Λειτουργικά 2021

Χαρίσης Αλένξανδρος 3361

Μερόλλι Αγγέλε 3278

Περιεχόμενα:

- Τρόπος καταγραφής
- Διάγνωση συστήματος
- Superblock
- Inode
- Directory entry
- File allocation table
- Καταγραφή των αλλαγών σε αρχείο

Τρόπος καταγραφής:

Σίγουρα θα μπορούσαμε να καταγράψουμε κάθε αλλαγή που συμβαίνει σε μια δομή με λεπτομέρεια, έως και στη μικρότερη μέθοδο όπου αλλάζει. Πχ όπως:

το **dir_ops** του **struct msdos_sb_info** αλλάζει στη **super()** των **namei_msdos.c** και **namei_vfat.c** ή το **mmu_private** του **struct msdos_inode_info** αλλάζει στη λειτουργία **fat_evict_inode()** στο **inode.c** μέσω της **fat_truncate_blocks()** που καλεί.

Ωστόσο, πιστεύουμε ότι δεν είναι αυτός ο σκοπός της άσκησης οπότε εστιάσαμε σε αλλαγές που γίνονται σε συγκεκριμμένα *"κλειδιά"* μέρη του κώδικα που επιτρέπουν και την ευκολότερη καταγραφή τους. Θα δείτε αργότερα σε λεπτομέρεια τι εννοούμε.

Διάγνωση Συστήματος

Για αρχή κοιτάμε τα structs που μας έχετε δώσει και πάμε σε κάθε λειτουργία από αυτές και βάζουμε ένα *printk* για να δούμε πως τρέχει το πρόγραμμα.

Για παράδειγμα:

Στο **struct super_operations fat_sops** του **fs/fat/inode.c** έχουμε:

```
static const struct super_operations fat_sops = {
    .alloc_inode    = fat_alloc_inode,
    .destroy_inode  = fat_destroy_inode,
    .write_inode    = fat_write_inode,
    .evict_inode    = fat_evict_inode,
    .put_super  = fat_put_super,
    .statfs     = fat_statfs,
    .remount_fs = fat_remount,

    .show_options   = fat_show_options,
};
```

Βρήκαμε τη κάθε μέθοδο και βάλαμε ένα *printk* που να προσδιορίζει τη κάθε λειτουργία.

Το ίδιο κάναμε για τα υπόλοιπα struct που μας έχουν δωθεί, ψάχνοντας με **~/bin/search** τη κάθε μέθοδο και το που βρίσκεται.

Τρέχοντας ύστερα την εντολή **./cptofs -l /tmp/vfatfile -p -t vfat lklfuse.c /** παίρνουμε το παρακάτω αποτέλεσμα.

Σκοπός μας είναι να αναγνωρίσουμε για αρχή πως τρέχει το πρόγραμμα.

```
[    0.024995] File op: generic_file_read_iter
[    0.025706] Superblock: fat_alloc_inode
[    0.025714] Superblock: fat_alloc_inode
[    0.025716] Superblock: fat_alloc_inode
[    0.026699] Dir: vfat_lookup
[    0.026710] Superblock: fat_alloc_inode
[    0.026715] Inode: fat_setattr
[    0.026719] Fat op: fat_ent_blocknr
[    0.026721] Fat op: fat_ent_bread
[    0.026734] Fat op: fat16_ent_set_ptr
[    0.026736] Fat op: fat16_ent_get
[    0.026737] Fat op: fat16_ent_put
[    0.026739] Fat op: fat_ent_blocknr
[    0.026754] Fat op: fat16_ent_set_ptr
[    0.026755] Fat op: fat16_ent_get
[    0.026756] Fat op: fat16_ent_put
[    0.026757] Fat op: fat_ent_blocknr
[    0.026759] Fat op: fat16_ent_set_ptr
[    0.026760] Fat op: fat16_ent_get
[    0.026770] Fat op: fat16_ent_put
[    0.026771] Fat op: fat_ent_blocknr
[    0.026773] Fat op: fat16_ent_set_ptr
[    0.026815] Fat op: fat16_ent_get
[    0.026844] Fat op: fat16_ent_put
[    0.026845] Fat op: fat_ent_blocknr
[    0.026847] Fat op: fat16_ent_set_ptr
[    0.026848] Fat op: fat16_ent_get
[    0.026849] Fat op: fat16_ent_put
[    0.026851] Fat op: fat_ent_blocknr
[    0.026852] Fat op: fat16_ent_set_ptr
[    0.026853] Fat op: fat16_ent_get
[    0.026854] Fat op: fat16_ent_put
[    0.026856] Fat op: fat_ent_blocknr
[    0.026857] Fat op: fat16_ent_set_ptr
[    0.026859] Fat op: fat16_ent_get
[    0.026860] Fat op: fat16_ent_put
[    0.026875] File op: generic_file_write_iter
[    0.026878] Inode: fat_write_begin
[    0.026885] Fat op: fat_ent_blocknr
[    0.026886] Fat op: fat_ent_bread
[    0.026888] Fat op: fat16_ent_set_ptr
```

```
[    0.026898] Fat op: fat16_ent_get
[    0.026899] Fat op: fat16_ent_put
[    0.026902] Fat op: fat_ent_blocknr
[    0.026904] Fat op: fat_ent_bread
[    0.026905] Fat op: fat16_ent_set_ptr
[    0.026906] Fat op: fat16_ent_get
[    0.026908] Fat op: fat16_ent_put
[    0.026909] Fat op: fat_ent_blocknr
[    0.026911] Fat op: fat_ent_bread
[    0.026912] Fat op: fat16_ent_set_ptr
[    0.026913] Fat op: fat16_ent_get
[    0.026915] Fat op: fat_ent_blocknr
[    0.026916] Fat op: fat_ent_bread
[    0.026917] Fat op: fat16_ent_set_ptr
[    0.026919] Fat op: fat16_ent_get
[    0.026920] Fat op: fat16_ent_put
[    0.026922] Fat op: fat_ent_blocknr
[    0.026923] Fat op: fat_ent_bread
[    0.026949] Fat op: fat16_ent_set_ptr
[    0.026951] Fat op: fat16_ent_get
[    0.026957] Inode: fat_write_end
[    0.026964] File op: generic_file_write_iter
[    0.026966] Inode: fat_write_begin
[    0.026971] Fat op: fat_ent_blocknr
[    0.026972] Fat op: fat_ent_bread
[    0.026974] Fat op: fat16_ent_set_ptr
[    0.026975] Fat op: fat16_ent_get
[    0.026976] Fat op: fat16_ent_put
[    0.026978] Fat op: fat_ent_blocknr
[    0.026979] Fat op: fat_ent_bread
[    0.026981] Fat op: fat16_ent_set_ptr
[    0.026982] Fat op: fat16_ent_get
[    0.026984] Fat op: fat_ent_blocknr
[    0.026985] Fat op: fat_ent_bread
[    0.026986] Fat op: fat16_ent_set_ptr
[    0.026987] Fat op: fat16_ent_get
[    0.026989] Fat op: fat16_ent_put
[    0.026990] Fat op: fat_ent_blocknr
[    0.026992] Fat op: fat_ent_bread
[    0.026993] Fat op: fat16_ent_set_ptr
[    0.026994] Fat op: fat16_ent_get
```

```
[    0.026996] Fat op: fat_ent_blocknr
[    0.026997] Fat op: fat_ent_bread
[    0.026999] Fat op: fat16_ent_set_ptr
[    0.027000] Fat op: fat16_ent_get
[    0.027001] Fat op: fat16_ent_put
[    0.027003] Fat op: fat_ent_blocknr
[    0.027004] Fat op: fat_ent_bread
[    0.027005] Fat op: fat16_ent_set_ptr
[    0.027007] Fat op: fat16_ent_get
[    0.027008] Fat op: fat_ent_blocknr
[    0.027009] Fat op: fat_ent_bread
[    0.027011] Fat op: fat16_ent_set_ptr
[    0.027012] Fat op: fat16_ent_get
[    0.027013] Fat op: fat16_ent_put
[    0.027015] Fat op: fat_ent_blocknr
[    0.027016] Fat op: fat_ent_bread
[    0.027018] Fat op: fat16_ent_set_ptr
[    0.027019] Fat op: fat16_ent_get
[    0.027025] Inode: fat_write_end
[    0.027029] File op: generic_file_write_iter
[    0.027030] Inode: fat_write_begin
[    0.027033] Fat op: fat_ent_blocknr
[    0.027034] Fat op: fat_ent_bread
[    0.027036] Fat op: fat16_ent_set_ptr
[    0.027037] Fat op: fat16_ent_get
[    0.027038] Fat op: fat16_ent_put
[    0.027040] Fat op: fat_ent_blocknr
[    0.027041] Fat op: fat_ent_bread
[    0.027043] Fat op: fat16_ent_set_ptr
[    0.027044] Fat op: fat16_ent_get
[    0.027045] Fat op: fat_ent_blocknr
[    0.027047] Fat op: fat_ent_bread
[    0.027048] Fat op: fat16_ent_set_ptr
[    0.027049] Fat op: fat16_ent_get
[    0.027051] Fat op: fat16_ent_put
[    0.027052] Fat op: fat_ent_blocknr
[    0.027053] Fat op: fat_ent_bread
[    0.027055] Fat op: fat16_ent_set_ptr
[    0.027056] Fat op: fat16_ent_get
[    0.027058] Fat op: fat_ent_blocknr
[    0.027059] Fat op: fat_ent_bread
```

```
[    0.027060] Fat op: fat16_ent_set_ptr
[    0.027062] Fat op: fat16_ent_get
[    0.027063] Fat op: fat16_ent_put
[    0.027064] Fat op: fat_ent_blocknr
[    0.027066] Fat op: fat_ent_bread
[    0.027067] Fat op: fat16_ent_set_ptr
[    0.027068] Fat op: fat16_ent_get
[    0.027070] Fat op: fat_ent_blocknr
[    0.027071] Fat op: fat_ent_bread
[    0.027072] Fat op: fat16_ent_set_ptr
[    0.027074] Fat op: fat16_ent_get
[    0.027075] Fat op: fat16_ent_put
[    0.027077] Fat op: fat_ent_blocknr
[    0.027080] Fat op: fat_ent_bread
[    0.027081] Fat op: fat16_ent_set_ptr
[    0.027083] Fat op: fat16_ent_get
[    0.027087] Inode: fat_write_end
[    0.027091] File op: generic_file_write_iter
[    0.027092] Inode: fat_write_begin
[    0.027094] Fat op: fat_ent_blocknr
[    0.027096] Fat op: fat_ent_bread
[    0.027097] Fat op: fat16_ent_set_ptr
[    0.027098] Fat op: fat16_ent_get
[    0.027100] Fat op: fat16_ent_put
[    0.027101] Fat op: fat_ent_blocknr
[    0.027103] Fat op: fat_ent_bread
[    0.027104] Fat op: fat16_ent_set_ptr
[    0.027105] Fat op: fat16_ent_get
[    0.027107] Fat op: fat_ent_blocknr
[    0.027108] Fat op: fat_ent_bread
[    0.027109] Fat op: fat16_ent_set_ptr
[    0.027111] Fat op: fat16_ent_get
[    0.027112] Fat op: fat16_ent_put
[    0.027113] Fat op: fat_ent_blocknr
[    0.027115] Fat op: fat_ent_bread
[    0.027116] Fat op: fat16_ent_set_ptr
[    0.027117] Fat op: fat16_ent_get
[    0.027122] Inode: fat_write_end
[    0.027130] File op: fat_file_release
[    0.027376] Inode: fat_writepages
[    0.027391] Superblock: fat_write_inode
[    0.027801] Superblock: fat_evict_inode
```

```
[    0.027807] Superblock: fat_destroy_inode
[    0.027809] Superblock: fat_evict_inode
[    0.027830] Superblock: fat_destroy_inode
[    0.027833] Superblock: fat_put_super
[    0.027845] Superblock: fat_evict_inode
[    0.027847] Superblock: fat_destroy_inode
[    0.027848] Superblock: fat_evict_inode
[    0.027850] Superblock: fat_destroy_inode
[    0.028017] reboot: Restarting system
myy601@myy601lab2:~/lkl/lkl-source/tools/lkl$
```

Παρατηρούμε ότι υπάρχει ένα μοτίβο που επαναλαμβάνεται με τις περισσότερες εντολές να αποτελούνται από λειτουργείες εγγραφών FAT. Στην αρχή δημιουργείται το superblock και συνεχώς δημιουργούνται καινούρια inodes και cluster για τη καταγραφή. Στο τέλος όλα επανακκινούντε.

SUPERBLOCK

Επειδή θέλουμε να δημιουργήσουμε ένα αρχείο, πάμε στο struct του *superblock* **msdos_sb_info** και προσθέτουμε μία μεταβλητή int που θα είνια ο file descriptor μας.

```
// my changes
int recordFile;
```

Έπειτα βρίσκουμε ότι το superblock αρχικοποιείται στο **fat_fill_super()** στο **inode.c** .

Βάζουμε ένα *printk* για να δούμε πότε τρέχει.

```
0.023052] File op: generic_file_read_iter
0.023581] Superblock Initialization
0.023616] Superblock: fat_alloc_inode
0.023618] Superblock: fat_alloc_inode
0.023620] Superblock: fat_alloc_inode
0.024247] Dir: vfat_lookup
0.024255] Superblock: fat_alloc_inode
0.024259] Inode: fat_setattr
0.024262] Fat op: fat_ent_blocknr
0.024263] Fat op: fat_ent_bread
0.024272] Fat op: fat16_ent_set_ptr
0.024274] Fat op: fat16_ent_get
```

Βλέπουμε πως είναι από τα πρώτα που γίνονται, οπότε πάμε στο σημείο όπου αρχικοποιούνται τα περισσότερα πεδία του **msdos_sb_info** και αρχικοποιώ και το file descriptor μου:

```
sbi->recordFile = sys_open("recording file", O_RDWR | O_CREAT | O_APPEND,
 S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
```

Βάζουμε τις επιλογές read/write, να δημιουργηθεί άμα δεν υπάρχει και τα καινούρια δεδομένα να γράφονται στο τέλος του αρχείου. Επίσης δώσαμε δικαιώματα read/write σε όλους.

Προσθέτουμε ένα *printk* για να δούμε άμα άνοιξε το αρχείο.

Πάμε στο τέλος της μεθόδου αυτής, αφού έχουνε οριστεί όσα πεδία μπορούσαν να οριστούν, και τα καταγράφουμε με ένα *printk*.

```
// my changes
    printk(KERN_INFO "SUPERBLOCK INFO\n"
                     "\t\tsbi->sec_per_clus = %u\n\t\tsbi->
cluster_bits = %u\n\t\tsbi->cluster_size = %u\n"
                     "\t\tsbi->fats = %u\n\t\tsbi->fat_bits = %u\n\t\tsbi-
>fat_start = %u\n\t\tsbi->fat_length = %lu\n"
                     "\t\tsbi->dir_start = %lu\n\t\tsbi-
>dir_entries = %u\n\t\tsbi->data_start = %lu\n"
                     "\t\tsbi->max_cluster = %lu\n\t\tsbi-
>root_cluster = %lu\n\t\tsbi->fsinfo_sector = %lu\n"
                     "\t\tsbi->prev_free = %u\n\t\tsbi-
>free_clusters = %u\n\t\tsbi->free_clus_valid = %u\n"
                     "\t\tsbi->dir_ops = %p\n\t\tsbi-
>dir_per_block = %u\n\t\tsbi->dir_per_block_bits = %u\n"
                     "\t\tsbi->vol_id = %u\n\t\tsbi->dirty = %u\n\t\tsbi-
>fatent_shift = %u\n-------------------------\n",
                      sbi->sec_per_clus, sbi->cluster_bits, sbi->cluster_size,
                      sbi->fats, sbi->fat_bits, sbi->fat_start, sbi->fat_length,
                      sbi->dir_start, sbi->dir_entries, sbi->data_start,
                      sbi->max_cluster, sbi->root_cluster, sbi->fsinfo_sector,
                      sbi->prev_free, sbi->free_clusters, sbi->free_clus_valid,
                      sbi->dir_ops, sbi->dir_per_block, sbi->dir_per_block_bits,
                      sbi->vol_id,sbi->dirty, sbi->fatent_shift
                      );
```

Εμείς καταγράψαμε μόνο τα πεδία που δεν είναι structs.

Επίσης παρατηρούμε πως από τις τελευταίες λειτουργείες που τρέχουν είναι η **fat_destroy_inode()**, οπότε πάμε και κλείνουμε το αρχείο εκεί. Βάζουμε κι ένα printk για να δούμε πως έκλεισε.

Τέλος προσθέτουμε `#include <linux/syscalls.h>` για τις λειτουργίες *sys_open, sys_write*, κλπ…

Το αποτέλεσμα:

```
[    0.028715] Inode: fat_write_end
[    0.028722] File op: fat_file_release
[    0.028927] Inode: fat_writepages
[    0.028942] Superblock: fat_write_inode
[    0.029219] Superblock: fat_evict_inode
[    0.029224] Superblock: fat_destroy_inode
[    0.029226] Closed file
[    0.029227] Superblock: fat_evict_inode
[    0.029254] Superblock: fat_destroy_inode
[    0.029256] Closed file
[    0.029256] Superblock: fat_put_super
[    0.029274] Superblock: fat_evict_inode
[    0.029276] Superblock: fat_destroy_inode
[    0.029277] Closed file
[    0.029278] Superblock: fat_evict_inode
[    0.029280] Superblock: fat_destroy_inode
[    0.029281] Closed file
[    0.029464] reboot: Restarting system
myy601@myy601lab2:~/lkl/lkl-source/tools/lkl$
```

```
  0.027326] File op: generic_file_read_iter
  0.027718] Superblock Initialization.
  0.027734] File opened.
  0.027745] Superblock: fat_alloc_inode
  0.027746] Superblock: fat_alloc_inode
  0.027748] Superblock: fat_alloc_inode
  0.028314] SUPERBLOCK INFO fat_fill_super
  0.028314]          sbi->sec_per_clus = 4
  0.028314]          sbi->cluster_bits = 11
  0.028314]          sbi->cluster_size = 2048
  0.028314]          sbi->fats = 2
  0.028314]          sbi->fat_bits = 16
  0.028314]          sbi->fat_start = 4
  0.028314]          sbi->fat_length = 200
  0.028314]          sbi->dir_start = 404
  0.028314]          sbi->dir_entries = 512
  0.028314]          sbi->data_start = 436
  0.028314]          sbi->max_cluster = 51093
  0.028314]          sbi->root_cluster = 0
  0.028314]          sbi->fsinfo_sector = 0
  0.028314]          sbi->prev_free = 2
  0.028314]          sbi->free_clusters = 4294967295
  0.028314]          sbi->free_clus_valid = 0
  0.028314]          sbi->dir_ops = 00005590c3901100
  0.028314]          sbi->dir_per_block = 16
  0.028314]          sbi->dir_per_block_bits = 4
  0.028314]          sbi->vol_id = 4096560365
  0.028314]          sbi->dirty = 0
  0.028314]          sbi->fatent_shift = 1
  0.028314] ------------------------
```

Έπειτα πάμε να εντοπίσουμε αλλαγές που γίνονται στο **struct msdos_sb_info:**

Με την εντολή **~/bin/search msdos_sb_info** βλέπουμε που αναφέρεται το συγκεκριμμένο struct, οπότε πάμε σε κάθε φάκελο που προσδιορίζεται και ψάχνουμε για αλλαγές στα πεδία του struct.

Καταλήγουμε να βάλουμε 2 ακόμα *printk*:

i) Στο αρχείο **inode.c** στη **fat_put_super()**, η οποία εμφανίζεται στο τέλος με όλα τα στοιχεία περασμένα, άρα και βάζουμε το ίδιο *printk* όπως και στο **fat_fill_super()** παραπάνω. Κάτι σαν τελική κατάσταση του superblock πριν τελειώσουμε.

ii) Στο αρχείο **misc.c** στη **fat_chain_add()** γιατί αλλάζει το **prev_free**

```
printk(KERN_INFO "SUPERBLOCK INFO fat_chain_add\n\t\tsbi->prev_free = %u\n",
                      sbi->prev_free);
```

Με αυτόν τον τρόπο τελειώσαμε από την καταγραφή του superblock (msdos_sb_info), δίχως να καταγράφουμε τα struct μέσα σε αυτό δυστυχώς.

INODE

Προχωρώντας παρακάτω, ψάχνουμε και τα υπόλοιπα structs μέσα στο **fat.h**. Συγκεκριμμένα το **msdos_inode_info**.

Πρώτο βήμα είναι να το καταγράψουμε ενώ αρχικοποιείται στο **fat_fill_inode()** στο **inode.c** ως εξής:

```
    printk(KERN_INFO "INODE\n"
                     "\t\tnr_caches = %d\n"
                     "\t\ti_start = %d\n"
                     "\t\ti_logstart = %d\n"
                     "\t\ti_attrs = %d\n"
                     "\t\ti_pos = %llu\n"
                     "\t\tmmu_private = %llu\n"
                     "\t\tcache_valid_id = %u\n",
                     MSDOS_I(inode)->nr_caches,
                     MSDOS_I(inode)->i_start,
                     MSDOS_I(inode)->i_logstart,
                     MSDOS_I(inode)->i_attrs,
                     MSDOS_I(inode)->i_pos,
                     MSDOS_I(inode)->mmu_private,
                     MSDOS_I(inode)->cache_valid_id
                     );
```

Παρατηρούμε πως καλείται από μία έως και περισσότερες φορές (αναλόγως το μέγεθος του αρχείου) η εξής ακολουθία:

Inode: fat_write_begin

….. fat op's …..

Inode: fat_write_end

Οπότε πάμε στη **fat_write_end()** και καταγράφουμε τις πληροφορίες του **msdos_inode_info** όπως και πριν με το ίδιο *printk*.

Το αποτέλεσμα (κάπου στη μέση όσων εκτυπώθηκαν στο terminal):

```
[    0.027095] SUPERBLOCK INFO fat_chain_add
[    0.027095]          sbi->prev_free = 8
[    0.027106] Fat op: fat_ent_blocknr
[    0.027111] Fat op: fat_ent_bread
[    0.027130] Fat op: fat16_ent_set_ptr
[    0.027135] Fat op: fat16_ent_get
[    0.027142] Inode: fat_write_end
[    0.027163] INODE
[    0.027163]          nr_caches = 1
[    0.027163]          i_start = 3
[    0.027163]          i_logstart = 3
[    0.027163]          i_attrs = 32
[    0.027163]          i_pos = 6465
[    0.027163]          mmu_private = 12288
[    0.027163]          cache_valid_id = 2
[    0.027243] File op: generic_file_write_iter
[    0.027248] Inode: fat_write_begin
```

Στο τέλος κάθε **fat_write_end()** παίρνουμε τα δεδομένα του *inode*.

Πηγαίνοντας στην μέθοδο **fat_direct_IO()** στο **inode.c** βλέπουμε ό,τι μπορεί να μην έχει ήδη αλλάξει το **mmu_private** (επειδή δε χρησιμοποιεί τη **write_begin()** όπως αναφέρεται σε ήδη υπάρχων σχόλια) οπότε καταγράφουμε την αλλαγή.

```
 */
loff_t size = offset + count;
if (MSDOS_I(inode)->mmu_private < size)
{
    printk(KERN_INFO "msdos_inode_info->mmu_private = %llu", size);
    return 0;
}
```

DIRECTORY ENTRY

Όπως και πριν, αναγνωρίζουμε μέσω **~/bin/search fat_slot_info** το που βρίσκονται αλλαγές.

Εμφανίζονται 3 πιθανοί φάκελοι:

namei_msdos.c : 8 εμφανίσεις
namei_vfat.c : 9 εμφανίσεις
dir.c : 6 εμφανίσεις (1 από αυτά είναι σε σχόλιο)

Έπειτα από αναζήτηση παρατηρούμε πως στα δύο πρώτα αρχεία, σε κάθε συνάρτηση που εμφανίζεται το **fat_slot_info** , είτε καλεί αμέσως μία μέθοδο από το αρχείο **dir.c** , είτε τη καλεί έμμεσα, καλώντας μία άλλη μέθοδο πρώτα.

Επομένως χρειάζεται να καταγράψουμε μόνο τις αλλαγές στο **dir.c** .

Βρίσκουμε λοιπόν τις εξής μεθόδους :

```
fat_search_long(...)
fat_scan(...)
fat_scan_logstart(...)
fat_remove_entries(...)
fat_add_entries(...)
```

Σε καθεμία από αυτές προσθέτουμε:

```
    printk(KERN_INFO "DIRECTORY TABLE: “METHOD NAME”\n"
                     "\t\ti_pos = %llu\n"
                     "\t\tslot_off = %llu\n"
                     "\t\tnr_slots = %d\n",
                     sinfo->i_pos,
                     sinfo->slot_off,
                     sinfo->nr_slots);
```

Δυστυχώς το μόνο ορατό αποτέλεσμα που μπορούμε να δούμε στην κονσόλα είναι:

```
Dir: vfat_lookup
Dir: vfat_create
DIRECTORY TABLE: fat_add_entries
         i_pos = 6465
         slot_off = 0
         nr_slots = 2
Superblock: fat_alloc_inode
INODE
```

Αλλά λειτουργεί κανονικά!

File Allocation Table

Μας μένει μόνο ένα struct από το **fat.h** που δεν έχουμε αναλύσει κι αυτό είναι το **fat_entry** που έχει να κάνει με τις λειτουργείες fat στο **fatent.c** .

~/bin/search fat_entry :

Αποτελέσματα σε 5 αρχεία, τα 4 από τα οποία μετά από έλεγχο των μεθόδων που καλούν, καταλήγουν όλα στο **fatent.c** . Τις αφήνουμε όπως έχει.

Η σκυτάλη περνάει στο αρχείο **fatent.c** όπου και γίνονται οι περισσότερες αναφορές.

Σε αυτό οι περισσότερες εμφανίσεις του **struct fat_entry** γίνονται μέσα στις λειτουργείες fat12/16/32 που έχουν αναφερθεί και στη εκφώνηση. Μέσω testing ανακαλύψαμε πως οι τιμές είναι ίδιες για μια ομάδα εντολών. Πχ:

```
.ent_blocknr   = fat12_ent_blocknr,
    .ent_set_ptr    = fat12_ent_set_ptr,
    .ent_bread  = fat12_ent_bread,
    .ent_get    = fat12_ent_get,
    .ent_put    = fat12_ent_put,
    .ent_next   = fat12_ent_next,
```

Γι αυτό βάζουμε σε μία μόνο *printk*, ώστε να μη γεμίσουμε το αρχείο. Εμείς επιλέξαμε να το τοποθετήσουμε στα **fat_ent_bread()** και **fat12_ent_bread()** .

```
printk("FILE ALLOCATION TABLE: \n"
           "\t\tentry = %d\n"
           "\t\tnr_bhs = %d\n"
           "\t\tu.ent12_p[0] = %p\n"
           "\t\tu.ent12_p[1] = %p\n"
           "\t\tu.ent16_p = %p\n"
           "\t\tu.ent32_p = %p\n",
            fatent->entry,
            fatent->nr_bhs,
            fatent->u.ent12_p[0],
            fatent->u.ent12_p[1],
            fatent->u.ent16_p,
            fatent->u.ent32_p);
```

Οι υπόλοιπες εμφανίσεις γίνονται κάθε φορά που διαβάζεται, γράφεται, ανακατανομείται (alloc) ή καταστρέφεται(free) ένα cluster. Δε προσθέτουμε κάτι γιατί κάνουν free τα πεδία μέσω `fatent_brelse` και χρησιμοποιούν άλλες μεθόδους που έχουνε ήδη *printk* μέσα, αν και έμμεσα. Εξαίρεση αποτελεί το **fat_ent_update_ptr()** στο οποίο επίσης προσθέτουμε το *printk*.

Αποτέλεσμα:

```
39] Fat op: fat_ent_blocknr
41] Fat op: fat_ent_bread
51] Fat op: fat16_ent_set_ptr
57] FILE ALLOCATION TABLE:
57]          entry = 3
57]          nr_bhs = 1
57]          u.ent12_p[0] = 00007fd4b1c00806
57]          u.ent12_p[1] = 00007ffe259e7ca0
57]          u.ent16_p = 00007fd4b1c00806
57]          u.ent32_p = 00007fd4b1c00806
99] Fat op: fat16_ent_get
01] Fat op: fat16_ent_put
03] Fat op: fat_ent_blocknr
05] Fat op: fat16_ent_set_ptr
07] FILE ALLOCATION TABLE:
07]          entry = 4
07]          nr_bhs = 1
07]          u.ent12_p[0] = 00007fd4b1c00808
07]          u.ent12_p[1] = 00007ffe259e7ca0
07]          u.ent16_p = 00007fd4b1c00808
07]          u.ent32_p = 00007fd4b1c00808
12] Fat op: fat16_ent_get
14] Fat op: fat16_ent_put
15] Fat op: fat_ent_blocknr
16] Fat op: fat16_ent_set_ptr
18] FILE ALLOCATION TABLE:
18]          entry = 5
18]          nr_bhs = 1
18]          u.ent12_p[0] = 00007fd4b1c0080a
18]          u.ent12_p[1] = 00007ffe259e7ca0
18]          u.ent16_p = 00007fd4b1c0080a
18]          u.ent32_p = 00007fd4b1c0080a
24] Fat op: fat16_ent_get
25] Fat op: fat16_ent_put
26] Fat op: fat_ent_blocknr
28] Fat op: fat16_ent_set_ptr
29] FILE ALLOCATION TABLE:
29]          entry = 6
29]          nr_bhs = 1
29]          u.ent12_p[0] = 00007fd4b1c0080c
29]          u.ent12_p[1] = 00007ffe259e7ca0
29]          u.ent16_p = 00007fd4b1c0080c
29]          u.ent32_p = 00007fd4b1c0080c
35] Fat op: fat16_ent_get
```

Καταγραφή των αλλαγών σε αρχείο

Έχωντας εκτυπώσει ό,τι δεδομένα από δομές αλλάζουν και θέλουμε να κρατήσουμε, δε μένει παρά να τα τοποθετήσουμε στο αρχείο μας .

Στην αρχή των αρχείων **dir.c**, **misc.c** και **fatent.c** προσθέτουμε το :

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

(Προσοχή: στο **inode.c** το έχουμε ήδη για *sys_open()*).

Οτιδήποτε έχουμε καταγράψει σαν αλλαγή των δομών με τα *printk*, θέλουμε να τα γράψουμε επίσης και μέσα στο αρχείο.

Για να το κάνουμε αυτό ακολουθούμε το ακόλουθο format εντολών, σε κάθε ξεχωριστό *printk*:

```
char buf[128];
sprintf(buf, "SUPERBLOCK: fat_chain_add\n"
             "\t\tsbi>prev_free = %u\n", sbi>prev_free);
    sys_write(sbi->recordFile, buf, sizeof(buf));
    sys_fsync(sbi->recordFile);
    sys_fdatasync(sbi->recordFile);
```

- Χρειαζόμαστε ένα array από χαρακτήρες για το sys_write, άρα δημιουργούμε ένα με το επιθυμητό μέγεθος, το οποίο ανά περίπτωση εξαρτάται.
- Μέσω της *sprintf* στέλνουμε output στο *buf*[] με συγκεκριμμένο format. Δηλαδή τα περνάμε όλα εκεί μέσα με λίγα λόγια.
- Στη sys_write:
  - Δηλώνουμε τον φάκελο στον οποίο θέλουμε να γράψουμε
  - Το string το οποίο θέλουμε να γράψουμε
  - Και το μέγεθος του, που σε αυτή τη περίπτωση είναι όσο και το size του *buf*[] διότι είναι *char* array. Αλλιώς θα χρειαζόταν να κάνω `sizeof(buf)/sizeof(type)`.
- Και τέλος *fsync* & *fdatasync* στον file descriptor μας για να μη “χάσουμε” τα δεδομένα.

Δεν αντικαθιστούμε τα *printk,* αλλά τα κρατάμε για να βλέπουμε τις αλλαγές.

Για καλύτερη κατανόηση του τι κάναμε ακριβώς, παρακάτω βρίσκεται το ίδιο “block“ εντολών που βάλαμε στο **fat_ent_bread()** στο **fatent.c** .

Τα σημεία που αλλάξαμε είναι **highlighted**.

```
static int fat_ent_bread(struct super_block* sb, struct fat_entry* fatent,
	int offset, sector_t blocknr)
{
	const struct fatent_operations* ops = MSDOS_SB(sb)->fatent_ops;

	// my changes
	printk(KERN_INFO "Fat op: fat_ent_bread\n");
	char buf[1024];

	WARN_ON(blocknr < MSDOS_SB(sb)->fat_start);
	fatent->fat_inode = MSDOS_SB(sb)->fat_inode;
	fatent->bhs[0] = sb_bread(sb, blocknr);
	if (!fatent->bhs[0]) {
		fat_msg(sb, KERN_ERR, "FAT read failed (blocknr %llu)",
			(llu)blocknr);
		return -EIO;
	}
	fatent->nr_bhs = 1;
	ops->ent_set_ptr(fatent, offset);

	printk("FILE ALLOCATION TABLE: \n"
		"\t\tentry = %d\n"
		"\t\tnr_bhs = %d\n"
		"\t\tu.ent12_p[0] = %p\n"
		"\t\tu.ent12_p[1] = %p\n"
		"\t\tu.ent16_p = %p\n"
		"\t\tu.ent32_p = %p\n",
		fatent->entry,
		fatent->nr_bhs,
		fatent->u.ent12_p[0],
		fatent->u.ent12_p[1],
		fatent->u.ent16_p,
		fatent->u.ent32_p);

	sprintf(buf, "FILE ALLOCATION TABLE: \n"
		"\t\tentry = %d\n"
		"\t\tnr_bhs = %d\n"
		"\t\tu.ent12_p[0] = %p\n"
		"\t\tu.ent12_p[1] = %p\n"
		"\t\tu.ent16_p = %p\n"
		"\t\tu.ent32_p = %p\n",
		fatent->entry,
		fatent->nr_bhs,
		fatent->u.ent12_p[0],
		fatent->u.ent12_p[1],
		fatent->u.ent16_p,
		fatent->u.ent32_p);
	sys_write(MSDOS_SB(sb)->recordFile, buf, sizeof(buf));
	sys_fsync(MSDOS_SB(sb)->recordFile);
	sys_fdatasync(MSDOS_SB(sb)->recordFile);

	return 0;
}
```

Το ίδιο εννοείται πως κάναμε σε κάθε printk που καταγράφουμε αλλαγή των δομών.

---------- Τέλος. ----------