

ΜΥΥ601

Λειτουργικά Συστήματα

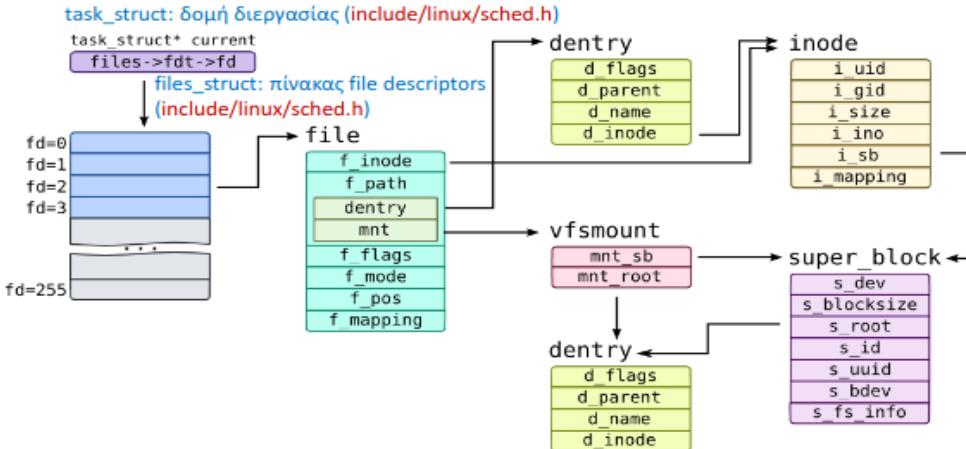
Εργαστήριο 2:
Υλοποίηση αρχείου καταγραφής
στο σύστημα αρχείων FAT του Linux

ΘΕΟΦΑΝΗΣ	ΓΕΩΡΓΑΚΗΣ	A.Μ : 4644
ΕΥΑΓΓΕΛΟΣ	ΜΠΑΛΛΟΣ	A.Μ : 4739
ΚΩΝΣΤΑΝΤΙΝΟΣ	ΓΚΙΟΥΛΗΣ	A.Μ : 4654

Εαρινό εξάμηνο 2021-2022

Πληροφορία αρχείων προς καταγραφή στο Journal

Συσχέτιση διεργασιών με αρχεία



ΜΥΥ601 : Λειτουργικά Συστήματα, Πανεπιστήμιο Ιωαννίνων

42

Οι πληροφορίες που τυπώνουμε είναι βασισμένες στη παραπάνω διαφάνεια. Συγκεκριμένα τυπώνουμε όλες τις πληροφορίες της παραπάνω ιεραρχικής αφαιρετικής δομής, συν μερικές παραπάνω πληροφορίες.

Επιπρόσθετα, κατανοούμε πως η δομή συστήματος fat αποτελείται από τα εξής στοιχεία:

- Superblock
- Inode
- File
- Dendry
- Vfsmount

Αυτά τα στοιχεία είναι προσβάσιμα από τα αρχεία inode.c και file.c αντίστοιχα. Συγκεκριμένα:

Κατανόηση των αρχείων του FAT που πειραχτηκαν

- **inode.c (lkl/lkl-source/fs/fat/inode.c)**

Στο αρχείο αυτό γίνονται οι διάφορες αλλαγές στα inodes και clusters. Περιέχει πληροφορίες για τα inode, superblock και dendry, μέσω structs και μερικά μεταβάλλονται με τις κλήσεις των διαφόρων συναρτήσεων που υπάρχουν εντός του κώδικα.

- **file.c (lkl/lkl-source/fs/fat/file.c)**

Το αρχείο αυτό υλοποιεί συναρτήσεις για τη δημιουργία και αλλαγή ενός αρχείου. Περιέχει πληροφορίες για τα file, και Vfsmount, καθώς και πρόσβαση στα inode, superblock και dendry, μέσω structs και μερικά μεταβάλλονται με τις κλήσεις των διαφόρων συναρτήσεων που υπάρχουν εντός του κώδικα.

- **cptofs.c (lkl/lkl-source/tools/lkl/cptofs.c)**

Αρχείο το οποίο χρησιμεύει στην αντιγραφή ενος αρχειου απο το συστημα (ext4) στο fat filesystem. Εδώ θα υλοποιηθεί η αρχικοποίηση και τελική επεξεργασία του αρχείου, καθώς και η αναδρομική κλήση της cptofs, με χρήση της execl(), για μεταφορά του journal στη vfat συσκευή.

ΔΙΑΔΙΚΑΣΙΑ ΕΠΙΛΥΣΗΣ

```
int fat_add_cluster(struct inode *inode)
{
    int err, cluster;

    printk(KERN_INFO "Super block: %p, Block size: %ld, Count: %d,\n"
           "Max Bytes: %ld, Flags: %lu, Stack Depth: %d,\n"
           "Read Only: %d, ID: %s, UUID: %u",
           inode->i_sb,
           inode->i_sb->s_blocksize,
           inode->i_sb->s_count,
           inode->i_sb->s_maxbytes,
           inode->i_sb->s_flags,
           inode->i_sb->s_stack_depth,
           inode->i_sb->s_READONLY_remount,
           inode->i_sb->s_id,
           inode->i_sb->s_uuid);
```

```
[ 0.022219] This architecture does not have kernel memory protection.
[ 0.023039] Super block: 00007fc76bb75800, Block size: 512, Count: 1,
[ 0.023039] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.023039] Read Only: 0, ID: vda, UUID: 1807178624
[ 0.023056] Super block: 00007fc76bb75800, Block size: 512, Count: 1,
[ 0.023056] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.023056] Read Only: 0, ID: vda, UUID: 1807178624
[ 0.023084] Super block: 00007fc76bb75800, Block size: 512, Count: 1,
[ 0.023084] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.023084] Read Only: 0, ID: vda, UUID: 1807178624
[ 0.023101] Super block: 00007fc76bb75800, Block size: 512, Count: 1,
[ 0.023101] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.023101] Read Only: 0, ID: vda, UUID: 1807178624
[ 0.023121] Super block: 00007fc76bb75800, Block size: 512, Count: 1,
[ 0.023121] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.023121] Read Only: 0, ID: vda, UUID: 1807178624
[ 0.023129] Super block: 00007fc76bb75800, Block size: 512, Count: 1,
[ 0.023129] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.023129] Read Only: 0, ID: vda, UUID: 1807178624
[ 0.023147] Super block: 00007fc76bb75800, Block size: 512, Count: 1,
[ 0.023147] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.023147] Read Only: 0, ID: vda, UUID: 1807178624
[ 0.023880] reboot: Restarting system
myy601@myy601lab2:~/lkl/lkl-source$
```

Αρχικά γράφουμε στον κωδικό της συνάρτησης fat_add_cluster ένα printk για να δούμε τα περιεχόμενα του super block να αλλάζουν κατά την προσθήκη ενός cluster, όπως το pointer του Superblock, το μέγεθος του block και χώρος που καταλαμβάνει (εικόνα 1). Η επιλογή έγινε διότι υπάρχει το struct Superblock μέσα στο inode.c. Με την printk καταλαβαίνουμε πως όλες οι αλλαγές που γίνονται, γίνονται μέσα στο ίδιο Superblock

```

212     // Superblock stays the same between runs, changes when compiling and re-setting up the vfat dev
213     printk(KERN_INFO "----- Super Block -----\\n"
214           "Spr blk: %p, Blk size: %ld, SBCnt: %d,\\n"
215           "Max Bytes: %ld, Flags: %lu, Stack Dep: %d,\\n"
216           "RONLY: %d, ID: %s, UUID: %u\\n"
217           "----- Inode -----\\n"
218           "Current inode: %p, Size: %ld,\\n"
219           "Byte: %hu, Blk bits: %hu,\\n"
220           "Cnt: %d, DIO Cnt: %lu, WRCnt: %lu\\n"\\
221           "----- Dendry -----\\n"
222           "Dendry: %p, Flags: %u, Par: %p, Name: %s, Inode: %p\\n\\n",
223           inode->i_sb,
224           inode->i_sb->s_blocksize,
225           inode->i_sb->s_count,
226           inode->i_sb->s_maxbytes,
227           inode->i_sb->s_flags,
228           inode->i_sb->s_stack_depth,
229           inode->i_sb->s_READONLY_remount,
230           inode->i_sb->s_id,
231           inode->i_sb->s_uuid,
232           inode,
233           inode->i_size,
234           inode->i_bytes,
235           inode->i_blkbits,
236           inode->i_count,
237           inode->i_dio_count,
238           inode->i_writecount,
239           inode->i_sb->s_root,
240           inode->i_sb->s_root->d_flags,
241           inode->i_sb->s_root->d_parent,
242           inode->i_sb->s_root->d_name.name,
243           inode->i_sb->s_root->d_inode
244 );

```

```

[ 0.016577] This architecture does not have kernel memory protection.
[ 0.017262] Super block: 00007f83b7b75800, Block size: 512, Count: 1,
[ 0.017262] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.017262] Read Only: 0, ID: vda, UUID: 3082247040
[ 0.017274] Current inode: 00007f83b740e7c0, Size: 0,
[ 0.017274] Byte: 0, Block bits: 9,
[ 0.017274] Count: 1, DIO Count: 0, Write Count: 1
[ 0.017289]
[ 0.017295] Super block: 00007f83b7b75800, Block size: 512, Count: 1,
[ 0.017295] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.017295] Read Only: 0, ID: vda, UUID: 3082247040
[ 0.017302] Current inode: 00007f83b740e7c0, Size: 0,
[ 0.017302] Byte: 0, Block bits: 9,
[ 0.017302] Count: 1, DIO Count: 0, Write Count: 1
[ 0.017309]
[ 0.017329] Super block: 00007f83b7b75800, Block size: 512, Count: 1,
[ 0.017329] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.017329] Read Only: 0, ID: vda, UUID: 3082247040
[ 0.017338] Current inode: 00007f83b740e7c0, Size: 4096,
[ 0.017338] Byte: 0, Block bits: 9,
[ 0.017338] Count: 1, DIO Count: 0, Write Count: 1
[ 0.017341]
[ 0.017346] Super block: 00007f83b7b75800, Block size: 512, Count: 1,
[ 0.017346] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.017346] Read Only: 0, ID: vda, UUID: 3082247040
[ 0.017352] Current inode: 00007f83b740e7c0, Size: 4096,
[ 0.017352] Byte: 0, Block bits: 9,
[ 0.017352] Count: 1, DIO Count: 0, Write Count: 1
[ 0.017358]
[ 0.017374] Super block: 00007f83b7b75800, Block size: 512, Count: 1,
[ 0.017374] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.017374] Read Only: 0, ID: vda, UUID: 3082247040
[ 0.017382] Current inode: 00007f83b740e7c0, Size: 8192,
[ 0.017382] Byte: 0, Block bits: 9,
[ 0.017382] Count: 1, DIO Count: 0, Write Count: 1
[ 0.017388]
[ 0.017393] Super block: 00007f83b7b75800, Block size: 512, Count: 1,
[ 0.017393] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.017393] Read Only: 0, ID: vda, UUID: 3082247040
[ 0.017400] Current inode: 00007f83b740e7c0, Size: 8192,
[ 0.017400] Byte: 0, Block bits: 9,
[ 0.017400] Count: 1, DIO Count: 0, Write Count: 1
[ 0.017403]
[ 0.017416] Super block: 00007f83b7b75800, Block size: 512, Count: 1,
[ 0.017416] Max Bytes: 4294967295, Flags: 1879050240, Stack Depth: 0,
[ 0.017416] Read Only: 0, ID: vda, UUID: 3082247040
[ 0.017427] Current inode: 00007f83b740e7c0, Size: 12288,
[ 0.017427] Byte: 0, Block bits: 9,
[ 0.017427] Count: 1, DIO Count: 0, Write Count: 1
[ 0.017434]
[ 0.017955] reboot: Restarting system
myy601@myy601lab2:~/lkl/lkl-source$ █

```

Έπειτα αλλάζουμε την printk για να δούμε δεδομένα για τις αλλαγές του inode με τη πρόσθεση clusters και παρατηρούμε πως μεταβάλλεται το size κατά 4 KB, όσο το size του buffer(που είναι και το size του cluster για την αποθήκευση δεδομένων) που διαβάζουμε από το ίδιο inode struct.

Αρχικά θα μελετήσουμε τις **αποτυχημένες** προσπάθειες επίλυσης, προτού προχωρήσουμε στην **επιτυχημένη**.

- Πρώτη **αποτυχημένη** προσπάθεια δημιουργίας και γραφής/ανάγνωσης του αρχείου Journal στο inode.c

```
119 int fat_add_cluster(struct inode *inode)
120 {
121     char *directory = "/journal";
122     int file = (int) sys_open(directory, O_RDWR|O_CREAT|O_TRUNC, 0666 );
123     printk(KERN_INFO "File Descriptor: %ld", file);
124     //scanf("%d", &tst);

171     loff_t wpos = 0;
172     //err = vfs_read(file, buff, 1, &pos);
173     char buffer[20];
174     size_t nbytes;
175     ssize_t bytes_written;
176     strcpy(buffer, "Hi there. :)\n");
177     nbytes = strlen(buffer);

178     err = sys_write(file, buffer, nbytes);
179     if (err)
180         return err;
181     //mdelay(2000000);

183
184     err = sys_close(file);
185     if (err)
186         return err;
```

Αρχικά δημιουργούμε μία μεταβλητή directory που είναι πίνακας χαρακτήρων, μία int μεταβλητή file που αποθηκεύει το index του αρχείου journal που θα δημιουργήσουμε μέσα στον φάκελο fat. Έπειτα τυπώνουμε το descriptor του journal που επιστρέφει η sys_open(). Προσπαθούμε να γράψουμε στο αρχείο με

την sys_write(), τις λέξεις “Hi there” σαν πρώτη προσπάθεια (με τη μεταβλητή buffer στην οποία γράφουμε μέσω της strcpy() και με το nbytes πόσα στοιχεία να γράψει από το buffer η sys_write()). Στο τέλος κλείνουμε το αρχείο για να μην δημιουργήσει προβλήματα κατα την εκτέλεση.

Αποτέλεσμα: Έκανε compile, η sys_open(), όμως επέστρεψε στο file descriptor τιμή -1 (την οποία αποτυπώναμε με τη χρήση της printk).

- Δεύτερη αποτυχημένη προσπάθεια δημιουργίας και γραφής/ανάγνωσης του αρχείου Journal

```

137 int fat_add_cluster(struct inode *inode)
138 {
139     char *directory = "journal";
140     mm_segment_t old_fs = get_fs();
141     set_fs(KERNEL_DS);
142     int file = (int) sys_open(directory, O_RDONLY|O_CREAT, 0666 );
143     /*printk(KERN_INFO "File Descriptor: %d", file);
144     struct file *filee = filp_open(directory, O_RDWR|O_CREAT|O_TRUNC, 0666 );
145     if (IS_ERR(filee)){
146         printk(KERN_INFO "File Open Failed: %d", PTR_ERR(filee));
147     }*/
148     //printk(KERN_INFO "File Descriptor: %s", filee->name);
149     //scanf("%d", &tst);
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199 //char buff[sizeof(inode->i_size)];
200
201 //loff_t wpos = 0;
202 //err = vfs_read(file, buff, 1, &pos);
203 char rdbuf[14];
204 char wrbuf[1];
205 size_t nbytes;
206 //ssize_t bytes_written;
207 strcpy(rdbuf, "Hi there. :\n");
208 nbytes = strlen(rdbuf);
209
210 int write = sys_write(file, rdbuf, nbytes);
211
212 sys_lseek(file, 0, SEEK_SET);
213 //err = vfs_read(filee, buffer, 25, &wpos);
214 printk(KERN_INFO);
215 while (sys_read(file, wrbuf, 1) == 1)
216     |  printk("%c", err, wrbuf[0]);
217 printk("\n");
218 //err = sys_write(file, buffer, nbytes);
219 //if (err)
220 //    return err;
221 //mdelay(2000000);
222
223 err = sys_close(file);
224 if (err)
225     |  return err;
226 set_fs(old_fs);

```

Συνειδητοποιήσαμε πως στο kernel space δεν μπορούμε να ανοίξουμε/γράψουμε αρχείο διότι το file system το καθιστά αδύνατο. Οπότε προσπαθούμε να αλλάξουμε το mode του filesystem στο kernelspace, ώστε να έχει την δυνατότητα να ανοίξει αρχεία. Αποθηκεύσαμε το αρχικό mode του filesystem σε old_fs = get_fs(), ώστε

να επιστρέψουμε στο αρχικό file system αφού ανοίξουμε και γράψουμε τα αρχεία που θέλουμε. Η εντολή sys_write() επέστρεψε θετική τιμή (άρα φαίνεται πως έγραψε επιτυχώς). Έπειτα αφού “γράψαμε” το “Hi there” εντός του Journal, προσπαθήσαμε και να το διαβάσουμε με την κλήση της sys_read(), 1 χαρακτήρα τη φορά και να το τυπώσουμε στο τερματικό.

Αποτέλεσμα: Η sys_open() και sys_close() φαίνεται πως δημιουργούσε και έκλεινε ένα αρχείο αλλά δεν το εμφάνιζε σε κανένα directory και ότι γράφαμε στο αρχείο δεν αποθηκευόταν και δεν μπορούσε να το διαβάσει η sys_read (επέστρεψε end of file).

```
157     char cwd[200];  
158     sys_getcwd(cwd, sizeof(cwd));  
159     printk(KERN_INFO "%s", cwd);
```

Δοκιμάσαμε τον παραπάνω κώδικα, ώστε να δούμε σε ποιο directory βρίσκεται το πρόγραμμα κατα την διάρκεια της εκτέλεσης, ώστε να καταλάβουμε που μπορεί να δημιουργείται το αρχείο αυτό. Επέστρεψε “/”, δηλαδή το root directory. Ελέγχαμε το root directory του ext4 και του vfat, δεν υπήρχε σε κανένα από τα δύο. Επιπρόσθετα, εκτελέσαμε την εντολή “find / | grep journal” στο ext4, αλλά δε βρήκαμε το αρχείο. Υποψιαζόμαστε πως μπορεί να έγραφε σε αρχείο με συμπεριφορά proc entry, καθώς τα δεδομένα δεν αποθηκευόταν.

- Τρίτη αποτυχημένη προσπάθεια δημιουργίας και γραφής/ανάγνωσης του αρχείου Journal

```

28 #include <linux/uaccess.h>
29 #include <linux/ioctl.h>
30
31 //////////////// NEW CODE
32 #include <linux/module.h>
33 #include <linux/pagemap.h>
34 #include <linux/mpage.h>
35 #include <linux/vfs.h>
36 #include <linux/seq_file.h>
37 #include <linux/parser.h>
38 #include <linux/uio.h>
39 #include <linux/blkdev.h>
40 #include <linux/backing-dev.h>
41 #include <asm/unaligned.h>
42 #include "fat.h"
43
44 #ifndef CONFIG_FAT_DEFAULT_IOCTLSET
45 /* if user don't select VFAT, this is undefined. */
46 #define CONFIG_FAT_DEFAULT_IOCTLSET ""
47 #endif
48
49 #define KB_IN_SECTORS 2
50 //////////////// NEW CODE
51 #define MAGIC 'M'
52 #define SET_DATA _IOW(MAGIC,1u,int)
53 #define GET_DATA _IOR(MAGIC,2u,int)
54 //////////////// NEW CODE
55
56 static unsigned int x = 0 ;
57 static unsigned int y = 100;

```

Line 53. Column 9

```

52 #define SET_DATA _IOW(MAGIC,1u,int)
53 #define GET_DATA _IOR(MAGIC,2u,int)
54 //////////////// NEW CODE
55
56 static unsigned int x = 0 ;
57 static unsigned int y = 100;
58 static dev_t device_num;
59 static struct class *cl;
60 static char *ptr = NULL;
61 static long mydevice_ioctl(struct file *f,unsigned int ioctl_num,unsigned long ioctl_param);
62 //////////////// NEW CODE
63 static struct file_operations fops=
64 {
65     //read= mydevice_read,
66     //write= mydevice_write,
67     //open= mydevice_open,
68     .unlocked_ioctl = mydevice_ioctl,
69     //release= mydevice_release,
70 };
71
72 //////////////// NEW CODE
73 //INPUT_OUTPUT CONTROL FUNCTION
74 static long mydevice_ioctl(struct file *f,unsigned int ioctl_num,unsigned long ioctl_param)
75 {
76     printk(KERN_INFO "INSIDE IOCTL FUNC\n");
77     switch(ioctl_num)
78     {
79         case SET_DATA :
80             copy_from_user(&x,(int*)ioctl_param,sizeof(int));
81             printk(KERN_INFO "DATA COPIED to variable x from user space: %d.\n",x);

```

Line 53. Column 9

```

70     };
71     ////////////////NEW CODE
72     //INPUT OUTPUT CONTROL FUNCTION
73     static long mydevice_ioctl(struct file *f,unsigned int ioctl_num,unsigned long ioctl_param)
74     {
75         printk(KERN_INFO "INSIDE IOCTL FUNC\n");
76         switch(ioctl_num)
77         {
78             case SET_DATA :
79                 copy_from_user(&x,(int*)ioctl_param,sizeof(int));
80                 printk(KERN_INFO "DATA COPIED to variable x from user space: %d.\n",x);
81                 break;
82             case GET_DATA:
83                 copy_to_user((int*)ioctl_param,&y,sizeof(int));
84                 printk(KERN_INFO "DATA COPIED to user from driver: %d.\n",y);
85                 break;
86         }
87         return 0;
88     }
89 */
90 /*
91 * A deserialized copy of the on-disk structure laid out in struct
92 * fat_boot_sector.
93 */
94 struct fat_bios_param_block {
95     u16 fat_sector_size;
96     u8 fat_sec_per_clus;
97     u16 fat_reserved;
98

```

Προσπαθήσαμε να κάνουμε ένα module (σαν μέρος του lkl). Προσπαθήσαμε να γράψουμε από το kernel space που βρίσκονται τα string με τις πληροφορίες των inode και file προς εγγραφή σε ένα buffer, το οποίο στο τέλος διαβάζει το cptofs. Η ιδέα είναι η μεταφορά των δεδομένων από κάθε αλλαγή των inode και file στο cptofs (δηλαδή ανάκτηση της πληροφορίας του kernel space από το user space).

Έπειτα στο cptofs θα αποθηκεύαμε τις αλλαγές στο αρχείο journal, αφού εκεί οι λειτουργίες του open() και write() είναι γνωστές.

```

////////

#define MAGIC 'M'
#define SET_DATA _IOW(MAGIC,1,u,int)
#define GET_DATA _IOR(MAGIC,2,u,int)

////////

```

```

536 ///////////////////////////////////////////////////////////////////NEW CODE
537
538 int test()
539 {
540
541     int fd;
542     //static unsigned int g = 0;
543     static unsigned int h = 0;
544
545     //open func
546     fd = open("/tmp/journal", O_RDWR|O_CREAT|O_TRUNC, 0666);
547     if(fd < 0)
548     {
549         printf("fail: %d\n", fd);
550         return -1;
551     }
552     scanf("%d", &g);
553     if(ioctl(fd, SET_DATA, &g) < 0)
554     {
555         printf("IOCTL FAIL g\n");
556         return -1;
557     }
558     if(ioctl(fd, GET_DATA, &h) < 0)
559     {
560         printf("IOCTL FAIL h\n");
561         return -1;
562     }
563     printf("the file is closed successfully with file descriptor: %d\n", fd);
564     return fd ;
565
566 }
567
568

```

Εδώ στο αρχείο του cptofs διαβάζει τα δεδομένα από τη μεταβλητή GET_DATA που τα στέλνει το kernel space και κάνει print τα δεδομένα αυτά. Πρώτου γίνει η write() θέλαμε να ελέγξουμε, αρχικά εαν τα δεδομένα στέλνονται και δεύτερον εαν στέλνονται σωστά.

Αποτέλεσμα: Τελικά, δεν γινόταν η σωστή μεταφορά των δεδομένων, ούτε το άνοιγμα της open() στο cptofs. Τα prints δεν έδειξαν καμία μεταφορά δεδομένων.

```

122 static struct cdev my_cdev;
123
124 static ssize_t copy_to_userspace(struct file *filp, char __user *buff, size_t len, loff_t *loff)
125 {
126     char bffrtst[] = "this is a test :(";
127     if(copy_to_user(buff, bffrtst, len))
128     {
129         printk(KERN_INFO "Read Failed for buffer: %s", buff);
130         return -EFAULT;
131     }
132 }

```

```

#include <linux/init.h>
#include <linux/module.h>
#include <linux/uaccess.h>
#include <linux/fs.h>
#include <linux/proc_fs.h>

// Module metadata
MODULE_AUTHOR("4739, 4654, 4644");
MODULE_DESCRIPTION("Hopefully mmap. Idk, this is a last ditch effort..");

static struct proc_dir_entry* proc_entry;

static ssize_t custom_read(struct file* file, char __user* user_buffer, size_t count, loff_t* offset)
{
    char message[] = "I really hope this works..";
    int mes_length = strlen(message);

    if (*offset > 0)
        return 0;

    copy_to_user(user_buffer, message, mes_length);
    *offset = mes_length

    return mes_length;
}

static struct file_operations fops =
{
    .owner = THIS_MODULE,
    .read = custom_read
};

static int __init kek_init(void) {
    proc_entry = proc_create("ihopethisworksffs", 0666, NULL, &fops);
    printk(KERN_INFO "Hello from kernel (?)");
    return 0;
}

static int __exit kek_exit(void) {
    proc_remove(proc_entry);
    printk(KERN_INFO "Goodbye from kernel (?)");
    return 0;
}

module_init(kek_init);
module_exit(kek_exit);

```

Μια δεύτερη προσπάθεια με αυτή τη μέθοδο με μικρές διαφορές. Αυτή η προσπάθεια ήταν κυρίως σαν module, σε αντίθεση με τη πρώτη προσπάθεια που ήταν μέρος του lkl (inode.c). Αυτό το module προσπαθήσαμε να το κάνουμε compile και να το εγκαταστήσουμε στο kernel του VM, αλλά τα εργαλεία που μας έχουν δοθεί δεν επαρκούν για αυτή την υλοποίηση. Επιπρόσθετα χρειάζεται μία φορά εγκατάσταση του module ως root, κάτι το οποίο αυξάνει τον πήχη δυσκολίας για απλούς χρήστες και βγαίνει εκτός του στόχου της άσκησης.

- Τέταρτη αποτυχημένη προσπάθεια δημιουργίας και γραφής/ανάγνωσης του αρχείου Journal

```

533 int kernel_mmap()
534 {
535     int _fdmem;
536     int *map = NULL;
537     const char memDevice[] = "/dev/mem";
538
539     /* open /dev/mem and error checking */
540     _fdmem = open( memDevice, O_RDWR | O_SYNC );
541
542     if (_fdmem < 0){
543         printf("Failed to open the /dev/mem !\n");
544         return 0;
545     }
546     else{
547         printf("open /dev/mem successfully !\n");
548     }
549
550     /* mmap() the opened /dev/mem */
551     map= (int *)mmap[0,sizeof (char *),PROT_READ|PROT_WRITE,MAP_SHARED,_fdmem,DDR_RAM_PHYS];
552
553     /* use 'map' pointer to access the mapped area! */
554     for (int i=0;i<100;i++)
555         printf("content: 0x%x\n",*(map+i));
556
557     /* unmap the area & error checking */
558     if (munmap(map,MAPPED_SIZE)==-1){
559         perror("Error un-mmapping the file");
560     }
561
562     /* close the character device */
563     close(_fdmem);
564 }
```

Έγινε προσπάθεια χρήσης mmap από το user space, για τη κατανομή θέσης μνήμης του kernel space (η οποία θέση, θα είναι προσβάσιμη από το userspace και το kernelspace ταυτόχρονα), στην οποία θα καταγράφαμε τα δεδομένα που είναι να προστεθούν στο Journal από το kernel space. Μέσω της πρόσβασης που μας παρέχει η mmap σε αυτή τη θέση μνήμης, θα διαβάζαμε αυτά τα δεδομένα από το userspace (cryptfs.c) και θα τα γράφαμε στο Journal. Το πρόβλημα με αυτή την υλοποίηση είναι πως για να γίνει η εκτέλεση της mmap

από το userspace, χρειάζεται δικαιώματα χρήστη root (sudo) κάθε φορά που θα εκτελείται η cryptofs, κάτι το οποίο διαφεύγει εκτός του στόχου της λειτουργίας της (ο χρήστης εαν έχει δικαιώματα root, μπορεί απλά να κάνει “sudo mount”, “cp” και “sudo umount”).

- Τελική **επιτυχημένη** προσπάθεια δημιουργίας και γραφής/ανάγνωσης του αρχείου Journal

cryptofs.c:

```
533 int main(int argc, char **argv)
534 {
535     ///////////////////////////////////////////////////////////////////
536     int fd;
537     // create / reset the journal file.
538     // inode.c and file.c open() calls do NOT have the O_TRUNC oflag !!!
539     fd = open("/tmp/journaltmp", O_RDWR|O_CREAT|O_TRUNC, 0666);
540
541     // Check for errors
542     if (!fd) {
543         fprintf(stderr, "Failed to open file");
544         return -1;
545     }
546     // Close the file - check for errors
547     if (close(fd)) {
548         fprintf(stderr, "Failed to close file");
549         return -1;
550     }
```

Αρχικά κάνουμε reset / δημιουργούμε το αρχείο “/tmp/journaltmp” με τη χρήση της open() με τις σημαίες O_CREAT και O_TRUNC (L. 539), ώστε να μην μένουν τα δεδομένα της προηγούμενης εκτέλεσης. Αμέσως μετά, κλείνουμε το αρχείο (L. 547).

```

608     ///////////////////////////////////////////////////////////////////
609     // Copies the journal to another file, then calls cptofs to copy the other file to the fs.
610     // Copy the journal file from to a new file, so that it does not get overwritten.
611     char *journ = "/tmp/journal";
612     if (strcmp(journ, cla.paths[0]) != 0)
613     { // !!!!! Check whether journal is currently being copied. If not, prepare journal for copy, else end the program.
614
615         // Open the two files | fds = fd source | fdd = fd destination.
616         int fds = open("/tmp/journaltmp", O_RDONLY, 0666);
617         int fdd = open("/tmp/journal", O_WRONLY|O_CREAT|O_TRUNC, 0666);
618
619         // Check if files opened successfully.
620         if (!fds | !fdd) {
621             fprintf(stderr, "Failed to open file");
622             return -1;
623         }
624
625         // Copy the file byte for byte.
626         char buff[1];
627         while (read(fds, buff, 1))
628         {
629             if (write(fdd, buff, 1) <= 0) {
630                 fprintf(stderr, "Failed to write to file");
631             }
632         }
633
634
635         // Close the file descriptors
636         if (close(fds) | close(fdd)) {
637             fprintf(stderr, "Failed to close file");
638             return -1;
639         }
640
641         // I. LOVE. C.
642         // Execute the same function but for the journal this time around.
643         execl("./cptofs", "./cptofs", "-i", "/tmp/vfatfile", "-p", "-t", "vfat", "/tmp/journal", "/", (char*) NULL);
644     }

```

Μετά την αντιγραφή στο filesystem, ελέγχουμε αν το αρχείο που αντιγράφουμε είναι το “/tmp/journal” (L. 611-612). Εάν είναι, το πρόγραμμα τερματίζει. Αλλιώς, αντιγράφουμε το αρχείο “/tmp/journaltmp” στο αρχείο “/tmp/journal”. Δηλαδή ανοίγουμε τα δύο αρχεία (L. 616-617) και το αντιγράφουμε byte προς byte (L. 626-633). Κατόπιν κλείνουμε τα αρχεία (L. 636) και εκτελούμε πάλι τη cptofs, με τη χρήση της execl, για το αρχείο “/tmp/journal” (L. 643), ώστε να αποθηκεύσουμε το journal και στο root του filesystem (“/journal”). Για να αποφύγουμε ατέρμονο κύκλο, κάνουμε τον έλεγχο που εξηγήσαμε παραπάνω (L. 611-612).

Inode.c:

```
35 //////////////////////////////////////////////////////////////////
36 #include <linux/fcntl.h>
37 #include <linux/syscalls.h>
38 #include <linux/fs.h>
39 #include <linux/string.h>
40 #include <linux/uaccess.h>
41 #include <linux/slab.h>
42 #include <linux/kernel.h>
43 #include <linux/init.h>
44
45 int open(const char *pathname, int flags, mode_t mode);
46 int close(const char *pathname);
47 ssize_t write(int fd, const void *buf, size_t count);
48 ssize_t read(int fd, const void *buf, size_t count);
49 off_t lseek(int fildes, off_t offset, int whence);
```

Κάναμε include μερικές βιβλιοθήκες (L. 36-43) και προσθέσαμε τα παραπάνω ορίσματα (L. 58-62), ώστε να μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις open(), close(), write(), read() και lseek(), καθώς δεν είναι διαθέσιμη η αντίστοιχη standard βιβλιοθήκη.

```

128 int fat_add_cluster(struct inode *inode)
129 {
130     int err, cluster;
131
132     err = fat_alloc_clusters(inode, &cluster, 1);
133
134     if (err)
135         return err;
136     /* FIXME: this cluster should be added after data of this
137      * cluster is writed */
138     err = fat_chain_add(inode, cluster, 1);
139     if (err)
140         fat_free_clusters(inode, cluster);
141
142
143
144     //////
145     // Open the journal
146     char *directory = "/tmp/journaltmp";
147     int file = open(directory, O_RDWR, 0666 );
148
149     // Check for open errors
150     if (!file)
151         printk(KERN_INFO "Open failed: %d", file);
152     else
153         printk(KERN_INFO "Open success: %d", file);
154

```

Όταν προστίθεται cluster, το καταγράφουμε στο journal. Συγκεκριμένα:

Ανοίγουμε το αρχείο “/tmp/journaltmp” (L. 146-154).

```

155 // Save string with the data to be written to the journal
156 char buffer[1536];
157 sprintf(buffer, "----- Super Block -----\\n"
158 "Spr blk: %p, Blk size: %ld, SBCnt: %d,\\n"
159 "Max Bytes: %ld, Flags: %lu, Stack Dep: %d,\\n"
160 "RONLY: %d, ID: %s, UUID: %u\\n"
161 "----- Inode -----\\n"
162 "Current inode: %p, Size: %ld,\\n"
163 "Byte: %hu, Blk bits: %hu,\\n"
164 "Cnt: %d, DIO Cnt: %lu, WRCnt: %lu\\n"\\n"
165 "----- Dendry -----\\n"
166 "Dendry: %p, Flags: %u, Par: %p, Name: %s, Inode: %p\\n\\n",
167 inode->i_sb,
168 inode->i_sb->s_blocksize,
169 inode->i_sb->s_count,
170 inode->i_sb->s_maxbytes,
171 inode->i_sb->s_flags,
172 inode->i_sb->s_stack_depth,
173 inode->i_sb->s_READONLY_remount,
174 inode->i_sb->s_id,
175 (char*) inode->i_sb->s_uuid,
176 inode,
177 inode->i_size,
178 inode->i_bytes,
179 inode->i_blkbits,
180 inode->i_count,
181 inode->i_dio_count,
182 inode->i_writecount,
183 inode->i_sb->s_root,
184 inode->i_sb->s_root->d_flags,
185 inode->i_sb->s_root->d_parent,
186 inode->i_sb->s_root->d_name.name,
187 inode->i_sb->s_root->d_inode
188 );

```

Καταγράφουμε τις αλλαγές που θα προστεθούν στο journal σε ένα buffer (L. 156-188).

Εδώ καταγράφουμε πληροφορίες για το inode, το superblock και το dendry του αρχείου.

```

190 // Find the end of the string, count how many characters long is the string
191 int str_size = 0;
192 while (1)
193 {
194     if (buffer[str_size] == '\0' || str_size >= 2048){
195         buffer[str_size] = '\n';
196         str_size++;
197         break;
198     }
199     else
200         str_size++;
201 }
202
203 // Go to the end of the file, to avoid overwriting existing data
204 lseek(file, 0, SEEK_END);
205
206 // Write aforementioned string to journal file
207 if (!write(file, buffer, str_size)) {
208     printk(KERN_INFO "Write failed");
209 }
210
211 // Close the journal file
212 if (close(file)) {
213     printk(KERN_INFO "Close failed");
214 }

```

Στη συνέχεια βρίσκουμε που τελειώνει το string στο buffer (L. 191-201), καθώς το buffer μας είναι αρκετά μεγαλύτερο, ώστε να γνωρίζουμε πόσους χαρακτήρες πρέπει να γράψουμε στο αρχείο. Κατόπιν με τη χρήση της lseek(), πηγαίνουμε στο τέλος του αρχείου (L. 204), ώστε να αποφύγουμε το να κάνουμε overwrite τις προηγούμενες εγγραφές. Τέλος ελέγχουμε αν ήταν επιτυχής η εγγραφή στο αρχείο (L. 207-209) και κλείνουμε το αρχείο (L. 212-214).

```

216 // Print the data that was written to the journal
217 printk(KERN_INFO "----- Super Block -----\\n"
218     "Spr blk: %p, Blk size: %ld, SBCnt: %d,\\n"
219     "Max Bytes: %ld, Flags: %lu, Stack Dep: %d,\\n"
220     "RONLY: %d, ID: %s, UUID: %u\\n"
221     "----- Inode -----\\n"
222     "Current inode: %p, Size: %ld,\\n"
223     "Byte: %hu, Blk bits: %hu,\\n"
224     "Cnt: %d, DIO Cnt: %lu, WRCnt: %lu\\n"\\
225     "----- Dendry -----\\n"
226     "Dendry: %p, Flags: %u, Par: %p, Name: %s, Inode: %p\\n\\n",
227     inode->i_sb,
228     inode->i_sb->s_blocksize,
229     inode->i_sb->s_count,
230     inode->i_sb->s_maxbytes,
231     inode->i_sb->s_flags,
232     inode->i_sb->s_stack_depth,
233     inode->i_sb->s_READONLY_remount,
234     inode->i_sb->s_id,
235     inode->i_sb->s_uuid,
236     inode,
237     inode->i_size,
238     inode->i_bytes,
239     inode->i_blkbits,
240     inode->i_count,
241     inode->i_dio_count,
242     inode->i_writecount,
243     inode->i_sb->s_root,
244     inode->i_sb->s_root->d_flags,
245     inode->i_sb->s_root->d_parent,
246     inode->i_sb->s_root->d_name.name,
247     inode->i_sb->s_root->d_inode
248 );
249 // Superblock stays the same between runs, changes when compiling and re-setting up the vfat dev

```

Κατόπιν τυπώνουμε τη καταχώρηση που κάναμε journal παραπάνω και στο τερματικό (L. 217-249).

File.c:

```
21 //////////////////////////////////////////////////////////////////
22 #include <linux/fcntl.h>
23 #include <linux/syscalls.h>
24 #include <linux/fs.h>
25 #include <linux/string.h>
26 #include <linux/uaccess.h>
27 #include <linux/slab.h>
28 #include <linux/kernel.h>
29 #include <linux/init.h>
30
31 int open(const char *pathname, int flags, mode_t mode);
32 int close(const char *pathname);
33 ssize_t write(int fd, const void *buf, size_t count);
34 ssize_t read(int fd, const void *buf, size_t count);
35 off_t lseek(int fildes, off_t offset, int whence);
```

Κάναμε include μερικές βιβλιοθήκες (L. 21-29) και προσθέσαμε τα παραπάνω ορίσματα (L. 31-35), ώστε να μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις open(), close(), write(), read() και lseek(), καθώς δεν είναι διαθέσιμη η αντίστοιχη standard βιβλιοθήκη.

```

169 static int fat_file_release(struct inode *inode, struct file *filp)
170 {
171     ///// ADDED CODE /////
172     // Open the journal
173     char *directory = "/tmp/journaltmp";
174     int file = open(directory, O_RDWR, 0666 );
175
176     // Check for open errors
177     if (!file)
178         printk(KERN_INFO "Open failed: %d", file);
179     else
180         printk(KERN_INFO "Open success: %d", file);
181
182     // Save string with the data to be written to the journal
183     char buffer[1536];
184     sprintf(buffer, "----- File -----\\n"
185             "Inode: %p, Path Dentry: %p, Path Mnt: %p,\\n"
186             "Mnt SB: %p, Mnt Dentry: %p, Flags: %u"
187             "Mode: %u, POS: %lld, Mapping: %p"
188             "Mapping Writable: %lu, Mapping NRPages: %lu\\n\\n",
189             filp->f_inode,
190             filp->f_path.dentry,
191             filp->f_path.mnt,
192             filp->f_path.mnt->mnt_sb,
193             filp->f_path.mnt->mnt_root,
194             filp->f_flags,
195             filp->f_mode,
196             filp->f_pos,
197             filp->f_mapping,
198             filp->f_mapping->i_mmap_writable,
199             filp->f_mapping->nrpages
200         );

```

Όταν προστίθεται cluster, το καταγράφουμε στο journal. Συγκεκριμένα:

Ανοίγουμε το αρχείο “/tmp/journaltmp” (L. 173-180).

Καταγράφουμε τις αλλαγές που θα προστεθούν στο journal σε ένα buffer (L. 183-200).

Εδώ καταγράφουμε πληροφορίες για το file, το dendry, το mnt και τα πεδία του, dendry και superblock, του αρχείου.

```

202     // Find the end of the string, count how many characters long is the string
203     int str_size = 0;
204     while (1)
205     {
206         if (buffer[str_size] == '\0' || str_size >= 2048){
207             buffer[str_size] = '\n';
208             str_size++;
209             break;
210         }
211         else
212             str_size++;
213     }
214
215     // Go to the end of the file, to avoid overwriting existing data
216     lseek(file, 0, SEEK_END);
217
218     // Write aforementioned string to journal file
219     if (!write(file, buffer, str_size)) {
220         printk(KERN_INFO "Write failed");
221     }
222
223     // Close the journal file
224     if (close(file)) {
225         printk(KERN_INFO "Close failed");
226     }
227
228     // Print the data that was written to the journal
229     printk(KERN_INFO "----- File -----\\n"
230           "Inode: %p, Path Dentry: %p, Path Mnt: %p,\\n"
231           "Mnt SB: %p, Mnt Dentry: %p, Flags: %u"
232           "Mode: %u, POS: %lld, Mapping: %p"
233           "Mapping Writable: %lu, Mapping NRPages: %lu\\n\\n",
234           filp->f_inode,
235           filp->f_path.dentry,
236           filp->f_path.mnt,
237           filp->f_path.mnt->mnt_sb,
238           filp->f_path.mnt->mnt_root,
239           filp->f_flags,
240           filp->f_mode,
241           filp->f_pos,
242           filp->f_mapping,
243           filp->f_mapping->i_mmap_writable,
244           filp->f_mapping->nrpages
245     );

```

Στη συνέχεια βρίσκουμε που τελειώνει το string στο buffer (L. 203-213), καθώς το buffer μας είναι αρκετά μεγαλύτερο, ώστε να γνωρίζουμε πόσους χαρακτήρες πρέπει να γράψουμε στο αρχείο. Κατόπιν με τη χρήση της lseek(), πηγαίνουμε στο τέλος του αρχείου (L. 216), ώστε να αποφύγουμε να κάνουμε overwrite τις προηγούμενες εγγραφές. Τέλος ελέγχουμε αν ήταν επιτυχής η εγγραφή στο αρχείο (L. 219-221) και κλείνουμε το αρχείο (L. 224-226). Κατόπιν τυπώνουμε τη καταχώρηση που κάναμε journal παραπάνω και στο τερματικό (L. 229-245).

Output του test στο τερματικό:

Στο τερματικό έχουμε προσθέσει την εκτύπωση των τιμών που εξηγήθηκαν παραπάνω.

Δηλαδή τυπώνουμε μερικά από τα πεδία των Superblock, Inode, Dendry, File και Mount

Συγκεκριμένα το output της κλήσης της τελικής έκδοσης της εργασίας μας είναι:

```
myy601@myy601lab2:~/lkl/lkl-source$ ./test.sh
make: Entering directory '/home/myy601/lkl/lkl-source/tools/lkl'
make -C tests test
for fs in vfat; do ./boot.sh -t $fs || exit 1; done
102400+0 records in
102400+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.167384 s, 626 MB/s
mutex          passed [1]
semaphore      passed [1]
join           passed [joined 139647763490560]
disk_add       passed [3 0]
getpid         passed [1]
syscall_latency  passed [avg/min/max lkl: 108/99/371 native: 108/99/5220]
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 [87838208]
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 kmsq maps misc stat iomem cry]
umount_fs       passed [proc: 0 0 0]
mount_dev        passed [0]
chdir           passed [0]
opendir         passed [4]
getdents64      passed [4 . . . ]
umount_dev      passed [0 0 0]
lo_ifup          passed [0]
gettid          passed [3433]
syscall_thread   passed []
many_syscall_threads  passed []
make: Leaving directory '/home/myy601/lkl/lkl-source/tools/lkl'

!!! CREATING VFAT DEVICE FILE !!!
```

```

mutex          passed [1]
semaphore      passed [1]
join           passed [joined 140425411847936]
disk add       passed [3 0]
[ 0.00000] Linux version 4.11.0 (myy601@myy601lab2) (gcc version 8.3.0 (Debian 8.3.0-6) ) #104 Mon May 23 18:54:30 EEST 2022
[ 0.00000] bootmem address range: 0x7fb7554a6000 - 0x7fb7564a5000
[ 0.00000] On node 0 totalpages: 4095
[ 0.00000] free_area_init_node: node 0, pgdat 55694566b1c0, node_mem_map 7fb7554aa450
[ 0.00000] Normal zone: 56 pages used for memmap
[ 0.00000] Normal zone: 0 pages reserved
[ 0.00000] Normal zone: 4095 pages, LIFO batch:0
[ 0.00000] pcpu-alloc: s0 r0 d32768 u32768 alloc=1*32768
[ 0.00000] pcpu-alloc: [0] 0
[ 0.00000] Built 1 zonelists in Zone order, mobility grouping off. Total pages: 4039
[ 0.00000] Kernel command line: mem=16M loglevel=8 virtio_mmio.device=292@0x1000000:1
[ 0.00000] PID hash table entries: 64 (order: -3, 512 bytes)
[ 0.00000] Dentry cache hash table entries: 2048 (order: 2, 16384 bytes)
[ 0.00000] Inode-cache hash table entries: 1024 (order: 1, 8192 bytes)
[ 0.00000] Memory available: 16028k/0k RAM
[ 0.00000] SLUB: Hwalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[ 0.00000] NR_IRQS:4096
[ 0.00000] lkl: irqs initialized
[ 0.00000] clocksource: lkl: mask: 0xfffffffffffffff max_cycles: 0x1cd42e4dfffb, max_idle_ns: 881590591483 ns
[ 0.00000] lkl: time and timers initialized (irq2)
[ 0.00004] pid_max: default: 4096 minimum: 301
[ 0.00016] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[ 0.00017] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)
[ 0.011482] console [lkl_console0] enabled
[ 0.011526] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[ 0.011674] NET: Registered protocol family 16
[ 0.014055] clocksource: Switched to clocksource lkl
[ 0.014360] NET: Registered protocol family 2
[ 0.014585] TCP established hash table entries: 512 (order: 0, 4096 bytes)
[ 0.014597] TCP bind hash table entries: 512 (order: 0, 4096 bytes)
[ 0.014603] TCP: Hash tables configured (established 512 bind 512)
[ 0.014672] UDP hash table entries: 128 (order: 0, 4096 bytes)
[ 0.014677] UDP-Lite hash table entries: 128 (order: 0, 4096 bytes)
[ 0.014805] virtio-mmio: Registering device virtio-mmio.0 at 0x1000000-0x1000123, IRQ 1.
[ 0.015156] workingset: timestamp_bits=62 max_order=12 bucket_order=0
[ 0.021918] io scheduler noop registered
[ 0.021942] io scheduler deadline registered
[ 0.022025] io scheduler cfq registered (default)
[ 0.022037] io scheduler mq-deadline registered
[ 0.022057] virtio-mmio virtio-mmio.0: Failed to enable 64-bit or 32-bit DMA. Trying to continue, but this might not work.
[ 0.028714] vda:
[ 0.029192] NET: Registered protocol family 10
[ 0.029537] Segment Routing with IPv6
[ 0.029573] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[ 0.029881] Warning: unable to open an initial console.
[ 0.029915] This architecture does not have kernel memory protection.

```

```

getpid          passed [1]
syscall_latency  passed [avg/min/max lkl: 279/260/602 native: 278/250/13895]
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 [87869547]
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 iomem cry]
umount_fs        passed [proc: 0 0 0]
mount_dev         passed [0]
chdir            passed [0]
opendir          passed [4]
getdents64       passed [4 ... ]
umount_dev       passed [0 0 0]
lo_ifup          passed [0]
gettid           passed [3555]
syscall_thread    passed []
many_syscall_threads []
[ 0.135641] reboot: Restarting system

!!! RUNNING TEST !!

[ 0.00000] Linux version 4.11.0 (myy601@myy601lab2) (gcc version 8.3.0 (Debian 8.3.0-6) ) #104 Mon May 23 18:54:30 EEST 2022
[ 0.00000] bootmem address range: 0x7fd545c00000 - 0x7fd546ff000
[ 0.00000] Built 1 zonelists in Zone order, mobility grouping on. Total pages: 25249
[ 0.00000] Kernel command line: mem=100M virtio_mmio.device=292@0x1000000:1
[ 0.00000] PID hash table entries: 512 (order: 0, 4096 bytes)
[ 0.00000] Dentry cache hash table entries: 16384 (order: 5, 131072 bytes)
[ 0.00000] Inode-cache hash table entries: 8192 (order: 4, 65536 bytes)
[ 0.00000] Memory available: 100752k/0k RAM
[ 0.00000] SLUB: Hwalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[ 0.00000] NR_IRQS:4096
[ 0.00000] lkl: irqs initialized
[ 0.00000] clocksource: lkl: mask: 0xfffffffffffffff max_cycles: 0x1cd42e4dfffb, max_idle_ns: 881590591483 ns
[ 0.00000] lkl: time and timers initialized (irq2)
[ 0.00006] pid_max: default: 4096 minimum: 301
[ 0.00022] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[ 0.00023] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)

```

```

[ 0.013471] console [lkl_console0] enabled
[ 0.013505] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[ 0.014393] NET: Registered protocol family 16
[ 0.020508] clocksource: Switched to clocksource lkl
[ 0.021097] NET: Registered protocol family 2
[ 0.024899] TCP established hash table entries: 1024 (order: 1, 8192 bytes)
[ 0.024920] TCP bind hash table entries: 1024 (order: 1, 8192 bytes)
[ 0.024924] TCP: Hash tables configured (established 1024 bind 1024)
[ 0.025550] UDP hash table entries: 128 (order: 0, 4096 bytes)
[ 0.025564] UDP-Lite hash table entries: 128 (order: 0, 4096 bytes)
[ 0.025678] virtio-mmio: Registering device virtio-mmio.0 at 0x1000000-0x1000123, IRQ 1.
[ 0.026112] workqueue: timestamp_bits=62 max_order=15 bucket_order=0
[ 0.032355] io scheduler noop registered
[ 0.032403] io scheduler deadline registered
[ 0.032516] io scheduler cfq registered (default)
[ 0.032542] io scheduler mq-deadline registered
[ 0.032581] virtio-mmio virtio-mmio.0: Failed to enable 64-bit or 32-bit DMA. Trying to continue, but this might not work.
[ 0.040902] vda:
[ 0.041585] NET: Registered protocol family 10
[ 0.043414] Segment Routing with IPv6
[ 0.043478] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[ 0.043885] Warning: unable to open an initial console.
[ 0.043940] This architecture does not have kernel memory protection.
[ 0.045720] Open success: 6
[ 0.045769] ----- Super Block -----
[ 0.045769] Spr blk: 00007fd54bb75800, Blk size: 512, SBCnt: 1,
[ 0.045769] Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
[ 0.045769] RDONLY: 0, ID: vda, UUID: 1270307712
[ 0.045769] ----- Inode -----
[ 0.045769] Current inode: 00007fd54b40e7c0, Size: 0,
[ 0.045769] Byte: 0, Blk bits: 9,
[ 0.045769] Cnt: 1, DIO Cnt: 0, WRCnt: 1
[ 0.045769] ----- Dendry -----
[ 0.045769] Dendry: 00007fd54b40d480, Flags: 2097159, Par: 00007fd54b40d480, Name: /, Inode: 00007fd54b40e550
[ 0.045769]
[ 0.045812] Open success: 6
[ 0.045831] ----- Super Block -----
[ 0.045831] Spr blk: 00007fd54bb75800, Blk size: 512, SBCnt: 1,
[ 0.045831] Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
[ 0.045831] RDONLY: 0, ID: vda, UUID: 1270307712
[ 0.045831] ----- Inode -----
[ 0.045831] Current inode: 00007fd54b40e7c0, Size: 0,
[ 0.045831] Byte: 0, Blk bits: 9,
[ 0.045831] Cnt: 1, DIO Cnt: 0, WRCnt: 1
[ 0.045831] ----- Dendry -----
[ 0.045831] Dendry: 00007fd54b40d480, Flags: 2097159, Par: 00007fd54b40d480, Name: /, Inode: 00007fd54b40e550
[ 0.045831]
[ 0.045887] Open success: 6
[ 0.045905] ----- Super Block -----
[ 0.045905] Spr blk: 00007fd54bb75800, Blk size: 512, SBCnt: 1,

```

```

[ 0.045905] Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
[ 0.045905] RDONLY: 0, ID: vda, UUID: 1270307712
[ 0.045905] ----- Inode -----
[ 0.045905] Current inode: 00007fd54b40e7c0, Size: 4096,
[ 0.045905] Byte: 0, Blk bits: 9,
[ 0.045905] Cnt: 1, DIO Cnt: 0, WRCnt: 1
[ 0.045905] ----- Dendry -----
[ 0.045905] Dendry: 00007fd54b40d480, Flags: 2097159, Par: 00007fd54b40d480, Name: /, Inode: 00007fd54b40e550
[ 0.045905]
[ 0.045933] Open success: 6
[ 0.045955] ----- Super Block -----
[ 0.045955] Spr blk: 00007fd54bb75800, Blk size: 512, SBCnt: 1,
[ 0.045955] Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
[ 0.045955] RDONLY: 0, ID: vda, UUID: 1270307712
[ 0.045955] ----- Inode -----
[ 0.045955] Current inode: 00007fd54b40e7c0, Size: 4096,
[ 0.045955] Byte: 0, Blk bits: 9,
[ 0.045955] Cnt: 1, DIO Cnt: 0, WRCnt: 1
[ 0.045955] ----- Dendry -----
[ 0.045955] Dendry: 00007fd54b40d480, Flags: 2097159, Par: 00007fd54b40d480, Name: /, Inode: 00007fd54b40e550
[ 0.045955]
[ 0.046036] Open success: 6
[ 0.046059] ----- Super Block -----
[ 0.046059] Spr blk: 00007fd54bb75800, Blk size: 512, SBCnt: 1,
[ 0.046059] Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
[ 0.046059] RDONLY: 0, ID: vda, UUID: 1270307712
[ 0.046059] ----- Inode -----
[ 0.046059] Current inode: 00007fd54b40e7c0, Size: 8192,
[ 0.046059] Byte: 0, Blk bits: 9,
[ 0.046059] Cnt: 1, DIO Cnt: 0, WRCnt: 1
[ 0.046059] ----- Dendry -----
[ 0.046059] Dendry: 00007fd54b40d480, Flags: 2097159, Par: 00007fd54b40d480, Name: /, Inode: 00007fd54b40e550
[ 0.046059]
[ 0.046099] Open success: 6
[ 0.046108] ----- Super Block -----
[ 0.046108] Spr blk: 00007fd54bb75800, Blk size: 512, SBCnt: 1,
[ 0.046108] Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
[ 0.046108] RDONLY: 0, ID: vda, UUID: 1270307712
[ 0.046108] ----- Inode -----
[ 0.046108] Current inode: 00007fd54b40e7c0, Size: 8192,
[ 0.046108] Byte: 0, Blk bits: 9,
[ 0.046108] Cnt: 1, DIO Cnt: 0, WRCnt: 1
[ 0.046108] ----- Dendry -----
[ 0.046108] Dendry: 00007fd54b40d480, Flags: 2097159, Par: 00007fd54b40d480, Name: /, Inode: 00007fd54b40e550
[ 0.046108]
[ 0.046155] Open success: 6
[ 0.046173] ----- Super Block -----
[ 0.046173] Spr blk: 00007fd54bb75800, Blk size: 512, SBCnt: 1,
[ 0.046173] Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
[ 0.046173] RDONLY: 0, ID: vda, UUID: 1270307712

```

```

[ 0.046173] ..... Inode .....  

[ 0.046173] Current inode: 00007fd54b40e7c0, Size: 12288,  

[ 0.046173] Byte: 0, Blk bits: 9,  

[ 0.046173] Cnt: 1, DIO Cnt: 0, WRCnt: 1  

[ 0.046173] ..... Dendry .....  

[ 0.046173] Dendry: 00007fd54b40d480, Flags: 2097159, Par: 00007fd54b40d480, Name: /, Inode: 00007fd54b40e550  

[ 0.046173]  

[ 0.046299] Open success: 5  

[ 0.046224] ..... File .....  

[ 0.046224] Inode: 00007fd54b40e7c0, Path Dentry: 00007fd54b40d540, Path Mnt: 00007fd54b82ac80  

[ 0.046224] Mnt Sb: 00007fd54b675800, Mnt Dentry: 00007fd54b40d480, Flags: 32770Mode: 491551, POS: 13958, Mapping: c0:e7:40:4b:d5:7f Writable: 0, Mapping NRPages: 4  

[ 0.046224]  

[ 0.009080] Linux version 4.11.0 (myy60@myy60lab2) (gcc version 8.3.0 (Debian 8.3.0-6) ) #104 Mon May 23 18:54:30 EEST 2022  

[ 0.009080] bootmem address range: 0x7fa941c00000 - 0x7fa947ffff000  

[ 0.009080] built 1 zoneList in Zone order, mobility grouping on. Total pages: 25249  

[ 0.009080] Kernel command line: mem=160M virt=160M mem-reserve=32@0x10000000:1  

[ 0.009080] PID hash table entries: 512 (order: 5, 131072 bytes)  

[ 0.009080] Dentry cache hash table entries: 16384 (order: 5, 4096 bytes)  

[ 0.009080] Inode cache hash table entries: 8192 (order: 4, 65536 bytes)  

[ 0.009080] Memory available: 100752k/8K RAM  

[ 0.009080] SLUB: HwAlign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1  

[ 0.009080] NR IRQS:4996  

[ 0.009080] Ikl: irqs initialized  

[ 0.009080] clocksource: lkl: mask: 0xffffffffffffffff max_cycles: 0x1cd42e4dff, max_idle_ns: 881590591483 ns  

[ 0.009080] Ikl: time and timers initialized (irq2)  

[ 0.009080] pid max: default: 4096 minimum: 301  

[ 0.009055] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)  

[ 0.009058] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)  

[ 0.008731] console [/dev/console0] enabled  

[ 0.008966] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns  

[ 0.009304] NET: Registered protocol family 16  

[ 0.012378] clocksource: Switched to clocksource lkl  

[ 0.012641] NET: Registered protocol family 2  

[ 0.012944] TCP established hash table entries: 1024 (order: 1, 8192 bytes)  

[ 0.012974] TCP bind hash table entries: 1024 (order: 1, 8192 bytes)  

[ 0.012985] TCP: Hash tables configured (established 1024 bind 1024)  

[ 0.013460] UDP hash table entries: 128 (order: 0, 4096 bytes)  

[ 0.013479] UDP-Lite hash table entries: 128 (order: 0, 4096 bytes)  

[ 0.013683] virtio-mmio: Registering device virtio-mmio.0 at 0x1000000-0x1000123, IRQ 1.  

[ 0.014382] workingset: timestamp_bits=62 max_order=15 bucket_order=0  

[ 0.020173] io scheduler noop registered  

[ 0.020198] io scheduler deadline registered  

[ 0.020270] io scheduler cfq registered (default)  

[ 0.020283] io scheduler mq-deadline registered  

[ 0.020302] virtio-mmio virtio-mmio.0: Failed to enable 64-bit or 32-bit DMA. Trying to continue, but this might not work.  

[ 0.027704] vda:  

[ 0.028477] NET: Registered protocol family 10  

[ 0.029231] Segment Routing with IPv6  

[ 0.029959] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver  

[ 0.029756] Warning: unable to open an initial console.

```

```

[ 0.029813] This architecture does not have kernel memory protection.  

[ 0.031478] Open success: 7  

[ 0.031544] ..... Super Block .....  

[ 0.031544] Spr blk: 00007fa947b75800, Blk size: 512, SBCnt: 1,  

[ 0.031544] Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,  

[ 0.031544] RDONLY: 0, ID: vda UUID: 1203198848  

[ 0.031544] ..... Inode .....  

[ 0.031544] Current inode: 00007fa94740e7c0, Size: 0,  

[ 0.031544] Byte: 0, Blk bits: 9,  

[ 0.031544] Cnt: 1, DIO Cnt: 0, WRCnt: 1  

[ 0.031544] ..... Dendry .....  

[ 0.031544] Dendry: 00007fa94740d480, Flags: 2097159, Par: 00007fa94740d480, Name: /, Inode: 00007fa94740e550  

[ 0.031544]  

[ 0.031635] Open success: 7  

[ 0.031671] ..... Super Block .....  

[ 0.031671] Spr blk: 00007fa947b75800, Blk size: 512, SBCnt: 1,  

[ 0.031671] Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,  

[ 0.031671] RDONLY: 0, ID: vda UUID: 1203198848  

[ 0.031671] ..... Inode .....  

[ 0.031671] Current inode: 00007fa94740e7c0, Size: 0,  

[ 0.031671] Byte: 0, Blk bits: 9,  

[ 0.031671] Cnt: 1, DIO Cnt: 0, WRCnt: 1  

[ 0.031671] ..... Dendry .....  

[ 0.031671] Dendry: 00007fa94740d480, Flags: 2097159, Par: 00007fa94740d480, Name: /, Inode: 00007fa94740e550  

[ 0.031671]  

[ 0.031775] Open success: 6  

[ 0.031798] ..... File .....  

[ 0.031798] Inode: 00007fa94740e7c0, Path Dentry: 00007fa94740d540, Path Mnt: 00007fa94782ac80,  

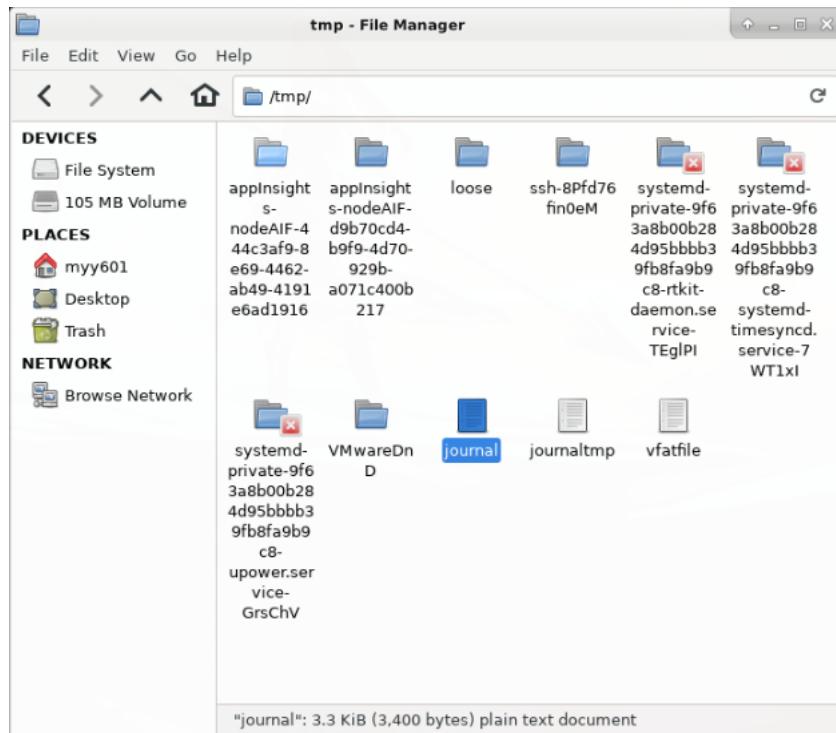
[ 0.031798] Mnt SB: 00007fa947b75800, Mnt Dentry: 00007fa94740d480, Flags: 32770Mode: 491551, POS: 3400, Mapping: c0:e7:40:47:a9:7f Writable: 0, Mapping NRPages: 1  

[ 0.031798]

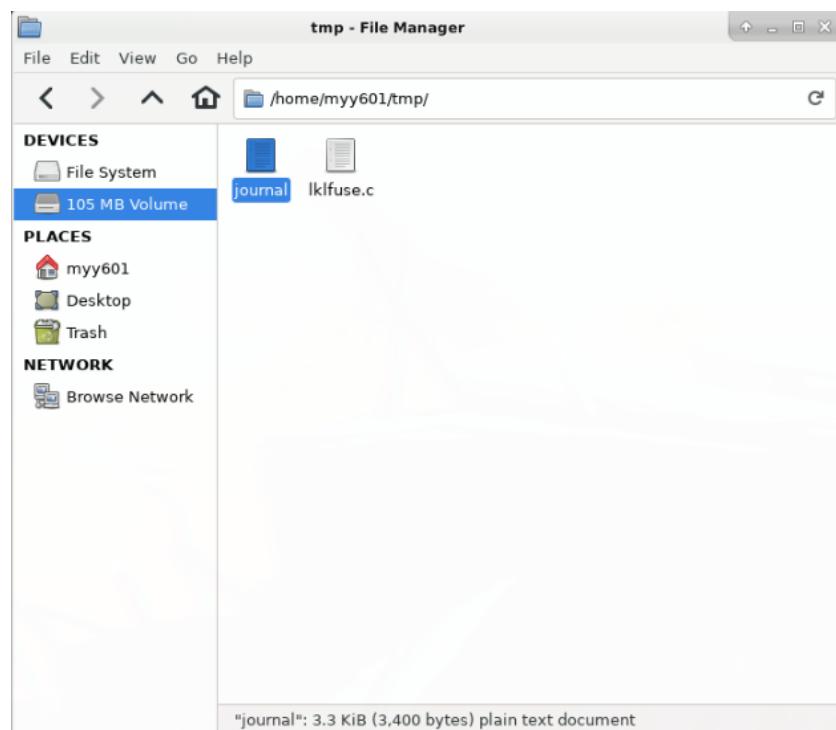
```

Ελέγχουμε πως έχουν δημιουργηθεί
τα ζητούμενα αρχεία journal:

ext4:



vfat device:



Ελέγχουμε, με το “vimdiff -y”, εάν το αρχείο /tmp/journal αντιγράφηκε σωστά στο root του vfat device.

```

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 0,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 0,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 4096,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 4096,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 4096,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 8192,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007
journalex4 [+]
41,22 Top journalvfat 41,22 Top
----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 4096,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 4096,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 8192,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007
journalex4 [+]
41,22 Top journalvfat 41,22 Top
----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 4096,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 4096,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 12288,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007

----- File -----
Inode: 00007ff3b40e7c0, Path Dentry: 00007ff3b40d540, Path Mnt: 00007ff3b82ac80,
Mnt SB: 00007ff3bb75800, Mnt Dentry: 00007ff3b40d480, Flags: 32770Mode: 491551, POS: 0
----- Super Block -----
Spr blk: 00007ff3bb75800, Blk size: 512, SBCnt: 1,
Max Bytes: 4294967295, Flags: 1879050240, Stack Dep: 0,
RONLY: 0, ID: vda, UUID: 1001872256
----- Inode -----
Current inode: 00007ff3b40e7c0, Size: 12288,
Byte: 0, Blk bits: 9,
Cnt: 1, DIO Cnt: 0, WRCnt: 1
----- Dendry -----
Dendry: 00007ff3b40d480, Flags: 2097159, Par: 00007ff3b40d480, Name: /, Inode: 00007
----- File -----
Inode: 00007ff3b40e7c0, Path Dentry: 00007ff3b40d540, Path Mnt: 00007ff3b82ac80,
Mnt SB: 00007ff3bb75800, Mnt Dentry: 00007ff3b40d480, Flags: 32770Mode: 491551, POS: 0
journalex4 [+]
89,1 Bot journalvfat 89,1 Bot
-- INSERT --

```

Σύμφωνα με το vimdiff τα αρχεία είναι ίδια.