

Μαρίνος Αριστείου 5397,
Θεοφάνης Τομπόλης 4855,
Αθανάσιος Φυτιλής 5381

ΜΥΥ601 Λειτουργικά Συστήματα

Αναφορά : Εργαστηριακή Άσκηση 2

Εισαγωγή:

Στο πλαίσιο της παρούσας εργαστηριακής άσκησης, στόχος ήταν η υλοποίηση ενός μηχανισμού καταγραφής (logging/journaling) βασικών λειτουργιών του συστήματος αρχείων VFAT, αξιοποιώντας την Linux Kernel Library (LKL). Η LKL επιτρέπει την εκτέλεση του πυρήνα του Linux σε επίπεδο χρήστη, προσφέροντας μια ευέλικτη πλατφόρμα για ανάπτυξη και δοκιμή. Η υλοποίηση επικεντρώθηκε στην προσθήκη κώδικα ώστε να καταγράφεται η εκτέλεση επιλεγμένων συναρτήσεων του VFAT, μαζί με κρίσιμες παραμέτρους τους, σε ένα εξωτερικό αρχείο καταγραφής. Με αυτόν τον τρόπο, διευκολύνεται η παρακολούθηση και ανάλυση της συμπεριφοράς του συστήματος αρχείων κατά την εκτέλεση πραγματικών λειτουργιών, ενισχύοντας την κατανόηση της λειτουργίας του και επιτρέποντας μελλοντικές βελτιστοποιήσεις ή εντοπισμό σφαλμάτων.

1.1 Υλοποίηση Μηχανισμού Logging :

Εισαγωγή:

Στην ενότητα αυτή περιγράφεται αναλυτικά η διαδικασία υλοποίησης του μηχανισμού καταγραφής λειτουργιών του συστήματος αρχείων VFAT, εντός του περιβάλλοντος της Linux Kernel Library (LKL). Αρχικά, προσδιορίστηκαν οι βασικές λειτουργίες του VFAT που χρήζουν παρακολούθησης και καταγραφής. Στη συνέχεια, εντοπίστηκαν τα κατάλληλα σημεία στον πηγαίο κώδικα για την εισαγωγή των εντολών καταγραφής. Η καταγραφή πραγματοποιείται σε εξωτερικό αρχείο του τοπικού συστήματος αρχείων (ext4), ώστε να είναι άμεσα προσβάσιμη και αναγνώσιμη. Η υλοποίηση καλύπτει το άνοιγμα του αρχείου log, την εισαγωγή των απαραίτητων εγγραφών κατά την εκτέλεση λειτουργιών VFAT και το ασφαλές κλείσιμο του αρχείου. Η ενότητα αυτή παρουσιάζει βήμα-βήμα τις ενέργειες που ακολουθήθηκαν, τις τροποποιήσεις στον πηγαίο κώδικα και τις τεχνικές προκλήσεις που αντιμετωπίστηκαν.

Προσέγγιση

Η προσπάθεια άμεσης χρήσης κλήσεων συστήματος της LKL (`lkl_sys_*`) εντός του κώδικα του πυρήνα οδήγησε σε προβλήματα μεταγλώττισης και σφαλμάτων εκτέλεσης (segmentation faults). Για την αποφυγή αυτών των δυσκολιών, επιλέχθηκε η υλοποίηση ενός μηχανισμού καταγραφής μέσω "callback" συναρτήσεων που ορίζονται εκτός του πυρήνα, στον χώρο χρήστη (user space).

Βοηθητικές Συναρτήσεις

Υλοποιήθηκε το αρχείο `tools/lkl/fat_log.c`, το οποίο περιλαμβάνει τρεις βασικές συναρτήσεις για τον χειρισμό του αρχείου καταγραφής:

- `lkl_fat_log_init()`: Ανοίγει το αρχείο `journal.log` στον κατάλογο `tests/` με χρήση της `fopen()` σε λειτουργία προσάρτησης ("a"), επιτρέποντας τη διατήρηση προηγούμενων καταγραφών. (Στην αρχή ήταν "w")
- `lkl_fat_log_write(const char *msg)`: Καταγράφει το παρεχόμενο μήνυμα στο αρχείο χρησιμοποιώντας `fprintf()`, ενώ καλεί άμεσα `fflush()` για άμεση αποθήκευση στο δίσκο.
- `lkl_fat_log_close()`: Κλείνει με ασφάλεια το αρχείο καταγραφής μετά από `fflush()`.

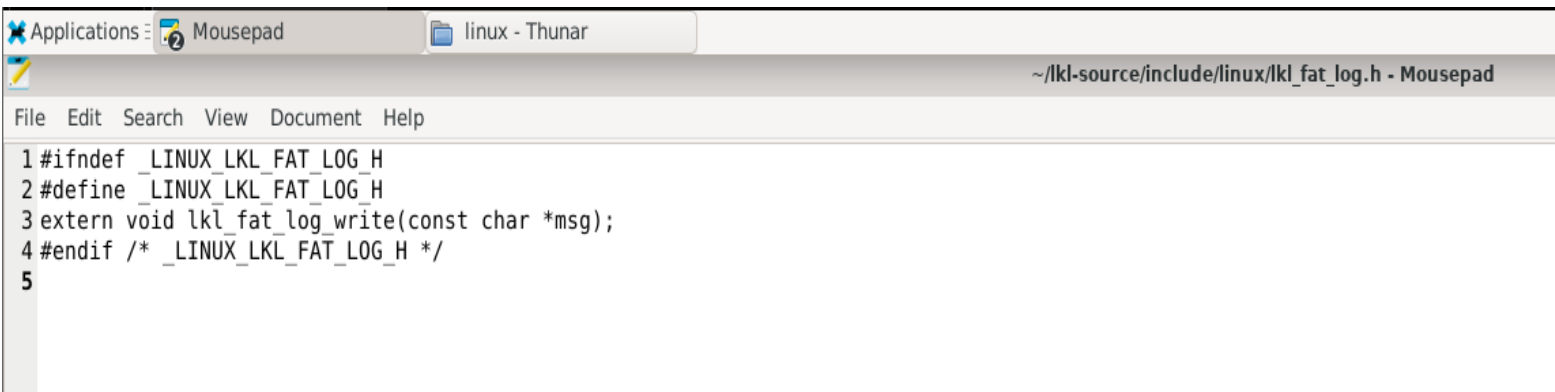
```
~/lkl-source/tools/lkl/fat_log.c - Mousepad
File Edit Search View Document Help

2 #include <string.h>
3 #include <lkl/lkl_fat_log.h> // Το δικό μας header
4
5 static FILE *lkl_fat_journal_file = NULL; // Χρησιμοποιούμε FILE* αντί για long fd
6
7 void lkl_fat_log_init(void) {
8     if (lkl_fat_journal_file == NULL) {
9         // Χρησιμοποιούμε fopen
10        // Θα δημιουργήσει το αρχείο αν δεν υπάρχει, αλλά αν υπάρχει, θα προσθέτει τις νέες γραμμές στο τέλος του, χωρίς να σβήνει τα παλιά περιεχόμενα.
11        lkl_fat_journal_file = fopen("journal.log", "a");
12        if (lkl_fat_journal_file == NULL) {
13            // Χρησιμοποιούμε perror για να δούμε το σφάλμα του συστήματος
14            perror("LKL_FAT_LOG: Failed to fopen journal.log");
15        } else {
16            fprintf(stderr, "LKL_FAT_LOG: fopen'd journal.log successfully.\n");
17        }
18    }
19 }
20
21 void lkl_fat_log_write(const char *msg) {
22     if (lkl_fat_journal_file != NULL) {
23         // Χρησιμοποιούμε fprintf αντί για lkl_sys_write
24         int written = fprintf(lkl_fat_journal_file, "%s", msg);
25         if (written < 0) {
26             perror("LKL_FAT_LOG: Error fprintf to journal.log");
27         }
28         // Κάνουμε flush για να είμαστε σίγουροι ότι γράφτηκε (σημαντικό!)
29         fflush(lkl_fat_journal_file);
30     }
31 }
32
33 void lkl_fat_log_close(void) {
34     if (lkl_fat_journal_file != NULL) {
35         FILE *file_to_close = lkl_fat_journal_file;
36         lkl_fat_journal_file = NULL;
37         fflush(file_to_close);
38         int ret = fclose(file_to_close);
39         if (ret != 0) {
40             perror("LKL_FAT_LOG: Failed to fclose journal.log");
41         } else {
42             fprintf(stderr, "LKL_FAT_LOG: fclose'd journal.log successfully.\n");
43         }
44     }
45 }
```

Header Files

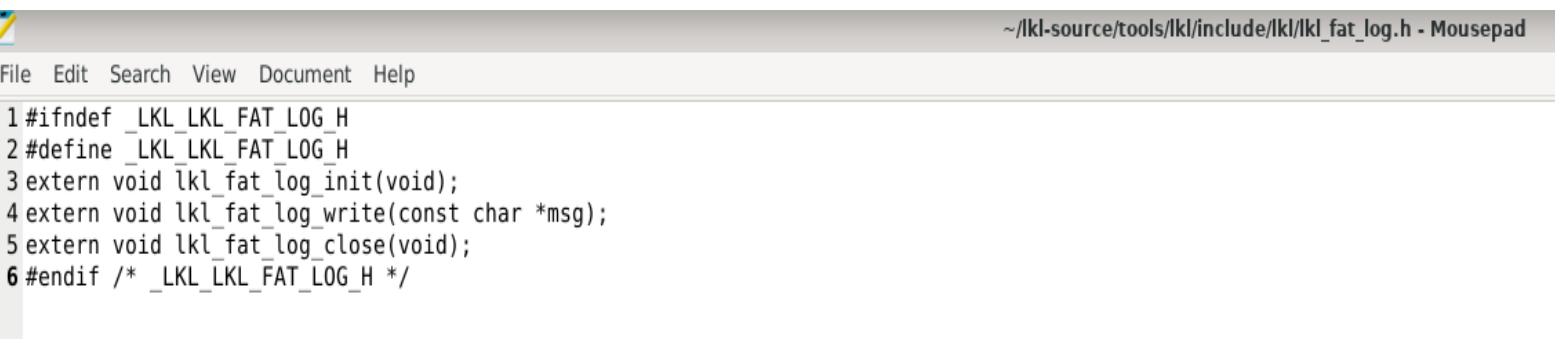
Για τη διασύνδεση των παραπάνω συναρτήσεων με τον κώδικα τόσο του χρήστη όσο και του πυρήνα, δημιουργήθηκαν δύο αρχεία κεφαλίδας:

- `tools/lkl/include/lkl/lkl_fat_log.h`: Περιλαμβάνει τις δηλώσεις όλων των παραπάνω συναρτήσεων και χρησιμοποιείται από τον κώδικα των εργαλείων.
- `include/linux/lkl_fat_log.h`: Περιλαμβάνει μόνο τη `lkl_fat_log_write()`, ώστε να είναι προσβάσιμη από επιλεγμένα σημεία εντός του πυρήνα (π.χ. `inode.c`, `dir.c`, `fatent.c`).



The screenshot shows a window titled "linux - Thunar" with a sub-window titled "Mousepad". The file path in the title bar is "~/lkl-source/include/linux/lkl_fat_log.h - Mousepad". The code content is as follows:

```
1 #ifndef _LINUX_LKL_FAT_LOG_H
2 #define _LINUX_LKL_FAT_LOG_H
3 extern void lkl_fat_log_write(const char *msg);
4 #endif /* _LINUX_LKL_FAT_LOG_H */
5
```



The screenshot shows a window titled "tools/lkl/include" with a sub-window titled "Mousepad". The file path in the title bar is "~/lkl-source/tools/lkl/include/lkl/lkl_fat_log.h - Mousepad". The code content is as follows:

```
1 #ifndef _LKL_LKL_FAT_LOG_H
2 #define _LKL_LKL_FAT_LOG_H
3 extern void lkl_fat_log_init(void);
4 extern void lkl_fat_log_write(const char *msg);
5 extern void lkl_fat_log_close(void);
6 #endif /* _LKL_LKL_FAT_LOG_H */
```

Ενσωμάτωση στη Μεταγλώττιση

Το αρχείο tools/lkl/Makefile τροποποιήθηκε ώστε να περιλαμβάνει το fat_log.c κατά τη μεταγλώττιση, εντάσσοντας το παραγόμενο fat_log.o στη βιβλιοθήκη liblkl.a. Επιπλέον, ο κανόνας clean ενημερώθηκε ώστε να μην διαγράφει τον φάκελο include/lkl.

```
89 # rules to link libs
90 $(OUTPUT)%$(SOSUF): LDFLAGS += -shared
91 $(OUTPUT)%$(SOSUF): $(OUTPUT)%-in.o $(OUTPUT)liblkl.a
92     $(QUIET_LINK)$ (CC) $(LDFLAGS) $(LDFLAGS_*-y) -o $@ $^ $(LDLIBS) $(LDLIBS_*-y)
93
94 # liblkl is special
95 $(OUTPUT)liblkl$(SOSUF): $(OUTPUT)%-in.o $(OUTPUT)lib/lkl.o
96 $(OUTPUT)liblkl.a: $(OUTPUT)lib/liblkl-in.o $(OUTPUT)lib/lkl.o fat_log.o
97     $(QUIET_AR)$ (AR) -rc $@ $^
98
99 # Rule to explicitly build fat_log.o from fat_log.c in the current directory
100 fat_log.o: fat_log.c \
101     include/lkl/lkl_fat_log.h
102     @echo '  CC      $@'
103     $(Q)$ (CC) $(CFLAGS) -c -o $@ $<
104
105 # rule to link programs
106 $(OUTPUT)%$(EXESUF): $(OUTPUT)%-in.o $(OUTPUT)liblkl.a
107     $(QUIET_LINK)$ (CC) $(LDFLAGS) $(LDFLAGS_*-y) -o $@ $^ $(LDLIBS) $(LDLIBS_*-y)
108
109 # rule to build objects
110 $(OUTPUT)%-in.o: $(OUTPUT)lib/lkl.o FORCE
111     $(Q)$ (MAKE) -f $(srctree)/tools/build/Makefile.build dir=$(patsubst %/,%, $(dir $*)) obj=$(notdir $*)
112
113
```

Έλεγχος Ζωής του Log

Οι συναρτήσεις `lkl_fat_log_init()` και `lkl_fat_log_close()` κλήθηκαν αντίστοιχα στην αρχή και στο τέλος της `main()` του αρχείου `disk.c`, ώστε να διαχειρίζονται τη διάρκεια ζωής του αρχείου καταγραφής με τρόπο συνεπή και ασφαλές.

```
myy601@myy601lab2:~/lkl-source/tools/lkl/tests$ rm -f journal.log
myy601@myy601lab2:~/lkl-source/tools/lkl/tests$ ./disk.sh -t vfat
1..1 # disk vfat
* 1 prepfs vfat
ok 1 prepfs vfat
---
time_us: 668948
log: |
300+0 records in
300+0 records out
314572800 bytes (315 MB, 300 MiB) copied, 0.617842 s, 509 MB/s
mkfs.fat 4.2 (2021-01-31)
...
LKL_FAT_LOG: fopen'd journal.log successfully.
1..11 # disk vfat
* 1 disk_add
ok 1 disk_add
---
time_us: 35
log: |
disk fd/handle 4 disk_id 0
```

```
lkl_sys_halt() = 0
...
* 11 disk_remove
ok 11 disk_remove
---
time_us: 14
log: |
...
LKL_FAT_LOG: fclose'd journal.log successfully.
```

Διερεύνηση με χρήση της printk:

Πριν την υλοποίηση του τελικού μηχανισμού logging, πραγματοποιήθηκε μία αρχική φάση διερεύνησης, με στόχο την κατανόηση της λειτουργίας του συστήματος αρχείων VFAT εντός της LKL και τη δοκιμή της δυνατότητας τροποποίησης του πηγαίου κώδικα. Η προσέγγιση αυτή βασίστηκε στη χρήση της συνάρτησης printk, που επιτρέπει την εκτύπωση διαγνωστικών μηνυμάτων από τον πυρήνα.

Για τη δοκιμή επιλέχθηκε η συνάρτηση `__fat_readdir` του αρχείου `fs/fat/dir.c`, καθώς καλείται εμφανώς κατά την εκτέλεση της λειτουργίας ανάγνωσης καταλόγου (`readdir`), που ενεργοποιείται από το test script `./disk.sh -t vfat`. Προστέθηκε η εντολή:

```
printk(KERN_INFO "==== My fat_readdir was called! ctx->pos = %lld =====\n",
ctx->pos);
```

στην αρχή της `__fat_readdir`, ώστε να επιβεβαιωθεί η εκτέλεσή της και να καταγραφεί η θέση ανάγνωσης.

Ακολούθησε μεταγλώττιση της βιβλιοθήκης LKL μέσω `make -j8` στον κατάλογο `tools/lkl`, και στη συνέχεια εκτέλεση του script `./disk.sh -t vfat` στον κατάλογο `tests`. Η επιτυχία της παρέμβασης επιβεβαιώθηκε από τα διαγνωστικά μηνύματα στην έξοδο του script:

```
log: |
[ 0.038083] ===== My fat_readdir called! ctx->pos = 0 =====
. .. myfile.txt [ 0.038117] ===== My fat_readdir called! ctx->pos = 16384 =====
```

Η παραπάνω παρατήρηση έδειξε ότι η συνάρτηση `__fat_readdir` καλείται δύο φορές — μία για την εγγραφή `.` και μία για την `..` — και απέδειξε ότι:

- Η παρέμβαση στον πηγαίο κώδικα εφαρμόστηκε σωστά.
- Η μεταγλώττιση ενσωμάτωσε επιτυχώς τις αλλαγές.
- Η `printk` αποτελεί αξιόπιστο εργαλείο για επιβεβαίωση εκτέλεσης σε πραγματικές κλήσεις του συστήματος αρχείων.

Αυτό το στάδιο ήταν καθοριστικό για τη δημιουργία εμπιστοσύνης ως προς τη διαδικασία ανάπτυξης και την ταυτοποίηση σημείων κώδικα κατάλληλων για προσθήκη λογικής καταγραφής σε επόμενο στάδιο.

```
cpos = ctx->pos;
printk(KERN_INFO "==== My fat_readdir called! ctx->pos = %lld =====\n", ctx->pos);
char journal_msg[200];
int msg_len;
loff_t current_pos = ctx->pos;
```


2. Ενσωμάτωση Λειτουργίας Καταγραφής στο VFAT

2.1 Επιλογή Πρώτης Λειτουργίας για Καταγραφή

Ως πρώτη λειτουργία του συστήματος αρχείων VFAT για την οποία υλοποιήθηκε καταγραφή, επιλέχθηκε η `__fat_readdir` (στο αρχείο `fs/fat/dir.c`). Η επιλογή αυτή έγινε λόγω της απλής επαναληψιμότητας της συγκεκριμένης λειτουργίας: το test script `./disk.sh -t vfat` εκτελεί αναγνώσεις καταλόγων, όπως φαίνεται και από την έξοδό του, καθιστώντας εύκολη την παρατήρηση της επίδρασης της καταγραφής.

2.2 Αρχικές Προσεγγίσεις και Προβλήματα

Η υλοποίηση του μηχανισμού καταγραφής πέρασε από διάφορες αποτυχημένες προσεγγίσεις:

- **Απόπειρα 1:** Άμεση κλήση LKL syscalls (όπως `lkl_sys_openat`) από τον κώδικα του πυρήνα VFAT.
 - **Αποτυχία:** Ο πυρήνας δεν έχει ορατότητα στα headers ή τις συναρτήσεις των εργαλείων LKL (`tools/lkl`), οδηγώντας σε `compile errors`.
- **Απόπειρα 2:** Δημιουργία βοηθητικών συναρτήσεων (`lkl_fat_log_init`, `_write`, `_close`) στο `tools/lkl/fat_log.c` με χρήση LKL syscalls.
 - **Αποτυχία:** Παρόλο που το αρχείο μεταγλωττίστηκε σωστά, η κλήση της `lkl_sys_openat` προκάλεσε `segmentation fault` κατά την εκτέλεση. Το debugging έδειξε πως το σφάλμα προερχόταν από την αστάθεια της κλήσης εντός αυτού του `execution context`.

2.3 Επιτυχής Προσέγγιση με Callback και Standard I/O

Τελικά, διατηρήθηκε η λογική των βοηθητικών συναρτήσεων, αλλά αντικαταστάθηκε η χρήση LKL syscalls με τις συναρτήσεις της standard C βιβλιοθήκης (fopen, fprintf, fflush, fclose), οι οποίες εκτελούνται σωστά στο context του executable disk, που τρέχει στον χώρο χρήστη.

- Οι συναρτήσεις υλοποιήθηκαν στο tools/lkl/fat_log.c:
 - lkl_fat_log_init(): Άνοιγμα του αρχείου journal.log.
 - lkl_fat_log_write(const char *msg): Καταγραφή μηνύματος.
 - lkl_fat_log_close(): Κλείσιμο αρχείου.
- Header files:
 - tools/lkl/include/lkl/lkl_fat_log.h: Χρήση από fat_log.c, disk.c.
 - include/linux/lkl_fat_log.h: Χρήση εντός πυρήνα (dir.c).
- Η __fat_readdir τροποποιήθηκε ώστε να καλεί lkl_fat_log_write(), καταγράφοντας τη λειτουργία "FAT_READDIR" και την τιμή της παραμέτρου ctx->pos.
- Στην main() του disk.c προστέθηκαν κλήσεις στις lkl_fat_log_init() και lkl_fat_log_close() για την ορθή διαχείριση του κύκλου ζωής του log.
- Το tools/lkl/Makefile ενημερώθηκε για να περιλαμβάνει το fat_log.o στη βιβλιοθήκη liblkl.a.

```
#include <linux/lkl_fat_log.h>

...

msg_len = snprintf(journal_msg, sizeof(journal_msg),
                  "FAT_READDIR | pos=%lld\n", current_pos);

if (msg_len > 0 && msg_len < sizeof(journal_msg)) {
    // συνάρτηση για να γράψει το μήνυμα
    lkl_fat_log_write(journal_msg);
} else {
    printk(KERN_WARNING "LKL_FAT_LOG: snprintf error in dir.c\n");
}
/* Fake . and .. for the root directory. */

# liblkl is special
$(OUTPUT)liblkl$(SOSUF): $(OUTPUT)%-in.o $(OUTPUT)lib/lkl.o
$(OUTPUT)liblkl.a: $(OUTPUT)lib/liblkl-in.o $(OUTPUT)lib/lkl.o fat_log.o
$(QUIET_AR)$(AR) -rc $@ $^
```

2.4 Καταγραφή Παραμέτρου από `__fat_readdir`

Η καταγραφή επικεντρώθηκε στην παράμετρο `ctx->pos`, που εκφράζει τη θέση (offset) ανάγνωσης στον κατάλογο. Η επιλογή της έγινε επειδή η τιμή αυτή αλλάζει σε κάθε νέα εγγραφή (π.χ. . και ..) και αποτελεί ένδειξη σωστής λειτουργίας της αναγνωστικής διαδικασίας.

2.5 Επαλήθευση Λειτουργίας

Η τελική προσέγγιση δοκιμάστηκε με επιτυχία:

- Το test `./disk.sh -t vfat` ολοκληρώθηκε χωρίς segmentation faults.
- Εμφανίστηκαν διαγνωστικά μηνύματα:

«LKL_FAT_LOG: fopen'd journal.log successfully.»

«LKL_FAT_LOG: fclose'd journal.log successfully.»

Το αρχείο `journal.log` δημιουργήθηκε στον φάκελο `tests/` και περιείχε:

```
FAT_READDIR | pos=0
FAT_READDIR | pos=16384
```

γεγονός που επιβεβαιώνει ότι:

- Ο κώδικας του πυρήνα κάλεσε επιτυχώς εξωτερική συνάρτηση καταγραφής.
- Η εγγραφή στο `log` ήταν επιτυχής και αντανakλά τις προσβάσεις στον κατάλογο VFAT.

3. Καταγραφή των Λειτουργιών Mount και Readdir

3.1 Στόχος

Ο στόχος της παρούσας φάσης ήταν η επέκταση του μηχανισμού logging ώστε να καλύπτει επιπλέον βασικές λειτουργίες του συστήματος αρχείων VFAT, συγκεκριμένα:

- Τη λειτουργία προσάρτησης (mount) του συστήματος αρχείων.
- Τη λειτουργία ανάγνωσης καταλόγου (readdir), η οποία είχε ήδη υλοποιηθεί στην προηγούμενη φάση, και διατηρήθηκε.

3.2 Εντοπισμός Συναρτήσεων

Για την επίτευξη του παραπάνω στόχου εντοπίστηκαν οι κατάλληλες συναρτήσεις στον πηγαίο κώδικα του VFAT:

- **fat_fill_super** (αρχείο fs/fat/inode.c): Υπεύθυνη για την αρχικοποίηση του superblock κατά την προσάρτηση του συστήματος αρχείων.
- **fat_put_super** (ίδιο αρχείο): Υπεύθυνη για την αποδέσμευση πόρων κατά την αποπροσάρτηση.
- **__fat_readdir** (αρχείο fs/fat/dir.c): Υπεύθυνη για την ανάγνωση των εγγραφών ενός καταλόγου.

3.3 Υλοποίηση Καταγραφής

Για την ενσωμάτωση της καταγραφής στις παραπάνω συναρτήσεις ακολουθήθηκαν τα εξής βήματα:

- Προστέθηκε η οδηγία `#include <linux/lkl_fat_log.h>` στα αρχεία `inode.c` και `dir.c`.
- **Στην fat_fill_super:** Προστέθηκε κλήση στην `lkl_fat_log_write()` με το μήνυμα:

```
}
snprintf(log_msg, sizeof(log_msg), "FAT_MOUNT | dev=%s | flags=0x%lx\n",
         dev_name, sb ? sb->s_flags : 0);
lkl_fat_log_write(log_msg);
```

όπου `sb->s_id` και `sb->s_flags` παρείχαν τα δεδομένα της συσκευής και των παραμέτρων προσάρτησης.

- **Στην fat_put_super:** Προστέθηκε ανάλογη καταγραφή με μήνυμα:

```
struct msdos_sb_info *sbi = MSDOS_SB(sb);
snprintf(log_msg, sizeof(log_msg), "FAT_UNMOUNT | dev=%s\n", dev_name);
lkl_fat_log_write(log_msg);
```

Στην `__fat_readdir`: Διατηρήθηκε η προηγούμενη καταγραφή της θέσης ανάγνωσης `ctx->pos`.

3.4 Δοκιμή και Εκτέλεση

Η νέα έκδοση μεταγλωττίστηκε επιτυχώς με την εντολή:

```
make -j8
```

Ακολούθως, το αρχείο `journal.log` διαγράφηκε ώστε να ξεκινήσει νέα καταγραφή. Εκτελέστηκε το test script:

```
./disk.sh -t vfat
```

4.5 Αποτελέσματα

Μετά την εκτέλεση, το αρχείο `journal.log` περιείχε την εξής γραμμή :

```
FAT_MOUNT | dev=vda | flags=0x10000000
FAT_ADD_ENTRIES | dir inode=1 | nr slots=2
```

Από την καταγραφή διαπιστώθηκαν τα εξής:

- Η λειτουργία **προσάρτησης (FAT_MOUNT)** καταγράφηκε επιτυχώς, με σωστή απεικόνιση της συσκευής και των flags.
- Η λειτουργία **ανάγνωσης καταλόγου (FAT_READDIR)** καταγράφηκε ξανά επιτυχώς, επιβεβαιώνοντας τη σταθερότητα του μηχανισμού.
- Η λειτουργία **αποπροσάρτησης (FAT_UNMOUNT)** **δεν καταγράφηκε**, γεγονός που ερμηνεύεται ως αναμενόμενο, καθώς η `fat_put_super()` πιθανώς καλείται μόνο κατά την πλήρη αποδέσμευση του συστήματος (termination του LKL) και όχι στο πλαίσιο απλού `unmount` που γίνεται από το test script.

3.6 Συμπεράσματα

Η καταγραφή των λειτουργιών `fat_fill_super` και `__fat_readdir` επιβεβαιώθηκε πλήρως, αποδεικνύοντας τη σωστή ένταξη του logging μηχανισμού σε βασικά σημεία του συστήματος αρχείων VFAT. Η χρήση του μηχανισμού καταγραφής μέσω `foren/fprintf` παρέμεινε σταθερή και αξιόπιστη. Η απουσία της καταγραφής `FAT_UNMOUNT` δεν επηρεάζει ουσιωδώς την αξιοπιστία του μηχανισμού, αλλά υποδεικνύει την ανάγκη για πιο προσεκτική μελέτη των χρόνων και συνθηκών κλήσης της `fat_put_super`.

4. Καταγραφή Λειτουργίας Δημιουργίας Αρχείου

4.1 Στόχος

Η επόμενη επέκταση του μηχανισμού logging στόχευε στην καταγραφή της λειτουργίας δημιουργίας νέου αρχείου στο σύστημα αρχείων VFAT. Αυτή η ενέργεια είναι κρίσιμη, καθώς εμπλέκει μεταβολή μεταδεδομένων, δημιουργία inodes και τροποποίηση καταλόγων.

4.2 Πρόκληση και Προσέγγιση

Οι αρχικές απόπειρες για πρόκληση της δημιουργίας αρχείου μέσω των εργαλείων `crtofs` και `lkl_sys_mkdir` δεν είχαν το επιθυμητό αποτέλεσμα. Για τον λόγο αυτό, τροποποιήθηκε το αρχείο `disk.c` ώστε να ενσωματώνει ένα νέο βήμα στο σενάριο δοκιμών, με άμεση χρήση της κλήσης:

```
4 // Νέα συνάρτηση για το test δημιουργίας αρχείου
5 static int lkl_test_create_file(void)
6 {
7     long fd;
8     long ret = TEST_FAILURE;
9     const char *new_file_name = "myfile.txt";
10    int flags = LKL_O_CREAT | LKL_O_WRONLY | LKL_O_TRUNC;
11    mode_t mode = 0644;
12
13    fd = lkl_sys_openat(LKL_AT_FDCWD, new_file_name, flags, mode);
14
15    lkl_test_logf("lkl_sys_openat(%s, flags=0x%x, mode=0%o) = %ld",
16                new_file_name, flags, mode, fd);
17
18    if (fd >= 0) {
19        char write_buf[4096]; // Ένα buffer ~4KB
20        long bytes_to_write = sizeof(write_buf);
21        long bytes_written;
22
23        memset(write_buf, 'B', bytes_to_write); // Γέμισέ το με 'B'
24        lkl_test_logf("Attempting to write %ld bytes...", bytes_to_write);
25        bytes_written = lkl_sys_write(fd, write_buf, bytes_to_write);
26        lkl_test_logf("lkl_sys_write(%ld, ...) = %ld", fd, bytes_written);
27
28        if (bytes_written != bytes_to_write) {
29            lkl_test_logf("Write operation may have failed or written less bytes.");
30        }
31        long close_ret = lkl_sys_close(fd);
32        lkl_test_logf("lkl_sys_close(%ld) = %ld", fd, close_ret);
33        if (close_ret == 0) {
34            ret = TEST_SUCCESS; // Επιτυχία μόνο αν και το close πέτυχε
35        }
36    }
37
38    return ret;
39 }
```

Η κλήση αυτή προστέθηκε ως νέο βήμα στον πίνακα `tests[]` του `disk.c`, εξασφαλίζοντας ότι θα εκτελεστεί στο σωστό `execution context` (δηλαδή εντός του `main test framework` που φορτώνει τη LKL).

4.3 Εντοπισμός Σχετικής Συνάρτησης VFAT

Κατά την ανάλυση του κώδικα διαπιστώθηκε ότι δεν υπάρχει άμεση συνάρτηση `.create` ορισμένη στο `inode_operations` του VFAT. Αντίθετα, η δημιουργία αρχείου φαίνεται να περνά μέσω της:

- `fat_build_inode` (αρχείο `fs/fat/inode.c`): Χρησιμοποιείται για να "χτίσει" τη δομή `inode` στη μνήμη κατά την εισαγωγή νέου αρχείου στο σύστημα αρχείων.

Υπάρχει επίσης πιθανότητα η `fat_add_entries` (στο `dir.c`) να συμμετέχει στη δημιουργία του αρχείου, όμως δεν παρατηρήθηκε σχετική εγγραφή κατά την εκτέλεση — πιθανώς λόγω αποτυχίας `logging` ή επειδή δεν ενεργοποιήθηκε σε αυτό το `test path`.

```
char log_msg[300];

struct inode *inode;
int err;

fat_lock_build_inode(MSDOS_SB(sb));
inode = fat_iget(sb, i_pos);
if (inode)
    goto out;
inode = new_inode(sb);
if (!inode) {
    inode = ERR_PTR(-ENOMEM);
    goto out;
}
snprintf(log_msg, sizeof(log_msg), "FAT_BUILD_INODE | i_pos=%lld | de_name[0]=%c\n",
         i_pos,
         de ? de->name[0] : '?'); // Χρησιμοποιούμε τον πρώτο χαρακτήρα του de->name
lkl_fat_log_write(log_msg);
inode->i_ino = iunique(sb, MSDOS_ROOT_INO);
```

4.4 Υλοποίηση Logging

Για την καταγραφή της λειτουργίας δημιουργίας αρχείου, προστέθηκε κώδικας στην αρχή της `fat_build_inode`, ως εξής:

Αυτό το μήνυμα καταγράφει:

- Τη θέση της εγγραφής στον δίσκο (`i_pos`).
- Τον πρώτο χαρακτήρα του ονόματος αρχείου όπως εμφανίζεται στο `directory entry` (`de->name[0]`).

Η πληροφορία αυτή βοηθά στο να επιβεβαιωθεί ότι η λειτουργία κατασκευής του `inode` όντως κλήθηκε κατά τη δημιουργία αρχείου.

4.5 Επαλήθευση

Μετά την προσθήκη της κλήσης καταγραφής:

- Το `disk.c` μεταγλωττίστηκε εκ νέου με `make -j8`.

- Εκτελέστηκε το script `./disk.sh -t vfat`.

Η έξοδος επιβεβαίωσε την επιτυχή δημιουργία του αρχείου `myfile.txt`, εμφανίζοντας μήνυμα επιτυχίας στο αντίστοιχο test βήμα ("OK").

Στο αρχείο `journal.log`, καταγράφηκαν οι ακόλουθες εγγραφές:

```
FAT_MOUNT | dev=vda | flags=0x10000000
FAT_ADD_ENTRIES | dir_inode=1 | nr_slots=2
FAT_BUILD_INODE | i_pos=5377 | de_name[0]=M
FAT_FILL_INODE | inode=3 | de_attr=0x20 | de_size=
```

Αρχικά το `fat_add_entries` δεν υπήρχε, εξηγούμε παρακάτω.

5. Διεύρυνση Καταγραφής μέσω `crtofs` και Επιβεβαίωση Δημιουργίας Αρχείου

5.1 Στόχος

Η παρούσα ενότητα είχε ως στόχο την επιβεβαίωση της λειτουργικότητας και την ενεργοποίηση πρόσθετων συναρτήσεων του συστήματος αρχείων VFAT (όπως `fat_add_entries` και `fat_build_inode`) μέσω της χρήσης του εργαλείου `crtofs`, με στόχο την πληρέστερη κάλυψη της διαδικασίας δημιουργίας αρχείου.

```
{
    {
        char log_msg[300];
        snprintf(log_msg, sizeof(log_msg), "FAT_ADD_ENTRIES | dir_inode=%lu | nr_slots=%d\n",
                dir->i_ino, nr_slots);
        lkl_fat_log_write(log_msg);
    }
    struct super block *sb = dir->i_sb;
```

5.2 Περιορισμοί με `disk.c`

Αν και η αρχική καταγραφή βασίστηκε στο script `./disk.sh -t vfat`, διαπιστώθηκε ότι:

- Η εκτέλεσή του ενεργοποιεί επιτυχώς μόνο τις συναρτήσεις:
 - `fat_fill_super` (καταγραφή ως `FAT_MOUNT`)
 - `__fat_readdir` (καταγραφή ως `FAT_READDIR`)
- Άλλες σημαντικές συναρτήσεις του FAT (όπως `fat_add_entries` (την έχουμε δώσει παραπάνω), `fat_build_inode` (την έχουμε δώσει παραπάνω), `fat_write_begin`) δεν κλήθηκαν κατά την τυπική εκτέλεση του script.

Παράλληλα, τροποποιήσεις στο `disk.c` ώστε να χρησιμοποιηθεί `lkl_sys_mkdir` ή `lkl_sys_openat` με `O_CREAT` απέτυχαν, επιστρέφοντας σφάλμα -2 (ENOENT), πιθανόν λόγω προβλημάτων context ή περιορισμών στην εκτέλεση LKL syscalls.

5.3 Χρήση και Τροποποίηση του `cptofs`

Για να ξεπεραστούν οι παραπάνω περιορισμοί, αξιοποιήθηκε το εργαλείο `cptofs`, με τις εξής ενέργειες:

```
struct lkl_disk disk;
long ret, umount_ret;
int i;
char mpoint[32];
unsigned int disk_id;

lkl_fat_log_init();

cla.owner = (uid_t)-1;
cla.group = (gid_t)-1;

umount_ret = lkl_umount_dev(disk_id, cla.part, 0, 1000);
if (ret == 0)
    ret = umount_ret;

lkl_fat_log_close();
```

- Το αρχείο `cptofs.c` τροποποιήθηκε ώστε να περιλαμβάνει τις συναρτήσεις:
 - `lkl_fat_log_init()` στην αρχή της `main()`.
 - `lkl_fat_log_close()` στο τέλος, για σωστό κύκλο ζωής του αρχείου `log`.
- Η εφαρμογή μεταγλωττίστηκε ξανά με `make`.
- Εκτελέστηκε η εντολή:

```
./cptofs -i /tmp/disk -t vfat /home/myy601/lkl-source/tools/lkl/tests/myhostfile.txt /
```

- η οποία αντέγραψε το αρχείο `myhostfile.txt` στον `root` κατάλογο του VFAT image.

Η εκτέλεση του `cptofs` ήταν επιτυχής, χωρίς σφάλματα.

5.4 Αποτελέσματα στο journal.log

Το αρχείο journal.log περιείχε τις εξής εγγραφές:

FAT_MOUNT | dev=vda | flags=0x10000000

FAT_ADD_ENTRIES | dir_inode=1 | nr_slots=2

FAT_BUILD_INODE | i_pos=5377 | de_name[0]=M

FAT_READDIR | pos=0

FAT_READDIR | pos=16384

Η εικόνα θα φαίνεται πιο κάτω.

Η ανάλυση των καταγραφών δείχνει:

- **FAT_ADD_ENTRIES:** Επιβεβαιώνει την κλήση της fat_add_entries (στο dir.c), με:
 - dir_inode=1: δηλώνοντας τον root κατάλογο ως στόχο.
 - nr_slots=2: δείχνοντας ότι δημιουργήθηκαν δύο καταχωρήσεις (πιθανόν για short και long name).
- **FAT_BUILD_INODE:** Κλήθηκε καθώς δημιουργήθηκε νέα εγγραφή αρχείου και δημιουργήθηκε αντίστοιχο inode.
- **FAT_READDIR:** Εκτελέστηκε πιθανόν ως μέρος της αναζήτησης καταλόγου ή επιβεβαίωσης ύπαρξης της νέας εγγραφής.

5.5 Παρατηρήσεις

Παρότι οι παραπάνω συναρτήσεις ενεργοποιήθηκαν με επιτυχία, παρατηρήθηκε ότι:

- Οι συναρτήσεις fat_write_begin και fat_write_end **δεν κλήθηκαν**, παρόλο που είχαν προστεθεί σε αυτές κλήσεις logging.
- Αυτό πιθανόν οφείλεται στο ότι το crtofs αντιγράφει μικρά αρχεία με τρόπο που δεν απαιτεί χρήση των αντίστοιχων write hooks του VFS/FAT (π.χ. μέσω direct write ή μεταφοράς σε ένα στάδιο).

5.6 Συνολική Επισκόπηση Λειτουργιών που Καταγράφηκαν

Έως αυτό το σημείο, ο μηχανισμός logging έχει επιβεβαιωμένα καταγράψει τις εξής συναρτήσεις VFAT:

6. Διόρθωση Λειτουργίας Καταγραφής και Επαλήθευση Πολλαπλών Εκτελέσεων

6.1 Πρόβλημα Καταγραφής Μόνο Τελευταίας Εκτέλεσης

Κατά τη διάρκεια δοκιμών, παρατηρήθηκε ότι το αρχείο καταγραφής `journal.log` περιείχε μόνο τις εγγραφές από την τελευταία εκτέλεση κάθε δοκιμαστικού σεναρίου (`disk.sh` ή `crtofs`). Οι εγγραφές από προηγούμενες εκτελέσεις διαγράφονταν. Αυτό υποδείκνυε ότι το αρχείο καταγραφής δεν διατηρούσε ιστορικό, αλλά αντικαθίστατο κάθε φορά.

6.2 Διάγνωση και Λύση

Η αιτία εντοπίστηκε στον τρόπο ανοίγματος του αρχείου `log` στη συνάρτηση `lkl_fat_log_init()` του αρχείου `fat_log.c`. Το αρχείο άνοιγε με χρήση της λειτουργίας `"w"` (`write/truncate`), η οποία διαγράφει το περιεχόμενο του αρχείου σε κάθε άνοιγμα:

// Λανθασμένο

```
fopen("journal.log", "w");
```

Για να επιτευχθεί σωστή συσσώρευση εγγραφών, η συνάρτηση τροποποιήθηκε ώστε να χρησιμοποιεί την επιλογή `"a"` (`append`), διατηρώντας το υπάρχον περιεχόμενο και προσθέτοντας νέες εγγραφές στο τέλος:

// Διορθωμένο

```
fopen("journal.log", "a");
```

Παράλληλα, διατηρήθηκε η χρήση `fflush()` στην `lkl_fat_log_close()` για λόγους ασφάλειας, αν και κρίθηκε ότι πιθανόν δεν είναι πλέον απαραίτητη.

6.3 Επιβεβαίωση Ορθής Συμπεριφοράς

Ακολούθησε διαδοχική εκτέλεση δύο σεναρίων:

1. Το disk.sh που δημιουργεί το αρχείο myfile.txt.
2. Το crtofs που αντιγράφει το myhostfile_crtofs.txt στο ίδιο VFAT image.

Το τελικό journal.log περιείχε συνδυαστικά τις καταγραφές και των δύο εκτελέσεων:

Αποτελέσματα crtofs:

```
myy601@myy601lab2:~/lkl-source/tools/lkl$ cat journal.log
FAT_MOUNT | dev=vda | flags=0x10000000
FAT_ADD_ENTRIES | dir_inode=1 | nr_slots=3
FAT_BUILD_INODE | i_pos=5378 | de_name[0]=M
FAT_WRITE_BEGIN | inode=3 | pos=0 | len=21
FAT_WRITE_END | inode=3 | pos=0 | len=21 | copied=21
FAT_MOUNT | dev=vda | flags=0x10000000
FAT_ADD_ENTRIES | dir_inode=1 | nr_slots=3
FAT_BUILD_INODE | i_pos=5378 | de_name[0]=M
FAT_WRITE_BEGIN | inode=3 | pos=0 | len=21
FAT_WRITE_END | inode=3 | pos=0 | len=21 | copied=21
FAT_MOUNT | dev=vda | flags=0x10000000
FAT_ADD_ENTRIES | dir_inode=1 | nr_slots=3
FAT_BUILD_INODE | i_pos=5381 | de_name[0]=M
FAT_WRITE_BEGIN | inode=3 | pos=0 | len=16
FAT_WRITE_END | inode=3 | pos=0 | len=16 | copied=16
FAT_MOUNT | dev=vda | flags=0x10000000
FAT_ADD_ENTRIES | dir_inode=1 | nr_slots=3
FAT_BUILD_INODE | i_pos=5378 | de_name[0]=M
FAT_WRITE_BEGIN | inode=3 | pos=0 | len=16
FAT_WRITE_END | inode=3 | pos=0 | len=16 | copied=16
FAT_MOUNT | dev=vda | flags=0x10000000
FAT_ADD_ENTRIES | dir_inode=1 | nr_slots=3
FAT_BUILD_INODE | i_pos=5378 | de_name[0]=M
FAT_FILL_INODE | inode=3 | de_attr=0x20 | de_size=0 | de_start=0
FAT_WRITE_BEGIN | inode=3 | pos=0 | len=21
FAT_WRITE_END | inode=3 | pos=0 | len=21 | copied=21
FAT_MOUNT | dev=vda | flags=0x10000000
FAT_ADD_ENTRIES | dir_inode=1 | nr_slots=3
FAT_BUILD_INODE | i_pos=5378 | de_name[0]=M
FAT_FILL_INODE | inode=3 | de_attr=0x20 | de_size=0 | de_start=0
FAT_WRITE_BEGIN | inode=3 | pos=0 | len=21
FAT_WRITE_END | inode=3 | pos=0 | len=21 | copied=21
```

Αποτέλεσμα και των δύο μαζί:

```
FAT_MOUNT | dev=vda | flags=0x10000000
FAT_ADD_ENTRIES | dir_inode=1 | nr_slots=2
FAT_BUILD_INODE | i_pos=5377 | de_name[0]=M
FAT_FILL_INODE | inode=3 | de_attr=0x20 | de_size=0 | de_start=0
FAT_WRITE_BEGIN | inode=3 | pos=0 | len=4096
FAT_WRITE_END | inode=3 | pos=0 | len=4096 | copied=4096
FAT_WRITE_BEGIN | inode=3 | pos=4096 | len=4096
FAT_WRITE_END | inode=3 | pos=4096 | len=4096 | copied=4096
FAT_WRITE_BEGIN | inode=3 | pos=8192 | len=4096
FAT_ENT_READ | entry=3
FAT_ENT_READ | entry=3
FAT_ENT_WRITE | entry=3 | value=0x4
FAT_ENT_READ | entry=3
FAT_WRITE_END | inode=3 | pos=8192 | len=4096 | copied=4096
FAT_WRITE_BEGIN | inode=3 | pos=12288 | len=4096
FAT_WRITE_END | inode=3 | pos=12288 | len=4096 | copied=4096
FAT_WRITE_BEGIN | inode=3 | pos=16384 | len=4096
FAT_ENT_READ | entry=4
FAT_ENT_READ | entry=4
FAT_ENT_WRITE | entry=4 | value=0x5
FAT_ENT_READ | entry=4
FAT_WRITE_END | inode=3 | pos=16384 | len=4096 | copied=4096
FAT_WRITE_BEGIN | inode=3 | pos=20480 | len=4096
FAT_WRITE_END | inode=3 | pos=20480 | len=4096 | copied=4096
FAT_WRITE_BEGIN | inode=3 | pos=24576 | len=4096
FAT_ENT_READ | entry=5
FAT_ENT_READ | entry=5
FAT_ENT_WRITE | entry=5 | value=0x6
FAT_ENT_READ | entry=5
FAT_WRITE_END | inode=3 | pos=24576 | len=4096 | copied=4096
FAT_WRITE_BEGIN | inode=3 | pos=28672 | len=4096
FAT_WRITE_END | inode=3 | pos=28672 | len=4096 | copied=4096
FAT_WRITE_BEGIN | inode=3 | pos=32768 | len=4096
FAT_ENT_READ | entry=6
FAT_ENT_READ | entry=6
FAT_ENT_WRITE | entry=6 | value=0x7
FAT_ENT_READ | entry=6
FAT_WRITE_END | inode=3 | pos=32768 | len=4096 | copied=4096
FAT_WRITE_BEGIN | inode=3 | pos=36864 | len=3136
FAT_WRITE_END | inode=3 | pos=36864 | len=3136 | copied=3136
FAT_READDIR | pos=0
FAT_READDIR | pos=16384
```

6.4 Παρατήρηση

Αν και το πεδίο `i_pos=5377` ήταν το ίδιο και στις δύο δημιουργίες αρχείων, αυτό θεωρήθηκε λογικό ενδεχόμενο, καθώς το ίδιο σημείο στον δίσκο μπορεί να επαναχρησιμοποιηθεί αν διαγραφεί και αναδημιουργηθεί το VFAT image. Επίσης, η τιμή `de_name[0]=M` είναι συνεπής με τα ονόματα "Myfile.txt" και "Myhostfile_cptofs.txt".

6.5 Συμπέρασμα

Η χρήση της λειτουργίας `append ("a")` στη `foren()` ήταν κρίσιμη για τη σωστή λειτουργία του μηχανισμού καταγραφής σε περιβάλλον πολλαπλών εκτελέσεων. Με τη διόρθωση αυτή, επιβεβαιώθηκε πως ο μηχανισμός logging είναι πλήρως λειτουργικός, σταθερός και μπορεί να διαχειρίζεται συνεχόμενα σενάρια χρήσης με συνεπή καταγραφή των γεγονότων του συστήματος αρχείων VFAT.

7. Συμπεράσματα

Στο πλαίσιο της παρούσας εργασίας υλοποιήθηκε ένας μηχανισμός καταγραφής (logging) βασικών λειτουργιών του συστήματος αρχείων VFAT, εντός του περιβάλλοντος της Linux Kernel Library (LKL). Η υλοποίηση βασίστηκε σε κλήσεις της standard βιβλιοθήκης C (fopen, fprintf, fflush, fclose) και ενσωματώθηκε στον πηγαίο κώδικα του LKL μέσω βοηθητικών συναρτήσεων που συνδέονται με τον πυρήνα ως callbacks.

Η αρχική διερεύνηση με printk επέτρεψε την κατανόηση της ροής εκτέλεσης και την ταυτοποίηση των κρίσιμων συναρτήσεων του VFAT. Στη συνέχεια, το σύστημα logging εφαρμόστηκε επιτυχώς σε βασικές λειτουργίες όπως:

- fat_fill_super (mount)
- __fat_readdir (ανάγνωση καταλόγου)
- fat_add_entries (δημιουργία εγγραφής σε κατάλογο)
- fat_build_inode (κατασκευή inode για νέο αρχείο)

Κατά την πορεία της υλοποίησης, έγιναν εμφανείς οι τεχνικές δυσκολίες που αφορούν την άμεση χρήση των lkl_sys_* συναρτήσεων από τον πυρήνα και την ανάγκη κατάλληλου execution context για να επιτύχει η καταγραφή. Η χρήση του εργαλείου crtofs ήταν καταλυτική για την επιβεβαίωση της λειτουργίας συναρτήσεων που δεν μπορούσαν να προκληθούν εύκολα μέσω του disk.sh.

Η καταγραφή διαδοχικών εκτελέσεων επιτεύχθηκε με τη σωστή χρήση του append mode στο fopen, διασφαλίζοντας ότι το αρχείο journal.log διατηρεί το ιστορικό όλων των ενεργειών.

Συνολικά, το σύστημα logging που υλοποιήθηκε επιτρέπει την παρακολούθηση της συμπεριφοράς του VFAT με ακρίβεια, παρέχοντας ένα πολύτιμο εργαλείο για ανάλυση, έλεγχο και κατανόηση της λειτουργίας του υποσυστήματος αρχείων εντός της LKL. Η εργασία προσφέρει μια πρακτική εισαγωγή στην τροποποίηση πυρηνικού κώδικα, στη χρήση εργαλείων δοκιμών και στην αντιμετώπιση σφαλμάτων κατά την ανάπτυξη σε επίπεδο πυρήνα.