ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΤΗΡΙΟ 2

Καραγιαννίδης Χρήστος , Α.Μ : 4375

Χριστόπουλος Κων/νος , Α.Μ : 4527

Αρχικά , πήγαμε στο αρχείο fs.h και προσθέσαμε ως πεδίο του struct του superblock το int myFileDescriptor το οποίο είχε ως σκοπό να αποθηκεύεται εκεί το file descriptor του αρχείου journal.txt

/home/myy601/lkl/lkl-source/include/linux/fs.h: (line 1293)

```
struct super_block {
struct list_head     s_list;        /* Keep this first */
dev_t                s_dev;         /* search index; _not_ kdev_t */
unsigned char        s_blocksize_bits;
unsigned long        s_blocksize;
loff_t               s_maxbytes;/* Max file size */
struct file_system_type    *s_type;
const struct super_operations    *s_op;
const struct dquot_operations    *dq_op;
const struct quotactl_ops*s_qcop;
const struct export_operations *s_export_op;
unsigned long        s_flags;
```

```
unsigned long           s_iflags;     /* internal SB_I_* flags */
unsigned long           s_magic;
struct dentry           *s_root;
struct rw_semaphore     s_umount;
int                     s_count;
int myFileDescriptor;                 /*Dhlwsh metablhths fd*/
atomic_t                s_active;
…
```

Έπειτα, η δομή struct super-block δημιουργείται από την μέθοδο alloc_super() στην μνήμη όταν το file system γίνεται mount. Επομένως δοκιμάσαμε εδώ να συσχετίσουμε την global μεταβλητή int FileDescriptor του mount.c με το πεδίο myFileDescriptor του superblock όμως δεν είχαμε το επιθυμητό αποτέλεσμα και γι' αυτό βάλαμε σε εκείνα τα μέρη του κώδικα σε σχόλια.(παρακάτω δίνεται και ο κώδικας που γράψαμε στο αρχείο mount.c)

/home/myy601/lkl/lkl-source/fs/super.c :

```
//#include "/home/myy601/lkl/lkl-source/fs/sysfs/mount.c"
static struct super_block *alloc_super(struct file_system_type *type, int flags,
struct user_namespace *user_ns){
struct super_block *s = kzalloc(sizeof(struct super_block),  GFP_USER);
static const struct super_operations default_op;
int i;
//s->myFileDescriptor = fileDescriptor;
```

```
//printk(KERN_INFO "ALLOC SUPER FILE DESCRIPTOR : %d \n", s->myFileDescriptor);
if (!s)
return NULL;
INIT_LIST_HEAD(&s->s_mounts);
s->s_user_ns = get_user_ns(user_ns);
…
```

προσπαθήσαμε σε αυτό το αρχείο(mount.c) να ανοίξουμε το αρχείο journal και να αποθηκεύσουμε την τιμή που επιστρέφει η sys_open() σε μια global μεταβλητή int FileDescriptor.

/home/myy601/lkl/lkl-source/fs/sysfs/mount.c

```
int fileDescriptor;     //global variable

static struct dentry *sysfs_mount(struct file_system_type *fs_type,
int flags, const char *dev_name, void *data){
struct dentry *root;
void *ns;
bool new_sb;

int byteNumber;
printk(KERN_INFO "mpika stin sysfs_mount");
fileDescriptor = sys_open("/Journal.txt", O_CREAT | O_RDWR, S_IRWXU | S_IRWXG | S_IRWXO);
printk(KERN_INFO "fd = %d", fileDescriptor);
if (!(flags & MS_KERNMOUNT)) {
```

```
if (!kobj_ns_current_may_mount(KOBJ_NS_TYPE_NET))
return ERR_PTR(-EPERM);
…
}
```

Το πρόβλημα που αντιμετωπίσαμε εδώ ήταν ότι ενώ η sys_open() επέστρεφε θετική τιμή, όταν προσπαθήσαμε να γράψουμε χρησιμοποιώντας το fd που είχαμε αποθηκεύσει μέσα στο superblock , μας χτυπούσε segmentation fault!

Η /home/myy601/lkl/lkl-source/fs/fat/namei_vfat.c καλεί την :

/home/myy601/lkl/lkl-source/fs/fat/inode.c

Δοκιμάσαμε να ανοίξουμε το αρχείο journal και εδώ μέσα στην συνάρτηση fat_fill_super().

```
sb->myFileDescriptor = sys_open("/journal.txt", O_CREAT | O_RDWR, S_IRWXU | S_IRWXG | S_IRWXO);
printk(KERN_INFO " sb->myFileDescriptor = %d", sb->myFileDescriptor);
```

αλλά όταν πηγαίναμε σε κάποιες άλλες συναρτήσεις να γράψουμε μέσα στο αρχείο αυτό μας χτυπούσε segmentation fault όπως φαίνεται παρακάτω.(Στο αρχείο fs/fat/fatent.c)

- η int check_if_writes, επιστρέφει τον αριθμό των bytes που γράφτηκαν στο αρχείο

```
static void fat_ent_blocknr(struct super_block *sb, int entry,
int *offset, sector_t *blocknr)
{
printk(KERN_INFO "mphke fat_ent_blocknr ");
struct msdos_sb_info *sbi = MSDOS_SB(sb);
int bytes = (entry << sbi->fatent_shift);
WARN_ON(entry < FAT_START_ENT || sbi->max_cluster <= entry);
```

```
*offset = bytes & (sb->s_blocksize - 1);
printk(KERN_INFO "*offset = (bytes & (sb->s_blocksize - 1)) = %d", *offset);
/*check_if_writes = sys_write(sb->myFileDescriptor, *offset, sizeof(int));
sys_fsync(sb->myFileDescriptor);
sys_fdatasync(sb->myFileDescriptor);
printk(KERN_INFO "check_if_writes = %d ", check_if_writes);*/

*blocknr = sbi->fat_start + (bytes >> sb->s_blocksize_bits);
printk(KERN_INFO "*blocknr = (sbi->fat_start + (bytes >> sb->s_blocksize_bits)) =
%d", *blocknr);
/*check_if_writes = sys_write(sb->myFileDescriptor, *blocknr, sizeof(sector_t));
sys_fsync(sb->myFileDescriptor);
sys_fdatasync(sb->myFileDescriptor);
printk(KERN_INFO "check_if_writes = %d ", check_if_writes);*/
…
…
```

Printk

Χρησιμοποιήσαμε αρκετές printk() και σε πολλά αρχεία του κώδικα προκειμένου να καταλάβουμε την σειρά εκτέλεσης των αρχείων , την σειρά κλήσεων των συναρτήσεων, την λειτουργεία των συναρτήσεων καθώς επίσης και για διάφορες τιμές μεταβλητών και πεδίων από structs.

Αναλυτικά όλα τα αρχεία που προσθέσαμε printk():

/home/myy601/lkl/lkl-source/arch/lkl/mm/bootmem.c

/home/myy601/lkl/lkl-source/fs/fat/fatent.c

/home/myy601/lkl/lkl-source/fs/fat/inode.c

/home/myy601/lkl/lkl-source/fs/fat/namei_vfat.c

/home/myy601/lkl/lkl-source/fs/fat/file.c

/home/myy601/lkl/lkl-source/fs/fat/cache.c

/home/myy601/lkl/lkl-source/fs/sysfs/mount.c

/home/myy601/lkl/lkl-source/fs/super.c

/home/myy601/lkl/lkl-source/tools/lkl/cptofs.c

/home/myy601/lkl/lkl-source/tools/lkl/tests/boot.c

(Δεν βάζουμε copy paste κώδικα από τις printk() αυτών των αρχείων γιατί μετά το report θα γίνει πολλών σελίδων.)

Ενδεικτικά :

- /home/myy601/lkl/lkl-source/arch/lkl/mm/bootmem.c

  Στην συνάρτηση *void__init bootmem_init(unsigned long mem_size)* σύμφωνα με το mem_size που δέχεται ως όρισμα υπολογίζει και εκτυπώνει το address range.

  Στην συνάρτηση void__init mem_init (void) υπολογίζεται και εκτυπώνεται η διαθέσιμη μνήμη.

```
myy601@myy601lab2:~/lkl/lkl-source/tools/lkl$ tests/boot -t vfat -d /tmp/vfatfile -p -P 0
mutex                passed [1]
semaphore            passed [1]
join                 passed [joined 140496089454336]
disk_add             passed [3 0]
[    0.000000] Linux version 4.11.0 (myy601@myy601lab2) (gcc version 8.3.0 (Debian 8.3.0-6) ) #51 Thu May 13 22:19:41 EEST 20
[    0.000000] mphke bootmem_init ------------------------------------------------------------------
[    0.000000]          mem_size = 16777216
[    0.000000] bootmem address range: 0x7fc7ca011000 - 0x7fc7cb010000
[    0.000000] On node 0 totalpages: 4095
[    0.000000] free_area_init_node: node 0, pgdat 55d0a19ae500, node_mem_map 7fc7ca0133b8
[    0.000000]   Normal zone: 56 pages used for memmap
[    0.000000]   Normal zone: 0 pages reserved
[    0.000000]   Normal zone: 4095 pages, LIFO batch:0
[    0.000000] pcpu-alloc: s0 r0 d32768 u32768 alloc=1*32768
[    0.000000] pcpu-alloc: [0] 0
[    0.000000] Built 1 zonelists in Zone order, mobility grouping off.  Total pages: 4039
[    0.000000] Kernel command line: mem=16M loglevel=8 virtio_mmio.device=292@0x1000000:1
[    0.000000] PID hash table entries: 64 (order: -3, 512 bytes)
[    0.000000] Dentry cache hash table entries: 2048 (order: 2, 16384 bytes)
[    0.000000] Inode-cache hash table entries: 1024 (order: 1, 8192 bytes)
[    0.000000] mphke mem_init---------------------------------------------------------------
[    0.000000]          max_mapnr = 4503565326573551
[    0.000000]          free_all_bootmem() : will put all memory onto the freelists
[    0.000000] Memory available: 16028k/0k RAM
```

  Αυτες οι 2 καλούνται παντα στην αρχη των κλησεων *tests/boot -t vfat -d /tmp/vfatfile -p -P 0* και *./cptofs -i /tmp/vfatfile -p -t vfat lklfuse.c*.

- /home/myy601/lkl/lkl-source/fs/fat/inode.c

Στην συνάρτηση fat_fill_super() καλείται η calc_fat_clusters() στην οποία υπολογίζεται το πεδίο cluster_size του superblock , το συνολο των διαθέσιμων sectors , το σύνολο των clusters καθώς επίσης και το πόσα sectors αντιστοιχούν σε κάθε cluster και όλα αυτά επιστρέφονται στην fat_fill_super().

- Το παρακάτω screenshot “τραβήχτηκε” κατά την εκτέλεση της εντολής *tests/boot -t vfat -d /tmp/vfatfile -p -P 0* . Παρατηρούμε, λοιπόν, ότι το σύστημα εκτελεί πρώτα τη συνάρτηση sysfs_init() η οποία βρίσκεται στο αρχείο mount.c του καταλόγου /fs/sysfs και στην συνέχεια την alloc_super() – η οποία βρίσκεται στο αρχείο super.c του καταλόγου /fs (όπου εκεί βρίσκεται η υλοποίηση του Virtual Filesystem και των συστημάτων αρχείων μέσα στους υποφακέλους) και μετά μόλις ολοκληρωθούν αυτές οι εκτελέσεις θα “τρέξουν” συναρτήσεις η οποίες εδράζονται μέσα στον φάκελο /fs/fat (συγκεκριμένα οι init_fat_fs() και fat_init_inodecache()) . Αυτό αποδεικνύεται και από τις διαφάνειες του φροντιστηρίου (Linux Internals, Filesystems, Linux Kernel Library, σελίδα 5), όπου παρουσιάζεται η αρχιτεκτονική του Linux Kernel και αιτιολογεί την πορεία “..System Call Interface -> Virtual Filesystem -> File Systems.. “

```
[    0.000000] lkl: time and timers initialized (irq2)
[    0.000007] pid_max: default: 4096 minimum: 301
[    0.000021] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[    0.000022] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)
[    0.000025] sysfs_init @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[    0.000065] mpainei alloc_super() @@@@@@@@@@@@@@@@@@@@@@@@@
[    0.000126] mpainei alloc_super() @@@@@@@@@@@@@@@@@@@@@@@@@
[    0.000191] mpainei alloc_super() @@@@@@@@@@@@@@@@@@@@@@@@@
[    0.000264] mpainei alloc_super() @@@@@@@@@@@@@@@@@@@@@@@@@
[    0.002220] mpainei alloc_super() @@@@@@@@@@@@@@@@@@@@@@@@@
[    0.005343] console [lkl_console0] enabled
[    0.005357] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[    0.005416] mpainei alloc_super() @@@@@@@@@@@@@@@@@@@@@@@@@
[    0.005469] NET: Registered protocol family 16
[    0.008742] clocksource: Switched to clocksource lkl
[    0.008808] mpainei alloc_super() @@@@@@@@@@@@@@@@@@@@@@@@@
[    0.008814] mpainei alloc_super() @@@@@@@@@@@@@@@@@@@@@@@@@
[    0.008910] NET: Registered protocol family 2
[    0.009043] TCP established hash table entries: 512 (order: 0, 4096 bytes)
[    0.009075] TCP bind hash table entries: 512 (order: 0, 4096 bytes)
[    0.009078] TCP: Hash tables configured (established 512 bind 512)
[    0.009126] UDP hash table entries: 128 (order: 0, 4096 bytes)
[    0.009138] UDP-Lite hash table entries: 128 (order: 0, 4096 bytes)
[    0.009220] virtio-mmio: Registering device virtio-mmio.0 at 0x1000000-0x1000123, IRQ 1.
[    0.010471] workingset: timestamp_bits=62 max_order=12 bucket_order=0
[    0.011573] mphke_init_fat_fs ############################
[    0.011574] mphke_fat_cache_init-----------------------------
[    0.011592]          fat_cache_cachep = -903113632 -----------------------------
[    0.011594] mphke_fat_init_inodecache ############################
[    0.011606]          fat_inode_cachep = 140496067073344 ############################
[    0.011608] err = 0 (auto pou epestrepse h fat_init_inodecache())
[    0.023081] io scheduler noop registered
[    0.023092] io scheduler deadline registered
[    0.023130] io scheduler cfq registered (default)
[    0.023132] io scheduler mq-deadline registered
[    0.023141] virtio-mmio virtio-mmio.0: Failed to enable 64-bit or 32-bit DMA.  Trying to continue, but this might not work.
[    0.029088]  vda:
```