Χήτα Δανάη – 4838

ΜΥΥ601 Λειτουργικά Συστήματα

Υλοποίηση αρχείου καταγραφής στο σύστημα αρχείων FAT του Linux

Περιεχόμενα:

I. Κατανόηση της λειτουργίας του συστήματος χρησιμοποιώντας τις εφαρμογές cptofs, cpfromfs, fs2tar ,boot και προσθήκη printk συναρτήσεων στο εσωτερικό του Linux Kernel Library.

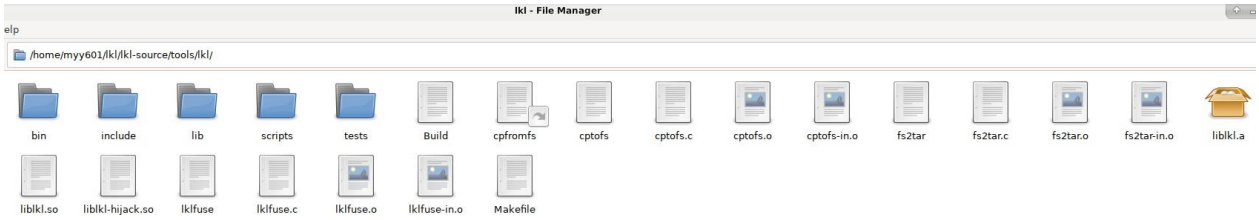
II. Αποθήκευση αλλαγών και τροποποιήσεων του FAT με χρήση συναρτήσεων open, write και άλλα .

III. Μια πιο ρεαλιστική υλοποίηση του συστήματος καταγραφής με χρήση των κλήσεων συστήματος sys_open, sys_write και άλλα.

I. Κατανόηση της λειτουργίας του συστήματος

Πρώτα από όλα, ανοίξαμε το τερματικό στην εικονική μηχανή (VMware Player v15) και έχοντας μετακινηθεί στον κατάλογο /lkl/lkl-source, εκτελέσαμε τις παρακάτω εντολές έτσι ώστε να γίνει compile (μεταγλώττιση):

- make -C tools/lkl clean
- make -j 4 -C tools/lkl



Βλέπουμε ότι όντως δημιουργήθηκαν αρχεία τύπου .h και .o, τα οποία μας βοηθάνε στην εκτέλεση κώδικα.

Στη συνέχεια μέσα στον κατάλογο /tools/lkl/ για να επιβεβαιωθεί ότι το σύστημα αρχείων είναι mounted, δηλαδή προσβάσιμο, εκτελέσαμε:

- mount -t vfat -o loop /tmp/vfatfile /vfat

Και μετά:

- make test

Αφού έγιναν αυτές οι αρχικές εντολές, εκτελέσαμε την εφαρμογή cptofs, με την οποία γίνεται αντιγραφή του αρχείου lklfuse.c στον κατάλογο / του /tmp/vfatfile, πληκτρολογώντας την εξής εντολή στο τερματικό :

- ./cptofs -i /tmp/vfatfile -p -t vfat lklfuse.c /

Επίσης, για να ελέγξουμε ότι το αρχείο αντιγράφηκε σωστά, κάναμε τα εξής (όπως ακριβώς αναγράφονται και στην εκφώνηση της άσκησης):

- su root (ωστέ να είμαστε χρήστες root)
- umount /vfat
- mount -t vfat -o loop /tmp/vfatfile /vfat

Πράγματι, παρατηρήσαμε την εξής αλλαγή στο σύστημα: Ότι μέσα στον κατάλογο /vfat/, στη ρίζα της εικονικής μηχανής, μπορούσαμε να δούμε το lklfuse.c ως εικονίδιο κανονικά.

Στο επόμενο μας βήμα, φτιάξαμε έναν φάκελο με το όνομα mytest που περιέχει το αρχείο lklfuse.c και το lklfuse2.c , ίδια μεταξύ τους, για να δούμε την συμπεριφορά του προγράμματος και με καταλόγους. Πάλι σε αυτή την περίπτωση δεν παρουσιάστηκε κάποιο πρόβλημα στην διαδικασία. Εκτελέσαμε την παρακάτω εντολή:

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

Στη συνέχεια, αναζητήσαμε τα αρχεία στα οποία αναγράφονται οι δομές που περιέχουν τις λειτουργίες των Inode, Superblock, File Allocation Table, Files, Directory Entries, καθώς και συναρτήσεις μέσα στο linux kernel library. Στην εκφώνηση μας δίνονται ακριβώς οι λειτουργίες και σε ποιο αρχείο βρίσκεται η καθεμία:

Οι λειτουργίες του superblock ορίζονται στο αρχείο *fs/fat/inode.c*, και πιο συγκεκριμένα στη δομή **struct super_operations fat_sops**. Στο ίδιο αρχείο, αλλά στη δομή **struct address_space_operations fat_aops**, βρίσκονται οι λειτουργίες μνήμης. Οι λειτουργίες εγγραφών FAT, ορίζονται στο αρχείο *fs/fat/fatent.c*, στη δομή **struct fatent_operations fat12/16/32_ops**. Οι λειτουργίες αρχείου ορίζονται στην **δομή struct file_operations fat_file_operations** του αρχείου *fs/fat/file.c*. Ορισμένες από τις λειτουργίες inode ορίζονται στην δομή **struct inode_operations fat_file_inode_operations** του αρχείου *fs/fat/file.c*. Τέλος, οι λειτουργίες των καταλόγων ορίζονται στην 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 από τις παραπάνω επιλέξαμε με δική μας πρωτοβουλία σε ποιες συναρτήσεις θα προσθέσουμε την εντολή printk, ώστε να δούμε αν και πότε εκτελείται η κάθε συνάρτηση. Παρακάτω, παραθέτουμε αναλυτικά ποια αρχεία τροποποιήσαμε και σε ποιες γραμμές τοποθετήσαμε το printk.

inode.c

- 186
- 193
- 199
- 206
- 232
- 250
- 286
- 324
- 659
- 744
- 757
- 770
- 824
- 851
- 890
- 934
- 1047

fatent.c

- 31
- 42
- 57
- 64
- 71
- 97
- 122
- 141
- 151
- 161
- 185
- 195
- 204
- 237
- 250
- 263

file.c

- 129
- 148
- 160
- 172
- 246
- 385
- 521

namei_vfat.c

- 785
- 825
- 856
- 883
- 927
- 950

Μετά την τοποθέτηση των printk, εκτελέσαμε και πάλι την εντολή:

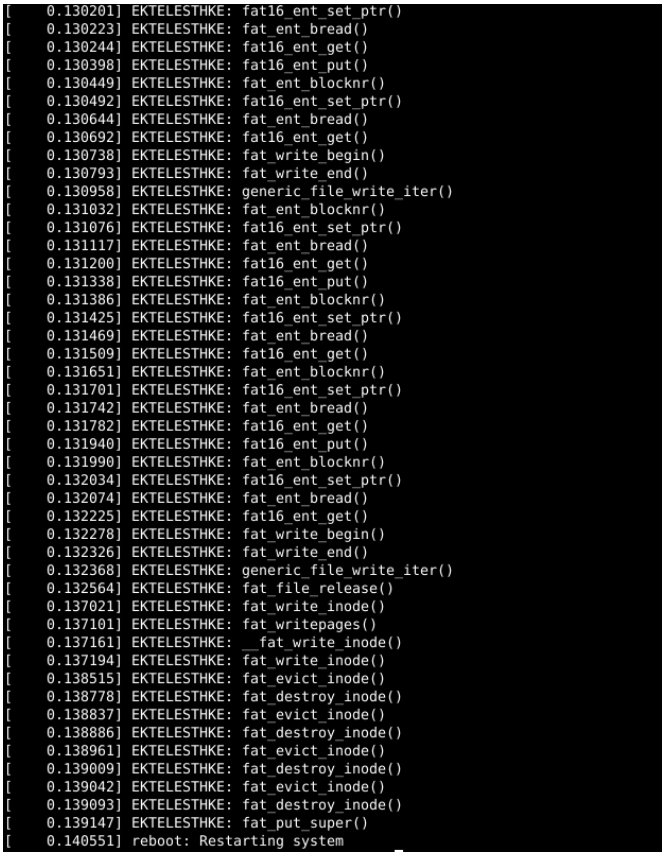
- ./cptofs -i /tmp/vfatfile -p -t vfat lklfuse.c /

Αυτή τη φορά στο τερματικό θα τυπωθούν και τα printk των λειτουργιών που εμπλέκονται κατά την προσπέλαση του αρχείου /tmp/vfatfile.

Τα αποτελέσματα που εμφανίστηκαν στο τερματικό είναι τα εξής:

```
myy601@myy601lab2:~/lkl/lkl-source/tools/lkl$ ./cptofs -i /tmp/vfatfile -p -t vfat lklfuse.c /
[    0.000000] Linux version 4.11.0 (myy601@myy601lab2) (gcc version 8.3.0 (Debian 8.3.0-6) ) #10 Mon May 23 19:43:05 EEST 2022
[    0.000000] bootmem address range: 0x7f0821c00000 - 0x7f0827fff000
[    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
[    0.000017] lkl: time and timers initialized (irq2)
[    0.000050] pid_max: default: 4096 minimum: 301
[    0.000120] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[    0.000139] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)
[    0.016482] console [lkl_console0] enabled
[    0.016545] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[    0.016929] NET: Registered protocol family 16
[    0.024373] clocksource: Switched to clocksource lkl
[    0.024616] NET: Registered protocol family 2
[    0.024900] TCP established hash table entries: 1024 (order: 1, 8192 bytes)
[    0.024937] TCP bind hash table entries: 1024 (order: 1, 8192 bytes)
[    0.024971] TCP: Hash tables configured (established 1024 bind 1024)
[    0.025380] UDP hash table entries: 128 (order: 0, 4096 bytes)
[    0.025419] UDP-Lite hash table entries: 128 (order: 0, 4096 bytes)
[    0.025606] virtio-mmio: Registering device virtio-mmio.0 at 0x1000000-0x1000123, IRQ 1.
[    0.028497] workingset: timestamp_bits=62 max_order=15 bucket_order=0
[    0.109081] io scheduler noop registered
[    0.109126] io scheduler deadline registered
[    0.109199] io scheduler cfq registered (default)
[    0.109219] io scheduler mq-deadline registered
[    0.109250] virtio-mmio virtio-mmio.0: Failed to enable 64-bit or 32-bit DMA.  Trying to continue, but this might not work.
[    0.116708]  vda:
[    0.117043] NET: Registered protocol family 10
[    0.119615] Segment Routing with IPv6
[    0.119672] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[    0.120519] Warning: unable to open an initial console.
[    0.120575] This architecture does not have kernel memory protection.
[    0.120606] EKTELESTHKE: generic_file_read_iter()
[    0.122675] EKTELESTHKE: fat_alloc_inode()
[    0.122795] EKTELESTHKE: fat_alloc_inode()
[    0.122837] EKTELESTHKE: fat_alloc_inode()
[    0.124024] EKTELESTHKE: fat_alloc_inode()
[    0.124149] EKTELESTHKE: vfat_create()
[    0.124224] EKTELESTHKE: fat_ent_blocknr()
[    0.124316] EKTELESTHKE: fat16_ent_set_ptr()
```

```
[    0.124316] EKTELESTHKE: fat16_ent_set_ptr()
[    0.124607] EKTELESTHKE: fat_ent_bread()
[    0.124652] EKTELESTHKE: fat16_ent_get()
[    0.124693] EKTELESTHKE: fat16_ent_put()
[    0.124752] EKTELESTHKE: fat_ent_blocknr()
[    0.124792] EKTELESTHKE: fat16_ent_set_ptr()
[    0.124932] EKTELESTHKE: fat_ent_bread()
[    0.124977] EKTELESTHKE: fat16_ent_get()
[    0.125016] EKTELESTHKE: fat16_ent_put()
[    0.125063] EKTELESTHKE: fat_ent_blocknr()
[    0.125103] EKTELESTHKE: fat16_ent_set_ptr()
[    0.125686] EKTELESTHKE: fat_ent_bread()
[    0.125711] EKTELESTHKE: fat16_ent_get()
[    0.125733] EKTELESTHKE: fat_ent_blocknr()
[    0.125755] EKTELESTHKE: fat16_ent_set_ptr()
[    0.125776] EKTELESTHKE: fat_ent_bread()
[    0.125794] EKTELESTHKE: fat16_ent_get()
[    0.125813] EKTELESTHKE: fat16_ent_put()
[    0.125835] EKTELESTHKE: fat_ent_blocknr()
[    0.125853] EKTELESTHKE: fat16_ent_set_ptr()
[    0.125877] EKTELESTHKE: fat_ent_bread()
[    0.125904] EKTELESTHKE: fat16_ent_get()
[    0.125931] EKTELESTHKE: fat_write_begin()
[    0.126011] EKTELESTHKE: fat_write_end()
[    0.126035] EKTELESTHKE: generic_file_write_iter()
[    0.126076] EKTELESTHKE: fat_ent_blocknr()
[    0.126157] EKTELESTHKE: fat16_ent_set_ptr()
[    0.126181] EKTELESTHKE: fat_ent_bread()
[    0.126203] EKTELESTHKE: fat16_ent_get()
[    0.126224] EKTELESTHKE: fat16_ent_put()
[    0.126379] EKTELESTHKE: fat_ent_blocknr()
[    0.126422] EKTELESTHKE: fat16_ent_set_ptr()
[    0.126465] EKTELESTHKE: fat_ent_bread()
[    0.126626] EKTELESTHKE: fat16_ent_get()
[    0.126674] EKTELESTHKE: fat_ent_blocknr()
[    0.126717] EKTELESTHKE: fat16_ent_set_ptr()
[    0.126758] EKTELESTHKE: fat_ent_bread()
[    0.126910] EKTELESTHKE: fat16_ent_get()
[    0.126959] EKTELESTHKE: fat16_ent_put()
[    0.127005] EKTELESTHKE: fat_ent_blocknr()
[    0.127045] EKTELESTHKE: fat16_ent_set_ptr()
[    0.127196] EKTELESTHKE: fat_ent_bread()
[    0.127244] EKTELESTHKE: fat16_ent_get()
[    0.127290] EKTELESTHKE: fat_ent_blocknr()
[    0.127332] EKTELESTHKE: fat16_ent_set_ptr()
[    0.127480] EKTELESTHKE: fat_ent_bread()
[    0.127528] EKTELESTHKE: fat16_ent_get()
[    0.127570] EKTELESTHKE: fat16_ent_put()
[    0.127570] EKTELESTHKE: fat16_ent_put()
[    0.127615] EKTELESTHKE: fat_ent_blocknr()
[    0.127764] EKTELESTHKE: fat16_ent_set_ptr()
[    0.127812] EKTELESTHKE: fat_ent_bread()
[    0.127854] EKTELESTHKE: fat16_ent_get()
[    0.127897] EKTELESTHKE: fat_ent_blocknr()
[    0.128051] EKTELESTHKE: fat16_ent_set_ptr()
[    0.128095] EKTELESTHKE: fat_ent_bread()
[    0.128134] EKTELESTHKE: fat16_ent_get()
[    0.128175] EKTELESTHKE: fat16_ent_put()
[    0.128221] EKTELESTHKE: fat_ent_blocknr()
[    0.128262] EKTELESTHKE: fat16_ent_set_ptr()
[    0.128424] EKTELESTHKE: fat_ent_bread()
[    0.128472] EKTELESTHKE: fat16_ent_get()
[    0.128517] EKTELESTHKE: fat_write_begin()
[    0.128569] EKTELESTHKE: fat_write_end()
[    0.128717] EKTELESTHKE: generic_file_write_iter()
[    0.128787] EKTELESTHKE: fat_ent_blocknr()
[    0.128880] EKTELESTHKE: fat16_ent_set_ptr()
[    0.129294] EKTELESTHKE: fat_ent_bread()
[    0.129319] EKTELESTHKE: fat16_ent_get()
[    0.129341] EKTELESTHKE: fat16_ent_put()
[    0.129364] EKTELESTHKE: fat_ent_blocknr()
[    0.129385] EKTELESTHKE: fat16_ent_set_ptr()
[    0.129406] EKTELESTHKE: fat_ent_bread()
[    0.129428] EKTELESTHKE: fat16_ent_get()
[    0.129450] EKTELESTHKE: fat_ent_blocknr()
[    0.129472] EKTELESTHKE: fat16_ent_set_ptr()
[    0.129492] EKTELESTHKE: fat_ent_bread()
[    0.129514] EKTELESTHKE: fat16_ent_get()
[    0.129705] EKTELESTHKE: fat16_ent_put()
[    0.129730] EKTELESTHKE: fat_ent_blocknr()
[    0.129751] EKTELESTHKE: fat16_ent_set_ptr()
[    0.129769] EKTELESTHKE: fat_ent_bread()
[    0.129789] EKTELESTHKE: fat16_ent_get()
[    0.129865] EKTELESTHKE: fat_ent_blocknr()
[    0.129889] EKTELESTHKE: fat16_ent_set_ptr()
[    0.129910] EKTELESTHKE: fat_ent_bread()
[    0.129931] EKTELESTHKE: fat16_ent_get()
[    0.129952] EKTELESTHKE: fat16_ent_put()
[    0.130031] EKTELESTHKE: fat_ent_blocknr()
[    0.130055] EKTELESTHKE: fat16_ent_set_ptr()
[    0.130077] EKTELESTHKE: fat_ent_bread()
[    0.130098] EKTELESTHKE: fat16_ent_get()
[    0.130177] EKTELESTHKE: fat_ent_blocknr()
[    0.130201] EKTELESTHKE: fat16_ent_set_ptr()
[    0.130223] EKTELESTHKE: fat_ent_bread()
[    0.130244] EKTELESTHKE: fat16_ent_get()
[    0.130398] EKTELESTHKE: fat16_ent_put()
```



Σύμφωνα με την τελευταία γραμμή του τερματικού *[0.140551] reboot: Restarting system*, αφού τερματίσει η εφαρμογή, επανεκκινείται ο μικροπυρήνας. Από αυτό καταλαβαίνουμε ότι το σύστημα δεν κλείνει ποτέ.

ΙΙ.Αποθήκευση αλλαγών καΙ τροποποΙήσεων του FAT με χρήση συναρτήσεων open, write καΙ άλλα.

Σύμφωνα με την εκφώνηση θα έπρεπε να γίνει καταγραφή των αλλαγών των δομών του συστήματος FAT όπως superblock, file allocation table, directory entries και δεδομένα αρχείων μέσα σε ένα αρχείο journal στο τοπικό σύστημα αρχείων ext4 της εικονικής μηχανής, χρησιμοποιώντας κανονικές κλήσεις συστήματος (open , write) .

Πρώτα από όλα , αντιληφθήκαμε πως θα εργαστούμε σε επίπεδο χρήστη και όχι μικροπυρήνα , οπότε αναζητήσαμε τα αρχεία που λειτουργούν ακριβώς με αυτό τον τρόπο. Τα αρχεία που εντοπίσαμε είναι τα cptofs.c , boot.c , fs2tar.c , οπότε προσπαθήσαμε να εργαστούμε πάνω σε αυτά .

Σκεφτήκαμε να υλοποιήσουμε συναρτήσεις που θα μας βοηθήσουν στο άνοιγμα και στην καταγραφή μιας δομής , το καταφέραμε αλλά όχι σε άριστο βαθμό , αντιμετωπίσαμε κάποια προβλήματα στην υλοποίηση .

Το βασικότερο όμως πρόβλημα μας , σε αυτό το 2^{ο} ερώτημα της εργασίας είναι το ότι δεν μπορούσαμε να βρούμε τρόπο στο πως θα πάρουμε τα δεδομένα που θέλουμε από τις δομές του μικροπυρήνα στο επίπεδο χρήστη ,έτσι ώστε να τις καταγράψουμε στο αρχείο journal ,που βρίσκεται στο ίδιο το FAT , με την μέθοδο write .

Επομένως , σε αυτό το κομμάτι της εργασίας, αντιμετωπίσαμε πολλές δυσκολίες και δεν έχουμε να σας παρουσιάσουμε κάποια ολοκληρωμένη υλοποίηση .

III.Μια πιο ρεαλιστική υλοποίηση του συστήματος καταγραφής με χρήση των κλήσεων συστήματος sys_open, sys_write και άλλα.

Αφού δεν καταφέραμε την δεύτερη υλοποίηση επειδή δεν μπορούσαμε να εντοπίσουμε πως θα μεταφέρουμε τις πληροφορίες από τον μικροπυρήνα στο επίπεδο χρήστη για να τις γράψουμε στο αρχείο καταγραφής, ασχοληθήκαμε με αυτή την πιο ρεαλιστική υλοποίηση , όπου οι αλλαγές γίνονται στα αρχεία του Linux Kernel Library (LKL), με εσωτερικές κλήσεις και άρα δεν χρειάζεται να μεταφέρουμε τις πληροφορίες στο επίπεδο χρήστη.

Πρώτα από όλα, αποθηκεύσαμε έναν ακέραιο (myfilenumber) σε μία από τις βασικές δομές του συστήματος FAT, στο αρχείο fat.h, μέσα στο superblock στη δομή msdos_sb_info.

Σε όποια αρχεία εκτελούμε sys_open και sys_write, γράφουμε στην αρχή τους:

- #include <linux/syscalls.h>

Στη συνέχεια, μεταφερθήκαμε στο αρχείο inode.c, στο οποίο αρχικοποιούνται τα πεδία της δομής struct msdos_sb_info. Στη συνάρτηση fat_fill_super, στη σειρά 1718, κάνουμε sys_open το αρχείο journal, στο οποίο θα γίνεται η καταγραφή κατά την αρχικοποίηση του συστήματος αρχείων, ορίζοντας τα δικαιώματά του. Η γραμμή κώδικα που γράψαμε είναι η εξής:

- sbi->myfilenumber = sys_open("journal.txt", O_CREAT | O_RDWR | O_APPEND | O_EXCL, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);

Ορίσαμε αυτά τα δικαιώματα για τους παρακάτω λόγους:

- O_CREAT: θέλουμε αν δεν υπάρχει αρχείο να το δημιουργεί.
- O_RDWR: για να μπορούμε να διαβάσουμε και να γράψουμε στο αρχείο.
- O_APPEND: κάθε write να γράφει στο τέλος του αρχείου τα δεδομένα.
- O_EXCL: να επιστρέψει σφάλμα σε περίπτωση που το αρχείο υπάρχει ήδη.
- S_IRUSR | S_IWUSR: θέλουμε ο χρήστης (user) να μπορεί να διαβάζει και να γράφει στο αρχείο.

- S_IRGRP | S_IWGRP: θέλουμε το group να μπορεί και να γράφει στο αρχείο.

Έπειτα, αναζητήσαμε στο τερματικό πού χρησιμοποιείται η κάθε μία από τις δομές του FAT. Στο αρχείο journal καταγράψαμε τις λειτουργίες που παρατηρήσαμε ότι έχουν την μεγαλύτερη μεταβλητότητα. Οι συναρτήσεις που επιλέξαμε να κάνουμε sys_write βρίσκονται στα αρχεία: inode.c, misc.c, fatent.c και namei_vfat.

Για την υλοποίηση της εντολής sys_write δημιουργήσαμε μια βοηθητική συνάρτηση στο αρχείο fat.h, με όνομα myfsync και όρισμα const struct msdos_sb_info *sbi. Η συνάρτηση αυτή είναι η εξής:

```
static inline void myfsync(const struct msdos_sb_info *sbi){
        sys_fsync(sbi->myfilenumber);
        sys_fdatasync(sbi->myfilenumber);
}
```

Οι εντολές που υλοποιούνται στη συνάρτηση μας αναγράφονται στις διαφάνειες του φροντιστηρίου.

Οπότε στις συναρτήσεις που τροποποιήσαμε καλούμε:

- printk(KERN_INFO "INSERTION: …..");
- sprintf(buf, ….);
- printk(KERN_INFO “BUF = %s\n\n”,buf);
- sys_write(sbi->myfilenumber, buf , sizeof(buf));
- myfsync(sbi);

Τα σημεία του κώδικα που μπήκαν οι συγκεκριμένες εντολές , επιλέχθηκαν από εμάς , με δική μας κρίση , ανάλογα με το ποιες λειτουργίες του FAT , θέλαμε να καταγραφούν στο αρχείο journal.

Τα αρχεία που τροποποιήθηκαν είναι τα εξής:

inode.c

- fat_set_state → 699 – 703, 711 – 715
- __ fat_write_inode → 907 – 911
- fat_fill_super → 1731 – 1735

misc.c

- fat_clusters_flush → 89 – 93, 98 – 102

namei_vfat

- vfat_build_slots → 649 – 653, 671 – 675

fatent.c

- fat_ent_access_init → 325 – 329

Υλοποιήσαμε ελέγχους που δείχνουν ότι οι εγγραφές αντιστοιχούν στις αναμενόμενες τροποποιήσεις από διάφορες εκτελέσεις εφαρμογών που προκαλούν δημιουργία αρχείων και καταλόγων στο σύστημα αρχείων FAT, χρησιμοποιώντας την εντολή cptofs που εκτελέσαμε και παραπάνω. Τα αποτελέσματα που τυπώθηκαν στο τερματικό (πέρα από αυτά που ήδη παραθέσαμε στην ενότητα Ι της αναφοράς μας) είναι τα εξής:

[0.019485] INSERTION: fat_fill_super

[0.019509] BUF = cluster_size= 2048, cluster_bits= 11, fats= 2

[0.019509]

[0.019509]

[0.019523] INSERTION: fat_ent_access_init

[0.019542] BUF = fatent_shift= 1, fatent_ops= 3355676256

[0.019542]

[0.019558] EKTELESTHKE: fat_alloc_inode()

[0.019560] EKTELESTHKE: fat_alloc_inode()

[0.019563] EKTELESTHKE: fat_alloc_inode()

[0.019918] INSERTION: fat_set_state

[0.019947] BUF = fat16.state = 1

[0.019947]

[0.020114] INSERTION: vfat_build_slots

[0.020186] BUF = id, attr, reserved,alias_checksum , start , name0_4 ,name5_10 , name11_12 = 0

[0.020186] ,0

[0.020186] , 0

[0.020186] , 0

[0.020186] , 62256161

[0.020186] , 62256174

[0.020186] , 62256188

[0.020186]

[0.020200] INSERTION: vfat_build_slots

[0.020203] BUF = name, attr, lcase, time, date = 62256160

[0.020203] ,32

[0.020203] , 0

[0.020203] , 0

[0.020203] , 33

.

.

.

[0.022369] EKTELESTHKE: __fat_write_inode()

[0.022375] INSERTION: __fat_write_inode

[0.022378] BUF = size , attr = 13958

[0.022378] ,32

[0.022378]

[0.022381] EKTELESTHKE: fat_write_inode()

[0.022547] EKTELESTHKE: fat_evict_inode()

[0.022569] EKTELESTHKE: fat_destroy_inode()

[0.022581] EKTELESTHKE: fat_evict_inode()

[0.022603] EKTELESTHKE: fat_destroy_inode()

[0.022605] INSERTION: fat_set_state

[0.022625] BUF = fat16.state = 0

[0.022625]