



ΕΤΟΣ: 2020-2021


2Η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ -ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΜΑΡΙΑ ΗΛΕΚΤΡΑ ΓΑΡΓΑΛΑ - Α.Μ.4330

ΔΗΜΟΣΘΕΝΗΣ ΖΑΓΚΑΣ – Α.Μ.4359

Email: cs04330@uoi.gr

Email: cs04359@uoi.gr

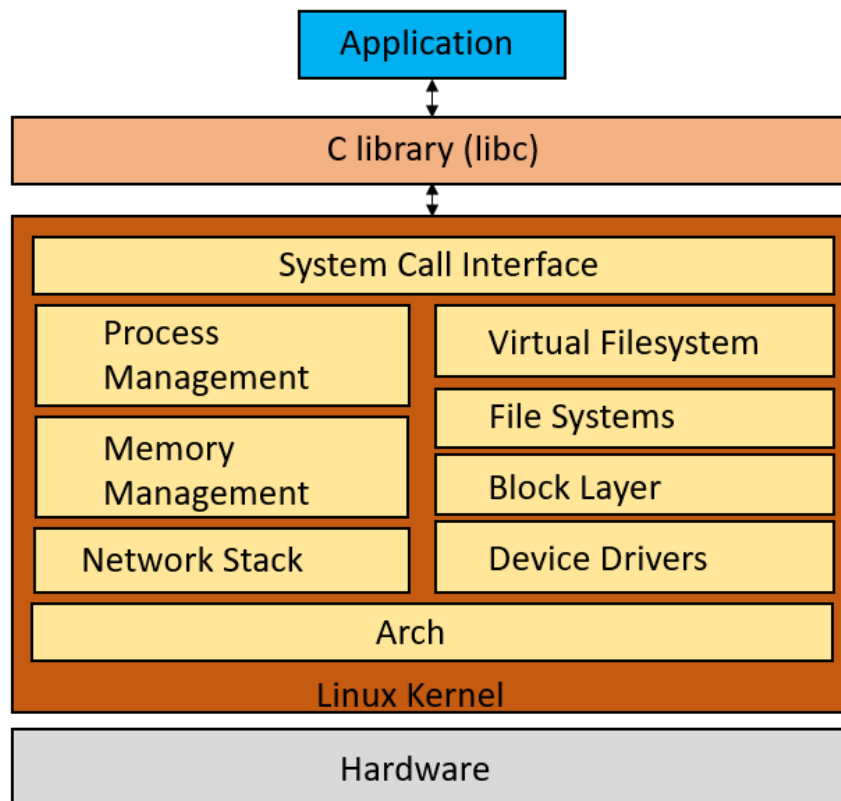


Υλοποίηση αρχείου καταγραφής στο σύστημα αρχείων FAT του Linux

Γενική περιγραφή δομή του Linux:

Περιγραφή Εικόνας 1:

Οι εφαρμογές «Application» τρέχουν σε επίπεδο χρήστη. Οι εφαρμογές χρησιμοποιούν βιβλιοθήκες, ώστε να ζητήσουν από το σύστημα λειτουργικότητα. Η libc είναι βιβλιοθήκη που τρέχει σε επίπεδο χρήστη.



Το επίπεδο πυρήνα, εκεί δηλαδή που υπάρχει ο πυρήνας του Linux, βρίσκεται κάτω από το επίπεδο χρήστη. Αποτελείται από τα εξής μέρη:

System Call Interface (διεπαφή αιτήσεων): Όταν η εφαρμογή θέλει να αποκτήσει προσβασιμότητα στον πυρήνα καλεί κλήση συστήματος

Process Management (διαχείριση διεργασιών): Παραδείγματος χάριν, διαχειρίζεται τις διεργασίες, υλοποιεί τα νήματα, είναι υπεύθυνο για την χρονοδρομολόγηση.

Memory Management (διαχείριση μνήμης): Διαχειρίζεται την φυσική και την εικονική μνήμη

Network stack: Υλοποίηση πρωτοκόλλων δικτύου

Virtual Filesystem: Διεπαφή συστημάτων αρχείων

File System: Υλοποίηση συστημάτων αρχείων

Block Layer: Επίπεδο μεταξύ δίσκου και συστήματος αρχείων

Device Drivers: Οδηγοί συσκευών

Arch: Υλοποίηση αρχιτεκτονικής

Ο πυρήνας δεν επιτρέπει στον προγραμματιστή να έχει πρόσβαση σε λειτουργίες βιβλιοθήκης, αφού ο πυρήνας είναι ανεξάρτητος από το επίπεδο χρήστη. Για να δουλέψουμε στον πυρήνα, θα πρέπει να χρησιμοποιούνται παρόμοιες συναρτήσεις.

Προβλήματα: Όταν προγραμματίζουμε σε επίπεδο πυρήνα, δεν υπάρχει προστασία μνήμης (πχ σε περίπτωση σφάλματος στην μνήμη ο πυρήνας κρασάρει και χρειάζεται reboot), το μέγεθος της στοίβας είναι πολύ μικρό και κατ' επέκταση, δε πρέπει να έχουμε πολλές μεταβλητές στην στοίβα, ούτε και αναδρομή. Επίσης δεν υποστηρίζεται το swapping, δηλαδή η λειτουργία κατά την οποία αν γεμίσει η μνήμη μεταφέρει τα αρχεία στον δίσκο.

Virtual Filesystem:

Αρχεία: περιλαμβάνουν τα δεδομένα του χρήστη.

Φάκελοι: περιλαμβάνουν αρχεία και φακέλους και βοηθούν στην οργάνωση των δεδομένων.

Το σύστημα αρχείων είναι μέθοδος αποθήκευσης δεδομένων(περιεχόμενα αρχείου) και μεταδεδομένων(πληροφορίες αρχείου όπως όνομα, μέγεθος, δικαιώματα, δείκτες σε block κλπ.) σε αρχεία και φακέλους. Τα δεδομένα αυτά αποθηκεύονται σε block δεδομένων.

Βασικές Δομές Συστήματος αρχείων(VF):

Block δεδομένων: αποθήκευση δεδομένων των αρχείων

Block μεταδεδομένων: αποθήκευση μεταδεδομένων όπου είναι οργανωμένα σε inodes

Bitmaps: πίνακας όπου δείχνει τα ελεύθερα block

Superblock: πληροφορίες για το σύστημα αρχείων. Βρίσκεται στην συνήθως στην αρχή του δίσκου

Inodes:

Κάθε inode σχετίζεται με ένα διαφορετικό αρχείο και περιλαμβάνει πληροφορίες μεταδεδομένων. Διατηρούνται σε πίνακα και κάθε inode έχει και έναν αριθμό όπου δείχνει στον πίνακα.

Στον κώδικα του Linux, το inode είναι ορισμένο ως ένα struct και περιέχει διάφορες μεταβλητές όπως:

το **struct hlist_node** (hash list, όπου περιέχει πληροφορίες για το αρχείο), **struct list head** (λίστα με inodes, superblocks), **uid_t** (περιέχει το id του κατόχου του αρχείου/φακέλου), **gid_t** (περιέχει το id του group), **loff_t** (μέγεθος αρχείου σε bytes), **struct timespec** (τελευταίος χρόνος τροποποίησης/αλλαγής), **umode_t i_mode** (αφορά τα δικαιώματα πρόσβασης).

Block δεδομένων:

Αποθηκεύονται τα δεδομένα των αρχείων. Επειδή τα block μπορεί να βρίσκονται διάσπαρτα στον δίσκο υπάρχουν δύο τρόποι ώστε να μπορούμε να βρούμε την θέση τους στον δίσκο:

Άμεσοι-έμμεσοι δείκτες: στο inode όπου υπάρχει ένας αριθμός σε δείκτη που δείχνει στα block

Πίνακας καταχώρησης αρχείων(File Allocation Table): το inode έχει έναν δείκτη στο πρώτο block του αρχείου και στην συνέχεια υπάρχει δείκτης στο επόμενο block. Το τελευταίο block δείχνει στο NULL ώστε να σηματοδοτείται το τέλος.

Superblocks: περιέχει πληροφορίες για το σύστημα αρχείων.

Dentry: καταχωρήσεις φακέλων.

File: η δομή χαρακτηρίζει κάθε φορά το αρχείο. Δημιουργείται με το open και διαλύεται με το close. Κάθε διεργασία έχει δικό της αντικείμενο file για την προσπέλαση του αρχείου.

Προσάρτηση Συστήματος Αρχείων:

Με το mount ενώνεται ένα σύστημα αρχείων σε έναν φάκελο. Με την διαδικασία αυτή, το σύστημα γίνεται ορατό.

Σφάλματα:

Σε περίπτωση σφάλματος όπου πχ ο υπολογιστής κρασάρει το σύστημα μένει σε ασυνεπή μορφή.

1^η λύση για να διατηρηθούν τα δεδομένα είναι να έχουμε μεταβλητή στο superblock, όπου όταν προσαρτίζεται το σύστημα τού προσδίδει την τιμή dirty. Εφόσον το σύστημα δεν προσαρτηθεί, θα έχει ακόμη την μεταβλητή dirty και θα λειτουργήσει σαν flag ώστε να ξεκινήσει μια διαδικασία ελέγχων επιδιόρθωσης. Η λύση αυτή δεν είναι βέλτιστη, αφού είναι χρονοβόρα.

Η 2^η λύση είναι η δημιουργία αρχείου log (καταγραφής) ώστε να αποθηκεύονται όλες οι λειτουργίες. Στην περίπτωση σφάλματος, οι λειτουργίες έχουν γραφεί στο αρχείο και μπορούν να ανακτηθούν από εκεί.

Σύστημα Αρχείων FAT

Το σύστημα αρχείων FAT οργανώνει τον δίσκο σε τομείς(sectors). Πολλοί sectors μαζί δημιουργούν ένα cluster. Τα δεδομένα ενός αρχείου γράφονται σε cluster. Εφόσον υπερβαίνουν το μέγεθος του cluster, μπορεί τα δεδομένα να γράφονται σε πολλαπλά clusters.

Το File Allocation Table είναι ένας πίνακας ευρετήριο όπου δημιουργείται κατά το format του συστήματος. Κατά την εκκίνηση δημιουργούνται δύο File Allocation Tables, όπου το 2^ο λειτουργεί ως αντίγραφο του 1^{ου} σε περίπτωση που χρειαστεί. Κάθε στοιχείο που βρίσκεται στον πίνακα είναι ένας δείκτης που δείχνει στο επόμενο cluster που αφορά ένα αρχείο. Στο τέλος του αρχείου, ο δείκτης δείχνει σε NULL, ώστε να ξέρουμε πως έφτασε στο τέλος.

Δομή FAT:

- *Reserved Sectors*: δεσμευμένη περιοχή, όπου αποθηκεύονται πληροφορίες
- *Fat Region*: αποθηκεύονται οι File Allocation Table
- *Root Directory Region*: περιλαμβάνονται αριθμοί αρχείων σε cluster και υπάρχει μόνο για FAT12 και FAT16
- *Data Region*: υπάρχουν εκεί τα clusters που περιέχουν δεδομένα αρχείων ή φακέλους

Οργάνωση FAT:

| |
|----------------------|
| Boot Area |
| FAT1 |
| FAT2 |
| Root Directory Table |
| Cluster1 |
| Cluster2 |
| ... |
| ... |

Στην αρχή του δίσκου υπάρχει η boot area όπου υπάρχουν πληροφορίες για το σύστημα αρχείων. Στην συνέχεια υπάρχει η περιοχή FAT Region1 και FAT Region2 που περιλαμβάνονται οι File Allocation Tables και μετά για τα συστήματα FAT12 και FAT16 υπάρχει το Root Directory Table. Τέλος, υπάρχουν τα clusters.

Linux Kernel Library (LKL)

Το LKL είναι μία βιβλιοθήκη όπου την διασυνδέουμε σε μορφή πυρήνα με την εφαρμογές και έτσι τρέχει σαν διεργασία και η εφαρμογή λειτουργεί χωρίς να επικοινωνεί με τον πυρήνα του Linux. Ουσιαστικά, είναι ένας τρόπος για να τροποποιήσουμε τον πυρήνα του Linux ώστε να λειτουργεί σαν βιβλιοθήκη σε επίπεδο χρήστη.

LKL Environments: μας βοηθάει να το τρέχουμε σε τρέχουμε σε διάφορα συστήματα. (Φάκελος tools/lkl)

Generic LKL architecture: εκεί υλοποιούνται νήματα, οι διαχειρίσεις των κλήσεων συστήματος κλπ. (Φάκελος arch/lkl)

Linux Kernel: πυρήνας Linux όπου περιλαμβάνει το VFS, NET, Process Manager κλπ.

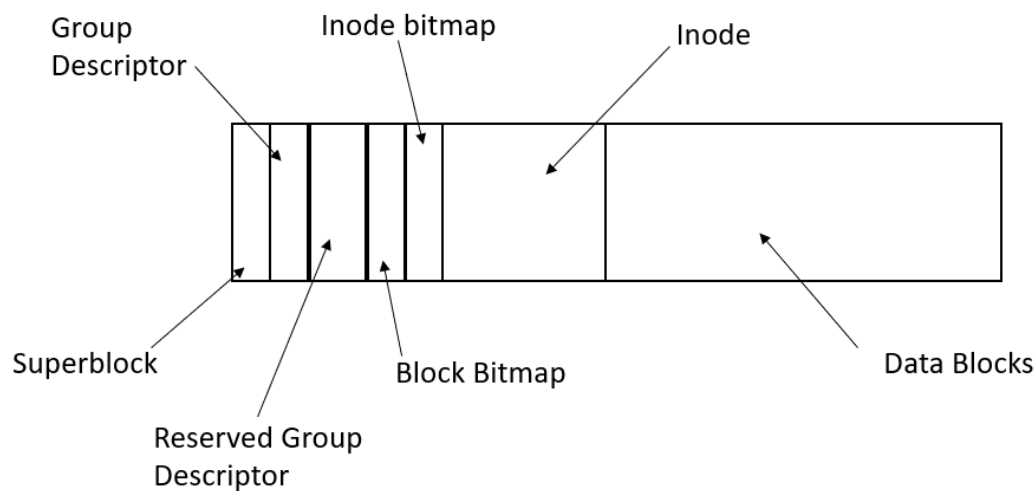
LKL System call API: διασυνδέει το LKL με την εφαρμογή. Μπορεί να διασυνδεθεί είτε με lkl_ μπροστά από τις κλήσεις συστήματος, είτε με προφόρτωση της εφαρμογής LKL στην εφαρμογή είτε με χρήση τροποποιημένης libc.

Περιγραφή 2^{ης} Εργαστηριακής Άσκησης

Για την Υλοποίηση της 2^{ης} Εργαστηριακής Άσκησης, πρέπει να χρησιμοποιήσουμε την βιβλιοθήκη Linux Kernel Library και σκοπός είναι η προσθήκη αρχείου καταγραφής στο σύστημα αρχείων FAT του Linux.

Το σύστημα αρχείων ext4 είναι το default σύστημα του Linux από το 2010 και αποτελεί βελτιωμένη έκδοση του ext3. Έχει περισσότερη χωρητικότητα και καλύτερη λειτουργία.

Το σύστημα ext4 έχει δομημένα τα δεδομένα όπως φαίνεται στην παρακάτω εικόνα:



Το superblock βρίσκεται μπροστά και είναι μία περιοχή του δίσκου και περιλαμβάνει πληροφορίες για το σύστημα (θέσεις των δεδομένων κλπ). Το group descriptor βρίσκεται μετά το superblock και περιέχει τα block δεδομένων στον δίσκο. Περιέχει τη θέση όπου βρίσκονται τα δεδομένα μπλοκ δεδομένων bitmap, inode Bitmap και inode Table σε ομάδα block. Το reserved group descriptor αφορά επεκτάσεις που ενδέχεται να γίνουν μελλοντικά στο σύστημα. Βρίσκεται μετά από το descriptor group και έχουν παρόμοια λειτουργία. Το block bitmap αναφέρεται στην χαρτογράφηση του διαθέσιμου χώρου στον δίσκο. Το bitmap αποτελείται από block όπου αναπαριστώνται με bit · αν το bit είναι 1 τότε το block είναι σε χρήση, ενώ αν είναι 0 τότε δεν είναι σε χρήση. Το inode table είναι μία δομή δεδομένων που αποθηκεύει δεδομένα για τα αρχεία. Το ext4 έχει συγκεκριμένο αριθμό inodes. Τα data blocks είναι σύνολο από block όπου περιέχουν τα δεδομένα των αρχείων.

```

myy601@myy601lab2:~/lkl/lkl-source$ ls -l
total 442016
drwx----- 34 myy601 myy601      4096 Sep 21  2017 arch
drwx----- 3 myy601 myy601      4096 Apr 29 11:55 block
drwx----- 2 myy601 myy601      4096 Apr 28 12:12 certs
-rw-r--r-- 1 myy601 myy601     3414 Sep 21  2017 circle.yml
-rw-r--r-- 1 myy601 myy601    18693 Sep 21  2017 COPYING
-rw-r--r-- 1 myy601 myy601   98253 Sep 21  2017 CREDITS
drwx----- 4 myy601 myy601    12288 Apr 29 11:50 crypto
drwx----- 120 myy601 myy601   12288 Apr 8 16:49 Documentation
drwx----- 130 myy601 myy601      4096 Apr 29 11:56 drivers
drwx----- 36 myy601 myy601      4096 Apr 28 12:13 firmware
drwx----- 74 myy601 myy601   12288 Apr 30 21:15 fs
drwx----- 30 myy601 myy601      4096 Apr 8 17:13 include
drwx----- 2 myy601 myy601      4096 Apr 30 21:15 init
drwx----- 2 myy601 myy601      4096 Apr 28 12:12 ipc
-rw-r--r-- 1 myy601 myy601     2888 Sep 21  2017 Kbuild
-rw-r--r-- 1 myy601 myy601       252 Sep 21  2017 Kconfig
drwx----- 17 myy601 myy601      4096 Apr 29 11:49 kernel
drwx----- 12 myy601 myy601    20480 Apr 29 11:56 lib
-rw-r--r-- 1 root root    149900832 Apr 30 21:15 lkl.o
-rw-r--r-- 1 myy601 myy601    399315 Sep 21  2017 MAINTAINERS
-rw-r--r-- 1 myy601 myy601    59352 Apr 8 17:17 Makefile
drwx----- 3 myy601 myy601      4096 Apr 29 11:49 mm
-rw-r--r-- 1 myy601 myy601         0 Apr 8 17:16 Module.symvers
drwx----- 67 myy601 myy601      4096 Apr 29 11:57 net
-rw-r--r-- 1 myy601 myy601       722 Sep 21  2017 README
lrwxrwxrwx 1 myy601 myy601        21 Apr 8 16:49 README.md -> Documentation/lkl.txt
drwx----- 27 myy601 myy601      4096 Sep 21  2017 samples
drwx----- 14 myy601 myy601      4096 Apr 28 12:12 scripts
drwx----- 10 myy601 myy601      4096 Apr 29 11:49 security
drwx----- 24 myy601 myy601      4096 Apr 28 12:13 sound
-rw-r--r-- 1 root root    721858 Apr 30 21:15 System.map
drwx----- 31 myy601 myy601      4096 Sep 21  2017 tools
drwx----- 2 myy601 myy601      4096 Apr 28 12:12 usr
drwx----- 4 myy601 myy601      4096 Apr 28 12:14 virt
-rw-r--r-- 1 root root   150766840 Apr 30 21:15 vmlinux
-rw-r--r-- 1 root root   150482976 Apr 30 21:15 vmlinux.o
myy601@myy601lab2:~/lkl/lkl-source$

```

Εκτελώντας στον φάκελο lkl-source την εντολή `ls -l`, εκτυπώνονται τα ονόματα των αρχείων, τα δικαιώματα, ο χρόνος κλπ.

Εκτελώντας στην συνέχεια την εντολή `lkl -li`, εκτός λαμβάνω περισσότερες πληροφορίες. Πριν από τα δικαιώματα, εκτυπώνεται στην οθόνη ένας αριθμός, όπου αντιστοιχεί στον αριθμό του inode του αρχείου και φανερώνει το σημείο του table όπου βρίσκεται το αρχείο.


```

myy601@myy601lab2:~/lkl/lkl-source$ ls -li
total 442016
153251 drwx----- 34 myy601 myy601      4096 Sep 21  2017 arch
156193 drwx----- 3 myy601 myy601      4096 Apr 29 11:55 block
156287 drwx----- 2 myy601 myy601      4096 Apr 28 12:12 certs
156293 -rw-r--r-- 1 myy601 myy601      3414 Sep 21  2017 circle.yml
150894 -rw-r--r-- 1 myy601 myy601     18693 Sep 21  2017 COPYING
150895 -rw-r--r-- 1 myy601 myy601    98253 Sep 21  2017 CREDITS
156294 drwx----- 4 myy601 myy601     12288 Apr 29 11:50 crypto
150896 drwx----- 120 myy601 myy601     12288 Apr 8 16:49 Documentation
156453 drwx----- 130 myy601 myy601      4096 Apr 29 11:56 drivers
162428 drwx----- 36 myy601 myy601      4096 Apr 28 12:13 firmware
302289 drwx----- 74 myy601 myy601     12288 Apr 30 21:15 fs
162888 drwx----- 30 myy601 myy601      4096 Apr 8 17:13 include
171065 drwx----- 2 myy601 myy601      4096 Apr 30 21:15 init
171066 drwx----- 2 myy601 myy601      4096 Apr 28 12:12 ipc
153245 -rw-r--r-- 1 myy601 myy601      2888 Sep 21  2017 Kbuild
153246 -rw-r--r-- 1 myy601 myy601       252 Sep 21  2017 Kconfig
303942 drwx----- 17 myy601 myy601      4096 Apr 29 11:49 kernel
171067 drwx----- 12 myy601 myy601     20480 Apr 29 11:56 lib
150888 -rw-r--r-- 1 root root    149900832 Apr 30 21:15 lkl.o
153247 -rw-r--r-- 1 myy601 myy601    399315 Sep 21  2017 MAINTAINERS
153248 -rw-r--r-- 1 myy601 myy601    59352 Apr 8 17:17 Makefile
171079 drwx----- 3 myy601 myy601      4096 Apr 29 11:49 mm
169138 -rw-r--r-- 1 myy601 myy601       0 Apr 8 17:16 Module.symvers
171081 drwx----- 67 myy601 myy601      4096 Apr 29 11:57 net
153249 -rw-r--r-- 1 myy601 myy601       722 Sep 21  2017 README
153250 lrwxrwxrwx 1 myy601 myy601       21 Apr 8 16:49 README.md -> Documentation/lkl.txt
171122 drwx----- 27 myy601 myy601      4096 Sep 21  2017 samples
171133 drwx----- 14 myy601 myy601      4096 Apr 28 12:12 scripts
426982 drwx----- 10 myy601 myy601      4096 Apr 29 11:49 security
171147 drwx----- 24 myy601 myy601      4096 Apr 28 12:13 sound
169655 -rw-r--r-- 1 root root     721858 Apr 30 21:15 System.map
305200 drwx----- 31 myy601 myy601      4096 Sep 21  2017 tools
171201 drwx----- 2 myy601 myy601      4096 Apr 28 12:12 usr
171202 drwx----- 4 myy601 myy601      4096 Apr 28 12:14 virt
169458 -rw-r--r-- 1 root root    150766840 Apr 30 21:15 vmlinux
168998 -rw-r--r-- 1 root root    150482976 Apr 30 21:15 vmlinux.o
myy601@myy601lab2:~/lkl/lkl-source$

```

Για να μάθουμε πληροφορίες για ένα αρχείο, εκτελούμε την εντολή stat «όνομα».
Τρέχοντας την εντολή για τον φάκελο πχ fs (stat fs), λαμβάνουμε το εξής:

```

myy601@myy601lab2:~/lkl/lkl-source$ stat fs
  File: fs
  Size: 12288          Blocks: 24          IO Block: 4096   directory
Device: 801h/2049d    Inode: 302289       Links: 74
Access: (0700/drwx-----)  Uid: ( 1000/  myy601)   Gid: ( 1000/  myy601)
Access: 2021-04-30 21:15:26.536556902 +0300
Modify: 2021-04-30 21:15:26.460554506 +0300
Change: 2021-04-30 21:15:26.460554506 +0300
 Birth: -
myy601@myy601lab2:~/lkl/lkl-source$

```

Λαμβάνουμε πληροφορίες όπως τον αριθμό του inode, το μέγεθος, τα δικαιώματα, τα block που καταλαμβάνει, σε ποια συσκευή βρίσκεται, πόσα links υπάρχουν στο αρχείο αυτό κλπ.

```

myy601@myy601lab2:~/lkl/lkl-source$ df -i /dev/sda1
Filesystem      Inodes   IUsed   IFree  IUse% Mounted on
/dev/sda1       488640  148869  339771   31% /
myy601@myy601lab2:~/lkl/lkl-source$ df -i /dev/sda2
Filesystem      Inodes   IUsed   IFree  IUse% Mounted on
udev            118278    386  117892    1% /dev
myy601@myy601lab2:~/lkl/lkl-source$ █

```

Στην παραπάνω εικόνα, εκτελώντας την εντολή `df -i /dev/sda1`, παρατηρούμε τα δύο διαμερίσματα του δίσκου. Οι πληροφορίες που μας δίνονται είναι οι εξής:

- Το σύστημα αρχείων στο οποίο αναφέρεται (Filesystem)
- Τον συνολικό αριθμό των inode σε αυτό το σύστημα αρχείων (Inodes)
- Τον αριθμό των inode που χρησιμοποιούνται (IUsed)
- Τον αριθμό των υπόλοιπων inode διαθέσιμων για χρήση (IFree)
- Το ποσοστό των χρησιμοποιημένων inode (IUse%)
- Το σημείο προσάρτησης για αυτό το σύστημα αρχείων (Mounted on)

Mount (προσάρτηση)

Για να μπορέσουμε να χρησιμοποιήσουμε ένα σύστημα αρχείων Linux, θα πρέπει να το προσαρτήσουμε (mount) τα μέρη του δίσκου. Για να γίνει αυτό θα πρέπει να χρησιμοποιήσουμε την εντολή `mount -t vfat -o loop /tmp/vfatfile /vfat` (μορφή: **mount -t** 'σύστημα αρχείων που θέλω να χρησιμοποιήσω' 'μέρος που θέλω να κάνω mount' 'το σημείο που θέλουμε να γίνει mount'). Για να κάνω umount το σύστημα (πχ τον φάκελο vfat) χρησιμοποιώ την εντολή `umount /vfat`.

Όταν κάνω αλλαγές στο σύστημα αρχείων, πολλές φορές χρειάζεται να κάνουμε umount και ξανά mount για να γίνουν ορατές οι αλλαγές.

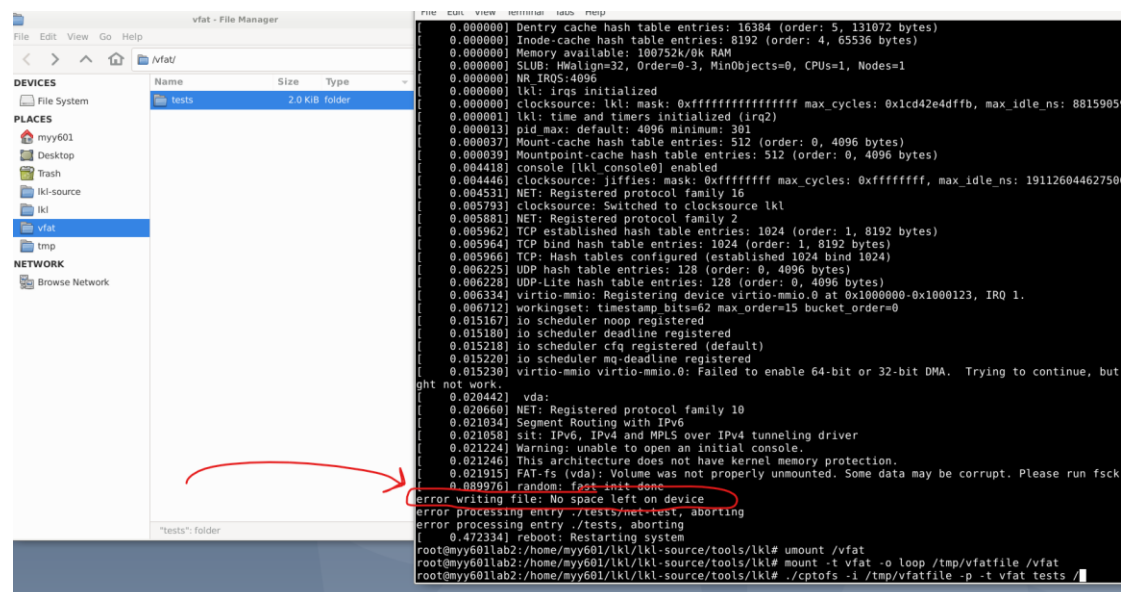
Το mountpoint είναι ένας κατάλογος όπου συνδέεται με ένα μέρος αποθήκευσης δεδομένων εκτός του δίσκου και των διαμερισμάτων του συστήματος.

Με την εντολή `./cptofs -i /tmp/vfatfile -p -t vfat lklfuse.c /` αντιγράφεται το αρχείο lklfuse.c στο vfatfile. Εκτελώντας την εντολή `cat /tmp/vfatfile` μπορούμε να δούμε ότι πράγματι τα περιεχόμενα του lklfuse.c βρίσκονται στο vfatfile. Μετά από αυτήν την

ενέργεια, θα περιμέναμε να έχει δημιουργηθεί ένα αντίγραφο στον φάκελο /vfat. Για να γίνει ορατή η αλλαγή αυτή, όπως προαναφέρθηκε, θα πρέπει να εκτελέσουμε `umount /vfat` και στην συνέχεια `mount -t vfat -o loop /tmp/vfatfile /vfat` ώστε να γίνουν ορατές οι αλλαγές.

Πράγματι, μετά από αυτές τις ενέργειες, στο /vfat έχει δημιουργηθεί αντίγραφο του αρχείου lkifuse.c.

Με την χρήση της `./cptofs -i /tmp/vfatfile -p -t vfat tests /` αντιγράφεται όλος ο φάκελος tests στο vfatfile, παρόλα αυτά λόγω έλλειψης χώρου κάποια αρχεία δεν αντιγράφονται.



Σκοπός της άσκησης είναι η δημιουργία αρχείου καταγραφής. Θέλουμε να δημιουργήσουμε ένα αρχείο όπου θα αποθηκεύουμε εκεί τις αλλαγές που γίνονται στο σύστημα, έτσι ώστε σε περίπτωση σφάλματος (πχ να κρασάρει το σύστημα), να μπορούμε να ανακτήσουμε τις ενέργειες από το αρχείο αυτό.

Αρχικά, μπορούμε να ανοίξουμε το αρχείο με την εντολή `open(...)`, σε επίπεδο χρήστη και να αποθηκεύσουμε το αρχείο σε έναν φάκελο της εικονικής μας μηχανής VMware. Θα έχουμε μέσα σε ένα αρχείο ένα file descriptor και θα προσθέτουμε κάθε φορά τα δεδομένα και μεταδεδομένα. Για να τα προσθέσουμε όμως, θα πρέπει να βρούμε τα κατάλληλα σημεία στον κώδικα, ώστε να γράψουμε τις πληροφορίες αυτές στο αρχείο.

Εφόσον θέλουμε να δημιουργήσουμε το αρχείο σε επίπεδο πυρήνα, μπορούμε να προσθέσουμε στον struct του superblock τον file descriptor όπου θα δείχνει στο αρχείο, για να καταλάβουμε χώρο σε αυτήν την περιοχή για το αρχείο μας. Η

δημιουργία του αρχείου θα μπορούσε να γίνει και σε άλλα σημεία. Στην συνέχεια, εκεί όπου γίνεται mount το σύστημα (προσαρτίζεται) θα δημιουργήσουμε το αρχείο με την κλήση συστήματος sys_open.

Για το πρώτο ερώτημα: Χρησιμοποιήσαμε το printk καθώς και το gdb, για να ελέγξουμε τα δεδομένα που βρίσκονται στις δομές αποθήκευσης στον δίσκο του FAT.

Προσθέτουμε printk στο αρχείο [/lkl-source/fs/fat/inode.c](#) στην συνάρτηση [__fat_write_inode](#) Γραμμές 862-866, στην συνάρτηση [fat_read_root](#) Γραμμές 1687-1402, στην συνάρτηση [fat_read_bpb](#) Γραμμές 1.

- Για το [fat_boot_sector](#): Ο αριθμός των fat ισούται με 2 και το μήκος του Fat ισούται με 200, όπως φαίνεται στην Εικόνα:

```
0.012701] This architecture does not have kernel memory protection.
0.012990] -----
0.012996] Info for fat boot sector:
0.012997]   Num of fats: 2
0.012998]   Length of fat: 200
0.013009] -----
```

- Για το [msdos_sb_info](#): Το μέγεθος του cluster είναι 2048, το root_cluster είναι 0, τα sectors ανά cluster είναι 4, η αρχή του root dentry είναι στο 512 και το μέγιστο cluster είναι 51093.

```
0.013023] -----
0.013024] Info from Msdos_sb_info:
0.013025]   Cluster_size is: 2048
0.013026]   Root_cluster is: 0
0.013027]   Sectors_per_clusters is: 4
0.013029]   Root_dentry_start is: 512
0.013030]   Max_cluster is: 51093
0.013031] -----
```

- Για το [msdos_dir_entry](#): Ο χρόνος δημιουργίας είναι 0, η ημερομηνία δημιουργίας είναι 33 και η ημερομηνία τελευταίας πρόσβασης είναι 33.

```
0.013031] -----
0.013535] Information for msdos_dir_entry
0.013540]   Ctime 0
0.013542]   Created date 33
0.013543]   Access date 33
0.013544] -----
0.013716] reboot: Restarting system
t@myy601lab2:/home/myy601/lkl/lkl-source/tools/lkl#
```

- Για το [inode](#): Η τιμή του s_count ισούται με 1, ο μέγιστος αριθμός των links είναι ίσος με 0, το options είναι NULL, το βάθος της στοίβας είναι 0, το μέγεθος του block σε bits είναι 9, το μέγεθος του block είναι 512 και ο μέγιστος αριθμός byte είναι 4294967295.

```

0.013334] -----
0.013334] Info for Inode (struct Superblock):
0.013336] Value of s_count from superblock (into inode) is: 1
0.013338] Max links is: 0
0.013339] Options from superblock is: (null)
0.013340] Depth of s_stack is: 0
0.013341] Blocksize bits is: 9
0.013343] Blocksize is: 512
0.013344] Max byte is: 4294967295
0.013345] -----

```

Τα αποτελέσματα των `printk`, εκτυπώνονται κατά την κλήση της `./cptofs -i /tmp/vfatfile -p -t vfat lklfuse.c /`.

Για το δεύτερο ερώτημα: Δημιουργία journal αρχείο σε επίπεδο χρήστη.

Προσπαθήσαμε να δημιουργήσουμε το αρχείο στο `/tools/lkl/tests/boot.c`, με το σκεπτικό ότι κάθε φορά θα θέλουμε να χρησιμοποιήσουμε τις εφαρμογές (`cptofs`, `cpfromfs`) θα πρέπει να κάνουμε boot το σύστημα, άρα πάντα θα δημιουργείται το αρχείο journal στην αρχή.

Το πρόβλημα που προκύπτει όμως, είναι ότι θέλουμε να κάνουμε write στην εφαρμογή `crtofs`, η οποία εκτελείται μετά την ολοκλήρωση της εφαρμογής boot και αυτό μας εμποδίζει να γράψουμε στο αρχείο.

Μία λύση θα ήταν να δημιουργούμε το αρχείο στην `crtofs`, μέσα στην οποία θα κάνουμε και write. Αρχικοποιούμε τον file descriptor στο αρχείο κεφαλίδας `/tools/lkl/include/lkl_host.c`, Γραμμή 56 (`int fd_journal;`).

```

641  /*-----*/
642
643  fd_journal = open("/home/myy601/journal.txt", O_CREAT | O_RDWR | O_APPEND, 0);
644  if(fd_journal < 0){
645      printf("Cannot open journal.txt file!\n");
646  }
647  /*-----*/
648  if (strstr(argv[0], "cptofs")) {

```

Βέβαια, η λύση αυτή δεν είναι η ιδανική λύση, καθώς για να δημιουργηθεί το αρχείο, είναι αναγκαίο να εκτελέσουμε την `cptofs`, η οποία καλεί λειτουργίες αντιγραφής.

Το file descriptor το αρχικοποιούμε στο αρχείο κεφαλίδας `/tools/lkl/include/lkl_host.h` Γραμμή 59.

```

58  /*-----*/
59  int fd_journal;
60  /*-----*/
61  /*-----*/
62  /*-----*/

```

Στην συνέχεια, εντοπίσαμε τις συναρτήσεις που επιφέρουν αλλαγές:

```
110
111 /*-----*/
112 /*-----*/
113 /*-----*/
114     char str[50];
115     snprintf(str, 50, "%d\n", fd);
116     write(fd_journal, "\nopen_src-> ", 10);
117     write(fd_journal, str, strlen(str));
118 /*-----*/
119 /*-----*/
120 /*-----*/
```

Γραμμή 114-117: Κάνουμε write στο αρχείο, πληροφορίες από την κλήση της συνάρτησης `open_src`.

```
146
147 /*-----*/
148 /*-----*/
149 /*-----*/
150     char str[50];
151     snprintf(str, 50, "%d\n", fd);
152     write(fd_journal, "\nOpen dst: Path: ", 22);
153     write(fd_journal, path, strlen(path));
154     write(fd_journal, str, strlen(str));
155 /*-----*/
156 /*-----*/
157 /*-----*/
```

Γραμμή 150-154: Γράφεται το μονοπάτι στο αρχείο journal, και στην Γραμμή 178-186 εγγράφεται το μέγεθος και τα δεδομένα που αποθηκεύονται.

```
175 /*-----*/
176 /*-----*/
177 /*-----*/
178     char str[100];
179     snprintf(str, 100, "%d\n", ret);
180     write(fd_journal, "\nRead src: \n", 12);
181     write(fd_journal, str, strlen(str));
182     write(fd_journal, "Buffer:\n", 9);
183     write(fd_journal, buf, strlen(buf));
184     write(fd_journal, "\n-----|-----", 68);
185     snprintf(str, 100, "\nLen: %d\n", len);
186     write(fd_journal, str, strlen(str));
187 /*-----*/
188 /*-----*/
189 /*-----*/
190
```

Με τον αντίστοιχο τρόπο, παίρνουμε δεδομένα από την crtofs όπως φαίνεται παρακάτω:

```
207 /*-----*/
208 /*-----*/
209 /*-----*/
210 char str[100];
211 snprintf(str, 100, "%d\n", ret);
212 write(fd_journal, "\ndst-> ", 6);
213 write(fd_journal, str, strlen(str));
214 /*-----*/
215 /*-----*/
216 /*-----*/
217
```

```
260 /*-----*/
261 /*-----*/
262 /*-----*/
263 char str[100];
264 snprintf(str, 100, "%d\n", fd_dst);
265 write(fd_journal, "\nfd_dst: ", 10);
266 write(fd_journal, str, strlen(str));
267
268 write(fd_journal, "Ptr: ", 5);
269 write(fd_journal, ptr, strlen(ptr));
270 write(fd_journal, "\n", 2);
271
272 snprintf(str, 100, "%ld\n", to_write);
273 write(fd_journal, "\nTo write: ", 11);
274 write(fd_journal, str, strlen(str));
275 /*-----*/
276 /*-----*/
277 /*-----*/
```

```
291 ret = len;
292 /*-----*/
293 /*-----*/
294 /*-----*/
295 char str[100];
296 snprintf(str, 100, "%ld\n", len);
297 write(fd_journal, "\ncopy-> ", 7);
298 write(fd_journal, str, strlen(str));
299 write(fd_journal, src, strlen(src));
300 write(fd_journal, "\n", 2);
301 write(fd_journal, dst, strlen(dst));
302 write(fd_journal, "\n", 2);
303 /*-----*/
304 /*-----*/
305 /*-----*/
```

```
343 /*-----*/
344 /*-----*/
345 /*-----*/
346 /*-----*/
347 char str[50];
348 snprintf(str, 50, "%d\n", ret);
349 write(fd_journal, "\nstat_src-> ", 10);
350 write(fd_journal, str, strlen(str));
351 /*-----*/
352 /*-----*/
353 /*-----*/
354
```

```
492 /*-----*/
493 /*-----*/
494 /*-----*/
495 char str[50];
496 snprintf(str, 50, "%d\n", ret);
497 write(fd_journal, "\ndoentry-> ", 12);
498 write(fd_journal, str, strlen(str));
499 /*-----*/
500 /*-----*/
501 /*-----*/
```



```

Terminal - myy601@myy601lab2: ~
File Edit View Terminal Tabs Help

    lklfuse.ro ? LKL_MS_RDONLY : 0, lklfuse.opts,
    mpoint, sizeof(mpoint));

    if (ret) {
        fprintf(stderr, "can't mount disk: %s\n", lkl_strerror(ret));
        goto out_halt;
    }

    ret = lkl_sys_chroot(mpoint);
    if (ret) {
        fprintf(stderr, "can't chdir to %s: %s\n", mpoint,
            lkl_strerror(ret));
        goto out_umount;
    }

    return 0;

out_umount:
    lkl_umount_dev(lklfuse.disk_id, lklfuse.part, 0, 1000);

out_halt:
    lkl_sys_halt();

out:
    return ret;
}

static void stop_lkl(void)
{
    int ret;

    ret = lkl_sys_chdir("/");
    if (ret)
        fprintf(stderr, "can't chdir to /: %s\n", lkl_strerror(ret));
    ret = lkl_sys_umount("/", 0);
    if (ret)
        fprintf(stderr, "failed to umount disk: %d: %s\n",
            lklfuse.disk_id, lkl_strerror

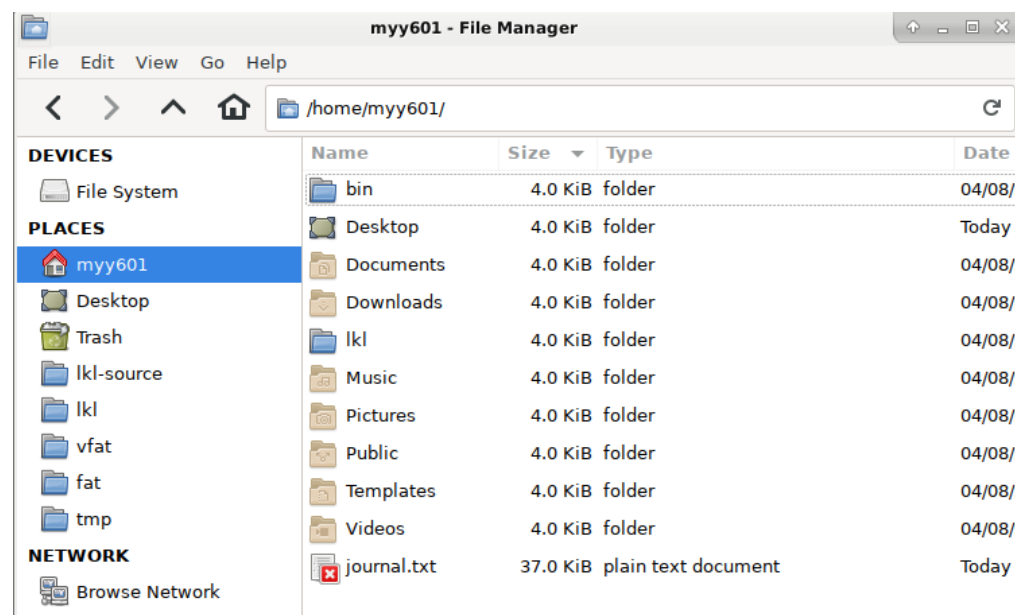
-----unab
Len: 4096

copy->0
./lklfuse.c
/mnt/0000fe00///lklfuse.c

doentry-> 0
root@myy601lab2:/home/myy601# cat journal.txt

```

Στην παραπάνω Εικόνα φαίνονται τα δεδομένα που είναι αποθηκευμένα στο αρχείο </home/myy601/journal.txt>. Παρακάτω φαίνεται η δημιουργία του αρχείου.



Για το τρίτο ερώτημα: Δημιουργία αρχείου σε επίπεδο πυρήνα.

Αρχικά, δημιουργήσαμε το αρχείο στον φάκελο **/fs/fat** με χρήση της συνάρτησης `sys_open` στο αρχείο `inode.c` (Γραμμή

```
58 /*-----*/
59 int fd_journal;
60 /*-----*/
61 /*-----*/
62 /*-----*/
```

Το path του αρχείου θα θέλαμε να καταλήγει στο `vfatfile`. Παρόλα αυτά, όταν στην εντολή δίνουμε μονοπάτι `/vfatfile` η δημιουργία αρχείου αποτύγχανε.

Έτσι, δημιουργήσαμε το αρχείο με path **/journal.txt** και το αρχείο δεν είναι ορατό στους φακέλους. Βέβαια, στο αρχείο έχουμε δυνατότητα εγγραφής και διαβάσματος κι αυτό φαίνεται εκτελώντας `sys_write` και `sys_read` στο αρχείο.

Στο αρχείο `inode.c`:

Συνάρτηση fat write inode

- Γραμμή 838: Αρχικοποίηση του πίνακα `c`.
- Γραμμή 839: Αρχικοποίηση του πίνακα `str` 1000 θέσεων, όπου θα χρησιμοποιηθεί για να κάνουμε `read` τα δεδομένα από το `journal`.
- Γραμμή 840: Αρχικοποίηση file descriptor.
- Γραμμή 878: Εκτέλεση της `sys_open`.
- Γραμμή 879: Έλεγχος δημιουργίας του αρχείου.
- Γραμμή 882-885: `printk` των πληροφοριών από το `msdos_dir_entry`
- Γραμμή 887-916: Εγγραφή των πληροφοριών στο `journal.txt`.
- Γραμμή 921: `lseek` με σκοπό να μεταβούμε με τον `pointer` στην αρχή του `journal`.
- Γραμμή 922-923: Διάβασμα και εκτύπωση των δεδομένων από το `journal`.

```
char c[50];
char str[1000];
int fd_journal_sys;
fd_journal_sys = sys_open("/journal.txt", O_CREAT | O_RDWR | O_APPEND, 0);
if(fd_journal_sys<0){
    printk(KERN_ERR "Cannot open journal!\n");
}
printk("Info for msdos_dir_entry\n");
printk(KERN_INFO " Ctime %d\n",raw_entry->ctime);
printk(KERN_INFO " Created date %d\n",raw_entry->cdate);
printk(KERN_INFO " Access date %d\n",raw_entry->adate);

sys_write(fd_journal_sys,"Ctime: ", 7);
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);
```

```

snprintf(c, 4, "%d\n", raw_entry->ctime);
sys_write(fd_journal_sys, c, strlen(c));
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

sys_write(fd_journal_sys, "Created date: ", 14);
sys_fsync(fd_journal_sys); sys_fdatasync(fd_journal_sys);

snprintf(c, 4, "%d\n", raw_entry->cdate);
sys_write(fd_journal_sys, c, strlen(c));
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

sys_write(fd_journal_sys, "Access date: ", 13);
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

snprintf(c, 4, "%d\n", raw_entry->adate);
sys_write(fd_journal_sys, c, strlen(c));
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);
printf("-----\n");

sys_lseek(fd_journal_sys, (off_t)0, 0);
sys_read(fd_journal_sys, str, 1000);
printf(KERN_INFO "Dats into journal\n%s", str);

```

Συνάρτηση fat_read_bpb

- Γραμμή 1816: αρχικοποίηση του πίνακα χαρακτήρων που θα χρησιμοποιηθεί για την αντιγραφή δεδομένων προς το journal.txt
- Γραμμή 1817: αρχικοποίηση file descriptor
- Γραμμή 1818: εκτέλεση της sys_open με flag O_APPEND, έτσι ώστε τα δεδομένα να γράφονται πάντα στο τέλος του αρχείου.
- Γραμμή 1819-1820: έλεγχος ορθής λειτουργίας της sys_open
- Γραμμή 1824-1826: printf των δεδομένων των οποίων θα αντιγράψουμε στο journal.
- Γραμμή 1828-1846: εγγραφή των δεδομένων στο journal.txt

```

char c[50];
int fd_journal_sys;
fd_journal_sys = sys_open("/journal.txt", O_CREAT | O_RDWR | O_APPEND, 0);
if(fd_journal_sys<0){
    printf(KERN_ERR "Cannot open journal!\n");
}
printf("-----\n");

```

```

printk("Info from fat_boot_sector:\n");
printk(KERN_INFO " Num of fats: %d\n",b->fats);
printk(KERN_INFO " Length of fat: %d\n",b->fat_length);

```

```

sys_write(fd_journal_sys,"Fats: ", 6);
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

```

snprintf(c, 4, "%d\n", b->fats);
sys_write(fd_journal_sys,c, strlen(c));
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

```

sys_write(fd_journal_sys,"Length of fat: ", 15);
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

```

snprintf(c, 4, "%d\n", b->fat_length);
sys_write(fd_journal_sys,c, strlen(c));
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

Συνάρτηση fat_read_root

- Γραμμή 1451: Αρχικοποίηση πίνακα
- Γραμμή 1452: Αρχικοποίηση file descriptor
- Γραμμή 1453: Δημιουργία αρχείου
- Γραμμή 1454: Έλεγχος δημιουργίας αρχείου
- Γραμμή 1459-1466: Δημιουργία printk για τα δεδομένα από το superblock στο inode
- Γραμμή 1471-1490: printk για τις πληροφορίες από το msdos_sb_info
- Γραμμή 1492-1733: εγγραφή των δεδομένων στο journal.txt.

```

char c[50];
int fd_journal_sys;
fd_journal_sys = sys_open("/journal.txt", O_CREAT | O_RDWR | O_APPEND, 0);
if(fd_journal_sys<0){
    printk(KERN_ERR "Cannot open journal!!\n");
}

```

```

printk("-----\n");
printk(KERN_INFO "Info from superblock in inode:\n");
printk(KERN_INFO "  Value of s_count is: %d\n",inode->i_sb->s_count);
printk(KERN_INFO "  Max links is: %d\n",inode->i_sb->s_max_links);

```

```

      .      .      .
      .      .      .

```

```

printk(KERN_INFO "   Sectors_per_clusters is: %d\n",sbi->sec_per_clus);
printk(KERN_INFO "   Root dentry start is: %d\n",sbi->dir_entries);
printk(KERN_INFO "   Max cluster is: %ld\n",sbi->max_cluster);

```

```

sys_write(fd_journal_sys,"\nValue of s_count: ", 19);
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

```

snprintf(c, 4, "%d\n", inode->i_sb->s_count);
sys_write(fd_journal_sys,c, strlen(c));
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

```

sys_write(fd_journal_sys,"Max links is: ", 14);
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

```

snprintf(c, 4, "%d\n", inode->i_sb->s_max_links);
sys_write(fd_journal_sys,c, strlen(c));
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

```

.      .
.      .
.      .

```

```

sys_write(fd_journal_sys,"\nMax cluster is: ", 17);
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

```

snprintf(c, 8, "%ld\n", sbi->max_cluster);
sys_write(fd_journal_sys,c, strlen(c));
sys_fsync(fd_journal_sys);
sys_fdatasync(fd_journal_sys);

```

Στο αρχείο *namei_vfat.c*:

Συνάρτηση vfat_build_slots.c

- Γραμμή 662-670: printk για τις πληροφορίες του msdos_dir_slot.

```

printk("-----\n");
printk("Info for msdos_dir_slot:\n");
printk(KERN_INFO " Sequence number for slot is: %d\n",ps->id);
printk(KERN_INFO " First 5 characters in name is: %s\n",ps->name0_4);

```

```

printk(KERN_INFO " Attridute byte is: %d\n",ps->attr);\
printk(KERN_INFO " 6 more characters in name is: %s\n",ps->name5_10);
printk(KERN_INFO " Starting cluster number is: %d\n",ps->start);
printk(KERN_INFO " Last 2 characters in name is: %s\n",ps->name11_12);
printk("-----\n");

```

Προσπαθήσαμε να γράψουμε τα δεδομένα αυτά από την συνάρτηση αυτήν στο journal.txt, αλλά εκτελώντας read μέσω του inode.c, δεν μπορούσαμε να τα διαβάσουμε. Δεν είμαστε σίγουροι αν γράφονται στο αρχείο και απλώς δε μπορούμε να τα διαβάσουμε ή αν δε γράφονται καθόλου.

Έχουμε βάλει τον κώδικα σε σχόλια για να μην επηρεάσει την εκτέλεση του προγράμματος (Γραμμή 673-740).

Επιχειρώντας να καλέσουμε την read από την συνάρτηση αυτήν, καταλήγουμε να χάνουμε δεδομένα από το inode.c.

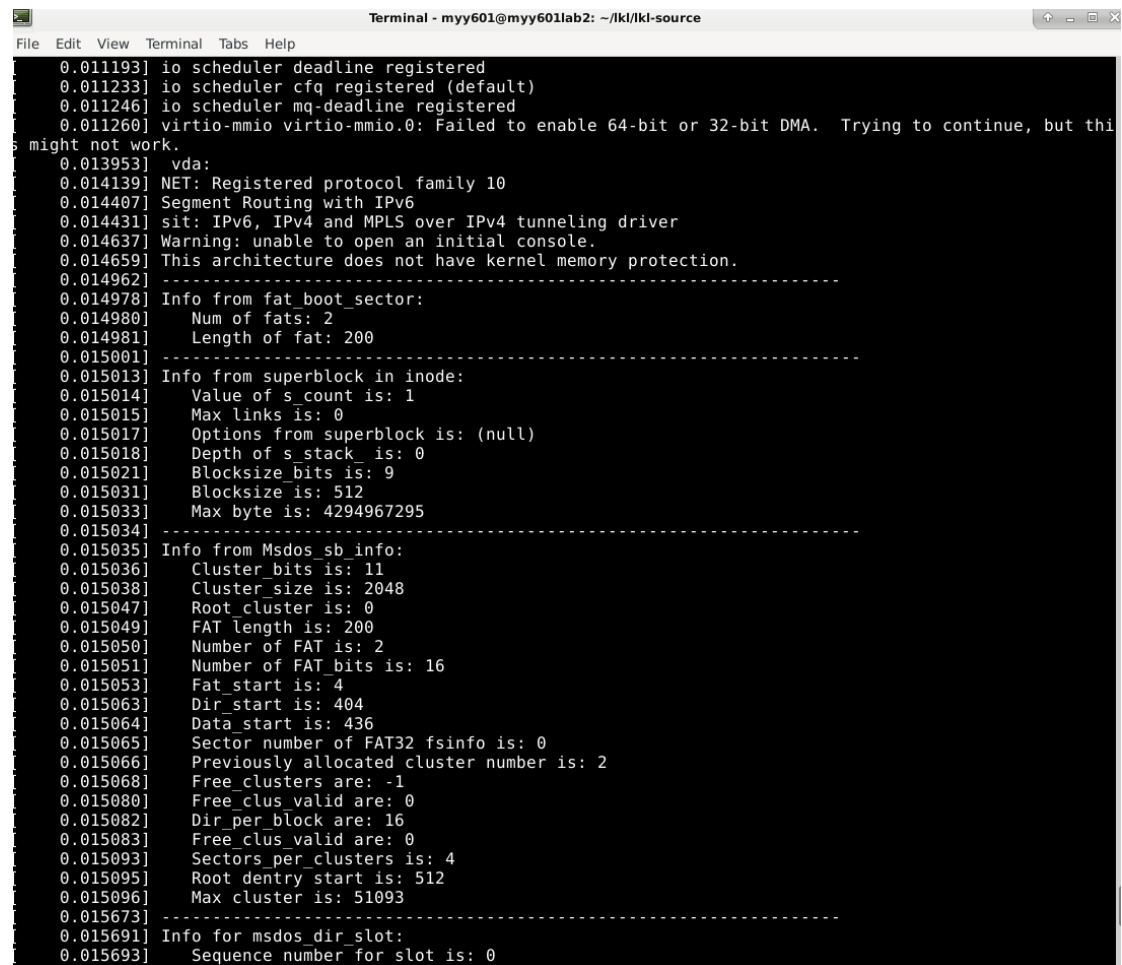
```

root@myy601lab2:/home/myy601/lkl/lkl-source/tools/lkl# make test
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, 0.305531 s, 343 MB/s
mutex                passed [1]
semaphore             passed [1]
join                  passed [joined 139645375379200]
disk_add              passed [3 0]
getpid                passed [1]
syscall_latency       passed [avg/min/max lkl: 79/70/601 native: 79/70/4970]
umask                 passed [22 777]
creat                 passed [0]
close                 passed [0]
failopen              passed [-2]
open                  passed [0]
write                 passed [5]
lseek                 passed [0 ]
read                  passed [5 test]
fstat                 passed [0 100721 5]
mkdir                 passed [0]
stat                  passed [0 40721]
nanosleep             passed [88123847]
pipe2                 passed [Hello world!]
epoll                 passed [Hello world!]
mount_fs              passed [proc: 0]
chdir                  passed [0]
opendir               passed [4]
getdents64            passed [4 . .. fs bus irq net sys tty kmsg maps misc stat i
mem cry]
umount_fs             passed [proc: 0 0 0]
mount_dev              passed [0]
chdir                  passed [0]
opendir               passed [6]
getdents64            passed [6 . .. ]
umount_dev             passed [0 0 0]
lo_ifup                passed [0]
gettid                 passed [81430]
syscall_thread         passed []
many_syscall_threads   passed []
root@myy601lab2:/home/myy601/lkl/lkl-source/tools/lkl#

```

Στην Εικόνα φαίνεται ότι το πρόγραμμα περνάει όλα τα tests που εκτελούνται κατά την εντολή **make test**.

Παρακάτω, παραθέτουμε εκτελέσεις του κώδικα:

A terminal window titled "Terminal - myy601@myy601lab2: ~/lkl/lkl-source" displays a series of kernel boot logs. The logs include messages about io scheduler registration, virtio-mmio DMA failure, vda registration, NET protocol family 10, IPv6 tunneling driver, console opening warning, kernel memory protection, and detailed FAT and MSDOS superblock information. The output is timestamped with values like 0.011193, 0.011233, etc.

```
0.011193] io scheduler deadline registered
0.011233] io scheduler cfq registered (default)
0.011246] io scheduler mq-deadline registered
0.011260] virtio-mmio virtio-mmio.0: Failed to enable 64-bit or 32-bit DMA. Trying to continue, but thi
might not work.
0.013953] vda:
0.014139] NET: Registered protocol family 10
0.014407] Segment Routing with IPv6
0.014431] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
0.014637] Warning: unable to open an initial console.
0.014659] This architecture does not have kernel memory protection.
0.014962] -----
0.014978] Info from fat_boot_sector:
0.014980]   Num of fats: 2
0.014981]   Length of fat: 200
0.015001] -----
0.015013] Info from superblock in inode:
0.015014]   Value of s_count is: 1
0.015015]   Max links is: 0
0.015017]   Options from superblock is: (null)
0.015018]   Depth of s_stack is: 0
0.015021]   Blocksize bits is: 9
0.015031]   Blocksize is: 512
0.015033]   Max byte is: 4294967295
0.015034] -----
0.015035] Info from Msdos_sb info:
0.015036]   Cluster_bits is: 11
0.015038]   Cluster_size is: 2048
0.015047]   Root_cluster is: 0
0.015049]   FAT_length is: 200
0.015050]   Number of FAT is: 2
0.015051]   Number of FAT_bits is: 16
0.015053]   Fat_start is: 4
0.015063]   Dir_start is: 404
0.015064]   Data_start is: 436
0.015065]   Sector number of FAT32 fsinfo is: 0
0.015066]   Previously allocated cluster number is: 2
0.015068]   Free_clusters are: -1
0.015080]   Free_clus_valid are: 0
0.015082]   Dir_per_block are: 16
0.015083]   Free_clus_valid are: 0
0.015093]   Sectors_per_clusters is: 4
0.015095]   Root_dentry_start is: 512
0.015096]   Max_cluster is: 51093
0.015673] -----
0.015691] Info for msdos_dir_slot:
0.015693]   Sequence number for slot is: 0
```

Στην παραπάνω Εικόνα φαίνονται τα printk όπου εκτυπώνονται κατά την εκτέλεση της εντολής `./cptofs -i /tmp/vfatfile -p -t vfat lklfuse.c /.`

```
terminal - myy601@myy601lab2: ~/lkl/lkl-source
File Edit View Terminal Tabs Help
0.015083] Free clus valid are: 0
0.015093] Sectors_per_clusters is: 4
0.015095] Root dentry start is: 512
0.015096] Max cluster is: 51093
0.015673] -----
0.015691] Info for msdos_dir_slot:
0.015693] Sequence number for slot is: 0
0.015695] First 5 characters in name is:
0.015696] Attribute byte is: 0
0.015698] 6 more characters in name is:
0.015700] Starting cluster number is: 0
0.015701] Last 2 characters in name is:
0.015714] -----
0.016183] Info for msdos_dir_entry
0.016198] Ctime 0
0.016200] Created date 33
0.016201] Access date 33
0.016207] -----
0.016210] Datas into journal
0.016210] Fats: 2
0.016210] Length of fat: 200
0.016210] Value of s_count: 1
0.016210] Max links is: 0
0.016210] Options from superblock is: (null)
0.016210] Depth of s_stack is: 0
0.016210] Blocksize_bits is: 9
0.016210] Blocksize is: 512
0.016210] Max byte is: 4294967
0.016210] Info from Msdos_sb info: 4294967
0.016210] Cluster_bits is: 11
0.016210] Cluster_size is: 204
0.016210] Root_cluster is: 0
0.016210] FAT length is: 200
0.016210] Number of FAT is: 2
0.016210] Number of FAT_bits is: 16
0.016210] Fat_start is: 4
0.016210] Dir_start is: 404
0.016210] Data_start is: 436
0.016210] Sector number of FAT32 fsinfo is: 0
0.016210] Previously allocated cluster number is: 2
0.016210] Free_clusters are: -1
0.016210] Free clus valid are: 0
0.016210] Sectors_per_clusters is: 4
0.016210] Root dentry start is: 512
0.016210] Max cluster is: 51093
0.016564] reboot: Restarting system
root@myy601lab2:/home/myy601/lkl/lkl-source/tools/lkl#
```

Στην παραπάνω Εικόνα φαίνονται τα δεδομένα που είναι αποθηκευμένα μέσα στο journal.txt.