

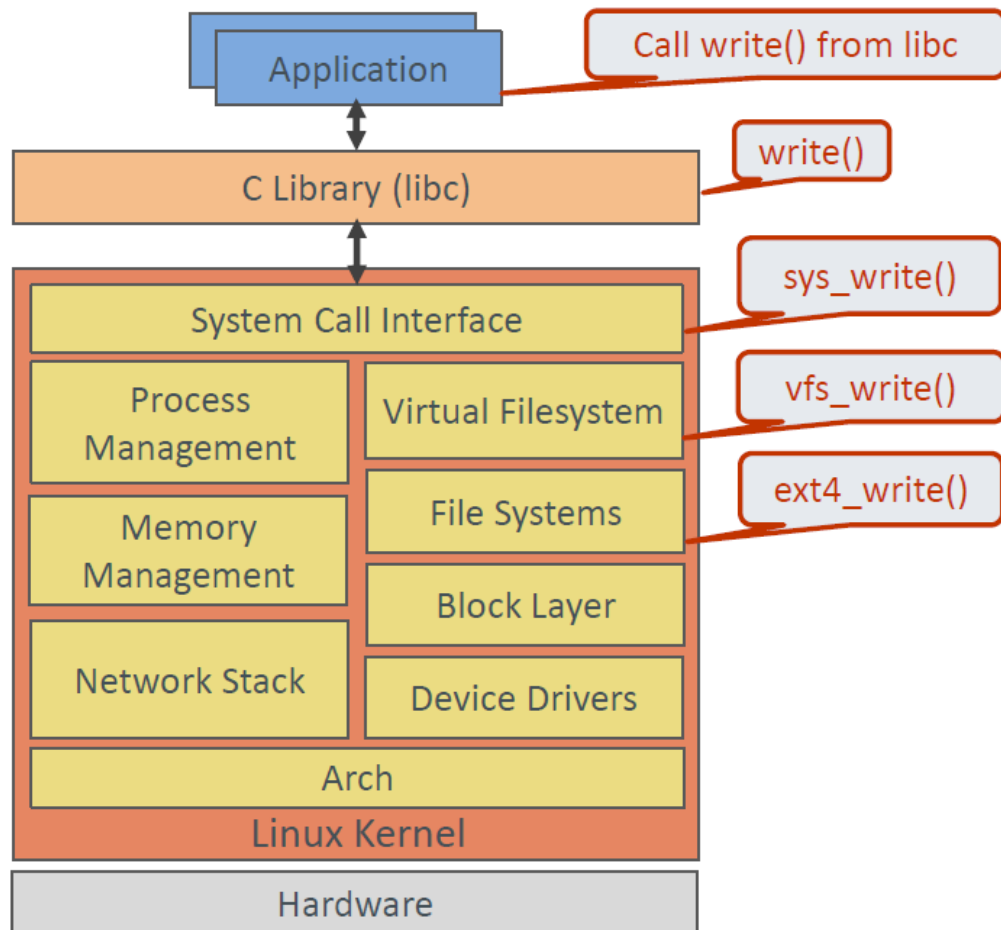
ΑΝΑΦΟΡΑ

Εργαστήριο 2: Υλοποίηση αρχείου καταγραφής στο
σύστημα αρχείων FAT του Linux



ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΙΚΙΔΗΣ (4387)
ΚΩΝΣΤΑΝΤΙΝΟΣ ΤΣΑΜΠΙΡΑΣ (4508)

Δομή του πυρήνα του Linux



Οι εφαρμογές που τρέχουν σε επίπεδο χρήστη αντιστοιχούν στο επίπεδο “Application” της παραπάνω εικόνας. Επικοινωνούν με τον πυρήνα μέσα από την χρήση βιβλιοθηκών (C Library) που τρέχουν στο επίπεδο χρήστη. Ο πυρήνας (ή το επίπεδο πυρήνα) διασυνδέει τις βιβλιοθήκες με το υλικό και είναι ανεξάρτητος από το επίπεδο χρήστη. Η επικοινωνία του προγραμματιστή με τον πυρήνα μπορεί να γίνει με κλήση των κατάλληλων συναρτήσεων. Το επίπεδο πυρήνα αποτελείται, όπως φαίνεται στην παραπάνω εικόνα, από τα εξής μέρη:

- **System Call Interface (διεπαφή αιτήσεων):** Αν μία εφαρμογή θέλει να επικοινωνήσει με τον πυρήνα, κάνει μία κλήση συστήματος.
- **Process Management (διαχείριση διεργασιών):** Υπεύθυνο για τις διεργασίες, τα νήματα και την χρονοδρομολόγηση.

- Memory Management (διαχείριση μνήμης): Υπεύθυνο για την διαχείριση της εικονικής και φυσικής μνήμης.
- Network Stack (στοίβα δικτύου): Υλοποιεί τα πρωτόκολλα δικτύωσης.
- Virtual File System (εικονικό σύστημα αρχείων): Διεπαφή για διάφορα συστήματα αρχείων.
- File System (σύστημα αρχείων): Υλοποιεί και διαχειρίζεται το σύστημα αρχείων.
- Block Layer: Υλοποιεί την διεπαφή που είναι υπεύθυνη για την διασύνδεση εφαρμογών και file system που χρησιμοποιούνται για διάφορες συσκευές αποθήκευσης.
- Device Drivers: Οδηγοί συσκευών
- Arch: Αρχιτεκτονική πυρήνα

Ο προγραμματισμός στο επίπεδο πυρήνα ενέχει κινδύνους όπως το μικρό μέγεθος στοίβας, την απουσία swap χώρου και απουσία προστασίας μνήμης όπου σε περίπτωση σφάλματος μνήμης θα έχουμε σφάλμα στον πυρήνα και η επανεκκίνηση του συστήματος είναι απαραίτητη για να επανέλθουμε σε κανονική λειτουργία.

Σύστημα Αρχείων

Αποτελείται από Αρχεία και Φακέλους, τα αρχεία περιέχουν τα δεδομένα που είναι χρήσιμα για τους χρήστες και οι φάκελοι είναι υπεύθυνοι για την οργάνωση των αρχείων και φακέλων. Κάθε αρχείο καθώς και το σύστημα αρχείων που χρησιμοποιούμε έχει μεταδεδομένα που αφορούν τα χαρακτηριστικά των αρχείων (όνομα, μέγεθος, δικαιώματα πρόσβασης, ιδιοκτήτης, χρόνος δημιουργίας/προσπέλασης, δείκτες στα block που δείχνουν στα δεδομένα) και χαρακτηριστικά του συστήματος αρχείων (τύπος, αριθμός block/inode και μέγεθος block).

Βασικές δομές ενός συστήματος αρχείων είναι:

- Block δεδομένων, που περιλαμβάνει τα περιεχόμενα του αρχείου.
- Block μεταδεδομένων/inode, δομή που αποθηκεύει τα μεταδεδομένα ενός αρχείου σε ένα ή περισσότερα inode.
- Bitmap, ένας πίνακας που μας δείχνει ποια block είναι ελεύθερα.

- Superblock, μεταδεδομένα που αφορούν το ίδιο το σύστημα αρχείων και τα χαρακτηριστικά του.

Βασικές δομές ενός εικονικού συστήματος αρχείων :

- Superblock: πληροφορίες για το σύστημα αρχείων
- Dentry: μια εγγραφή καταλόγου που αντιπροσωπεύει ένα φάκελο ή αρχείο, σε ένα μονοπάτι.
- Inode: πληροφορίες/ μεταδεδομένα για ένα συγκεκριμένο αρχείο ή φάκελο.
- File: διασύνδεση ενός αρχείου με μια διεργασία.

Ένα σύστημα αρχείων μπορεί να προσαρτηθεί στο σύστημά μας σε ένα φάκελο κάνοντας mount. Αν το σύστημά μας κρασάρει, μπορεί να το αφήσει σε ασυνεπή μορφή. Για την επίλυση του προβλήματος μπορούμε είτε να έχουμε στο superblock μια μεταβλητή η οποία μας υποδεικνύει αν το σύστημά μας είναι σε συνεπή ή ασυνεπή κατάσταση. Αν το σύστημά μας είναι ασυνεπές, με την χρήση κατάλληλου εργαλείου (fsck), γίνεται ο έλεγχος και η επιδιόρθωση του συστήματος, η οποία είναι μια χρονοβόρα διαδικασία. Διαφορετικά μπορούμε να κάνουμε καταγραφή των λειτουργιών σε ένα αρχείο (journal) πριν γίνει η ενημέρωση των δομών του συστήματος αρχείων, ώστε σε περίπτωση αποτυχίας να γίνει η επανάληψη των λειτουργιών που βρίσκονται μέσα στο journal.

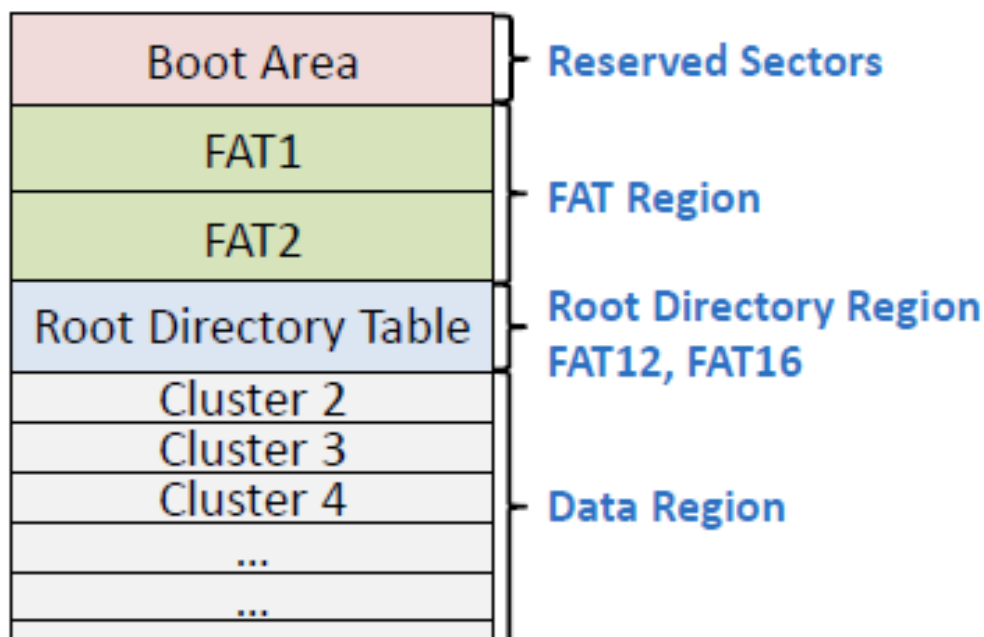
Σύστημα αρχείων FAT

Το σύστημα αρχείων FAT αποτελείται από τομείς (sectors). Πολλαπλά sectors μαζί δημιουργούν clusters, στα οποία αποθηκεύονται τα δεδομένα ενός αρχείου. Επίσης σε κάθε cluster περιλαμβάνεται ένας δείκτης ο οποίος δείχνει στο επόμενο cluster εκτός αν είναι το τελευταίο cluster που τότε ο δείκτης έχει την τιμή NULL.

Το File Allocation Table είναι ένας πίνακας ευρετήριο που δημιουργείται στατικά κατά τη διαμόρφωση. Κάθε στοιχείο του FAT είναι ένας δείκτης που δείχνει στο επόμενο cluster που αφορά ένα αρχείο. Το FAT διατηρεί δυο αντίγραφα του πίνακα ένα για την κύρια χρήση και ένα για back up σε περίπτωση που χρειαστεί.

Η δομή του FAT περιλαμβάνει:

- Reserved Sectors: δεσμευμένη περιοχή, όπου αποθηκεύονται πληροφορίες
 - Boot Sector: πληροφορίες για το μέσο αποθήκευσης
 - Filesystem Information Sector: πληροφορίες για το σύστημα αρχείων
- Fat Region: αποθηκεύονται οι File Allocation Table
- Root Directory Region (FAT12/FAT16 μόνο): δεσμευμένη περιοχή, όπου αποθηκεύονται πληροφορίες.
- Data Region: η περιοχή με τα clusters που περιέχουν τα δεδομένα αρχείων ή φακέλους.

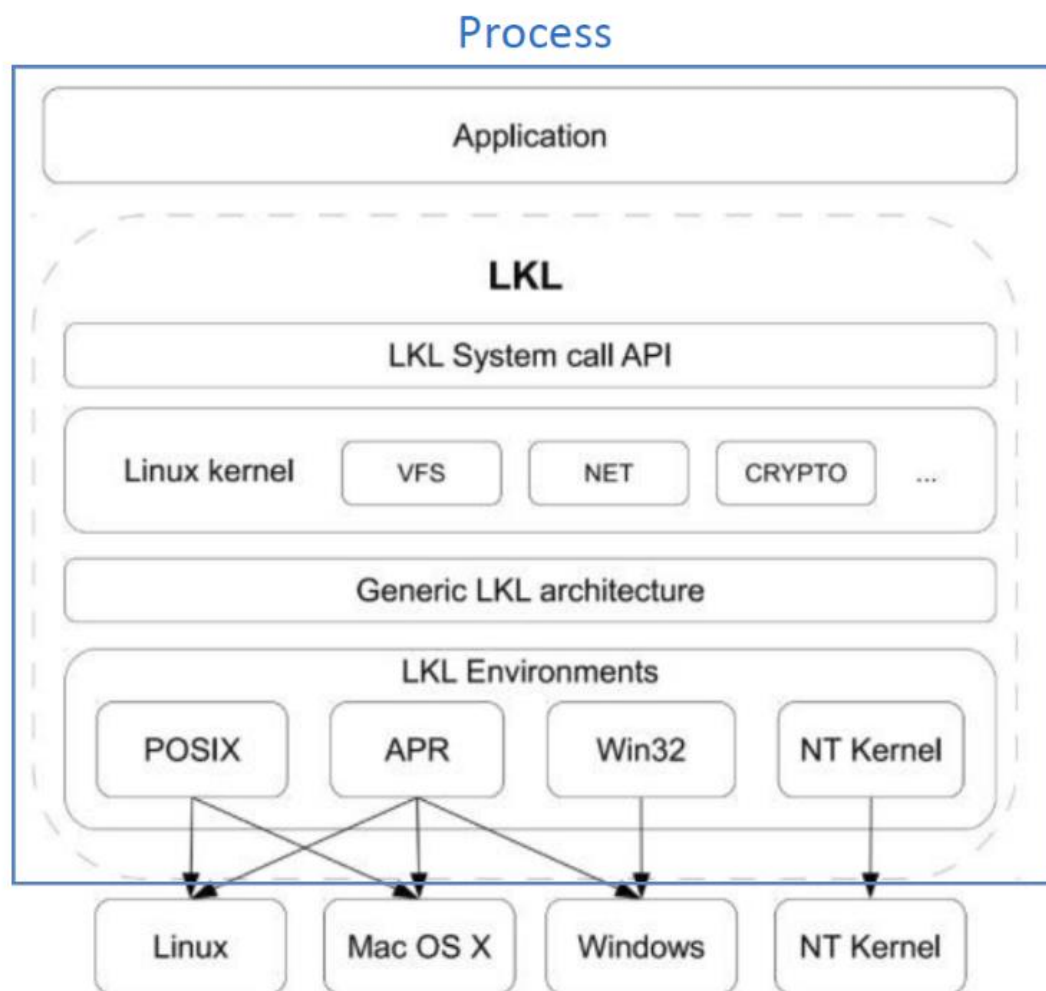


Linux Kernel Library

Το LKL είναι μία βιβλιοθήκη που στοχεύει στην όσο το δυνατόν περισσότερη επαναχρησιμοποίηση κώδικα του πυρήνα του Linux με ελάχιστη προσπάθεια και χαμηλό κόστος συντήρησης. Παραδείγματα χρήσης του LKL είναι η δημιουργία εφαρμογών σε επίπεδο χρήστη (είτε σε Linux είτε σε άλλα λειτουργικά συστήματα) για να μπορούν να κάνουν ανάγνωση ή/και εγγραφή σε συστήματα αρχείων Linux, χρήση της στοίβας δικτύωσης του Linux, δημιουργία οδηγών πυρήνα για άλλα λειτουργικά συστήματα για να μπορούν να διαβάζουν συστήματα αρχείων Linux κλπ. Με το LKL, ο κώδικας πυρήνα μπορεί να γίνει compile

σε ένα object αρχείο και να συνδεθεί άμεσα με εφαρμογές. Το API που προσφέρει είναι βασισμένο στην διεπαφή του Linux για κλήσεις συστήματος. Ουσιαστικά επιτρέπει σε εφαρμογές να χρησιμοποιούν κώδικα πυρήνα στις εφαρμογές τους σαν μία βιβλιοθήκη.

Αρχιτεκτονική του LKL



LKL Environments: Έχει κώδικα που σχετίζεται με μία πλατφόρμα λογισμικού (πχ POSIX). Υλοποιείται στο `tools/lkl`

Generic LKL Architecture: Γενική αρχιτεκτονική επεξεργαστή που δεν σχετίζεται με την πλατφόρμα υποδοχής. Υλοποιεί νήματα, system calls κλπ. Υλοποιείται στο `arch/lkl`

Linux Kernel: Ο πυρήνας του Linux με πολλά από τα γνωστά υποσυστήματα του (VFS, FS, PM, MM, Net), ωστόσο δεν υποστηρίζονται όλες οι λειτουργίες πυρήνα (πχ προστασία επιπέδου χρήστη – πυρήνα)

LKL System Call API: Η διεπαφή για σύνδεση με τις εφαρμογές, η οποία επιτυγχάνεται με 3 τρόπους, είτε την προσθήκη του “lkl_” μπροστά από τις κλήσεις συστήματος, είτε με την προφόρτωση του LKL στην εφαρμογή, είτε με μία τροποποιημένη libc (που εσωτερικά χρησιμοποιεί σύμβολα του LKL, πχ, open() -> sys_open()). Υλοποιείται στα tools/lkl/lib και arch/lkl.

Η άσκηση

Ζητείται να υλοποιήσουμε ένα αρχείο καταγραφής τροποποιήσεων (journal) για το σύστημα αρχείων FAT στην LKL.

Κατανόηση του FAT και προσθήκη printk() σε κατάλληλες θέσεις και ερμηνεία των αποτελεσμάτων.

Με βάση την εκφώνηση, μπορούμε να αντλήσουμε δεδομένα για την υλοποίηση του FAT στο Linux.

Superblock: struct super_operations fat_ops στο fs/fat/inode.c

Λειτουργίες μνήμης: struct address_space fat_aops στο fs/fat/inode.c

Λειτουργίες εγγραφών FAT: struct fatent_operations fat12_ops, struct fatent_operations fat16_ops, struct fatent_operations fat32_ops στο fs/fat/fatent.c

Λειτουργίες αρχείου: struct file_operations fat_file_operations στο fs/fat/file.c

Λειτουργίες Inode: struct inode_operations fat_file_inode_operations στο fs/fat/file.c

Λειτουργίες καταλόγων: struct inode_operations msdos_dir_inode_operations στο fs/fat/namei_msdos.c (FAT) και fs/fat/namei_vfat.c (VFAT)

Για αρχή, ξεκινήσαμε να μελετάμε το αρχείο inode.c στο οποίο υπάρχουν στοιχεία που αφορούν το Superblock, συγκεκριμένα:

Στη συνάρτηση fat_read_bpb (bpb = bios param block) του inode.c, υπάρχουν στοιχεία που αφορούν το Superblock, όπως ο αριθμός των FATs, το μέγεθος ενός Sector, τον συνολικό αριθμό των Sector, το s_count, το s_blocksize, το s_max_links, το s_stack_depth και το s_maxbytes του Superblock. Αυτά τα στοιχεία τα εμφανίζουμε στο τερματικό με printk στις γραμμές 1471-1480 του αρχείου inode.c.

```
C inode.c 9+ x C boot.c 1 C fs.c
fs > fat > C inode.c > fat_read_bpb(super_block *, fat_boot_sector *, int, fat_bios_param_block *)
1470
1471     printk(KERN_INFO "-----");
1472     printk(KERN_INFO "Inside fat_read_bpb of inode.c");
1473     printk(KERN_INFO "Number of FATs: %d\n", bpb->fat_fats);
1474     printk(KERN_INFO "Fat Sector Size: %d\n", bpb->fat_fat_length);
1475     printk(KERN_INFO "Number of Sectors: %d\n", bpb->fat_total_sect);
1476     printk(KERN_INFO "Superblock s_count: %d\n", sb->s_count);
1477     printk(KERN_INFO "Superblock s_blocksize: %d\n", sb->s_blocksize);
1478     printk(KERN_INFO "Superblock s_max_links: %ld\n", sb->s_max_links);
1479     printk(KERN_INFO "Superblock s_stack_depth: %ld\n", sb->s_stack_depth);
1480     printk(KERN_INFO "-----");
```

Οι printk αυτές μετά από εκτέλεση της εντολής

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

```
0.023240] this architecture does not have kernel memory p
0.023615] -----
0.023616] Inside fat_read_bpb of inode.c
0.023620] Number of FATs: 2
0.023622] Fat Sector Size: 200
0.023623] Number of Sectors: 204800
0.023624] Superblock s_count: 1
0.023625] Superblock s_blocksize: 512
0.023626] Superblock s_max_links: 0
0.023627] Superblock s_stack_depth: 0
0.023627] -----
```

(εφόσον έχει προηγηθεί make clean και make στον ~/lkl/lkl-source/tools/lkl), θα εμφανίσει το παρακάτω αποτέλεσμα.

Από την εικόνα διαπιστώνουμε πως έχουμε 2 FAT, το μέγεθος του sector είναι 200, ο αριθμός των sector είναι 204800, το superblock reference counter είναι 1 (s_count), το μέγεθος του block είναι 512 Bytes, το

s_max_links και s_stack_depth είναι 0. Στη συνάρτηση αυτή, διαβάζουμε πληροφορίες που υπάρχουν τόσο στο superblock όσο και στο boot sector του FAT.

Στη συνάρτηση fat_fill_super του inode.c, διαβάζει το superblock ενός MS-DOS συστήματος αρχείων, αντίστοιχα και εδώ λαμβάνουμε πληροφορίες σχετικές με το superblock του FAT, όπως τον αριθμό των

```
0.024367] -----  
[ 0.024367] Inside fat_fill_super of inode.c  
[ 0.024369] Sectors per cluster: 4  
[ 0.024371] Cluster size: 2048  
[ 0.024371] Directories per block: 16  
[ 0.024372] Free_clusters: -1  
[ 0.024373] FAT starts at: 4  
[ 0.024434] FAT length: 200  
[ 0.024436] FAT s_maxbytes (maximum file size): 4294967295  
[ 0.024437] -----
```

Sectors ανά cluster που είναι 4, το μέγεθος του cluster που είναι 2048, το directory entries per block που είναι 16, το free clusters που είναι -1 (undefined ακόμα), το FAT start για το πού ξεκινάει το FAT, το FAT length που αφορά το μέγεθος του FAT (σε sectors) και το μέγιστο μέγεθος που μπορεί να έχει ένα αρχείο στο FAT που είναι 4294967295. Τις πληροφορίες αυτές τις παίρνουμε με printk() στις γραμμές 1875-1884.

```
C inode.c 9+ x C boot.c 1 C fs.c  
fs > fat > C inode.c > fat_fill_super(super_block *, void *, int, int, void(*) (super_block *))  
1874  
1875     printk(KERN_INFO "-----");  
1876     printk(KERN_INFO "Inside fat_fill_super of inode.c");  
1877     printk(KERN_INFO "Sectors per cluster: %d\n", sbi->sec_per_clus);  
1878     printk(KERN_INFO "Cluster size: %d\n", sbi->cluster_size);  
1879     printk(KERN_INFO "Directories per block: %d\n", sbi->dir_per_block);  
1880     printk(KERN_INFO "Free_clusters: %d\n", sbi->free_clusters);  
1881     printk(KERN_INFO "FAT starts at: %d\n", sbi->fat_start);  
1882     printk(KERN_INFO "FAT length: %ld\n", sbi->fat_length);  
1883     printk(KERN_INFO "FAT s_maxbytes (maximum file size): %lld\n", sb->s_maxbytes);  
1884     printk(KERN_INFO "-----");  
1885
```

Σε αυτή την συνάρτηση παρατηρούμε πως έχουμε την αρχικοποίηση αρκετών πεδίων του msdos_sb_info, που ίσως είναι και λογικό από την ονομασία της συνάρτησης αυτής, η οποία υποδηλώνει το «γέμισμα» του superblock του FAT.

Από τα παραπάνω, πήραμε μία ιδέα για το που βρίσκονται τιμές που είναι χρήσιμες για το FAT και ενδεχομένως να τις χρησιμοποιήσουμε όταν θελήσουμε να καταγράψουμε τις αλλαγές που γίνονται σε αυτό.

Καταγραφή αλλαγών που γίνονται στις δομές του FAT σε ένα journal αρχείο στο τοπικό σύστημα αρχείων ext4.

Η καταγραφή των αλλαγών που γίνονται στο FAT γίνεται μέσα από ένα αρχείο καταγραφής στο οποίο κρατάμε πληροφορίες σχετικά με την κατάσταση του FAT και τυχόν αλλαγών που γίνονται όταν προσθέτουμε/διαγράφουμε/τροποποιούμε αρχεία εντός αυτού. Σκοπός του είναι να μπορούμε να καταλάβουμε ότι κάτι πήγε στραβά και να μπορέσουμε να επαναφέρουμε το σύστημα σε μία λειτουργική κατάσταση χρησιμοποιώντας τα δεδομένα που έχει το journal. Στο FAT δεν έχει υλοποιηθεί αρχείο καταγραφής, όπως υπάρχει στα περισσότερα σύγχρονα συστήματα αρχείων (NTFS, ext4). Στόχος μας είναι να φτιάξουμε ένα αρχείο καταγραφής για το FAT, το οποίο για αρχή θα το δημιουργούμε στο ext4 και αργότερα θα προσπαθήσουμε να το υλοποιήσουμε και εντός του FAT.

Αρχικά, αυτό που δοκιμάσαμε αρχικά ήταν μέσα στο αρχείο boot.c να ανοίξουμε/δημιουργήσουμε ένα αρχείο στο επίπεδο χρήστη, το αρχείο filejournal στις γραμμές 181-187.

```

181 // -----
182 printf("=====\n");
183 printf("Create filejournal at /tmp/filejournal\n");
184 int filejournal = open("/tmp/filejournal", O_CREAT, S_IRWXU | S_IRWXG | S_IRWXO );
185 printf("int value of filejournal: %d\n", filejournal);
186 printf("=====\n");
187 // -----

```

```

myy601@myy601lab2:~/lkl/lkl-source$ make -C tools/lkl test
make: Entering directory '/home/myy601/lkl/lkl-source/tools/lkl'
make -C tests test
for fs in vfat; do ./boot.sh -t $fs || exit 1; done
102400+0 records in
102400+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.301104 s, 348 MB/s
mutex                passed [1]
semaphore            passed [1]
join                 passed [joined 140592825706240]
disk_add             passed [3 0]
getpid               passed [1]
syscall_latency      passed [avg/min/max lkl: 74/70/461 native: 83/60/14367]
umask                 passed [22 777]
=====
Create filejournal at /tmp/filejournal
int value of filejournal: 4
=====
creat                passed [0]
close                 passed [0]
failopen              passed [-2]
open                  passed [0]

```

Το οποίο δουλεύει, τρέχοντας την εντολή `make -C tools/lkl test`

Και πράγματι, μεταβαίνοντας στον φάκελο `/tmp/`, βλέπουμε να έχει δημιουργηθεί το αρχείο `filejournal`.

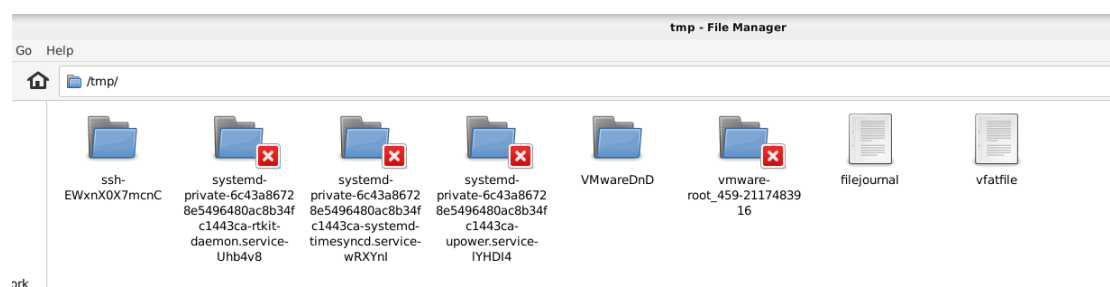
Ωστόσο, όταν πήγαμε να ανοίξουμε αυτό το αρχείο από το `inode.c` για να

```

C inode.c 9+ x  C boot.c 1  C fs.c
fs > fat > C inode.c > fat_read_bpb(super_block *, fat_boot_sector *, int, fat_bios_param_block *)
1481
1482 printk(KERN_INFO "Trying to open filejournal created from boot");
1483 int filejournal = sys_open("/tmp/filejournal", O_RDWR, 0);
1484 printk(KERN_INFO "filejournal int value: %d\n", filejournal);
1485 sys_write(filejournal, "Test writing to filejournal\n", strlen("Test writing to filejournal\n"));
1486

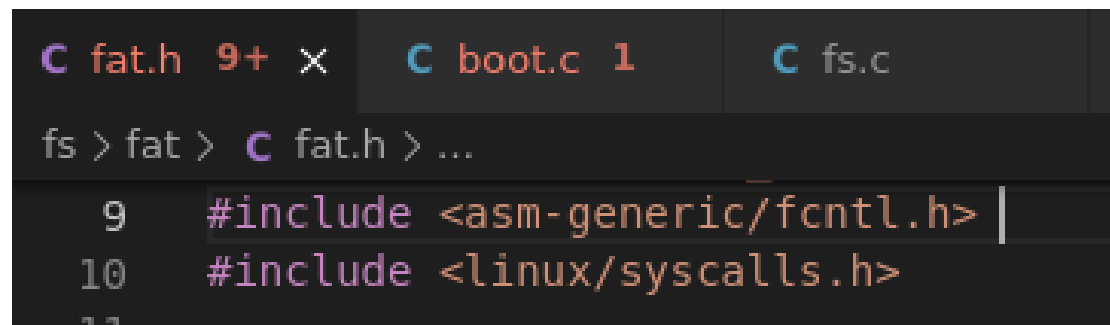
```

κάνουμε μέσα από το `inode` εγγραφή στοιχείων σχετικά με το FAT, το `int` που πήρε σαν αποτέλεσμα η `sys_open` ήταν `-2` (file not found).



```
[ 0.038330] -----  
[ 0.038334] Trying to open filejournal created from boot  
[ 0.038349] filejournal int value: -2
```

Για να καλέσουμε την `sys_open` και `sys_write` μέσα στην `inode`, ενσωματώσαμε τις `asm-generic/fcntl.h` και `linux/syscalls.h` στην `fat.h`



The screenshot shows a code editor with three tabs: `fat.h` (9 lines), `boot.c` (1 line), and `fs.c`. The `fat.h` tab is active, showing the following code:

```
fs > fat > C fat.h > ...  
9  #include <asm-generic/fcntl.h>  
10 #include <linux/syscalls.h>  
11
```

Στόχος μας με αυτό ήταν, εφόσον το αρχείο άνοιγε, να γράφαμε δεδομένα από το `superblock`, το `inode`, το `dentry` μέσα σε αυτό το αρχείο, από κάθε συνάρτηση που μεταβάλλει κάτι στο σύστημα αρχείων, αυτά που εκτυπώνουμε παραπάνω με τα `printk()`, να τα ενσωματώνουμε και στο `filejournal`, για αρχή σε ένα αρχείο που βρίσκεται στο `ext4` και μετά να το εισάγουμε μέσα στο `FAT`, με έναν `file descriptor` στο `superblock`, και όταν προκύπτουν αλλαγές στις δομές του συστήματος αρχείων, να γίνεται προσπέλαση του `filejournal` και αποθήκευση των επικείμενων αλλαγών. Το αρχείο καταγραφής θα πρέπει να δημιουργείται είτε κατά την διαμόρφωση του συστήματος αρχείων είτε στο πρώτο `mount` του σε ένα σύστημα (αν υπάρχει ήδη, δεν θα πρέπει να γίνει `overwrite` των περιεχομένων του).

Αρχεία που τροποποιήθηκαν

`fat.h`

`boot.c`

`inode.c`