++) {
    char *kaddr = ext2_get_folio(dir, i, 0, foliop);
    if(IS_ERR(kaddr))
        return ERR_CAST(kaddr);

    de = (ext2_dirent *) kaddr;
    kaddr += ext2_last_byte(dir, i) - reclen;
    while((char *) de <= kaddr){
        if(de->rec_len == 0){
            ext2_error(dir->i_sb, __func__, "Zero length directory entry");
            folio_release_kmap(*foliop, de);
            goto out;
        }
        if(ext2_match(namelen, name, de)){
            ei->i_dir_start_lookup = i;
            goto found;
        }
        de = ext2_next_entry(de);
    }
    folio_release_kmap(*foliop, kaddr);
}
/*If we reach this that means
 next folio is past the blocks we've got (i >= npages)
 return appropriate error message*/
```

```
    ext2_error(dir->i_sb, __func__,
               "dir %lu size %lld exceeds block count %llu",
               dir->i_ino, dir->i_size,
               (unsigned long long)dir->i_blocks);
    goto out;

out:
    return ERR_PTR(-ENOENT);
found:
    return de;
}
```

ext2_get_inode (από το αρχείο inode.c)

Η συνάρτηση **ext2_get_inode** καλείτε για να εντοπιστεί συγκεκριμένα inode στον δίσκο, βάση του αριθμού του.

Καταρχάς βρίσκει μέσω του superblock (δείκτης στην δομή super_block δίνεται σαν παράμετρος στην συνάρτηση) το blocksize του συστήματος αρχείων, καθώς και το size των inodes και τον αριθμό των inodes ανά block group. Στην συνέχεια ελέγχει ότι ο αριθμός που έχει δοθεί ως παράμετρος είναι έγκυρο inode (επιστρέφοντας με error εφόσον δεν είναι) και υπολογίζει κατάλληλα το block group στο οποίο βρίσκεται το inode αυτό. Έπειτα φέρνει στη μνήμη το block group descriptor για το παραπάνω block group (με χρήση της **ext2_get_group_desc**), από το οποίο βρίσκει τον αριθμό του block από το οποίο ξεκινάει το inode table, και τελικά φέρνει στην μνήμη το block που βρίσκεται το ζητούμενο inode, και επιστρέφει δείκτη στην αρχή του inode.

```
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;
    printk("Fez is here\n");
```

```
    *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) / EXT2_INODES_PER_GROUP(sb);
    gdp = ext2_get_group_desc(sb, block_group, NULL);
    if(!gdp)
        goto egdp;
    /* Figure out the offset within the block group inode table */
    offset = ((ino - 1) % EXT2_INODES_PER_GROUP(sb)) * EXT2_INODE_SIZE(sb);
    block = le32_to_cpu(gdp->bg_inode_table) +
            (offset >> EXT2_BLOCK_SIZE_BITS(sb));
    if(!(bh = sb_bread(sb, block)))
        goto eio;
    /* Return the pointer to the appropriate ext2_inode */
    *p = bh;
    offset &= (EXT2_BLOCK_SIZE(sb) - 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);
egdp:
    return ERR_PTR(-EIO);
}
```

ext2_iget (από το αρχείο inode.c)

Η συνάρτηση **ext2_iget** καταρχάς ελέγχει αν το ζητούμενο inode υπάρχει ήδη στην μνήμη και αν ναι επιστρέφει δείκτη προς τον καλούντα. Σε διαφορετική περίπτωση, χρησιμοποιεί την **ext2_get_inode** (που είδαμε παραπάνω) για να φέρει το inode στην μνήμη και στην συνέχεια δημιουργεί ένα VFS inode και αρχικοποιεί κατάλληλα τα πεδία του, ανάλογα πάντα και με το

είδος του αρχείου στο οποίο αναφέρεται το inode (regular file, directory, symbolic link fast ή regular, special file). Τέλος, ενημερώνει την δομή **ex2_inode_info**, απελευθερώνει την μνήμη και επιστρέφει το VFS inode.

```
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;
    uid_t i_uid;
    gid_t i_gid;
    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)) {
        printk("We have error on raw_inode\n");
        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);
printk("We got mode %d for inode\n", raw_inode->i_mode);
i_uid = (uid_t)le16_to_cpu(raw_inode->i_uid);
i_gid = (gid_t)le16_to_cpu(raw_inode->i_gid);
i_uid_write(inode, i_uid);
i_gid_write(inode, i_gid);
set_nlink(inode, le16_to_cpu(raw_inode->i_links_count));
inode_set_atime(inode, (signed)le32_to_cpu(raw_inode->i_atime), 0);
inode_set_ctime(inode, (signed)le32_to_cpu(raw_inode->i_ctime), 0);
inode_set_mtime(inode, (signed)le32_to_cpu(raw_inode->i_mtime), 0);
ei = EXT2_I(inode);
//ei->i_dtime = le32_to_cpu(raw_inode->i_dtime);
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;
} 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])));
            printk("We have the special inode: %d\n", inode->i_mode);
        }
        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->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;
}
```

ext2_allocate_in_bg (από το αρχείο balloc.c)

Τέλος, η **ext2_allocate_in_bg** εντοπίζει για συγκεκριμένο block group το πρώτο ελεύθερο block μέσω του bitmap του, και προσπαθεί να δεσμεύσει count το πολύ συνεχόμενα blocks σε αυτό, τροποποιώντας και κατάλληλα την τιμή της count ανάλογα με τα πόσα blocks κατάφερε να δεσμεύσει. Τέλος επιστρέφει -1 σε περίπτωση αποτυχίας, ενώ αλλιώς το offset μέσα στο block bitmap, του πρώτου block του οποίου δέσμευσε.

```
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 = 0;

    first_free_bit = 0;

    ext2_grpblk_t grp_goal = find_next_usable_block(first_free_bit, bitmap_bh,
                                                    nblocks);
    if(grp_goal < 0){
        goto fail_access;

    }

    for (; num < *count && grp_goal < nblocks; grp_goal++) {
        if (ext2_set_bit_atomic(sb_bgl_lock(EXT2_SB(sb), group),
                    grp_goal, bitmap_bh->b_data)) {
            if (num == 0)
                continue;
            break;
        }
        num++;
    }

    if (num == 0)
        goto fail_access;

    *count = num;
```

```
    return grp_goal - num;

fail_access:
    return -1;
}
```

Έλεγχος λειτουργίας ext2_lite

Ολοκληρώνοντας την εργαστηριακή άσκηση θέλουμε να ελέγξουμε την ορθή λειτουργία του κώδικα μας. Σύμφωνα με τον εργαστηριακό οδηγό και τις εντολές που μας δίνονται δημιουργούμε ένα ext2-lite image με στόχο να το προσαρτήσουμε. Εκτελούμε :

```
$ touch ext2-lite.img && truncate -s 128M ext2-lite.img
$ mkfs.ext2 -b 1024 -L "ext2-lite fs" -O none -m 0 ./ext2-lite.img
$ mount -t ext2-lite -o loop ./ext2-lite.img /mnt
```

Όπως βλέπουμε παρακάτω το σύστημα κάνει mount

```
root@utopia:~# cat /proc/mounts | grep 'ext2-lite'
/dev/loop0 /mnt ext2-lite rw,relatime,errors=continue 0 0
```