Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

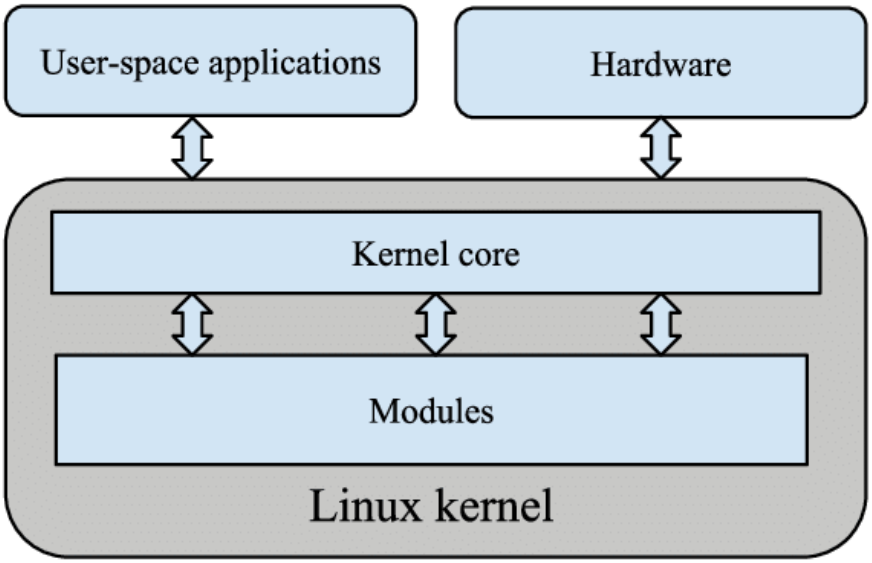
Προπτυχιακό Μάθημα: «Λειτουργικά Συστήματα»

2η Εργαστηριακή Άσκηση

Μέλη Ομάδας - Α.Μ.:

Παναγιώτης Ντουνέτας - 2781

Γεώργιος Κρομμύδας – 3260



ΙΩΑΝΝΙΝΑ,

2021

Αναφορά 2ης Εργαστηριακής Άσκησης

(Υλοποίηση αρχείου καταγραφής στο σύστημα αρχείων FAT του Linux)

(Κατανόηση της λειτουργίας του LKL, αρχείο καταγραφής, μεταφορά από και προς το vfatfile)

Μέρος Α: Ζητούμενα, ανάλυση διαδικασίας, πρώτες ιδέες.

Μας δίνεται μία εικονική μηχανή (Debian) η οποία περιέχει το source code του Linux για εκτέλεση σε επίπεδο χρήστη, ή αλλιώς η βιβλιοθήκη LKL (Linux Kernel Library). Στόχος είναι, η παραγωγή ενός αρχείου το οποίο θα κρατάει αλλαγές που προκύπτουν στο σύστημα, με σκοπό τη διασφάλιση της επαναφοράς του συστήματος αν αυτό έρθει σε ασυνεπή μορφή. Αυτό θα γίνει με την αποθήκευση δεδομένων ή/και μετα-δεδομένων στο αρχείο καταγραφής, έτσι ώστε να διοχετεύσει το σύστημα με τις τελευταίες τιμές που είχε πριν αυτό έρθει σε δυσλειτουργία.

Ο κώδικας της LKL είναι χωρισμένος σε φακέλους ανάλογα την λειτουργία που θέλει να εξυπηρετήσει κάθε φορά. Για παράδειγμα στο φάκελο fs γίνονται οι λειτουργίες υλοποίησης του VFAT και του συστήματος αρχείων, στο mm η διαχείριση της μνήμης, στο tools διάφορα εργαλεία επικοινωνίας μεταξύ της LKL και του vfatfile κ.ο.κ .

Όσον αφορά το σύστημα FAT , είναι ιεραρχικά δομημένο ως προς τον τρόπο αποθήκευσης των αρχείων. Διαχειρίζεται το superblock, τα inodes, αρχεία και καταλόγους μέσω λειτουργιών που μας επιτρέπουν να παρατηρούμε λεπτομερείς αλλαγές.

Η διαδικασία που ακολουθήσαμε στηρίζεται κυρίως στις φροντιστηριακές διαφάνειες και στην παράγραφο της εκφώνησης στη σελίδα 3 [Η υλοποίησης του FAT στο Linux… για το VFAT] που περιέχει ενδεικτικές δομές του κώδικα. Έτσι, ψάχνοντας μία προς μία αυτές τις δομές και τα περιεχόμενά τους, οδηγούμαστε πιο βαθιά στον κώδικα, μέσα σε πολλές διαφορετικές συναρτήσεις και αρχεία. Σε αυτή την ιδέα στηρίξαμε και την διαδικασία της εφαρμογής των δοκιμαστικών εκτυπώσεων του 1ου ερωτήματος (τοποθετήσαμε printk() σε κάθε σημείο όπου ορίζεται κάθε πεδίο του κάθε struct που δίνεται στην εκφώνηση). Σκοπός είναι να παρατηρήσουμε τη σειρά, καθώς και ποιες δομές καλούνται σε μία απλή κλήση της εφαρμογής cptofs (copy to filesystem).

Έχουμε κατασκευάσει 2 ειδών αρχεία που δοκιμάζουμε να αποθηκεύσουμε στο vfatfile:

- **.TXT αρχεία:** test1.txt, περιέχει ένα από μήνυμα μίας γραμμής, test2.txt, περιέχει την εκφώνηση της άσκησης.
- **.C αρχείο:** test3.c περιέχει μια απλή διαδικασία εκτύπωσης hello world. (παρόμοια δοκιμάσαμε και το lklfuse.c , απλά βάλαμε ένα μικρότερο αρχείο για ευκολία)

Θέλουμε να παρατηρήσουμε τις αλλαγές στην εκτέλεση, καθώς και πως επηρεάζονται οι λειτουργίες, για αυτό το λόγο έχουμε δημιουργήσει test files με διαφορετικό μέγεθος και τύπο. Το μικρότερο σε μέγεθος είναι το test1.txt και το μεγαλύτερο το test2.txt. Θα εξηγήσουμε αναλυτικά ότι υλοποιήσαμε με παραδείγματα κώδικα που προσθέσαμε ή που υπήρχε ήδη, μέσω του LKL_DOX, καθώς και μηνύματα του τερματικού.

<u>Βήμα 1ο: Τοποθέτηση printk() στα ενδεδειγμένα σημεία και εξήγηση του αποτελέσματος.</u>

Σύμφωνα με την εκφώνηση, δίνονται κάποιες δομές που αφορούν τις παρακάτω λειτουργίες:

- **Superblock:** struct *super_operations fat_ops* (fs/fat/inode.c)
- **Μνήμη:** struct *address_space_operations fat_aops* (fs/fat/inode.c)
- **Εγγραφές FAT:** struct *fatent_operations fat12_ops*, *struct fatent_operations fat16_ops*, *struct fatent_operations fat32_ops* (fs/fat/fatent.c)
- **Αρχείο:** struct *file_operations fat_file_operations* (fs/fat/file.c)
- **Inode:** struct *inode_operations fat_file_inode_operations* (fs/fat/file.c)
- **Κατάλογοι:** struct *inode_operations msdos_dir_inode_operations* (fs/fat/namei_msdos.c για FAT) και struct *inode_operations vfat_dir_inode_operations* (fs/fat/namei_vfat.c για VFAT). (Μάλλον τυπογραφικό της εκφώνησης εφόσον δεν βρήκαμε πρόθεμα msdos, άλλα βρήκαμε όμοια συνάρτηση με vfat).

Οπότε, σε κάθε ένα από τα πεδία των δομών αυτών, βάζουμε printk() έτσι ώστε να δούμε τη σειρά εκτέλεσης , καθώς και ποιες από αυτές εκτελούνται. Πιο συγκεκριμένα:

Από κάθε δομή έχουμε τις παρακάτω εφαρμογές printk():

Με την βοήθεια κυρίως του LKL_DOX προσπαθούμε να εντοπίσουμε που ορίζονται οι παρακάτω συναρτήσεις (οι περισσότερες βρίσκονται στο ίδιο αρχείο με το struct με μερικές εξαιρέσεις):

- **Superblock:** struct *super_operations fat_ops* (fs/fat/inode.c):

fat_alloc_inode, fat_destroy_inode, fat_write_inode, fat_evict_inode, fat_put_super, fat_statfs, fat_remount, fat_show_options. Επίσης, προστέθηκε printk() στη fat_flush_inodes γιατί το όνομα παραπέμπει σε "άδειασμα", οπότε πιθανότατα θα γίνεται προς το τέλος και ίσως φανεί χρήσιμο για αργότερα.

- **Μνήμη:** struct *address_space_operations fat_aops* (fs/fat/inode.c):

fat_readpage, fat_readpages, fat_writepage, fat_writepages, fat_write_begin, fat_write_end, fat_direct_IO, _fat_bmap.

- **Εγγραφές FAT:** struct *fatent_operations fat12_ops*, *struct fatent_operations fat16_ops*, *struct fatent_operations fat32_ops* (fs/fat/fatent.c): fat12_ent_blocknr, fat12_ent_set_ptr, fat12_ent_bread, fat12_ent_get, fat12_ent_put, fat12_ent_next, ίδιες για 16 και 32.

- **Αρχείο:** struct *file_operations fat_file_operations* (fs/fat/file.c): Εδώ υπάρχουν μερικές συναρτήσεις, ορισμένες σε διαφορετικά αρχεία: generic_file_llseek(fs/read_write.c),
generic_file_read_iter(mm/filemap.c),
generic_file_write_iter(mm/filemap.c),
generic_file_mmap(mm/filemap.c), fat_file_release, fat_generic_ioctl, fat_generic_compat_ioctl, fat_file_fsync,
generic_file_splice_read(fs/splice.c), fat_fallocate.

- **Inode:** struct *inode_operations fat_file_inode_operations* (fs/fat/file.c): fat_setattr, fat_getattr.

- **Κατάλογοι:** struct *inode_operations msdos_dir_inode_operations* (fs/fat/namei_msdos.c για FAT) και struct *inode_operations vfat_dir_inode_operations*(fs/fat/namei_vfat.c για VFAT):

→ msdos_create, msdos_lookup, msdos_unlink, msdos_mkdir, msdos_rmdir, msdos_rename, fat_setattr, fat_getattr (set και get attr στο fs/fat/file.c) για MSDOS.

→ vfat_create, vfat_lookup, vfat_unlink, vfat_mkdir, vfat_rmdir, vfat_rename, fat_setattr, fat_getattr (set και get attr στο fs/fat/file.c) για VFAT.

Όλα τα printk() τοποθετούνται μόλις τελειώσει η δήλωση όλων τον εντολών. Αλλιώς υπάρχει περίπτωση κάπου ενδιάμεσα να καλείται κάποια άλλη συνάρτηση με printk() και να βγάλουμε λάθος συμπεράσματα.

Εκτελούμε ένα παράδειγμα ./cptofs, με το δοκιμαστικό αρχείο test1.txt (επειδή τα υπόλοιπα αρχεία βγαίνουν ~15 σελίδες σαν copy-paste εδώ), για να δούμε, λοιπόν, τι κάνουν τα print που προσθέσαμε:

Κάνουμε ./cptofs -i /tmp/vfatfile -p -t vfat test1.txt / :

[1]###mm/filemap.c-INSIDE GENERIC_FILE_READ_ITER###

[2] Creating our journal...

[3] File opened.

[4] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###

[5] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###

[6] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###

[7] ###fs/fat/namei_vfat.c - INSIDE VFAT_LOOKUP###

[8] ###fs/fat/namei_vfat.c - INSIDE VFAT_CREATE###

[9] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###

[10] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[11] ###fs/fat/inode.c - INSIDE FAT_WRITE_BEGIN###

[12] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###

[13] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###

[14] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###

[15] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###

[16] ###fs/fat/fatent.c - INSIDE FAT16_ENT_PUT###

[17] ###fs/fat/inode.c - INSIDE FAT_WRITE_END###

[18] ###fs/fat/file.c - INSIDE FAT_FILE_RELEASE###

[19] ###fs/fat/inode.c - INSIDE FAT_WRITE_INODE###

[20] ###fs/fat/inode.c - INSIDE FAT_WRITE_INODE###

[21] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###

[22] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###

[23] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###

[24] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###

[25]###fs/fat/inode.c - INSIDE FAT_PUT_SUPER###

[26] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###

[27] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###

[28]###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###

[29] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###

Τοποθετούμε μόνο για την αναφορά (για καλύτερη στοίχιση και περιγραφή των αποτελεσμάτων) τα αποτελέσματα από το τερματικό, αριθμημένα από το 1, στη θέση που κανονικά είναι ο χρόνος της printk().

Προς το παρόν αγνοούμε τις 3 πρώτες γραμμές (αφορούν το αρχείο).

Παρατηρούμε γενικά ότι υπάρχει ένα μοτίβο: Οι γραμμές 4,5,6,9 δείχνουν ότι η fat_alloc_inode() εκτελείται, συνεπώς (και από την δήλωσή της) βλέπουμε πως κάνει δέσμευση μνήμης cache για το αρχείο που θέλουμε να μεταφέρουμε – αντιγράψουμε στο vfatfile. Το μοτίβο αυτό ακολουθείται και στην αποδέσμευση καθώς οι γραμμές 21-24,26-29 συνοδεύουν κάθε fat_alloc_inode() με ένα fat_evict_inode() και fat_destroy_inode(). Η vfat_lookup() είναι υπεύθυνη για τη γραμμή 9 εξ αιτίας της εντολής της:

```
inode = fat_build_inode(sb, sinfo.de, sinfo.i_pos);
```

Άρα οι γραμμές 4,5,6 είναι ανεξάρτητες λειτουργικά από την 9 (συμβαίνουν για άλλο λόγο). Οι 4,5,6 πιθανότατα αφορούν το superblock ενώ η 9 αφορά το αρχείο που θα αντιγραφεί. Αυτό το καταλαβαίνουμε διότι, σε κάθε μία από τις vfat_lookup() και vfat_create() εφαρμόζεται αμοιβαίος αποκλεισμός με το superblock:

```
mutex_lock(&MSDOS_SB(sb)->s_lock);
```
**Από vfat_lookup() **

```
mutex_lock(&MSDOS_SB(sb)->s_lock);
```
**Από vfat_create() **

Οι γραμμές 10-17 αφορούν την εγγραφή του αρχείου. Σαν γενική προσέγγιση, γίνεται ενημέρωση των λειτουργιών fat16 για την εγγραφή. Σύμφωνα με την παρακάτω πρόταση:

FAT16:υποστηρίζει ονόματα αρχείων μέχρι 8 χαρακτήρες για το όνομα + 3 για τον τύπο του αρχείου και μέχρι 4GB αποθηκευτικό χώρο

στηρίζουμε το γεγονός ότι χρησιμοποιούνται οι συναρτήσεις του fat16, μιας και το αρχείο μας πληρεί τις προϋποθέσεις για τη χρήση τους.

Εδώ λογικά, γίνεται και η ανάθεση των clusters του αρχείου με βάση το παρακάτω:

Cluster

- Συνεχόμενη περιοχή στο δίσκο προκαθορισμένου μεγέθους που περιλαμβάνει πολλαπλούς τομείς
- Τα δεδομένα ενός αρχείου αποθηκεύονται σε πολλαπλά cluster
- Κάθε cluster περιλαμβάνει δείκτη στο επόμενο cluster ενός αρχείου ή NULL αν πρόκειται για το τελευταίο cluster

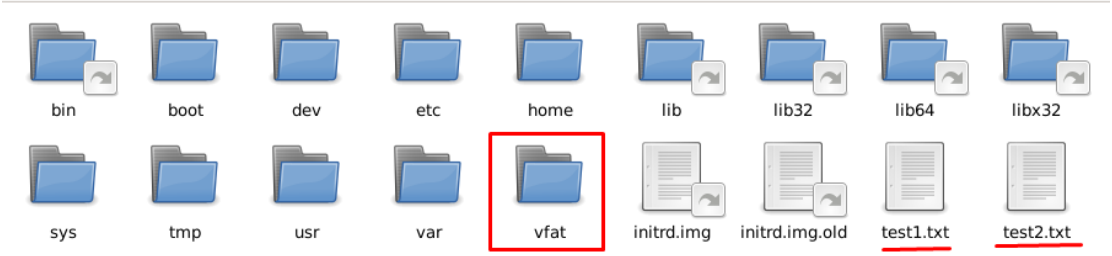
Στη γραμμή 18 έχουμε την fat_file_release(). Από τον κώδικά της παρατηρούμε: `fat_flush_inodes(inode->i_sb, inode, NULL);` , σηματοδοτώντας το τέλος της μεταφοράς του αρχείου και διαγραφή ή εκκαθάριση του inode.

Στη γραμμή 25 εμφανίζεται η fat_put_super(). Είναι η συνάρτηση που καλείται ουσιαστικά τελευταία από τις βασικές, και αυτή με τη σειρά της καλεί τα `iput(sbi->fsinfo_inode)` και `iput(sbi->fat_inode)` που η περιγραφή τους δείχνει ότι είναι υπεύθυνα για την καταστροφή των τελικών inodes (iput(): *Puts an inode, dropping its usage count. If the inode use count hits zero, the inode is then freed and may also be destroyed*).

Τέλος, οι γραμμές 19, 20 προέρχονται από την fat_write_inode(). Ουσιαστικά, σαν λογική σειρά, τώρα γίνεται και η απελευθέρωση των clusters (`err = fat_clusters_flush(sb);`).

Αυτά ισχύουν για ένα πολύ μικρό μικρό αρχείο. Εκτελώντας μεγαλύτερα αρχεία , παρατηρούμε ότι υπάρχει ομοιότητα στα αρχικά και τελικά βήματα της εκτέλεσης με μια σημαντική διαφορά. Αν για παράδειγμα τρέξουμε το test2.txt που περιέχει όλη την εκφώνηση της άσκησης, παρατηρούμε πως επαναλαμβάνεται το κομμάτι των γραμμών 10-17 (του παραπάνω παραδείγματος) που αφορά την εγγραφή του αρχείου και παρατηρούμε ότι αυξάνονται κατά πάρα πολύ και οι ενδιάμεσες λειτουργίες fat16 που καλούνται. Αυτό γίνεται διότι, αυξάνεται ο όγκος των δεδομένων που πρέπει να αποθηκευτεί οπότε αυτό έρχεται και δένει με την πρόταση: "Τα δεδομένα ενός αρχείου αποθηκεύονται σε πολλαπλά cluster".

Όσον αφορά τις εντολές που χρησιμοποιήσαμε, έγινε δοκιμή όλων των προτεινόμενων. Η cptofs κάνει επιτυχώς τη μεταφορά από το σύστημά μας στο vfatfile. Η cpfromfs μετέφερε το αρχείο της επιλογής μας στο γονικό φάκελο του vfat (που προσομοίωνε το εσωτερικό του vfatfile στο οποίο αποθηκεύονταν τα αρχεία, παρακάτω το αποτέλεσμα της cpfromfs **τα αρχεία προερχόμενα από την cpfromfs δεν ήταν δυνατό να διαγραφούν**):



Η εντολή test/boot καταλάβαμε πως ήταν ένας συνδυασμός:

α) από το mount και την έναρξη του vfatfile στην εικονική μηχανή και

β) από τα αποτελέσματα της εντολής make test, καθώς εμφανίζονται ανάμεικτα τα μηνύματα της κάθε λειτουργίας από αυτές τις διαδικασίες ανάλογα την σειρά εκτέλεσής τους.

Για να είναι ορατά τα αρχεία που μεταφέρουμε στο vfatfile, απαιτείται να γίνει mount του vfatfile (mount -t vfat -o loop /tmp/vfatfile /vfat) στο φάκελο /tmp (του συστήματος της εικονικής μηχανής). Τα αρχεία δεν ήταν ορατά αν το filesystem ήταν ήδη mounted, οπότε έπρεπε πρώτα να γίνει unmount (umount /vfatfile). Η διαδικασία αυτή γίνεται ως super user (root) με την εντολή στο τερματικό su. Το vfat αποτελεί το σημείο προσάρτησης (mount point).

Εφόσον πήραμε τη γενική ιδέα για το πως λειτουργεί η διαδικασία της αποθήκευσης στο vfatfile, όπως και τη ροή του κώδικα κατά προσέγγιση, είμαστε σε ευνοϊκή θέση να μελετήσουμε τη δομή του FAT , που είναι αποθηκευμένο στη διαδρομή fs/fat.

<u>Βήμα 2ο: Επεξεργασία των λεπτομερειών υλοποίησης του FAT (συναρτήσεις και δομές)</u>

Αρχικά μελετάμε τις δομές που βρίσκονται στο μονοπάτι

include/uapi/linux/msdos_fs.h .

→ struct __fat_dirent: Κρατάει πληροφορίες για το directory entry.

→ struct fat_boot_sector: Περιέχει τις απαραίτητες πληροφορίες για τα στοιχεία που απαρτίζουν το FAT (μέσο αποθήκευσης) , βλέπουμε ότι περιέχει μεταβλητές που αφορούν τα sectors και clusters, καθώς και διαφορετικές μεταβλητές ανάλογα με το ποια μορφή του FAT θα εκτελεστεί (16 ή 32).

→ struct fat_boot_fsinfo: Κρατάει πληροφορίες για το FAT πάλι , ‘οπως υποδηλώνει και το όνομα (fsinfo – file system information).

→ struct msdos_dir_entry: Πληροφορίες για το directory entry, μόνο που εδώ το πρόθεμα αλλάζει και γίνεται msdos, οπότε αφορά λειτουργία του λειτουργικού συστήματος.

→ struct msdos_dir_slot: Πληροφορίες για τα slots, πάλι για το λειτουργικό σύστημα.

Βλέπουμε ότι τα παραπάνω structs περιέχουν πεδία που κρατάνε τιμές σχετικές με την αρχικοποίηση του συστήματος. Επίσης, το αρχείο msdos_fs.h περιέχει αρκετά defines με αρχικές τιμές που αφορούν τεχνικές λεπτομέρειες της υλοποίησης του FAT.

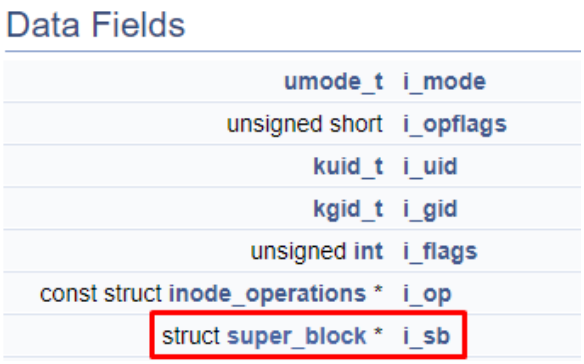
Επίσης, μελετάμε και τις υπόλοιπες ενδεδειγμένες δομές και λειτουργίες (από τις φροντιστηριακές διαφάνειες σελ. 52-53, οι λειτουργίες ταυτίζονται με αυτές που βάλαμε τα Printk() στο 1ο ερώτημα οπότε δεν θα τις αναφέρουμε ξανά και θα δούμε μόνο τις δομές):

Για το File Allocation Table: Ελέγχουμε τα περιεχόμενα των αρχείων fat.h και fatent.c και βλέπουμε ότι περιέχουν την αρχικοποίηση του fat_entry, που δίνεται σαν argument ή καλείται στις συναρτήσεις του FAT, καθώς και πολλών άλλων συναρτήσεων.

Για το superblock (struct msdos_sb_info) : Όπως αναγράφεται, η δομή αυτή περιέχει πεδία που αφορούν το superblock. Ψάχνουμε στο fat.h την αρχικοποίηση του και διαπιστώνουμε το εξής σχόλιο στην αρχή: ***"MS-DOS file system in-core superblock data"*** . Άρα εδώ καταλαβαίνουμε πως ότι αλλαγή ή καταγραφή κάνουμε θα γίνει σε επίπεδο πυρήνα. Μένει να δούμε όμως, που αρχικοποιούνται τα πεδία. Μέσα στο μονοπάτι fs/fat (ψάχνουμε εδώ γιατί πλέον δουλεύουμε μέσα στο FAT), ανοίγουμε τερματικό και εκτελούμε: ***~/bin/search msdos_sb_info***. Θα ψάξει εκεί μέσα για την αρχικοποίηση της δομής , έτσι ώστε να δούμε πως χρησιμοποιείται. Βλέπουμε πως αρχικοποιείται ως:

> ***struct msdos_sb_info *sbi = MSDOS(sb) ή MSDOS(inode→i_sb) ή MSDOS(dir→i_sb) ή MSDOS(dentry->d_sb)***

Ψάχνοντας τα διαθέσιμα αρχεία του ***fs/fat*** βλέπουμε πως οι περισσότερες συναρτήσεις δέχονται ως όρισμα το ***sb*** ή το ***inode*** ή τα υπόλποιπα αναφερόμενα (από το inode γίνεται αναφορά μερικές φορές στο sb μέσο του πεδίου i_sb του inode, ομοίως για τα υπόλοιπα, για αυτό έχουμε και διαφορετικές αρχικοποιήσεις). Βλέπουμε στη δομή inode το πεδίο:



Έτσι πηγαίνουμε σε κάθε ένα από τα αρχεία που αναφέρονται (namei_msdos.c, dir.c, file.c, namei_vfat.c, inode.c, cache.c) και ψάχνουμε το "**sbi→**", που υποδηλώνει αναφορά στην τιμή του struct. Καταλήγουμε στο αρχείο ***inode.c***, και πιο συγκεκριμένα στη συνάρτηση **fat_fill_super()**. Βρίσκουμε λοιπόν, ότι εδώ μέσα αρχικοποιούνται μαζικά μερικά πεδία του msdos_sb_info κάτω από ένα mutexlock που αφορά το sbi. Άρα η συνάρτηση fat_fill_super (που μάλλον μεταφράζεται σε FAT fill superblock) είναι υπεύθυνη για την αρχικοποίηση του.

[Όλες αυτές οι πληροφορίες θα φανούν αργότερα χρήσιμες για το πως δημιουργούμε το αρχείο και πως κάνουμε τις καταγραφές των τιμών].

Για το inode (struct msdos_inode_info): Ψάχνουμε όμοια στο fat.h αρχικά. Πάνω από τη δήλωση του υπάρχει το μήνυμα: "***MS-DOS file system inode data in memory***". Άρα τα πεδία του είναι υπεύθυνα για τη διαχείριση της μνήμης όπως πχ από που ξεκινάει το κάθε Inode (int i_start). Πάλι με bin/search, ψάχνουμε να δούμε πώς καλείται το struct και σε ποια αρχεία και λαμβάνουμε δύο χρήσεις:

→ struct msdos_inode_info *i = MSDOS_I(inode);

→ struct msdos_inode_info *ei = (struct msdos_inode_info *)foo;

Επίσης εδώ, ψάχνουμε στα αρχεία που εμφανίζονται(inode.c, nfs.c, cache.c) για τυχόν αρχικοποιήσεις ή αλλαγές.

Για τα Directory Entires (struct fat_slot_info): Η συνάρτηση που δίνεται (msdos_slot__info), την ψάξαμε με το search και δεν την βρήκαμε πουθενά σε ολόκληρο κιόλας το lkl-source. Στο προτεινόμενο αρχείο fat.h υπάρχει μόνο η fat_slot_info. Εκτελούμε search για τη fat_slot_fino και έχουμε:

→ struct fat_slot_info sinfo;

→ struct fat_slot_info old_sinfo, sinfo;

και εντοπίζεται στα αρχεία namei_msdos.c, dir.c, namei_vfat.c, nfs.c, στα οποία ψάχνουμε για αρχικοποιήσεις ή αλλαγες.

Για το File (struct file): Κάνοντας search file struct στο fs/fat πήραμε πάρα πολλά αποτελέσματα, διότι εμφάνιζε και παραπλήσιους (δηλαδή το string που έψαχνε η search ‘struct file *’ που είχαμε εφαρμόσει παρόμοια και στις υπόλοιπες) τύπους με το file πχ file_system_type. Οπότε πάμε σε ένα προς ένα τα .c αρχεία του καταλόγου και αναζητούμε: “struct file *” έτσι ώστε να πάρουμε μόνο το σημείο που δηλώνεται το σκέτο file. Έτσι έχουμε δύο ειδών εμφανίσεις:

→ struct file *file = iocb→ki_filp; (στο inode.c)

και σαν παράμετροι σε συναρτήσεις ως: struct file *filp , struct file *file.

Τα αρχεία στα οποία εμφανίζονται κλήσεις της file είναι: dir.c, inode.c, file.c και παρόμοια ψάχνουμε αρχικοποιήσεις ή αλλαγες.

Έχουμε μελετήσει τις συναρτήσεις και τις δομές που έχουν δοθεί καθώς και τα περιεχόμενά τους. Βρήκαμε πως εμφανίζεται το καθένα από αυτά και σε ποια σημεία, καθιστώντας τώρα ευκολότερη τη διαδικασία να σκεφτούμε που μπορούμε να επεξεργαστούμε το αρχείο καταγραφής (journal) και τις τιμές που πρέπει να καταγράψουμε.

<u>Βήμα 3ο: Διαδικασία δημιουργίας αρχείου και καταγραφής δεδομένων (incomplete)</u>

Το αρχείο καταγραφής (Journal) είναι υπεύθυνο να κρατά πληροφορίες (δεδομένα και μέτα-δεδομενα) , έτσι ώστε αν το σύστημα μας έρθει σε μία μη-λειτρουγική κατάσταση, να αντληθούν τα περιεχόμενα του και να επαναφέρουμε το σύστημα. Όπως είχε αναφερθεί στις τελευταίες διαλέξεις, το FAT δεν διαθέτει δικό του σύστημα journaling. Στην άσκηση μας είχε δοθεί παράδειγμα συστημάτων αρχείων (ex. Ext4) που διαθέτουν δικό τους σύστημα. Η δική μας υλοποίηση βασίστηκε σε μία πολύ πιο "απλή" διαδικασία από το παραπάνω σύστημα (το ext4 που βασίζεται στην καταγραφή του jbd2, που με μία έρευνα που κάναμε ήταν αρκετά δύσκολο να βρούμε έστω έναν παρόμοιο τρόπο να τροφοδοτήσουμε και το δικό μας file system) πιο πολύ σαν ιδέα. Δηλαδή, θέλουμε να ανοίξουμε ένα αρχείο, στο οποίο θα καταγράφονται αλλαγές ή αρχικοποιήσεις μεταβλητών που αφορούν πεδία δομών (superblock, inodes, direcotry entries, files) κάθε φορά που ο κώδικας περνάει από το κάθε σημείο και μπορεί να επιφέρει κάποια αλλαγή. Έχουμε επεξεργαστεί τα παρακάτω αρχεία για το άνοιγμα, κλείσιμο του αρχείου και καταγραφή: **fat.h** (προσθέσαμε μία δική μας συνάρτηση που θα καλείται να γράφει στο αρχείο κάθε φορά μία αλλαγή), **inode.c, fatent.c, cache.c, dir.c.** Παρακάτω θα δείξουμε και θα εξηγήσουμε τις προσθήκες μας:

Αρχείο fat.h που περιέχει συναρτήσεις που αφορούν το FAT:

Προσθέτουμε στη γραμμή 286 την συνάρτησή μας (εκεί τελειώνουν οι συναρτήσεις που υλοποιούνται μέσα στο fat.h και αφορούν γενικά το superblock ή το FAT) και από κάτω ακολουθούν συναρτήσεις που υλοποιούνται σε κάθε ένα από τα αρχεία του FAT. Η συνάρτηση είναι η εξής:

```
static inline void mywritetofile(const struct msdos_sb_info *sbi, char *filepr, int copysize){
        printk(KERN_INFO "We save from file: %s ", filepr); //Typwsh sthn othoni
        sys_write(sbi->fop, filepr, copysize); //Grafoume sto arxeio
        sys_fsync(sbi->fop); //flush kai eggrafi sto disko
        memset(filepr,0,sizeof(filepr));//Adeiazei ton pinaka.
}
```

Η **mywritetofile** είναι μία συνάρτηση που γράφει στο ήδη ανοιχτό αρχείο έναν pointer (***filepr από το fileprinter**) για μέγεθος **copysize**. Το όρισμα **const struct msdos_sb_info *sbi** (το είδος εξαρτάται από που ανοίγει το αρχείο μας) έχει αυτή τη μορφή διότι, κάθε φορά που την καλούμε θέλουμε για το συγκεκριμένο **sbi** που θα είναι ανοιχτός ο file opener μας. Χωρίς το **const,** το όρισμα κάθε φορά θα ζητούσε ένα νέο sbi που δεν σχετίζεται με τον file opener. Οπότε δεν εκτύπωνε τίποτα.

Ο τύπος static inline void χρησιμοποιήθηκε όπως τις συναρτήσεις που ήδη υπήρχαν στο αρχείο. Επειδή θέλουμε η συνάρτηση να είναι ορατή σε κάθε αρχείο που καταγράφουμε μία αλλαγή (και το κοινό όλων

των αρχείων που επεξεργαζόμαστε είναι ότι κάνουν include το fat.h) την κατασκευάζουμε έτσι, όπως δηλαδή είναι λειτουργικές και ορατές οι υπόλοιπες συναρτήσεις εκεί μέσα. Οι λειτουργίες που εξυπηρετεί είναι:

- Εμφανίζει στην οθόνη τι αποθηκεύεται στο αρχείο στη μορφή:

  We save from file: path_filename(struct που ανήκει το πεδίο) =>
  Changed: field = value

- Γράφει στο αρχείο με την sys_write(όνομα_fileopener, pointer, μήκος).

- Κάνει flush τη μνήμη που είναι προσωρινά γραμμένο το μήνυμα και γράφει στο δίσκο την αλλαγή. Ο λόγος που δεν κάνουμε και fdatasync() είναι διότι και οι δύο αυτές συναρτήσεις κάνουν την ίδια δουλειά. Όμως η fdatasync() είναι λιγότερο αξιόπιστη σε θέμα ενημέρωσης του χρόνου καταγραφής, αλλά αυτομάτως την κάνει πιο γρήγορη διότι σε αντίθεση με την fsync() δεν περνάει το χρόνο ενημέρωσης στο δίσκο. Πλέον, στα πιο σύγχρονα όμως συστήματα κάνουν ακριβώς την ίδια δουλειά. (Στο μάθημα είχε αναφερθεί ότι μπορούμε να χρησιμοποιήσουμε και τις 2 ταυτόχρονα διότι δεν μας απασχολεί τόσο το τεχνικό κομμάτι της διαδικασίας αυτής, καθώς ότι θα μπορούσε να γίνει μια φόρα πριν κλείσει το αρχείο. Δοκιμάσαμε όλες τις περιπτώσεις και είδαμε ότι το αποτέλεσμα παραμένει το ίδιο)

- Αντιγράφει την τιμή 0 στον πίνακα – pointer (μπορεί η memset να το περνάει σαν integer αλλά αποθηκεύει την unsigned char τιμή του) που χρησιμοποιούμε έτσι ώστε αν κληθεί ξανά η mywritetofile στην ίδια συνάρτηση ο πίνακας θα είναι γεμάτος 0,

κάνοντας reset το περιεχόμενό του. (δεν χρειάζεται σε όλες τις περιπτώσεις που καλείται αλλά δεν μας επηρεάζει κιόλας).

- Καλούμε τη βιβλιοθήκη <linux/syscalls.h> (γραμμή 9) για να αναγνωριστεί η εντολή sys_write().

Αρχείο inode.c, άνοιγμα του αρχείου , αρχικοποίηση του fop:

Όπως περιγράψαμε παραπάνω από την παρατήρηση στη σελ. 13-14 για το fat_fill_super() , είδαμε ότι εκεί γίνονται αρχικοποιήσεις του superblock. Η ιδέα είναι ότι, εφόσον το superblock είναι το "πρώτο" πράγμα που δημιουργείται και αποτελεί ένα "υπερσύνολο" που περιέχει τις υπόλοιπες λειτουργίες του FAT. Είμαστε πλέον ήδη σε κώδικα που τρέχει στον πυρήνα (kernel). Μπορούμε λοιπόν να πούμε ότι είναι ένα ασφαλές σημείο να δημιουργηθεί το αρχείο. Επίσης σύμφωνα με το σχόλιο στη γραμμή 1664: "since we're only just about to mount it", μιλώντας για το FAT, σημαίνει ότι ο κώδικας σε αυτό το σημείο είναι λίγο πριν το mount. Άρα μπορούμε να προχωρήσουμε στη δημιουργία του αρχείου. Αρχικά ορίζουμε στο header file (fat.h) μία μεταβλητή σαν πεδίο του msdos_sb_info, εφόσον αυτό αφορά το superblock:

int fop; /* anafora sto arxeio katagrafis fop (File OPener) */

Είναι ο ίδιος file opener (ή file descriptor πιο σωστά) που αναφέραμε παραπάνω στην συνάρτηση mywritetofile. Δηλώνουμε στην αρχή τη μεταβλητή char *save που θα δοθεί σαν όρισμα στην sys_write και περιέχει το όνομα – διαδρομή για το αρχείο:

char *save ="journal.txt"; //path - onoma arxeiou gia pou dinoume sthn sys_open san orisma.

Έπειτα, στις γραμμές 1752-1757 γίνεται το άνοιγμα του αρχείου (μετά την αρχικοποίηση των πεδίων του msdos_sb_info):

```
printk(KERN_INFO "Creating our journal...");

sbi->fop = sys_open(save, O_RDWR | O_APPEND | O_CREAT, S_IRUSR |  S_IWUSR); //Dhmiourgia arxeiou katagrafis me ta apraithta dikaiwmata.


if(sbi->fop >= 0){ // Gia na doume oti to arxeio ontws anoigei.

        printk(KERN_INFO "File opened.");

}
```

- Εμφανίζουμε ένα συμβολικό μήνυμα για τη δημιουργία.
- Αρχικοποιούμε τον file descriptor (file opener) ώστε να ανοίγει το αρχείο. **Save** είναι το char πεδίο που αναφέραμε πριν, **O_RDWR | O_APPEND | O_CREAT** είναι τι επιλογές επεξεργασίας θέλουμε να έχει το αρχείο όταν ανοίγει (read write, append - γράψιμο στο τέλος, create – δημιουργία αν δεν υπάρχει ήδη το αρχείο) και **S_IRUSR | S_IWUSR** τα δικαιώματα χρήστη. (Το flag και mode είναι από τις φροντιστηριακές διαφάνειες Τ1 σελ49-50)
- Τέλος, κάνουμε έναν απλό έλεγχο ότι το αρχείο άνοιξε και μπορούμε να γράψουμε.

Εφόσον το αρχείο είναι πλέον ανοιχτό, μπορούμε να ξεκινήσουμε τη διαδικασία της καταγραφής.

Τι σημαίνει καταγραφή?

Για κάθε μία από τις αρχικοποιήσεις που συνέβαιναν στα struct που αναλύσαμε παραπάνω (sbi, ei ή i, sinfo ή old_sinfo, file ή filp) ψάχνουμε να δούμε που υπάρχουν οι μεταβλητές αρχικοποίησης του struct μαζί με ένα πεδίο της δομής τους. (πχ στη μορφή μεταβλητή→πεδίο)

Πως κάνουμε την καταγραφή από τον κώδικα σε αρχείο?

Για αυτή τη δουλειά πειραματιστήκαμε με τις εντολές πυρήνα kzalloc() και kfree() για τη δέσμευση και αποδέσμευση χώρου του fileprinter. Ουσιαστικά, δηλώνουμε τον βοηθητικό pointer – πίνακα στον οποίο θα περνάμε το μήνυμα μας και θα το εκτυπώνουμε στην οθόνη και θα γράφουμε στο αρχείο. Η διαδικασία γίνεται ως εξής, αρχικά:

char *fileprinter; //Voithitikos pinakas.

fileprinter = kzalloc(CPSIZE, GFP_NOWAIT); //Desmeysi xwrou.

- Το **CPSIZE** (copysize) είναι ένα πεδίο το οποίο γίνεται define στην αρχή του **fat.h** (εφόσον είναι το header file, μπαίνει εκεί σαν define, διότι η **kzalloc()** χρειάζεται την τιμή από τον preprocessor). Το **CPSIZE** είναι μία μεταβλητή ίση με 1024, συμβολικά στο μέγεθος 1Kb. Οπότε στην **kzalloc()** δεσμεύεται χώρος 1024bytes γεμάτος μηδενικά. Το μέγεθος 1024 μας βολεύει για δυο λόγους: ξέρουμε ακριβώς τη μνήμη που δεσμεύεται από την **kzalloc()** και μας βοηθάει αργότερα στον έλεγχο για τη σωστή εγγραφή στο αρχείο.

- Το όρισμα **GFP_NOWAIT** σημαίνει ότι ή διαδικασία της δέσμευσης δεν θα σταματήσει από τίποτα. (θα επισυνάψουμε και την πηγή που αναφέρει “Allocation will not sleep”).

Έτσι λοιπόν έχουμε αρχικοποιήσει τον πίνακα με 1024 θέσεις (αρκετές για να καταγράψουμε ένα οποιοδήποτε μήνυμα). Οπότε , μόλις συναντήσουμε μία αλλαγή κάνουμε (παράδειγμα):

```
1. sbi->fsinfo_sector = 1;

2. //Kratame tis allages gia eggrafh sto arxeio

3. sprintf(fileprinter, "fs/fat/inode.c => Changed(msdos_sb_info):
       fsinfo_sector = %ld",sbi->fsinfo_sector);

4. mywritetofile(sbi, fileprinter, CPSIZE);
```

Αρχικά μέσω της **sprintf** (γρ. 3) περνάμε μέσα στον **fileprinter** το μήνυμα που επιθυμούμε , ότι και αν αυτό περιέχει. Ύστερα, τον δίνουμε σαν όρισμα στην **mywritetofile** (γρ. 4) για να κάνει την εκτύπωση και εγγραφή στο αρχείο.

Σημείωση: Η γραμμή 1 υπήρχε ήδη στον κώδικα και την καταγράφουμε με την διαδικασία που περιγράψαμε. Στο εσωτερικό της mywritetofile έχουμε την εντολή:

```
sys_write(sbi->fop, filepr, copysize);
```

Ο λόγος που το 3ο όρισμα δεν είναι στο μέγεθος του filepr (δηλαδή αντί να βάλουμε sizeof(filepr)) είναι διότι αυτό επέστρεφε την τιμή 8, δηλαδή την τιμή μίας char μεταβλητής σε x64 σύστημα (σαν το Debian της άσκησης). Οπότε , το βάλαμε απλά σε CPSIZE κάθε φορά που το καλούμε μίας και δεν παίζει τόσο ρόλο στην καταγραφή. (Παίζει ρόλο διότι αν μπορούσαμε να ανοίξουμε και να δούμε το αρχείο θα

βλέπαμε το μήνυμά μας να ακολουθείται από μηδενικά, απλά προς το παρόν δεν μας απασχολεί)

Τέλος, μόλις τελειώσουν οι καταγραφές στη συνάρτηση που είμαστε, κάνουμε **kfree(fileprinter)** για την αποδέσμευση. Αν όμως, μέσα ***στο ίδιο αρχείο, ή και σε επόμενο*** συναντήσουμε αλλαγές που πρέπει να καταγραφούν, τότε επαναλαμβάνουμε τη διαδικασία:

1) Δήλωση **char *fileprinter** και δέσμευση μεγέθους του με **kzalloc()**.

2) Εντοπίζουμε την αλλαγή ή αρχικοποίηση (με **search(ctrl+f)** στο αρχείο για κάθε ένα από τα sbi, ei ή i, sinfo ή old_sinfo, file ή filp με το σύμβολο → για να το βρίσκουμε πιο εύκολα) και τοποθετούμε εκεί που ακριβώς συμβαίνει την **sprintf()** και καλούμε την **mywritetofile()**.

3) Αποδέσμευση χώρου με **kfree()**.

Πως ξέρουμε ότι δουλεύει σωστά?

Δοκιμαστικά, στην αρχή της υλοποίησής μας, δεν είχαμε συνάρτηση και απλά για τις πρώτες 3 καταγραφές που κάναμε είχαμε όλες τις γραμμές που εκτυπώνουν και γράφουν κάτω από κάθε αλλαγή που συναντούσαμε. Είδαμε λοιπόν, ότι παίρνουμε τα ίδια αποτελέσματα εφόσον κάναμε και τη συνάρτηση και τα εκτυπώναμε μαζί. Έτσι για να είναι και πιο εύκολο στην ανάγνωση αλλά και σωστό προγραμματιστικά, κάναμε τη συνάρτηση.

Ακολουθούμε την ίδια τακτική για κάθε συνάρτηση στο ίδιο αρχείο ή και διαφορετικά αρχεία που μελετάμε (στην περίπτωσή μας 4 αρχεία: inode.c, cache.c, fatent.c, dir.c).

Οι συνολικές καταγραφές που κάνουμε είναι οι εξής:

Υπόλοιπα αρχεία και οι καταγρεφές τους:

- inode.c: Συνάρτηση:
  - static void init_once()
    - γραμμές πεδίων καταγραφής: 793, 794
  - int fat_fill_super(): γραμμές→ 1737-1746 (αρχικοποίηση πεδίων msdos_sb_info), 1775-1777, 1780, 1788, 1810, 1815, 1816,1828 και 1830, 1835, 1836, 1838, 1839, 1854, 1867, 1875 και 1877, 1893, 1901, 1907, 1914, 1934, 1947, 1965, 1977.
- cache.c: Συνάρτηση:
  - static void __fat_cache_inval_inode(): γραμμές→ 201, 210, 216.
- dir.c: Συνάρτηση:
  - int fat_search_long(): γραμμές→ 532-536.
  - static int fat_ioctl_readdir(): γραμμές→ 794.
  - int fat_scan(): γραμμές→ 987, 988, 998-1000.
  - int fat_scan_logstart(): γραμμές→ 1027,1028, 1038-1040
  - int fat_remove_entries(): γραμμές→ 1104, 1106, 1108.
  - int fat_add_entries(): γραμμές→ 1355, 1464-1467.
- fatent.c: Συνάρτηση:
  - void fat_ent_access_init(): 350,351, 354,355, 358,359.
  - int fat_alloc_clusters(): 594,602, 627,628.
  - int fat_free_clusters(): 704.

- int fat_count_free_clusters(): 804, 805.

Εδώ να προσθέσουμε ότι: Μερικές μεταβλητές που ήταν τύπου struct,τις βρήκαμε μέσω του lkl_dox και φυσικά όσων καταφέραμε να βρούμε τη σωστή τιμή (σε όσες δεν βρήκαμε τη σωστή τιμή εκτυπώνεται σε παρένθεση ποια τιμή έχει) τις παραθέτουμε εδώ:

Γραμμή που εκτυπώνονται:

- inode.c:
  - 1969: sbi→fat_inode→i_sb→s_count (άλλη τιμή)
  - 1980: sbi→fsinfo_inode→i_ino
- dir.c:
  - 540: sinfo→de→name και sinfo→bh→b_state (άλλη τιμή σε όλες τις αναφερόμενες γραμμές)
  - 1111: sinfo→de→name
  - 1470: sinfo→de→name και sinfo→bh→b_state

**Σημείωση: Στο αρχείο fatent.c υπάρχει σε σχόλια ένα έλεγχος που δοκιμάζαμε αν το αρχείο γράφει σωστά και είναι πολυ αρχικού σταδίου, πριν καν βρούμε που θα γίνει sys_close() κτλπ.

Αρχείο inode.c, κλείσιμο του αρχείου, εκτύπωση αποτελεσμάτων:

Το κλείσιμο του αρχείου γίνεται στη συνάρτηση fat_put_super(). Στην αρχή είχαμε κάνει την παρατήρηση ότι είναι ουσιαστικά η "τελευταία" συνάρτηση που καλείται πριν τον τερματισμό. (Φυσικά δεν είναι γιατί κυριολεκτικά μιλώντας, είναι άλλη). Είδαμε, ότι καλεί συναρτήσεις που καταστρέφουν τα inodes, οπότε για αυτό την ονομάσαμε τελευταία.

```
1. static void fat_put_super(struct super_block *sb)
2. {
3.     struct msdos_sb_info *sbi = MSDOS_SB(sb);
4.     int ban = 0; //metavliti test gia to lseek
5.     printk(KERN_INFO "###fs/fat/inode.c - INSIDE FAT_PUT_SUPER###");
6.     fat_set_state(sb, 0, 0);
7.     iput(sbi->fsinfo_inode);
8.     iput(sbi->fat_inode);
9.     call_rcu(&sbi->rcu, delayed_free);
10.    /*to parakatw prokeitai gia enan elegxo gia an diapistwsoume oti to arxeio grafetai mexri telous, kleinei, periexei ta swsta bytes.
11.    To lseek epistrfei swsta bytes pou exoun graftei me thn ektelesh ths cptofs sto termatiko, to arxeio omws den einai emfanes kapou.
```

12. Epishs, an to arxeio den anoiksei apo th stigmh pou klithei o file opener, epistrefei -9 kai den grafetai tipota. Opote me to kleisimo tou arxeioy,

13. h 2h lseek pou kaleitai epistrefei -9 , dhladh den yparxoun bytes na diavastoun apo kapou. H fclose epistrefei epishs -9, dhladh thn nea timh pou exei o file opener

14. efoson ton apodesmeysame apo th leitoyrgia. */

15. //----------

16. ban = sys_lseek(sbi->fop,0,SEEK_CUR);

17. printk(KERN_INFO "------ FILE DETAILS ------ \n");

18. printk(KERN_INFO "BEFORE CLOSE:file bytes = %d, file des value = %d \n", ban, sbi->fop);

19. sys_close(sbi->fop);

20. ban = sys_lseek(sbi->fop,0,SEEK_CUR);

21. printk(KERN_INFO "AFTER CLOSE:file bytes = %d, sys_close return value = %ld", ban, sys_close(sbi->fop));

22. printk(KERN_INFO "---------- \n");

23. //----------*/

24.}

Στις γραμμές 6-9 είναι οι λειτουργίες που είχες εξ αρχής η συνάρτηση.

Αυτές με τη σειρά του καλούν τις υπόλοιπες συναρτήσεις υπεύθυνες για τον τερματισμό της καταγραφής. Εμείς προσθέτουμε τις γραμμές 10-23.

Αρχικά, εμφανίζουμε τι υπάρχει μέσα στο αρχείο από τις καταγραφές που έχουμε κάνει καθώς και την τιμή του file opener (file descriptor). Κλείνουμε το αρχείο και εκτυπώνουμε πάλι το μέγεθος του αρχείου και την τιμή της sys_close. Ύστερα, βλέπουμε την εκτύπωση το αποτελέσματος στο τερματικό της διαδικασίας που περιγράψαμε παραπάνω:

[0.013057] Creating our journal...

[0.013080] File opened.

[0.013084] We save from file: fs/fat/inode.c => Initialized(msdos_sb_info): cluster_size = 2048, cluster_bits = 11, sec_per_clus = 4

[0.013084] fats = 2, fat_bits = 0, fat_start = 4

[0.013084] fat_length = 200, root_cluster = 0, free_clusters = -1

[0.013084] free_clus_valid = 0, prev_free = 2

[0.013084]

[0.013086] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013094] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): vol_id = 3945486744

[0.013094]

[0.013095] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013099] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): dir_per_block = 16, dir_per_block_bits = 4, dir_start = 404, dir_entries = 512

[0.013099]

[0.013100] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013103] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): data_start = 436

[0.013103]

[0.013104] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013107] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): fat_bits = 16

[0.013107]

[0.013108] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013112] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): dirty = 0

[0.013112]

[0.013114] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013117] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): max_cluster = 51093

[0.013117]

[0.013120] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013123] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): free_clusters = -1

[0.013123]

[0.013125] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013127] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): prev_free = 2

[0.013127]

[0.013129] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013133] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): prev_free = 2

[0.013133]

[0.013134] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013138] We save from file: fs/fat/fatent.c(msdos_sb_info) => Changed: fatent_shift = 1, fatent_ops = 94308020330080

[0.013138]

[0.013139] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013143] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): nls_disk = cp437

[0.013143]

[0.013144] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013147] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): nls_io = iso8859-1

[0.013147]

[0.013149] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013152] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###

[0.013155] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013155]

[0.013157] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013160] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013160]

[0.013161] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013164] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013164]

[0.013166] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013169] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013169]

[0.013170] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013174] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013174]

[0.013176] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013179] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013179]

[0.013180] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013184] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013184]

[0.013185] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013188] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013188]

[0.013189] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013194] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013194]

[0.013195] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013198] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013198]

[0.013199] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013203] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013203]

[0.013204] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013207] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013207]

[0.013208] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013213] We save from file: fs/fat/inode.c(msdos_inode_info) => Changed: nr_caches = 0, cache_valid_id = 1

[0.013213]

[0.013214] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013217] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): fat_inode (count) = 1

[0.013217]

[0.013218] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013221] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###

[0.013222] We save from file: fs/fat/inode.c => Changed(msdos_sb_info): fsinfo_inode = 2

[0.013222]

[0.013225] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013228] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###

[0.013493] ###fs/fat/namei_vfat.c - INSIDE VFAT_LOOKUP###

[0.013505] ###fs/fat/namei_vfat.c - INSIDE VFAT_CREATE###

[0.013512] We save from file: fs/fat/dir.c(fat_slot_info) => Changed: slot_off = 0, bh = (null)

[0.013512]

[0.013514] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013527] We save from file: fs/fat/dir.c(fat_slot_info) => Changed: nr_slots = 2

[0.013527]

[0.013528] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013532] We save from file: fs/fat/dir.c(fat_slot_info) => Changed: slot_off = 0, de = TEST1 TXT bh = 43, i_pos = 6465

[0.013532]

[0.013534] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013537] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###

[0.013713] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013718] ###fs/fat/inode.c - INSIDE FAT_WRITE_BEGIN###

[0.013723] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###

[0.013724] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###

[0.013737] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###

[0.013738] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###

[0.013740] ###fs/fat/fatent.c - INSIDE FAT16_ENT_PUT###

[0.013742] We save from file: fs/fat/fatent.c(msdos_sb_info) => Changed: prev_free = 3

[0.013742]

[0.013743] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013751] ###fs/fat/inode.c - INSIDE FAT_WRITE_END###

[0.013758] ###fs/fat/file.c - INSIDE FAT_FILE_RELEASE###

[0.013902] ###fs/fat/inode.c - INSIDE FAT_WRITE_INODE###

[0.013912] ###fs/fat/inode.c - INSIDE FAT_WRITE_INODE###

[0.013962] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###

[0.013966] We save from file: fs/fat/cache.c(msdos_inode_info) => Changed: cache_valid_id = 2

[0.013966]

[0.013968] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013973] We save from file: fs/fat/cache.c(msdos_inode_info) => Changed: cache_valid_id = 2

[0.013973]

[0.013974] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013977] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###

[0.013979] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###

[0.013982] We save from file: fs/fat/cache.c(msdos_inode_info) => Changed: cache_valid_id = 2

[0.013982]

[0.013984] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013986] We save from file: fs/fat/cache.c(msdos_inode_info) => Changed: cache_valid_id = 2

[0.013986]

[0.013988] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.013990] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###

[0.013992] ###fs/fat/inode.c - INSIDE FAT_PUT_SUPER###

[0.013999] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###

[0.014001] We save from file: fs/fat/cache.c(msdos_inode_info) => Changed: cache_valid_id = 2

[0.014001]

[0.014002] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.014006] We save from file: fs/fat/cache.c(msdos_inode_info) => Changed: cache_valid_id = 2

[0.014006]

[0.014008] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.014010] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###

[0.014011] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###

[0.014013] We save from file: fs/fat/cache.c(msdos_inode_info) => Changed: cache_valid_id = 2

[0.014013]

[0.014015] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.014017] We save from file: fs/fat/cache.c(msdos_inode_info) => Changed: cache_valid_id = 2

[0.014017]

[0.014019] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###

[0.014021] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###

[0.014023] ###fs/read-write.c - INSIDE GENERIC_FILE_LLSEEK###

[0.014024] ------ FILE DETAILS ------

[0.014027] BEFORE CLOSE:file bytes = 40960, file des value = 0

[0.014028] AFTER CLOSE:file bytes = -9, sys_close return value = -9

[0.014029] -------------------------

[0.014148] reboot: Restarting system

Μετράμε συνολικά 40 εγγραφές (οι γραμμές με τους έντονους χαρακτήρες) με το τρέχον παράδειγμά μας για ./cptofs -i /tmp/vfatfile -p -t vfat test1.txt / .

Στις τελευταίες 4 γραμμές βλέπουμε τα στοιχεία του αρχείου. 40960 Bytes γράφτηκαν στο αρχείο. Αν κάνουμε 40960/40 = 1024 , δηλαδή το σωστό CPSIZE που είχαμε ορίσει. (εδώ αναφερόμασταν ότι το CPSIZE θα βοηθήσει και στο έλεγχο).

Τέλος θα θέλαμε να εκφράσουμε κάποιες ιδέες, που δεν βρήκαμε πως υλοποιούνται αλλά πως σκεφτήκαμε:

- Το αρχείο μας δεν εμφανίζεται σε κανένα φάκελο και ξέρουμε ότι φταίει σε πρώτη φάση το path που δίνουμε. Ίσως να φταίει και το που γίνεται το άνοιγμα και το κλείσιμο, προσπαθήσαμε το πάμε όσο πιο κοντά γίνεται με τη δική μας λογική.

- Με πολλές δοκιμές είδαμε τα pahts που επέστρεφε η cptofs και σαν destination έδινε /mount/και στη συνέχεια ενα 16αδικό που είδαμε πως κατασκευάζεται μέσα στη cptofs. Το source path ήταν αυτό της tools/lkl που έτρεχε η cptofs. Σκεφτήκαμε αν μπορούσαμε να πάρουμε το destination path για χρήση αλλά μετά σκεφτήκαμε οτι για να γίνει χρήση του θα έπρεπε να είναι ήδη mounted το vfatfile.
- Κάναμε καταγραφή μόνο των πεδίων που μας οδήγησαν οι διαφάνειες και δε προλάβαμε δυστυχώς να δούμε περισσότερες διότι η έρευνα πάνω στην άσκηση πήρε πάρα πολλές μέρες.
- Παρατηρήσαμε ότι τελικά δεν έχει σημασία ο τύπος του αχείου, παρα μόνο το μέγεθός του.
- Τέλος θα επισυνάψουμε τις πηγές που μάθαμε πληροφορίες για τεχνικά κομμάτια της άσκησης:
  - Έλεγχος τιμών επιστροφής της fclose() - https://stackoverflow.com/questions/19056309/not-checking-closes-return-value-how-serious-really
  - Σύγκριση fsync και fdatasync - https://www.informit.com/articles/article.aspx?p=23618&seqNum=5
  - Χρήση συναρτήσεων πυρήνα και ορισμάτων- https://www.kernel.org/doc/htmldocs/kernel-api/API-kmalloc.html

- Memset και εξήγηση - https://www.tutorialspoint.com/c_standard_library/c_function_memset.htm

Ευχαριστούμε για το χρόνο σας και καλη διόρθωση! Παρακάτω θα αφήσουμε μερικά screenshot από πράγματα που ήδη εξηγήσαμε.

1) Τα printk() του 1ου ερωτήματος.

```
[    0.011065] ###mm/filemap.c - INSIDE GENERIC_FILE_READ_ITER###
[    0.011280] Creating our journal...
[    0.011293] File opened.
[    0.011296] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###
[    0.011304] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###
[    0.011305] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###
[    0.011560] ###fs/fat/namei_vfat.c - INSIDE VFAT_LOOKUP###
[    0.011569] ###fs/fat/namei_vfat.c - INSIDE VFAT_CREATE###
[    0.011581] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###
[    0.014652] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###
[    0.014658] ###fs/fat/inode.c - INSIDE FAT_WRITE_BEGIN###
[    0.014663] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###
[    0.014664] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###
[    0.014679] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###
[    0.014680] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###
[    0.014682] ###fs/fat/fatent.c - INSIDE FAT16_ENT_PUT###
[    0.014687] ###fs/fat/inode.c - INSIDE FAT_WRITE_END###
[    0.014694] ###fs/fat/file.c - INSIDE FAT_FILE_RELEASE###
[    0.014831] ###fs/fat/inode.c - INSIDE FAT_WRITE_INODE###
[    0.014840] ###fs/fat/inode.c - INSIDE FAT_WRITE_INODE###
[    0.014894] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###
[    0.014898] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###
[    0.014899] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###
[    0.014903] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###
[    0.014904] ###fs/fat/inode.c - INSIDE FAT_PUT_SUPER###
[    0.014911] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###
[    0.014913] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###
[    0.014915] ###fs/fat/inode.c - INSIDE FAT_EVICT_INODE###
[    0.014919] ###fs/fat/inode.c - INSIDE FAT_DESTROY_INODE###
[    0.015023] reboot: Restarting system
root@myy601lab2:/home/myy601/lkl/lkl-source/tools/lkl#
```

2) Πόσες παραπάνω κλήσεις συναρτήσεων (και αυτές οι κλήσεις ήταν πάνω από 4 φορές) για το μεγάλο αρχείο test2.txt.

```
[    0.014458] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###
[    0.014466] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###
[    0.014467] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###
[    0.014680] ###fs/fat/namei_vfat.c - INSIDE VFAT_LOOKUP###
[    0.014690] ###fs/fat/namei_vfat.c - INSIDE VFAT_CREATE###
[    0.014704] ###fs/fat/inode.c - INSIDE FAT_ALLOC_INODE###
[    0.022212] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###
[    0.022221] ###fs/fat/inode.c - INSIDE FAT_WRITE_BEGIN###
[    0.022228] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###
[    0.022230] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###
[    0.022248] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###
[    0.022250] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###
[    0.022252] ###fs/fat/fatent.c - INSIDE FAT16_ENT_PUT###
[    0.022257] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###
[    0.022259] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###
[    0.022260] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###
[    0.022261] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###
[    0.022263] ###fs/fat/fatent.c - INSIDE FAT16_ENT_PUT###
[    0.022264] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###
[    0.022266] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###
[    0.022267] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###
[    0.022268] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###
[    0.022270] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###
[    0.022271] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###
[    0.022272] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###
[    0.022278] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###
[    0.022279] ###fs/fat/fatent.c - INSIDE FAT16_ENT_PUT###
[    0.022281] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###
[    0.022282] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###
[    0.022284] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###
[    0.022285] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###
[    0.022290] ###fs/fat/inode.c - INSIDE FAT_WRITE_END###
[    0.022362] ###mm/filemap.c - INSIDE GENERIC_FILE_WRITE_ITER###
[    0.022366] ###fs/fat/inode.c - INSIDE FAT_WRITE_BEGIN###
[    0.022377] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###
[    0.022378] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###
[    0.022380] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###
[    0.022381] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###
[    0.022383] ###fs/fat/fatent.c - INSIDE FAT16_ENT_PUT###
[    0.022385] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###
[    0.022386] ###fs/fat/fatent.c - INSIDE FAT_ENT_BREAD###
[    0.022387] ###fs/fat/fatent.c - INSIDE FAT16_ENT_SET_PTR###
[    0.022388] ###fs/fat/fatent.c - INSIDE FAT16_ENT_GET###
[    0.022390] ###fs/fat/fatent.c - INSIDE FAT_ENT_BLOCKNR###
```