

# **ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ**

## **2Η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ**

**Μάιος 2022**

**Υλοποίηση αρχείου καταγραφής στο σύστημα  
αρχείων FAT του Linux**

**Μέλη Ομάδας :**

**Αθανασία - Δανάη Τσαούση 3349**

**Κωνσταντίνος Γεωργίου 4333**

**Πηνελοπη Ελευθεριάδη 3221**

## ΠΕΡΙΕΧΟΜΕΝΑ

Στόχος Εργασίας.....	3
Πρώτο Στάδιο - Κατανόηση.....	3
Πρόσθεση printf() στο πηγαίο κώδικα της LKL.....	4
Εκτέλεση με δοκιμαστικό αρχείο.....	7
Εκτυπώσεις στον τερματικό .....	8
Περιήγηση και ανάλυση των συναρτήσεων .....	10
Αποτύπωση ιδεών για journaling με κανονικές κλήσεις συστήματος.....	15
Superblock και προσάρτηση.....	17
Δομές αποθήκευση στο msdos_fs.h και ~/bin/search.....	19
Εγγραφή στο αρχείο καταγραφής.....	20
Βασικές Δομές FAT.....	26
Έξοδος στον τερματικό.....	34
Επίλογος.....	36

## Στόχος Εργασίας

Στόχος της εργασίας αυτής είναι η επέκταση του συστήματος αρχείων FAT στην βιβλιοθήκη του πυρήνα Linux, που εκτελείται σε επίπεδο χρήστη, με τη δημιουργία ενός αρχείου καταγραφής (journaling) που αποθηκεύει όλες τις τροποποιήσεις που συμβαίνουν στο σύστημα αρχείων κατά την εκτέλεση εφαρμογών.

Η υλοποίηση αυτού του αρχείου έχει σκοπό την επαναφορά του συστήματος σε περίπτωση αποτυχίας.

## Πρώτο Στάδιο-Κατανόηση

Αρχικά, προσπαθήσαμε να κατανοήσουμε την λειτουργία των διάφορων καταλόγων του source code της LKL. Περιηγηθήκαμε στους καταλόγους arch/lkl, tools/lkl, fs/fat και mm και προσπαθήσαμε να κατανοήσουμε τον ρόλο κάθε καταλόγου.

Προσπαθήσαμε σε πρώτο επίπεδο να κατανοήσουμε την δομή και τις βασικές λειτουργίες κάθε δομής, όπως περιγράφεται στην εκφώνηση. Θα αναλύσουμε κάποιες, όπως:

- Inodes(μπλοκ μεταδιδόμενων) : μεταδεδομένα (πχ δικαιώματα πρόσβασης, ιδιοκτήτης, χρόνοι προσπέλασης) ενός αρχείου αποθηκεύονται σε μια δομή που ονομάζεται inode.
- Superblock : Περιλαμβάνει μεταδεδομένα υψηλότερου επιπέδου που σχετίζονται συνολικά με το σύστημα αρχείων
- Dentry: μια εγγραφή καταλόγου που αντιπροσωπεύει ένα στοιχείο (φάκελο ή αρχείο) ενός μονοπατιού
- File: δομή που περιγράφει ένα ανοικτό αρχείο που είναι συσχετισμένο με μια διεργασία

Ύστερα, επικεντρωθήκαμε στο σύστημα αρχείων FAT. Κατανοήσαμε πως διαχειρίζονται τις διάφορες λειτουργίες του superblock, των inodes, των αρχείων και των καταλόγων και πως χωρίζονται στα διάφορα αρχεία (πχ inode.c, fatten.c) και στα αντίστοιχα structs.

Σύμφωνα ,λοιπόν, με τα παραπάνω προχωρήσαμε στο πρώτο βήμα της άσκησης.

Δημιουργήσαμε ένα δοκιμαστικό αρχείο , το **mytest** και με χρήση και του αποσφαλματωτή gbd και τις οδηγίες της εκφώνησης περάσαμε βήμα βήμα από τις συναρτήσεις και καταλάβαμε την ροή του κώδικα.

### Πρόσθεση printk() στο πηγαίο κώδικα της LKL

Έπειτα , στις κατάλληλες συναρτήσεις στους καταλόγους fat/ και /mm προσθέσαμε την συνάρτηση **printk(KERN\_INFO ".....")** , όπου στο κενό δίνουμε το όνομα της συνάρτησης που περνάει το πρόγραμμα κατά την εκτέλεση της εφαρμογής. Έτσι, θα καταλάβουμε με ποια σειρά εκτελούνται τα πεδία των δομών.

Παράδειγμα από το αρχείο

```
long fat_generic_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    struct inode *inode = file_inode(filp);
    u32 __user *user_attr = (u32 __user *)arg;

    printk(KERN_INFO "reached fat_generic_ioctl");

    switch (cmd) {
    case FAT_IOCTL_GET_ATTRIBUTES:
        return fat_ioctl_get_attributes(inode, user_attr);
    case FAT_IOCTL_SET_ATTRIBUTES:
        return fat_ioctl_set_attributes(filp, user_attr);
    case FAT_IOCTL_GET_VOLUME_ID:
        return fat_ioctl_get_volume_id(inode, user_attr);
    default:
        return -ENOTTY; /* Inappropriate ioctl for device */
    }
}
```

Πιο αναλυτικά, οι δομές για τις αντίστοιχες λειτουργίες είναι :

1. Η δομή struct super\_operations fat\_sops του αρχείου fs/fat/inode.c για τις λειτουργίες του **superblock**.
2. Η δομή struct address\_space\_operations fat\_aops του αρχείου fs/fat/inode.c για τις λειτουργίες της **μνήμης**.
3. Η δομή struct fatent\_operations fat12/16/32\_ops του αρχείου fs/fat/fatent.c για τις λειτουργίες των **εγγραφών FAT**.

4. Η δομή struct file\_operations fat\_file\_operations του αρχείου fs/fat/file.c για τις **λειτουργίες αρχείου**.
5. Η δομή struct inode\_operations fat\_file\_inode\_operations του αρχείου fs/fat/file.c για τις λειτουργίες **inode**.
6. Η δομή struct inode\_operations msdos\_dir\_inode\_operations του αρχείου fs/fat/namei\_msdos.c για το **FAT** και fs/fat/namei\_vfat.c για το **VFAT**.

Σε κάθε δομή , λοιπόν, προηγήθηκαμε σε κάθε συνάρτηση που έχει και τοποθετήσαμε το αντίστοιχο printk.

Συγκεκριμένα :

- Στην δομή **struct address\_space\_operations fat\_aops**  
Στο αρχείο inode.c κατευθυνθήκαμε στις συναρτήσεις  
fat\_readpage, fat\_readpages, fat\_writepage,  
fat\_writepages, fat\_write\_begin, fat\_write\_end,  
fat\_direct\_IO, \_fat\_bmap.  
Παράδειγμα printk :  
**printk(KERN\_INFO "WE ARE IN fat\_readpage");**
- Στην δομή **struct inode\_operations fat\_file\_inode\_operations** , οι συναρτήσεις fat\_setattr και fat\_getattr που βρίσκονται στο file.c
- Στις δομές **struct fatent\_operations fat12\_ops , struct fatent\_operations fat16\_ops, struct fatent\_operations fat32\_ops** στο αρχείο fatent.c τοποθετήσαμε τα αντίστοιχα  
printk() στις συναρτήσεις fat12\_ent\_blocknr,  
fat12\_ent\_set\_ptr, fat12\_ent\_bread, fat12\_ent\_get,  
fat12\_ent\_put, fat12\_ent\_next, fat16\_ent\_set\_ptr,  
fat16\_ent\_get, fat16\_ent\_put, fat16\_ent\_next,  
fat32\_ent\_set\_ptr, fat32\_ent\_get, fat32\_ent\_put,  
fat32\_ent\_next.

- Στην δομή ***struct super\_operations fatsops*** στην οποία περιλαμβάνονται οι συναρτήσεις `fat_alloc_inode`, `fat_destroy_inode`, `fat_write_inode`, `fat_evict_inode`, `fat_put_super`, `fat_statfs`, `fat_remount`.
- Στην δομή ***struct file\_operations fat\_file\_operations*** του αρχείου `file.c` στην οποία περιλαμβάνονται οι συναρτήσεις `fat_file_release`, `fat_generic_ioctl`, `fat_generic_compat_ioctl`, `fat_file_fsync`, `fat_fallocate`. Ακόμα βρήκαμε την συνάρτηση `generic_file_llseek` στο αρχείο `read_write.c` και προσθέσαμε `printk()`, όπως και τις συναρτήσεις `generic_file_read_iter`, `generic_file_write_iter`, `generic_file_mmap` στο αρχείο `filemap.c`, καθώς και την συνάρτηση `generic_file_splice_read` στο `splice.c`.
- Στην δομή ***struct super\_operations fat\_sops*** του αρχείου `inode.c` προσθέσαμε `printk()` στις συναρτήσεις `fat_alloc_inode`, `fat_destroy_inode`, `fat_write_inode`, `fat_evict_inode`, `fat_put_super`, `fat_statfs`, `fat_remount`, `fat_show_options`.
- Στην δομή ***struct inode\_operations vfat\_dir\_inode\_operations*** του αρχείου `namei_vfat` προσθέσαμε `printk()` στις συναρτήσεις `vfat_create`, `vfat_lookup`, `vfat_unlink`, `vfat_mkdir`, `vfat_rmdir`, `vfat_rename`. Εντοπίσαμε τις 2 τελευταίες συναρτήσεις `fat_set_attr`, `fat_getattr` να ορίζονται στο `file.c`. Έχουμε ήδη προσθέσει τα `printk()` εξαιτίας της δομής `struct inode_operations fat_file_inode_operations` στην οποία περιλαμβάνονται αυτές οι συναρτήσεις.

## Εκτέλεση με δοκιμαστικό αρχείο

Έπειτα, όπως μας καθοδηγεί και η εκφώνηση, εκτελούμε τις εντολές στο lkl/lkl-source

```
-make -j8 clean; make -j8 -C tools/lkl clean
```

```
-make -j8 -C tools/lkl
```

Και ύστερα στο tools/lkl ,όπου έχουμε τοποθετήσει και το δοκιμαστικό αρχείο μας mytest, εκτελούμε τις

```
-make test
```

```
./cptofs -i /tmp/vfatfile -p -t vfat mytest /
```

Δημιουργήσαμε ένα αρχείο txt μικρού μεγέθους με όνομα mytest και έπειτα εκτελέσαμε την εντολή cptofs ώστε να δούμε μέσω των εκτυπώσεων στον τερματικό από ποιες συναρτήσεις «περνάει» το πρόγραμμα. Οι εκτυπώσεις είναι οι υποφαινόμενες στη παρακάτω εικόνα και θα μελετήσουμε μια μια τις συναρτήσεις για τυχόν συσχετίσεις μεταξύ πεδίων και θα παρατηρήσουμε τις μεταβλητές ώστε τελικά να καταλήξουμε στα σημεία που πιστεύουμε ότι γίνονται αλλαγές και χρειάζεται να τις καταγράψουμε. Να σημειωθεί επίσης ότι υπάρχουν δυο είδη φράσεων στις εκτυπώσεις επειδή χωρίσαμε την αναλογία των printf() για να μοιραστεί ο φόρτος.

## Εκτυπώσεις στον τερματικό

```
0.026780] reached generic_file_read_iter
0.027198] reached fat_alloc_inode
0.027218] reached fat_alloc_inode
0.027220] reached fat_alloc_inode
0.027685] reached vfat_lookup
0.027736] reached vfat_create
0.027748] reached fat_alloc_inode
0.027762] reached generic_file_write_iter
0.027765] WE ARE IN fat_write_begin
0.027769] WE ARE IN fat_ent_blocknr
0.027783] WE ARE IN fat_ent_bread
0.027798] WE ARE IN fat16_ent_set_ptr
0.027811] WE ARE IN fat16_ent_get
0.027813] WE ARE IN fat16_ent_next
0.027815] WE ARE IN fat16_ent_get
0.027816] WE ARE IN fat16_ent_next
0.027818] WE ARE IN fat16_ent_get
0.027821] WE ARE IN fat16_ent_next
0.027822] WE ARE IN fat16_ent_get
0.027824] WE ARE IN fat16_ent_put
0.027830] WE ARE IN fat_write_end
0.027838] reached fat_file_release
0.032482] reached fat_write_inode
0.032509] WE ARE IN fat_writepages
0.032525] reached fat_write_inode
0.032767] reached fat_evict_inode
0.032786] reached fat_destroy_inode
0.032788] reached fat_evict_inode
0.032794] reached fat_destroy_inode
0.032796] reached fat_put_super
0.032806] reached fat_evict_inode
0.032808] reached fat_destroy_inode
0.032810] reached fat_evict_inode
0.032811] reached fat_destroy_inode
0.033045] reboot: Restarting system
```

Οπότε, παρατηρούμε την σειρά που εκτελούνται οι συναρτήσεις :

1. generic\_file\_read\_iter
2. fat\_alloc\_inode
3. vfat\_lookup
4. vfat\_create
5. fat\_alloc\_inode\_
6. generic\_write\_iter



7. fat\_write\_begin
8. fat\_ent\_blocknr
9. fat\_ent\_bread
- 10.fat16\_ent\_set\_ptr
- 11.fat16\_ent\_get
- 12.fat16\_ent\_put
- 13.fat\_write\_end
- 14.fat\_file\_release
- 15.fat\_write\_inode
- 16.fat\_writepages
- 17.fat\_evict\_inode
- 18.fat\_destroy\_inode
- 19.fat\_put\_super

## Περιήγηση και ανάλυση των συναρτήσεων

Σε αυτό το σημείο της εργασίας αποφασίσαμε πριν προβούμε σε οποιεσδήποτε αποφάσεις για τα σημεία που πιστεύουμε ότι συμβαίνουν αλλαγές ώστε να τις καταγράψουμε μετά ,να κατανοήσουμε σε βάθος τη διαδρομή συναρτήσεων που ακολουθείται. Διαβάσαμε τον κώδικα κάθε συνάρτησης ξεχωριστά ώστε να καταλάβουμε το ρόλο και το λόγο χρήσης τους. Παρακάτω υλοποιούμε μια ανάλυση των συναρτήσεων. Παρατηρήθηκε ακόμη ότι πολλές από τις συναρτήσεις χρησιμοποιούν κλειδαριές(mutexes) για αμοιβαίο αποκλεισμό.

1. Αρχικά βλέπουμε ότι διέρχεται από την **generic\_file\_read\_iter()** , η οποία βρίσκεται στο αρχείο **filemap.c** και τα πεδία της είναι **struct kiocb \* iocb** (kernel I/O control block ), **struct iov\_iter \* iter** (θέση για data read).Είναι μια από τις λειτουργίες αρχείων και αυτό γίνεται εύκολα αντιληπτό μιας και ανήκει στη δομή **struct file\_operations fat\_file\_operations**.Αντιπροσωπεύει τη ρουτίνα **read\_iter()** για όλα τα συστήματα αρχείων και χρησιμοποιεί τη **page cache** απευθείας. Ο πυρήνας του Linux χρησιμοποιεί την **page cache** , η οποία αναπαριστά την κύρια κρυφή μνήμη του δίσκου του πυρήνα. Όταν πρόκειται να γίνει ανάγνωση ή ακόμη και εγγραφή σε δίσκο αποτελεί σύνηθες φαινόμενο ο πυρήνας να αναφέρεται στη **page cache**.
2. Έπειτα εισέρχεται 3 φορές στην **fat\_alloc\_inode()**,η οποία βρίσκεται στο αρχείο **inode.c** και έχει ως πεδίο το **struct super\_block \*sb**). Είναι μια από τις λειτουργίες γνωστές ως **super operations** για τη διαχείριση του **superblock**. Είναι υπεύθυνη για το εκχωρεί τα **inode** στο **Fat**. Παρατηρούμε ότι δεσμεύει μνήμη με τέτοιο τρόπο ώστε στη περίπτωση του πυρήνα Linux, να δημιουργεί ένα σύστημα διαχείρισης μνήμης (**slab**), που εκχωρεί μνήμη σε διαφορετικά αντικείμενα του πυρήνα. Αυτό το επιτυγχάνει με την χρήση της εντολής **kmem\_cache\_alloc** , ώστε να κρατά τα **Msdos inode** αντικείμενα ,να πραγματοποιεί γρήγορα την εκχώρηση (**allocation**) και τελικά να κάνει απελευθέρωση πίσω στη μνήμη (**Free**).

```
struct msdos_inode_info *ei;  
ei = kmem_cache_alloc(fat_inode_cache, GFP_NOFS);
```

3. Άλλη μια συνάρτηση στην οποία εισέρχεται είναι η **vfat\_lookup()** , η οποία βρίσκεται στο αρχείο **namei\_vfat.c** και τα πεδία της είναι **struct inode\* dir**, **struct dentry\* dentry** , **unsigned int flags**. Είναι μια από τις λειτουργίες καταλόγου(**dentry**). Ο ρόλος της είναι να αναζητά μια δοσμένη εγγραφή σε έναν κατάλογο και να εξάγει το κατάλληλο **inode**. Εφόσον δίνεται ως όρισμα το **dentry** (μια εγγραφή καταλόγου που αντιπροσωπεύει ένα στοιχείο -φάκελο ή αρχείο – ενός μονοπατιού) , έχει πρόσβαση και στο **inode** του αρχικού καταλόγου. Ο κώδικας αναζητά με την βοήθεια της συνάρτησης **vfat\_find()**(βλέπει εικόνα παρακάτω) στο κατάλογο στο **dir** ελέγχει αν το **&dentry->d\_name** ταιριάζει με το όνομα κάποια εγγραφή.

```
err = vfat_find(dir, &dentry->d_name, &sinfo);
```

Αν βρεθεί κάποια εγγραφή με το ίδιο όνομα τότε επιστρέφει **NULL** και συσχετίζει το **inode** με την εγγραφή, αλλιώς επιστρέφει **ERROR\_PTR(err)**. Ακολουθεί επίσης μια διαδικασία για να ελέγξει αν έχει γίνει **corruption** στο σύστημα αρχείων πριν κάνει την συσχέτιση , με το να εξετάζει αν έχει ξαναγίνει **lookup** γι' αυτό το **inode** με διαφορετικό **dentry->d\_name**.

4. Επίσης περνά από την συνάρτηση **vfat\_create()** , η οποία βρίσκεται στο αρχείο **namei\_vfat.c** και τα πεδία της είναι **struct inode\* dir**, **struct dentry\* dentry** , **unmode\_t mode**, **bool excl**. Είναι μια από τις λειτουργίες καταλόγου(**dentry**). Είναι υπεύθυνη να δημιουργεί ένα **inode** σύμφωνα με το ορίσματα που της δίνονται. Εισάγει μια νέα εγγραφή με τη βοήθεια της εντολής :

```
err = vfat_add_entry(dir, &dentry->d_name, 0, 0, &ts, &sinfo);
```

Ορίζει το **inode** σε συσχέτιση με το **dentry** με τη βοήθεια της εντολής: **d\_instantiate(dentry, inode);** .

5. Ξανά διέρχεται από την **fat\_alloc\_inode()**. Αυτό πιστεύουμε ότι συμβαίνει για να γίνει εκχώρηση(**allocation**) του νέου **inode** που δημιουργήθηκε στη γραμμή 4 από την **vfat\_create()**.
6. Βλέπουμε ότι διέρχεται από την **generic\_file\_write\_iter()** , η οποία βρίσκεται στο αρχείο **filemap.c** και τα πεδία της είναι **struct kiocb \* iocb**(**IO** κατάσταση δομής: **file** , **offset** κτλ.),

struct iovec \* **from**(iovec με data για εγγραφή). Είναι μια από τις λειτουργίες αρχείων και αυτό γίνεται εύκολα αντιληπτό μιας και ανήκει στη δομή **struct file\_operations fat\_file\_operations**. Από ότι παρατηρούμε είναι υπεύθυνη για την εγγραφή δεδομένων στη page cache (**A**) πριν γίνει το sync με τον δίσκο (**B**).

```
{
    struct file *file = iocb->ki_filp;
    struct inode *inode = file->f_mapping->host;
    ssize_t ret;

    inode_lock(inode);
    ret = generic_write_checks(iocb, from);
    if (ret > 0)
        ret = __generic_file_write_iter(iocb, from);
    inode_unlock(inode);

    if (ret > 0)
        ret = generic_write_sync(iocb, ret);
    return ret;
}
```

A

B

7. Με την κλήση της **fat\_write\_begin** ξεκινά η εγγραφή. Ανήκει στο αρχείο **inode.c** και αποτελεί μια λειτουργία μνήμης. Τα ορίσματα

( struct file *	file,
struct address_space *	mapping,
loff_t	pos,
unsigned	len,
unsigned	flags,
struct page **	pagep,
void **	fsdata

της είναι αυτά που φαίνονται στην εικόνα αριστερά. Το σύστημα αρχείων σε αυτό σημείο πρέπει να κάνει εγγραφή len bytes στο δοσμένο offset pos του αρχείου file.

### Σημείωση:

Αυτό που συμπεραίνουμε από τις γραμμές **8 με 13** είναι ότι σε αυτό το τμήμα γίνεται η αντιγραφή του αρχείου. Σύμφωνα με την εκφώνηση, στο FAT αντιστοιχεί μια εγγραφή ανά cluster με

μέγεθος εγγραφής ίσο με 12,16 ή 32 bits. Προφανώς, εδώ βλέπουμε ότι αφού καλούνται οι `fat16_ent_set_ptr`, `fat16_ent_get` και `fat16_ent_put`, εκτελείται η `fat16`. Οπότε, σε αυτό το σημείο θα γίνει η ενημέρωση και θα αποθηκευτούν τα clusters για το αρχείο μας. Όπως γνωρίζουμε, cluster είναι ένας οργανωμένος χώρος στον δίσκο, όπου χωρίζεται σε προκαθορισμένο πλήθος από συνεχομένους τομείς.

13. Με την **`fat_write_end`** τελειώνει η εγγραφή και γίνονται οι

```
( struct file *      file,
  struct address_space * mapping,
  loff_t             pos,
  unsigned           len,
  unsigned           copied,
  struct page *      pagep,
  void *             fsdata
```

απαραίτητοι έλεγχοι. Ανήκει στο αρχείο **`inode.c`** και αποτελεί μια λειτουργία μνήμης. Τα ορίσματα της είναι αυτά που φαίνονται στην εικόνα αριστερά. Σε αυτό το σημείο τελειώνει η εγγραφή. Το πεδίο `len` είναι το ίδιο με τη `fat_write_begin` και το `copied` είναι τα συνολικά bytes που αντιγράφηκαν τελικά. Το `fsdata` περνιέται από τη `fat_write_begin` που επιστέφει σε αυτό `void *`.

14. Έπειτα περνά από τη **`fat_file_release`**, η οποία ανήκει στο αρχείο **`file.c`** και αποτελεί μια λειτουργία αρχείου. Τα ορίσματα της είναι ένα `struct inode*` **`inode`** και `struct file*` **`filp`**. Υποθέτουμε ότι γίνετε flush του `inode` και απελευθερώνονται (free) τα εκχωρημένα (allocated) clusters. Τέλος χρησιμοποιείται η εντολή `congestion_wait()`, πιθανόν για να περιμένει να τελειώσουν όλες οι διεργασίες στο σύστημα.

15. Ακολουθεί **`fat_write_inode`**, η οποία βρίσκεται στο αρχείο **`inode.c`** και έχει ως ορίσματα το `struct inode*` **`inode`** και `struct writeback_control*` **`wbc`**. Είναι μια από τις λειτουργίες γνωστές ως super operations για τη διαχείριση του superblock. Παρατηρούμε σε αυτό το σημείο του κώδικα

```
mutex_lock(&MSDOS_SB(sb)->s_lock);

err = fat_clusters_flush(sb);
```

```
mutex_unlock(&MSDOS_SB(sb)->s_lock);
```

ότι με χρήση κλειδαριών γίνεται αμοιβαίος αποκλεισμένος για την κρίσιμη περιοχή `err = fat_clusters_flush(sb)`, όπου όπως περιμέναμε γίνονται flush τα clusters.

16. Ακολουθεί η **fat\_writepages** , ανήκει στο αρχείο **inode.c** και αποτελεί μια λειτουργία μνήμης. Τα ορίσματα της είναι `struct address_space * mapping`, `struct writeback_control * wbc`. Καλεί την `mpage_writepages()` για να ολοκληρωθεί.

17. Ακολουθεί η **fat\_evict\_inode** , η οποία βρίσκεται στο αρχείο **inode.c** και έχει ως όρισμα το `struct inode* inode` .Είναι μια από τις λειτουργίες γνωστές ως super operations για τη διαχείριση του superblock. Διαγράφει το inode από τη page cache με την εντολή `truncate_inode_pages()` , διαγράφει το inode από την μνήμη με την εντολή `clear_inode()`.

18. Η **fat\_destroy\_inode** , η οποία βρίσκεται στο αρχείο **inode.c** και έχει ως όρισμα το `struct inode* inode` .Είναι μια από τις λειτουργίες γνωστές ως super operations για τη διαχείριση του superblock. Πρακτικά «καταστρέφει» το inode κάνοντας ένα RCU cleanup.

19. Η **fat\_put\_super** , η οποία βρίσκεται στο αρχείο **inode.c** και έχει ως όρισμα το `struct inode* inode`.

### Αποτύπωση ιδεών για journaling με κανονικές κλήσεις συστήματος

Αρχικά πρέπει να γίνει αποθήκευση των εγγραφών των τροποποιήσεων του FAT κατά την εκτέλεση των εφαρμογών στο ext4 της εικονικής μηχανής χρησιμοποιώντας κανονικές κλήσεις συστήματος εισόδου/εξόδου (π.χ., open, write, κλπ). Με τις εγγραφές αυτές σας έπειτα πρέπει να καταγράψουμε τις αλλαγές των δομών του συστήματος FAT και δεδομένα αρχείων όχι απλώς τις κλήσεις συστήματος.

Διαβάζοντας για το σύστημα αρχείων ext4 σε βιβλία και στο διαδίκτυο είδαμε ότι διαθέτει παραπάνω από μια λειτουργία journaling (modes = writeback/ordered/journal). Η δικιά μας περίπτωση είναι το journal mode. Ακόμα αυτό που πιστεύουμε με μεγάλη σιγουριά είναι ότι πρέπει να χρησιμοποιηθούν οι χρονικές σημάνσεις που υπάρχουν στις εγγραφές οι οποίες υποδεικνύουν την ώρα και μέρα τελευταίας τροποποίησης.

Η λογική είναι να ορίσουμε έναν file descriptor ανοίγοντας στην εφαρμογή μας με χρήση της open() το αρχείο καταγραφής που θα δημιουργήσουμε και με τη κλήση της write() να καταγράψουμε σε αυτό τις αλλαγές. Οι κανονικές κλήσεις συστήματος θα καλέσουν τις αντίστοιχες κλήσεις συστήματος, αυτές με την σειρά τους θα καλέσουν τις αντίστοιχες vfs. Οι vfs συναρτήσεις καλούνται από τις syscall κλήσεις λογικά με κάποιο κατάλληλο syscall number. Στο vfs file system level αντιστοιχούν και η inode cache, η directory cache και η page cache. Σε ένα επίπεδο πιο μέσα θα κληθεί η αντίστοιχη open() του ext4 και αυτό με τη σειρά του θα ανατρέξει στις συσκευές block που αντιπροσωπεύει τον δίσκο.

Με βάση τα παραπάνω αφού ανοίγουμε το αρχείο καταγραφής θα καλέσουμε τη write(), η οποία θα καλέσει την συνάρτηση write του file object, που συνδέεται με το κανονικό αρχείο στο ext4. Έπειτα θα κληθεί

το journal στο jdb (το jbd που χρησιμοποιείται από το ext4 είναι το jbd2) και έτσι αποκτάται πρόσβαση στα metadata blocks και αυτά προστίθενται σε μια λίστα μεταφοράς .Το jdb θα σταματήσει να τρέχει όταν ολοκληρωθούν οι καταγραφές των εγγραφών στο αρχείο εγγραφής .Έπειτα γίνεται το transaction των metadata και των data.

Δεν καταφέραμε να υλοποιήσουμε τον κώδικα γιατί μας λείπουν κάποια βασικά κομμάτια για να ολοκληρωθεί η ιδέα μας ,ώστε να υλοποιηθεί. Δεν μπορούμε να καταλάβουμε πως θα αποκτήσουμε πρόσβαση στα blocks που περιέχουν τα δεδομένα αρχείων και στα metadata για τα αρχεία του Fat .

BOOT SECTOR | SUPER BLOCK | INODE BLOCK | DATA BLOCKS

Κάθε αρχείο έχει ένα directory entry που αποθηκεύει το όνομα του αρχείου ,το αρχικό cluster και την χρονική σήμανση timestamp και άλλες χρήσιμες πληροφορίες .Αυτή περιέχει την τελευταία πρόσβαση που καταγράφηκε και το πότε δημιουργήθηκε ,οπότε θα μπορούμε με σύγκριση να ξέρουμε αν έγινε τροποποίηση της εγγραφής. Επίσης ένα directory entry έχει δείκτες που δείχνουν απευθείας σε data blocks. Οπότε καταλαβαίνουμε ότι θα μας ήταν αρχικά πολύ χρήσιμο να αποκτήσουμε πρόσβαση στα directory entries.



## Superblock και Προσάρτηση

Μέσα στην διαδικασία αναζήτησης για το που θα τοποθετηθεί ο κώδικας εισαγωγής δεδομένων στο αρχείο καταγραφής μας καταλήξαμε σε μια διαφάνεια από το βοηθητικό αρχείο που μας έχετε παραδώσει.

Ένα εργαλείο που μας βοήθησε να εντοπίσουμε τις δομές αποθήκευσης στον δίσκο ήταν ο σύνδεσμος [LKL DOX](#) που μας δόθηκε στην εκφώνηση. Αναζητήσαμε τις δομές αυτές και είδαμε ότι εμπεριέχονται όλες στο αρχείο `msdos_fs.h`, οπότε σε αρχικό επίπεδο ξεκινήσαμε να ανακαλύψουμε τον κώδικα των δομών του και να βρούμε που υπάρχουν βασικές αλλαγές ώστε να τα προσθέσουμε στο `journal`.

Διαβάσαμε στις διαφάνειες ότι η δομή `struct super_block` δημιουργείται από τη μέθοδο `alloc_super()` στη μνήμη όταν το σύστημα αρχείων προσαρτείται (**mounted**). Έτσι προσπαθήσαμε να ακολουθήσουμε τη διαδρομή που ακολουθείται κατά την προσάρτηση (`mount`). Ξεκινώντας από το αρχείο `namei_msdos.c` παρατηρήσαμε τη συνάρτηση **`init_msdos_fs()`** στην οποία αρχικοποιείται το `superblock`.

### ♦ `init_msdos_fs()`

```
static int __init init_msdos_fs ( void )
```

Definition at line 673 of file `namei_msdos.c`.

```
674 {  
675     return register_filesystem(&msdos_fs_type);  
676 }
```

References `msdos_fs_type`, and `register_filesystem()`.

Από τις αναφορές προχωρήσαμε στην **msdos\_fs\_type()** και είδαμε ότι είναι δομή που βρίσκεται στο ίδιο αρχείο. Οι αρχικές τιμές των πεδίων της είναι αυτές στην εικόνα αριστερά. Επειδή μελετάμε την πορεία κατά την προσάρτηση παρατηρήσαμε το πεδίο **mount** μιας και το όνομα παραπέμπει σε κάτι σχετικό. Βλέποντας την δομή **msdos\_mount**. Βλέπουμε ότι επιστρέφει μια δομή **mount\_bdev** η οποία ορίζεται στο **fs.h**.

#### ◆ msdos\_fs\_type

```
struct file_system_type msdos_fs_type
```

Initial value:

```
= {
    .owner      = THIS_MODULE,
    .name       = "msdos",
    .mount      = msdos_mount,
    .kill_sb    = kill_block_super,
    .fs_flags   = FS_REQUIRES_DEV,
}
```

```
{
    return mount_bdev(fs_type, flags, dev_name, data, msdos_fill_super);
}
```

Στο τελευταίο πεδίο καλεί την **msdos\_fill\_super**, η οποία απλά καλεί την **fat\_fill\_super** με τα κατάλληλα ορίσματα.

#### ◆ msdos\_fill\_super()

```
static int msdos_fill_super ( struct super_block * sb,
                             void *          data,
                             int              silent
                             )
```

Definition at line 652 of file **namei\_msdos.c**.

```
653 {
654     return fat_fill_super(sb, data, silent, 0, setup);
655 }
```

Η συνάρτηση **fat\_fill\_super** δεσμεύει/εκχωρεί μνήμη και αρχικοποιεί τα πεδία της δομής **msdos\_sb\_info / sbi**.

```
char out[50];

/*
 * GFP_KERNEL is ok here, because while we do hold the
 * superblock lock, memory pressure can't call back into
 * the filesystem, since we're only just about to mount
 * it and have no inodes etc active!
 */
sbi = kzalloc(sizeof(struct msdos_sb_info), GFP_KERNEL);
```

Συμπεραίνουμε λοιπόν ότι αν θέλουμε να ορίσουμε ένα νέο πεδίο στην δομή **msdos\_sb\_info / sbi** πρέπει να το προσθέσουμε στα πεδία της στο αρχείο **fat.h** και έπειτα να πάμε στο αρχείο **inode.c** και να το αρχικοποιήσουμε μαζί με τα υπόλοιπα στην συνάρτηση **fat\_fill\_super**.

Οπότε, καταλάβαμε ότι θα πρέπει στην **fat\_fill\_super** να ανοίξουμε το αρχείο καταγραφής μας, ως εξής

```
sbi->myid = sys_open("myjournal",O_RDWR | O_CREAT | O_EXCL,  
S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);
```

και έπειτα να τοποθετήσουμε τον ορισμό του file descriptor του αρχείου καταγραφής μας:

```
int myid; .
```

### ΔΟΜΕΣ ΑΠΟΘΗΚΕΥΣΗΣ ΣΤΟ ΑΡΧΕΙΟ msdos\_fs.h και ~/bin/search

Στην δομή msdos\_fs.h υπάρχουν οι δομές

- \_\_fat\_dirent
- fat\_boot\_sector
- fat\_boot\_fsinfo
- msdos\_dir\_entry
- msdos\_dir\_slot

Για κάθε δομή λοιπόν θέλαμε να δούμε που αλλάζουν τιμή οι μεταβλητές των δομών. Με την εντολή ~/bin/search αναζητούμε αναδρομικά .

->Ξεκινάμε από την **\_\_fat\_dirent**. Παρακάτω έχουμε το screenshot από την εκτέλεση της ~/bin/search

```
Terminal - myy601@myy601lab2: ~/lkl/lkl-source
File Edit View Terminal Tabs Help
myy601@myy601lab2:~/lkl/lkl-source$ ~/bin/search __fat_dirent | more\
> ^C
myy601@myy601lab2:~/lkl/lkl-source$ ~/bin/search __fat_dirent | more
FAT_IOCTL_FILLDIR_FUNC(fat_ioctl_filldir, __fat_dirent)
    struct __fat_dirent __user *dl = (struct __fat_dirent __user *)arg;
    if (!access_ok(VERIFY_WRITE, dl, sizeof(struct __fat_dirent[2])))
-rwxrwx-rw- 1 myy601 myy601 35603 Sep 21 2017 ./fs/fat/dir.c
struct __fat_dirent {
#define VFAT_IOCTL_READDIR_BOTH    _IOR('r', 1, struct __fat_dirent[2])
#define VFAT_IOCTL_READDIR_SHORT  _IOR('r', 2, struct __fat_dirent[2])
-rw-r--r-- 1 myy601 myy601 6888 Sep 21 2017 ./include/uapi/linux/msdos_fs.h
myy601@myy601lab2:~/lkl/lkl-source$
```

Προηγήθήκαμε ,λοιπόν, στην dir.c όμως ενώ χρησιμοποιούνται μεταβλητές της δομής \_\_fat\_dirent, δεν υπάρχει καμία αλλαγή.

->Συνεχίζουμε με την **fat\_boot\_sector**, όπου μας κατευθύνει στις Inode.c, msdos\_fs.h και msdos.c

```
myy601@myy601lab2:~/lkl/lkl-source$ ~/bin/search fat_boot_sector | more
* fat_boot_sector.
    struct fat_boot_sector *b;
    b = (struct fat_boot_sector *) bh->b_data;
static bool fat_bpb_is_zero(struct fat_boot_sector *b)
static int fat_read_bpb(struct super_block *sb, struct fat_boot_sector *b,
    struct fat_boot_sector *b, int silent,
    error = fat_read_bpb(sb, (struct fat_boot_sector *)bh->b_data, silent,
        (struct fat_boot_sector *)bh->b_data, silent, &bpb);
-rwxrwx-rw- 1 myy601 myy601 52626 May 16 00:10 ./fs/fat/inode.c
struct fat_boot_sector {
-rw-r--r-- 1 myy601 myy601 6888 Sep 21 2017 ./include/uapi/linux/msdos_fs.h
    struct fat_boot_sector *fb;
        fb = (struct fat_boot_sector *) data;
-rw-r--r-- 1 myy601 myy601 16201 Sep 21 2017 ./block/partitions/msdos.c
myy601@myy601lab2:~/lkl/lkl-source$
```

Παρατηρήσαμε ότι οι μεταβλητές της αλλάζουν μόνο στην inode.c και συγκεκριμένα στις γραμμές 690 με 723.

### Εγγραφή στο αρχείο καταγραφής

Εδώ θα κάνουμε μια παρένθεση ώστε να εξηγήσουμε πως υλοποιήσαμε τις εγγραφές στο journal. Οι προσθήκες που θα αναφέρουμε συμβαίνουν σε κάθε αρχείο που εντοπίσαμε αλλαγές .

Αυτά που κάναμε λοιπόν είναι τα εξής :

- Αρχικά έχουμε δημιουργήσει μια συνάρτηση , την **insert**, στο fat.h, αφού από εδώ θα είναι ορατή σε όλα τα αρχεία που κάνουμε ερευνά και εισάγουμε τις αλλαγές.  
Έχει την εξής μορφή:

```
static void insert(int v,char buf[2042],int bufsize){  
  
    printk(KERN_INFO "STARTING WRITING IN JOURNAL\n");  
    sys_write(v,buf,bufsize);  
    sys_fsync(v);  
    sys_fdatasync(v);  
    printk(KERN_INFO "END OF WRITING IN JOURNAL\n");  
}
```

- Δέχεται ως ορίσματα έναν ακέραιο v ,ο οποίος είναι ο file descriptor, τον buffer που κρατάμε τις αλλαγές κάθε φορά και το μέγεθος του buffer.
- Με χρήση της printk δίνουμε μηνύματα ενημέρωσης για το στάδιο που βρισκόμαστε, στην αρχή ή στο τέλος της εγγραφής.
- Έπειτα, με τις εντολές που μας έχουν δοθεί από το βοηθητικό εργαστήριο T7 , εκτελούμε εγγραφή στο αρχείο (**sys\_write**) και flush την μνήμη και write στον δίσκο με τις **sys\_fsync** και **sys\_fdatasync**.
- Εισαγωγή του **#include <linux/syscalls.h>** για να μπορέσουμε να καλέσουμε τις παραπάνω εντολές.

Μέσα στον κώδικα που γίνονται οι αλλαγές κάνουμε τα εξής:

- Έχουμε προσδιορίσει το μέγεθος του buffer στην αρχή του αρχείου:

```
#define BUFSIZE 1000
```

- Τυπώνουμε στο τερματικό σε ποια συνάρτηση και σε ποιο αρχείο βρισκόμαστε.
- Ορίζουμε τον buffer μας και προσθέτουμε την μεταβλητή που αλλάζει τιμή.
- Τυπώνουμε το περιεχόμενο του buffer για έλεγχο .
- Τέλος, τυπώνουμε ότι τέλειωσε η διαδικασία.

Ένα παράδειγμα κώδικα παρουσιάζουμε εδώ, όπου στο αρχείο inode.c στην συνάρτηση fat\_set\_state βλέπουμε ότι αλλάζει τιμή η μεταβλητή state και κάνουμε τα παρακάτω:

```
687 char obuffermoy[BUFSIZE];
688 b = (struct fat_boot_sector *) bh->b_data;
689
690 if (sbi->fat_bits == 32) {
691     if (set)
692         b->fat32.state |= FAT_STATE_DIRTY;
693     else
694         b->fat32.state &= ~FAT_STATE_DIRTY;
695
696     //##### PRINT #####
697     printk(KERN_INFO " EIMASTE STO FAT SET STATE sto inode\n");
698     sprintf(obuffermoy, "fat32.state =%u\n", b->fat32.state);
699     printk(KERN_INFO "O BUFFER EXEI MESA = %s\n", obuffermoy);
700
701     int v = sbi->myid;
702     insert(v,obuffermoy, BUFSIZE);
703     printk(KERN_INFO "FINISHED\n");
704
705 } else /* fat 16 and 12 */ {
706     if (set)
707         b->fat16.state |= FAT_STATE_DIRTY;
708     else
709         b->fat16.state &= ~FAT_STATE_DIRTY;
710     int v = sbi->myid;
711     printk(KERN_INFO " EIMASTE STO FAT SET STATE sto inode\n");
712     sprintf(obuffermoy, "fat32.state =%u\n", b->fat16.state);
713     printk(KERN_INFO "O BUFFER EXEI MESA = %s\n", obuffermoy);
714
715     insert(v,obuffermoy, BUFSIZE);
716     printk(KERN_INFO "END OF WRITING IN JOURNAL\n");
717 }
718 }
```

->Ξανά κάνουμε την ίδια διαδικασία για την fat\_boot\_fsinfo

```

myy601@myy601lab2:~/lkl/lkl-source$ ~/bin/search fat_boot_fsinfo | more
    struct fat_boot_fsinfo *fsinfo;
    fsinfo = (struct fat_boot_fsinfo *)bh->b_data;
-rwxrw-rw- 1 myy601 myy601 7700 Sep 21  2017 ./fs/fat/misc.c
    struct fat_boot_fsinfo *fsinfo;
    fsinfo = (struct fat_boot_fsinfo *)fsinfo_bh->b_data;
-rwxrw-rw- 1 myy601 myy601 52626 May 16 00:10 ./fs/fat/inode.c
struct fat_boot_fsinfo {
-rw-r--r-- 1 myy601 myy601 6888 Sep 21  2017 ./include/uapi/linux/msdos_fs.h

```

Οπότε, περιηγηθήκαμε στις misc.c και inode.c. Με Ctrl+F ψάξαμε μεταβλητές fat\_boot\_fsinfo και είδαμε ότι υπάρχουν αλλαγές στο αρχείο misc.c.

Συγκεκριμένα , βλέπουμε ότι αλλάζουν οι μεταβλητές next\_cluster και free\_clusters. Όπως και πριν προσθέτουμε κώδικα για να εγγράψουμε τις αλλαγές στο journal.

->Συνεχίζουμε με την επόμενη δομή, την **msdos\_dir\_entry**. Εδώ βλέπουμε ότι έχουμε αρκετές αλλαγές σε πολλά αρχεία.

```

    struct msdos_dir_entry de;
    struct msdos_dir_entry *dotdot_de;
-rwxrw-rw- 1 myy601 myy601 17052 Sep 21 2017 ./fs/fat/namei_msdos.c
        struct msdos_dir_entry *de)
        | (de - (struct msdos_dir_entry *)bh->b_data);
        struct buffer_head **bh, struct msdos_dir_entry **de)
*pos += sizeof(struct msdos_dir_entry);
*de = (struct msdos_dir_entry *)((*bh)->b_data + offset);
        struct msdos_dir_entry **de)
    (*de - (struct msdos_dir_entry *)(*bh)->b_data) <
    *pos += sizeof(struct msdos_dir_entry);
        struct buffer_head **bh, struct msdos_dir_entry **de,
        const struct msdos_dir_entry *de,
    struct msdos_dir_entry *de;
    struct msdos_dir_entry *de;
    if (cpos & (sizeof(struct msdos_dir_entry) - 1)) {
    ctx->pos = cpos - (nr_slots + 1) * sizeof(struct msdos_dir_entry);
        struct msdos_dir_entry **de)
        struct msdos_dir_entry **de)
    struct msdos_dir_entry *de;
    struct msdos_dir_entry *de;
    struct msdos_dir_entry *de, *endp;
        endp = (struct msdos_dir_entry *) (bh->b_data + sb->s_blocksize);
    struct msdos_dir_entry *de;
    while (nr_slots && de >= (struct msdos_dir_entry *)bh->b_data) {
    struct msdos_dir_entry *de;
    de = (struct msdos_dir_entry *)bhs[0]->b_data;
        int *nr_cluster, struct msdos_dir_entry **de,
    size = nr_slots * sizeof(struct msdos_dir_entry);
    offset = copy - sizeof(struct msdos_dir_entry);
    *de = (struct msdos_dir_entry *)((*bh)->b_data + offset);
    struct msdos_dir_entry *uninitialized_var(de);
-rwxrw-rw- 1 myy601 myy601 35603 Sep 21 2017 ./fs/fat/dir.c
    struct msdos_dir_entry *de;
        de = (struct msdos_dir_entry *)slots;
    de = (struct msdos_dir_entry *)ps;
    struct msdos_dir_entry *dotdot_de;
-rwxrw-rw- 1 myy601 myy601 27800 May 5 17:11 ./fs/fat/namei_vfat.c
    struct msdos_dir_entry *de;
        const struct msdos_dir_entry *de)
static inline void fat_set_start(struct msdos_dir_entry *de, int cluster)
        struct msdos_dir_entry **de);
        struct msdos_dir_entry *de, loff_t i_pos);
extern int fat_fill_inode(struct inode *inode, struct msdos_dir_entry *de);
-rwxrw-rw- 1 myy601 myy601 14437 May 16 00:08 ./fs/fat/fat.h
int fat_fill_inode(struct inode *inode, struct msdos_dir_entry *de)
        struct msdos_dir_entry *de, loff_t i_pos)
    struct msdos_dir_entry *raw_entry;

```



```

    struct msdos_dir_entry *raw_entry;
    raw_entry = &((struct msdos_dir_entry *) (bh->b_data))[offset];
    inode->i_size = sbi->dir_entries * sizeof(struct msdos_dir_entry
);
    sbi->dir_per_block = sb->s_blocksize / sizeof(struct msdos_dir_entry);
    * sizeof(struct msdos_dir_entry) / sb->s_blocksize;
-rwxrw-rw- 1 myy601 myy601 52626 May 16 00:10 ./fs/fat/inode.c
    struct msdos_dir_entry *de ;
    de = (struct msdos_dir_entry *)bh->b_data;
    struct msdos_dir_entry *de;
    de = (struct msdos_dir_entry *) parent_bh->b_data;
    struct msdos_dir_entry *de;
-rwxrw-rw- 1 myy601 myy601 8016 Sep 21 2017 ./fs/fat/nfs.c
#define MSDOS_DPS (SECTOR_SIZE / sizeof(struct msdos_dir_entry))
#define MSDOS_DIR_BITS 5 /* log2(sizeof(struct msdos_dir_entry)) */
struct msdos_dir_entry {
-rw-r--r-- 1 myy601 myy601 6888 Sep 21 2017 ./include/uapi/linux/msdos_fs.h

```

Οπότε , μπήκαμε και ψάξαμε για μεταβλητές μέσα στα αρχεία `namei_msdos.c`, `dir.c`, `fat.h`, `inode.c` και `nfs.c`.

```

174 struct msdos_dir_entry {
175     __u8    name[MSDOS_NAME];/* name and extension */
176     __u8    attr;           /* attribute bits */
177     __u8    lcase;          /* Case for base and extension */
178     __u8    ctime_cs;       /* Creation time, centiseconds (0-199) */
179     __le16   ctime;          /* Creation time */
180     __le16   cdate;          /* Creation date */
181     __le16   adate;          /* Last access date */
182     __le16   starthi;        /* High 16 bits of cluster in FAT32 */
183     __le16   time,date,start; /* time, date and first cluster */
184     __le32   size;           /* file size (in bytes) */
185 };

```

-Στην `namei_msdos.c` στην `msdos_add_entry` έχουμε 5 αλλαγές για τις μεταβλητές `name`, `attr`, `lcase`, `time`, `cdate` και `date` στις γραμμές 239-204.

-Στην `dir.c` στην συνάρτηση `fat_remove_entries` στην γραμμή 1019 βλέπουμε την αλλαγή της μεταβλητής `name`. Χρειάζεται όμως την προσθήκη της μεταβλητής `msdos_sb_info` ώστε να καλέσουμε το `myid` για την εγγραφή στο `journal` μας. Τέλος, μέσα στην `while` ,οπού αλλάζει τιμή η μεταβλητή , κάνουμε τον αντίστοιχο κώδικα για εγγραφή που έχουμε αναλύσει παραπάνω.

Στο ίδιο αρχείο στην `fat_remove_entries` έχουμε την ίδια αλλαγή και εκτελούμε τις ιδίες ενέργειες.

Επίσης, βρήκαμε αλλαγές στην fat\_alloc\_new\_dir, όπου βλέπουμε αλλαγές στις μεταβλητές name, attr, ctime, ctime\_cs, adate και size.

-Στην namei\_vfat.c στην συνάρτηση vfat\_build\_slots ανακαλυψαμε ότι οι μεταβλητές name, attr, lcase, time, date, ctime δέχονται αλλαγή στις γραμμές 665-868.

-Στην inode.c στην συνάρτηση \_\_fat\_write\_inode βλέπουμε αλλαγή στις μεταβλητές size, attr.  
Τέλος, στην nfs.c δεν βρήκαμε κάτι που να χρειάζεται καταγραφή.

->Τέλος, έχουμε την **msdos\_dir\_slot**

```
myy601@myy601lab2:~/lkl/lkl-source$ ~/bin/search msdos_dir_slot | more
struct msdos_dir_slot *ds;
ds = (struct msdos_dir_slot *)*de;
ds = (struct msdos_dir_slot *)*de;
-rwxrw-rw- 1 myy601 myy601 35603 Sep 21 2017 ./fs/fat/dir.c
struct msdos_dir_slot *slots, int *nr_slots)
struct msdos_dir_slot *ps;
struct msdos_dir_slot *slots;
-rwxrw-rw- 1 myy601 myy601 27800 May 5 17:11 ./fs/fat/namei_vfat.c
struct msdos_dir_slot {
-rw-r--r-- 1 myy601 myy601 6888 Sep 21 2017 ./include/uapi/linux/msdos fs.h
```

Με την ίδια λογική καταλήγουμε στις dir.c και namei\_vfat.c.  
Στην dir.c δεν βρήκαμε κάποια αλλαγή ενώ στην namei\_vfat.c βρήκαμε μετατροπές στις μεταβλητές id, attr, reserved, alias\_checksum, start, name5\_10 και name11\_12 στις γραμμές 641-649. Οπότε και εδώ κάνουμε την καταγραφή αυτών των μεταβλητών.

## ΒΑΣΙΚΕΣ ΔΟΜΕΣ FAT

Αφού, λοιπόν, τελειώσαμε με το αρχείο msdos\_fs.h προσπαθήσαμε να καταλάβουμε που αλλού θα χρειαστεί να παρέμβουμε για να καταγράψουμε. Έτσι, με γνώμονα μας το βοηθητικό υλικό από το φροντιστήριο T7 ,παρατηρήσαμε ότι για κάθε δομή της FAT (File Allocation Table, Superblock, Inode,Dentries,Files) συμμετέχουν κάποια συγκεκριμένα αρχεία. Αρχικά, για τα πρώτα 4 βλέπουμε ότι υπάρχουν δομές στο αρχείο fs/fat/fat.h.

Θα μιλήσουμε όμως για κάθε δομή FAT ξεχωριστά :

- ❖ FILE ALLOCATION TABLE: Βλέπουμε ότι το FAT entry ξεκινά στο fs/fat/fat.h Η στην δομή struct fat\_entry. Έτσι αποφασίσαμε ότι θα κινηθούμε όπως παραπάνω, δηλαδή με εκτέλεση της ~/bin/search.

```
struct fat_entry {  
    int entry;  
    union {  
        u8 *ent12_p[2];  
        __le16 *ent16_p;  
        __le32 *ent32_p;  
    } u;  
    int nr_bhs;  
    struct buffer_head *bhs[2];  
    struct inode *fat_inode;  
};
```

Στην επόμενη σελίδα έχουμε και το αποτέλεσμα της εκτέλεσης. Ξεκινάμε οπότε να ψάχνουμε στα αρχεία misc.c, file.c, fatten.c,cache.c.

- misc.c : δεν πιστεύουμε ότι γίνεται κάποια αλλαγή οπότε δεν προσθέσαμε κάτι στο journal.

- file.c : επίσης δεν βρήκαμε κάποια αλλαγή

- fatten.c στις γραμμές 49 και 50 βλέπουμε αλλαγή στις μεταβλητές ent12\_p στην συνάρτηση fat12\_ent\_set\_ptr.

Στην `fat16_ent_set_ptr` βλέπουμε αλλαγή στις μεταβλητές `ent16_p` στις γραμμές 92-93.

Στην `fat32_ent_set_ptr` βλέπουμε αλλαγή στις μεταβλητές `ent32_p` στις γραμμές 111-112.

Στην `fat12_ent_bread` οι `fat_inode` και `nr_bhs`.

Στην `fat_ent_bread` οι `fat_inode` , `nr_bhs` και `bhs`.

Στην `fat16_ent_put` υπάρχει αλλαγή στην `ent16_p`.

Στην `fat32_ent_put` υπάρχει αλλαγή στην `ent32_p`.

Στην `fat12_ent_next` υπάρχει αλλαγή στην `next`.

Στην `fat16_ent_next` υπάρχει αλλαγή στην `entry ent16_p` .

Στην `fat_ent_update_ptr` υπάρχει αλλαγή στην `nr_bhs`.

Τέλος, στην `fat_alloc_clusters` καταγράφουμε την `entry`.

- `cache.c` : Δεν υπάρχουν αλλαγές.

- ❖ **Superblock** : συνεχίζουμε με την δομή `superblock`, και όπως μας υποδεικνύει η σελίδα 52 καταλήγουμε στην `fat.h` και συγκεκριμένα στην δομή `msdos_sb_info`. Όπως και πριν, εκτελούμε `~/bin/search msdos_sb_info`.

```
struct msdos_sb_info *sbi = MSDOS_SB(d17->d_sb);  
-rw-rw-rw- 1 myy601 myy601 19449 May 18 16:31 ./fs/fat/namei_msdos.c  
    struct msdos_sb_info *sbi = MSDOS_SB(sb);  
    struct msdos_sb_info *sbi = MSDOS_SB(sb);  
  
void fat_time_unix2fat(struct msdos_sb_info *sbi, struct timespec  
-rw-rw-rw- 1 myy601 myy601 8619 May 18 16:07 ./fs/fat/misc.c  
    struct msdos_sb_info *sbi = MSDOS_SB(sb);  
  
-rw-rw-rw- 1 myy601 myy601 42168 May 19 00:25 ./fs/fat/dir.c  
    struct msdos_sb_info *sbi = MSDOS_SB(inode->i_sb);  
  
static int fat_getow_set_time(struct msdos_sb_info *sbi, struct  
    struct msdos_sb_info *sbi = MSDOS_SB(dentry->d_sb);  
-rw-rw-rw- 1 myy601 myy601 13755 May 13 21:48 ./fs/fat/file.c  
    struct msdos_sb_info *sbi = MSDOS_SB(sb);  
  
struct msdos_sb_info *sbi = MSDOS_SB(sb);  
-rw-rw-rw- 1 myy601 myy601 27946 May 19 02:08 ./fs/fat/fatent.c  
    struct msdos_sb_info *sbi = MSDOS_SB(dir->i_sb);
```

```

    struct msdos_sb_info *sbi;
    sbi = kzalloc(sizeof(struct msdos_sb_info), GFP_KERNEL)
-rwxrw-rw- 1 myy601 myy601 54924 May 18 16:02 ./fs/fat/inode.c
    struct msdos_sb_info *sbi = MSDOS_SB(sb);
    struct msdos_sb_info *sbi = MSDOS_SB(inode->i_sb);
    struct msdos_sb_info *sbi = MSDOS_SB(sb);
    struct msdos_sb_info *sbi = MSDOS_SB(sb);
-rwxrw-rw- 1 myy601 myy601 8016 Sep 21 2017 ./fs/fat/nfs.c
    struct msdos_sb_info *sbi = MSDOS_SB(sb);
    struct msdos_sb_info *sbi = MSDOS_SB(inode->i_sb);
-rwxrw-rw- 1 myy601 myy601 9403 Sep 21 2017 ./fs/fat/cache.c

```

```

63 struct msdos_sb_info {
64     unsigned short sec_per_clus; /* sectors/cluster */
65     unsigned short cluster_bits; /* log2(cluster_size) */
66     unsigned int cluster_size; /* cluster size */
67     unsigned char fats, fat_bits; /* number of FATs, FAT bits (12,16 or 32) */
68     unsigned short fat_start;
69     unsigned long fat_length; /* FAT start & length (sec.) */
70     unsigned long dir_start;
71     unsigned short dir_entries; /* root dir start & entries */
72     unsigned long data_start; /* first data sector */
73     unsigned long max_cluster; /* maximum cluster number */
74     unsigned long root_cluster; /* first cluster of the root directory */
75     unsigned long fsinfo_sector; /* sector number of FAT32 fsinfo */
76     struct mutex fat_lock;
77     struct mutex nfs_build_inode_lock;
78     struct mutex s_lock;
79     unsigned int prev_free; /* previously allocated cluster number */
80     unsigned int free_clusters; /* -1 if undefined */
81     unsigned int free_clus_valid; /* is free_clusters valid? */
82     struct fat_mount_options options;
83     struct nls_table *nls_disk; /* Codepage used on disk */
84     struct nls_table *nls_io; /* Charset used for input and display */
85     const void *dir_ops; /* Opaque; default directory operations */
86     int dir_per_block; /* dir entries per block */
87     int dir_per_block_bits; /* log2(dir_per_block) */
88     unsigned int vol_id; /*volume ID*/
89
90     int fatent_shift;
91     const struct fatent_operations *fatent_ops;
92     struct inode *fat_inode;
93     struct inode *fsinfo_inode;
94
95     struct ratelimit_state ratelimit;
96
97     spinlock_t inode_hash_lock;
98     struct hlist_head inode_hashtable[FAT_HASH_SIZE];
99
100    spinlock_t dir_hash_lock;
101    struct hlist_head dir_hashtable[FAT_HASH_SIZE];
102
103    unsigned int dirty; /* fs state before mount */
104    struct rcu_head rcu;
105

```

Από τα παραπάνω screenshot βλέπουμε ότι πρέπει να κατευθυνθούμε στα αρχεία namei\_msdos.c, misc.c, dir.c, file.c, fatten.c, inode.c, nfs.c και cache.c.

- namei\_msdos.c : Δεν βρέθηκε κάποια αλλαγή.
- misc.c : Δεν βρέθηκε κάποια αλλαγή.
- dir.c: Δεν βρέθηκε κάποια αλλαγή.
- file.c: Δεν βρέθηκε κάποια αλλαγή.
- fatten.c: Στις γραμμές 502-512 fat\_ent\_access\_init έχουμε αλλαγή στις μεταβλητές fatten\_shift και fatten\_ops. Καταγράφουμε τις αλλαγές για κάθε case(32,16,12).

Στην fat\_alloc\_clusters έχουμε αλλαγές στις μεταβλητές prev\_free, free\_clusters και free\_clus\_valid.

Στην fat\_free\_clusters έχουμε αλλαγή στην free\_clusters.

Στην fat\_count\_free\_clusters έχουμε τις free\_clusters και free\_clus\_valid.

- inode.c: Στην fat\_fill\_super στις γραμμές 1755 με 1765 έχουμε αλλαγή στις μεταβλητές cluster\_size, cluster\_bits, fats, fat\_bits, fat\_start, fat\_length, root\_cluster, free\_clusters, free\_clus\_valid, prev\_free. Επίσης, στις 1788-1790 για τις fat\_bits, fat\_length, , root\_cluster.

Στην 1805 και 1819 η fsinfo\_sector, στην 1850 η free\_clus\_valid και στις 1851-1852 οι free\_clusters και prev\_free.

Στην 1884 και 1898 η vol\_id και 1912-1916 οι

```
sbi->dir_per_block
sbi->dir_per_block_bits = ffs(sbi->dir_per_block) - 1;

sbi->dir_start = sbi->fat_start + sbi->fats * sbi->fat_length;
sbi->dir_entries = bpb.fat_dir_entries;
```

Στην 1939 αλλάζει η μεταβλητή

```
sbi->data_start
```

Στην 1959 καταγράφουμε την

```
sbi->fat_bits
```

Στις 1974 και 1976 την

```
sbi->dirty
```

Ενώ στην 2012

```
sbi->max_cluster
```

Τέλος, στις 2027, 2043 και 2093 αλλάζουν αντίστοιχα οι free\_clusters, prev\_free, nls\_disk και nls\_io.

- nfs.c: Δεν βρέθηκε κάποια αλλαγή.
- cache.c: Δεν βρέθηκε κάποια αλλαγή.

❖ Inode : και για την δομή inode λοιπόν καταλήγουμε στο struct msdos\_inode\_info της fat.h. Η εκτέλεση ~/bin/search μας δίνει τα αρχεία που θα περιηγηθούμε , τα οποία είναι τα:

```
struct msdos_inode_info {
static inline struct msdos_inode_info *MSDOS_I(struct inode *inode)
    return container_of(inode, struct msdos_inode_info, vfs_inode);
-rwxrwx-rw- 1 myy601 myy601 14559 May 18 14:51 ./fs/fat/fat.h
    struct msdos_inode_info *i;
    struct msdos_inode_info *ei;
    struct msdos_inode_info *ei = (struct msdos_inode_info *)foo;
                                   sizeof(struct msdos_inode_info),
-rwxrwx-rw- 1 myy601 myy601 66115 May 20 16:58 ./fs/fat/inode.c
    struct msdos_inode_info *i;
-rwxrwx-rw- 1 myy601 myy601 8016 Sep 21 2017 ./fs/fat/nfs.c
    struct msdos_inode_info *i = MSDOS_I(inode);
-rwxrwx-rw- 1 myy601 myy601 9403 Sep 21 2017 ./fs/fat/cache.c
```

inode.c,nfs.c,cache.c.

-inode.c: στην συνάρτηση init\_once προσθέτουμε τις μεταβλητές nr\_caches και cache\_valid\_id στο αρχείο καταγραφής.

-nfs.c:Δεν υπάρχει κάτι να καταγράψουμε.

-cache.c:Στην \_\_fat\_cache\_inval\_inode προσθέτουμε την μεταβλητή nr\_caches.



- ❖ Directory Entry : Για το Directory entries αναζητήσαμε στο αρχείο fat.h την struct msdos\_slot\_info όμως δε την βρήκαμε (ψάξαμε και σε άλλους καταλόγους). Καταλήξαμε στην υπόθεση ότι πρόκειται για τη δομή fat\_slot\_info οπότε κάναμε ~/bin/search σε αυτήν.

```

myy601@myy601lab2:~/lkl/lkl-source$ ~/bin/search fat_slot_info | more
        struct fat_slot_info *sinfo)
    struct fat_slot_info sinfo;
        struct timespec *ts, struct fat_slot_info *sinfo)
    struct fat_slot_info sinfo;
    struct fat_slot_info sinfo;
    struct fat_slot_info sinfo;
    struct fat_slot_info sinfo;
    struct fat_slot_info old_sinfo, sinfo;
-rwxrw-rw- 1 myy601 myy601 19449 May 18 16:31 ./fs/fat/namei_msdos.c
        int name_len, struct fat_slot_info *sinfo)
    * The ".." entry can not provide the "struct fat_slot_info" information
        struct fat_slot_info *sinfo)
            struct fat_slot_info *sinfo)
int fat_remove_entries(struct inode *dir, struct fat_slot_info *sinfo)
        struct fat_slot_info *sinfo)
-rwxrw-rw- 1 myy601 myy601 42168 May 19 00:25 ./fs/fat/dir.c
    struct fat_slot_info sinfo;
        struct fat_slot_info *sinfo)
            struct fat_slot_info *sinfo)
    struct fat_slot_info sinfo;
    struct fat_slot_info sinfo;
    struct fat_slot_info sinfo;
    struct fat_slot_info sinfo;
    struct fat_slot_info sinfo;
    struct fat_slot_info old_sinfo, sinfo;
-rwxrw-rw- 1 myy601 myy601 29527 May 19 00:58 ./fs/fat/namei_vfat.c
struct fat_slot_info {
        int name_len, struct fat_slot_info *sinfo);
    struct fat_slot_info *sinfo);
        struct fat_slot_info *sinfo);
            struct fat_slot_info *sinfo);
extern int fat_remove_entries(struct inode *dir, struct fat_slot_info *sinfo);
-rwxrw-rw- 1 myy601 myy601 14559 May 18 14:51 ./fs/fat/fat.h
    struct fat_slot_info sinfo;
-rwxrw-rw- 1 myy601 myy601 8016 Sep 21 2017 ./fs/fat/nfs.c

```

-namei\_msdos.c: Δεν υπάρχει κάποια αλλαγή.

-dir.c: Δεν υπάρχει κάποια αλλαγή.

-namei\_vfat.c: Δεν υπάρχει κάποια αλλαγή.



-nfs.c: Στην fat\_search\_long υπάρχει αλλαγή στις μεταβλητές slot\_off, nr\_slots, de, bh, i\_pos.

Στην fat\_scan οι slot\_off, nr\_slots, , i\_pos, όπως και στην fat\_scan\_logstart .

Τελος, στην fat\_add\_entries προσθέτουμε οι μεταβλητές nr\_slots, de,bh,i\_pos.

❖ Files : Εκτελούμε και εδώ ~/bin/search file (include/Linux/fs.h)

```
-rwxrwx-rw- 1 myy601 myy601 19449 May 18 16:31 ./namei_msdos.c
- rwxrwx-rw- 1 myy601 myy601 8619 May 18 16:07 ./misc.c
- rwxrwx-rw- 1 myy601 myy601 45665 May 21 00:03 ./dir.c
- rwxrwx-rw- 1 myy601 myy601 13755 May 13 21:48 ./file.c
- rwxrwx-rw- 1 myy601 myy601 29527 May 19 00:58 ./namei_vfat.c
- rwxrwx-rw- 1 myy601 myy601 66806 May 20 23:32 ./inode.c
- rwxrwx-rw- 1 myy601 myy601 8016 May 20 23:32 ./nfs.c
- rwxrwx-rw- 1 myy601 myy601 10719 May 20 23:36 ./cache.c
```

Η δομή fs.h που βρίσκεται ,όπως μας υποδεικνύεται από τις διαφάνειες, στο include\linux είναι η

```

struct file {
    union {
        struct llist_node    fu_llist;
        struct rcu_head      fu_rcuhead;
    } f_u;
    struct path              f_path;
    struct inode             *f_inode;      /* cached value */
    const struct file_operations *f_op;

    /*
     * Protects f_ep_links, f_flags.
     * Must not be taken from IRQ context.
     */
    spinlock_t              f_lock;
    atomic_long_t           f_count;
    unsigned int            f_flags;
    fmode_t                 f_mode;
    struct mutex            f_pos_lock;
    loff_t                  f_pos;
    struct fown_struct      f_owner;
    const struct cred       *f_cred;
    struct file_ra_state    f_ra;

    u64                     f_version;
#ifdef CONFIG_SECURITY
    void                    *f_security;
#endif
    /* needed for tty driver, and maybe others */
    void                    *private_data;

#ifdef CONFIG_EPOLL
    /* Used by fs/eventpoll.c to link all the hooks to this file */
    struct list_head        f_ep_links;
    struct list_head        f_tfile_llink;
#endif /* #ifdef CONFIG_EPOLL */
    struct address_space    *f_mapping;
} __attribute__((aligned(4))); /* lest something weird decides that 2 is OK */

```

Από την εκτέλεση κατευθυνόμαστε στα αρχεία:

-namei\_msdos.c : Δεν βρέθηκε κάποια αλλαγή.

-misc.c : Δεν βρέθηκε κάποια αλλαγή.

-dir.c : Στην fat\_ioctl\_readdir αλλάζει η f\_pos

-file.c : Δεν βρέθηκε κάποια αλλαγή.

-namei\_vfat.c : Δεν βρέθηκε κάποια αλλαγή.

-inode.c : Δεν βρέθηκε κάποια αλλαγή.

-nfs.c : Δεν βρέθηκε κάποια αλλαγή.

-cache.c : Δεν βρέθηκε κάποια αλλαγή.

## Έξοδος στον τερματικό

Παρακάτω παρουσιάζουμε κάποια screenshot από εκτελέσεις του τερματικού

```
myy601@myy601lab2:~/lkl/lkl-source/tools/lkl$ ./cptofs -i /tmp/vfatfile -p -t vfat mytest /
[ 0.000000] Linux version 4.11.0 (myy601@myy601lab2) (gcc version 8.3.0 (Debian 8.3.0-6) ) #35 Tue May 24 23:24:47 EEST 2022
[ 0.000000] bootmem address range: 0x7fea89c00000 - 0x7fea8ffff000
[ 0.000000] Built 1 zonelists in Zone order, mobility grouping on. Total pages: 25249
[ 0.000000] Kernel command line: mem=100M virtio_mmio.device=292@0x1000000:1
[ 0.000000] PID hash table entries: 512 (order: 0, 4096 bytes)
[ 0.000000] Dentry cache hash table entries: 16384 (order: 5, 131072 bytes)
[ 0.000000] Inode-cache hash table entries: 8192 (order: 4, 65536 bytes)
[ 0.000000] Memory available: 100752k/0k RAM
[ 0.000000] SLUB: HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[ 0.000000] NR_IRQS:4096
[ 0.000000] lkl: irqs initialized
[ 0.000000] clocksource: lkl: mask: 0xffffffffffffffff max_cycles: 0x1cd42e4dffb, max_idle_ns: 881590591483 ns
[ 0.000011] lkl: time and timers initialized (irq2)
[ 0.000011] pid_max: default: 4096 minimum: 301
[ 0.000026] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[ 0.000028] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)
[ 0.007829] console [lkl_console0] enabled
[ 0.007853] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[ 0.007933] NET: Registered protocol family 16
[ 0.009849] clocksource: Switched to clocksource lkl
[ 0.009967] NET: Registered protocol family 2
[ 0.010318] TCP established hash table entries: 1024 (order: 1, 8192 bytes)
[ 0.010344] TCP bind hash table entries: 1024 (order: 1, 8192 bytes)
[ 0.010350] TCP: Hash tables configured (established 1024 bind 1024)
[ 0.010689] UDP hash table entries: 128 (order: 0, 4096 bytes)
[ 0.010728] UDP-Lite hash table entries: 128 (order: 0, 4096 bytes)
[ 0.010832] virtio-mmio: Registering device virtio-mmio.0 at 0x1000000-0x1000123, IRQ 1.
[ 0.011619] workingset: timestamp_bits=62 max_order=15 bucket_order=0
[ 0.033752] io scheduler noop registered
[ 0.033782] io scheduler deadline registered
[ 0.033835] io scheduler cfq registered (default)
[ 0.033853] io scheduler mq-deadline registered
[ 0.033867] virtio-mmio virtio-mmio.0: Failed to enable 64-bit or 32-bit DMA. Trying to continue, but this might not work.
[ 0.038381] vda:
[ 0.038687] NET: Registered protocol family 10
[ 0.039320] Segment Routing with IPv6
[ 0.039366] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[ 0.039796] Warning: unable to open an initial console.
[ 0.039846] This architecture does not have kernel memory protection.
[ 0.039968] reached generic_file_read_iter in filemap
[ 0.041157] EIMASTE STO fat_fill_super sto inode.c
[ 0.041233] 0 BUFFER EXEI MESA = sbi->cluster_size =2048,sbi->cluster_bits=(null),sbi->fats = ,sbi->fat_bits=
```

```

[ 0.039846] This architecture does not have kernel memory protection.
[ 0.039968] reached generic_file_read_iter in filemap
[ 0.041157] EIMASTE STO fat_fill_super sto inode.c
[ 0.041233] 0 BUFFER EXEI MESA = sbi->cluster_size =2048,sbi->cluster_bits=(null),sbi->fat_bits=
[ 0.041239] STARTING WRITING IN JOURNAL
[ 0.041245] reached generic_file_write_iter in filemap
[ 0.041259] END OF WRITING IN JOURNAL
[ 0.041264] FINISHED
[ 0.041267] EIMASTE STO fat_fill_super sto inode.c
[ 0.041270] 0 BUFFER EXEI MESA = sbi->vol_id =-196441731
[ 0.041270]
[ 0.041273] STARTING WRITING IN JOURNAL
[ 0.041465] reached generic_file_write_iter in filemap
[ 0.041488] END OF WRITING IN JOURNAL
[ 0.041549] FINISHED
[ 0.041553] EIMASTE STO fat_fill_super sto inode.c
[ 0.041558] 0 BUFFER EXEI MESA = sbi->dir_per_block =16, sbi->dir_per_block_bits =4,sbi->dir_start =404,sbi->dir_entries =(null)
[ 0.041558]
[ 0.041562] STARTING WRITING IN JOURNAL
[ 0.041567] reached generic_file_write_iter in filemap
[ 0.041579] END OF WRITING IN JOURNAL
[ 0.041586] FINISHED
[ 0.041589] EIMASTE STO fat_fill_super sto inode.c
[ 0.041594] 0 BUFFER EXEI MESA = sbi->data_start =436
[ 0.041594]
[ 0.041598] STARTING WRITING IN JOURNAL
[ 0.041601] reached generic_file_write_iter in filemap
[ 0.041606] END OF WRITING IN JOURNAL
[ 0.041643] FINISHED
[ 0.041649] EIMASTE STO fat_fill_super sto inode.c
[ 0.041654] 0 BUFFER EXEI MESA = sbi->fat_bits =
[ 0.041654]
[ 0.041660] STARTING WRITING IN JOURNAL
[ 0.041721] reached generic_file_write_iter in filemap
[ 0.041732] END OF WRITING IN JOURNAL
[ 0.041876] FINISHED
[ 0.041883] EIMASTE STO fat_fill_super sto inode.c
[ 0.041886] 0 BUFFER EXEI MESA = sbi->dirty =0
[ 0.041886]
[ 0.041915] STARTING WRITING IN JOURNAL
[ 0.041922] reached generic_file_write_iter in filemap
[ 0.041927] END OF WRITING IN JOURNAL
[ 0.042186] FINISHED
[ 0.042192] EIMASTE STO fat_fill_super sto inode.c
[ 0.042195] 0 BUFFER EXEI MESA = sbi->max_cluster =51093

```

```

[ 0.041643] FINISHED
[ 0.041649] EIMASTE STO fat_fill_super sto inode.c
[ 0.041654] 0 BUFFER EXEI MESA = sbi->fat_bits =
[ 0.041654]
[ 0.041660] STARTING WRITING IN JOURNAL
[ 0.041721] reached generic_file_write_iter in filemap
[ 0.041732] END OF WRITING IN JOURNAL
[ 0.041876] FINISHED
[ 0.041883] EIMASTE STO fat_fill_super sto inode.c
[ 0.041886] 0 BUFFER EXEI MESA = sbi->dirty =0
[ 0.041886]
[ 0.041915] STARTING WRITING IN JOURNAL
[ 0.041922] reached generic_file_write_iter in filemap
[ 0.041927] END OF WRITING IN JOURNAL
[ 0.042186] FINISHED
[ 0.042192] EIMASTE STO fat_fill_super sto inode.c
[ 0.042195] 0 BUFFER EXEI MESA = sbi->max_cluster =51093
[ 0.042195]
[ 0.042199] STARTING WRITING IN JOURNAL
[ 0.042228] reached generic_file_write_iter in filemap
[ 0.042236] END OF WRITING IN JOURNAL
[ 0.042243] FINISHED
[ 0.042269] EIMASTE STO fat_ent_access_init sto fatent.c
[ 0.042274] 0 BUFFER EXEI MESA = sbi->fatent_shift =1,sbi->fatent_ops = 804257376
[ 0.042274]
[ 0.042279] STARTING WRITING IN JOURNAL
[ 0.042299] reached generic_file_write_iter in filemap
[ 0.042303] END OF WRITING IN JOURNAL
[ 0.042309] FINISHED
[ 0.042334] EIMASTE STO fat_fill_super sto inode.c
[ 0.042337] 0 BUFFER EXEI MESA = sbi->nls_disk =804247072
[ 0.042337]
[ 0.042340] STARTING WRITING IN JOURNAL
[ 0.042343] reached generic_file_write_iter in filemap
[ 0.042347] END OF WRITING IN JOURNAL
[ 0.042368] FINISHED
[ 0.042371] EIMASTE STO fat_fill_super sto inode.c
[ 0.042374] 0 BUFFER EXEI MESA = sbi->nls_io =804248608
[ 0.042374]
[ 0.042376] STARTING WRITING IN JOURNAL
[ 0.042379] reached generic_file_write_iter in filemap
[ 0.042381] END OF WRITING IN JOURNAL
[ 0.042384] FINISHED
Segmentation fault

```

## Επίλογος

Όπως βλέπουμε στην εικόνα παραπάνω, μας εμφανίζει Segmentation fault. Δεν είμαστε σίγουροι για τον λόγο .

Η αλήθεια είναι ότι μας κίνησε πολύ την περιέργεια το ερώτημα 2 και εμβαθύναμε αρκετά τις γνώσεις μας πάνω σε αυτό. Θα θέλαμε πολύ να μας λυθούν αυτές οι απορίες κάποια στιγμή !