Laborator 9 - Drivere de sisteme de fişiere (Linux) partea 1

Obiectivele laboratorului

- Dobândirea de cunoştinţe legate de VFS, montarea unui sistem de fişiere, superbloc, diversele suporturi posibile pentru sisteme de fişiere.
- Cunoştinţe legate de diferenţele între drivere-le pentru sisteme de fişiere cu suport fizic(disc în acest caz) şi sisteme de fişiere virtuale.

Cuvinte cheie

- VFS
- superbloc
- inode
- dentry
- file
- get_sb/kill/sb
- buffer_head

Materiale ajutătoare

- Slide-uri de suport pentru laborator
- SO2 Reference Card

Virtual Filesystem

Virtual Filesystem este o componentă a nucleului care se ocupă de tratarea tuturor apelurilor de sistem relative la sisteme de fişiere. VFS este o interfaţă între utilizator şi un sistem de fişiere particular care simplifică implementarea acestuia din urmă şi oferă o integrare facilă a mai multor sisteme de fişiere. În acest fel, implementarea unui sistem de fişiere este realizată prin folosirea API-ului pus la dispoziţie de VFS, părţile generice de comunicaţie cu dispozitivul hardware şi subsistemul de I/O fiind rezolvate de acesta.

Din punct de vedere funcţional sistemele de fişiere pot fi grupate în:

- sisteme de fişiere pentru disc (ext2, ext3, reiserfs, fat, ntfs, etc.)
- sisteme de fişiere pentru reţea (nfs, smbfs, ncp, etc.)
- sisteme de fişiere virtuale (procfs, sysfs, sockfs, pipefs, etc.)

Fiecare sistem de fişiere este de obicei montat de pe mediul pentru care a fost construit. În particular însă, VFS-ul poate folosi drept dispozitiv de tip bloc virtual un fişier normal, deci se pot monta sisteme de fişiere pentru disc peste fişiere normale.

Ideea de bază a VFS-ului este de a oferi un singur model de fişier, care să poată reprezenta fişierele din orice sistem de fişiere. Driver-ul de sistem de fişiere este responsabil pentru aducerea la numitorul comun. Astfel se poate crea o singură structură de directoare care conţine întreg sistemul. Va exista un sistem de fişiere care va fi rădăcina, restul fiind montate în diverse directoare ale acestuia.

Modelul general al sistemului de fişiere

Modelul general al sistemului de fişiere, la care trebuie să se reducă orice sistem de fişiere implementat, este format din mai multe entităţi cu rol bine definit: superbloc, inode, file şi dentry. Aceste entităţi sunt metadatele sistemului de fişiere (conţin informaţii despre date sau despre alte metadate).

Entităţile modelului interacţionează cu ajutorul unor subsisteme ale VFS sau ale nucleului: cache-ul de dentry-uri, cache-ul de inode-uri, buffer cache-ul. Fiecare entitate este tratată ca un obiect: are o structură de date asociată şi un pointer la o tabelă de metode. Inducerea unui comportament particular al fiecărei componente este făcut prin înlocuirea metodelor asociate.

superbloc

superblocul stochează informaţiile necesare unui sistem de fişiere montat:

- zonele de inode-uri, de blocuri
- dimensiunea blocului sistemului de fişiere
- lungimea maximă a numelor fişierelor
- dimensiunea maximă a fişierelor
- locaţia inode-ului rădăcină

Localizare:

- În cazul sistemelor de fişiere pentru discuri, superblocul are un corespondent în primul bloc al acestora (Filesystem Control Block).
- În **VFS**, toate superbloc-urile sistemelor de fişiere sunt reţinute într-o listă de structuri struct super_block şi metodele în struct super_operations

inode

inode-ul (**index node**) menţine informaţii despre un fişier în sensul general (abstractizare): fişier obişnuit (regular file), director, fişier special (pipe, fifo), dispozitiv de tip bloc, dispozitiv de tip caracter, link, sau orice poate fi abstractizat ca fişier.

Un inode menţine informaţii precum:

- tipul fişierului;
- dimensiunea fişierului;
- drepturile de acces;
- timpul de acces sau modificare;
- poziţionarea datelor pe disc (pointeri către blocurile de pe disc ce conţin date).

 În general, inode-ul nu deţine numele fişierului. Numele este reţinut de entitatea dentry. Astfel, un inode poate avea mai multe nume (hardlink-uri).

Localizare:

- La fel ca şi superblocul, inode-ul are un corespondent pe disc. inode-urile de pe disc sunt, în general, grupate într-o zonă specializată (zonă de inode-uri), separată de zona de blocuri de date; în unele sisteme de fişiere, structurile echivalente inode-urilor sunt răspândite în structura sistemului de fişiere (FAT);
- ca entitate **VFS**, inode-ul este reprezentat de structura struct inode şi de operaţiile cu aceasta definite în structura struct inode_operations.

Fiecare inode este în general identificat de un număr. Pe Linux, argumentul `-i` la comanda `ls` precizează numărului inode-ului asociat fişierului:

```
razvan@valhalla:~/school/2008-2009/so2/wiki$ ls -i
1277956 lab10.wiki  1277962 lab9.wikibak  1277964 replace_lxr.sh
1277954 lab9.wiki   1277958 link.txt      1277955 tema4.wiki
```

file

file este componenta din modelul general al sistemului de fişiere care se apropie cel mai mult de utilizator. Structura există doar ca entitate VFS în memorie şi nu are corespondent fizic pe disc.

În vreme ce inode-ul abstractizează un fişier situat pe disc, file-ul abtractizează un fişier deschis. Din punctul de vedere al procesului, entitatea file abstractizează fişierul. Din punctul de vedere al implementării sistemului de fişiere, însă, inode-ul este entitatea care abstractizează fişierul.

Structura file menţine informaţii precum:

- poziţia cursorului de fişier (file pointer);
- drepturile de deschidere ale fişierului;
- pointer către inode-ul asociat (eventual indexul acestuia).

Localizare:

- Structura struct file reprezintă entitatea VFS asociată, iar structura struct file_operations operaţiile cu aceasta.

dentry

dentry (directory entry) realizează asocierea între un inode şi numele fişierului.

În general o structură dentry conţine două câmpuri:

- un întreg care identifică inode-ul;
- un şir de caractere reprezentând numele acestuia.

dentry reprezintă o componentă specifică dintr-o cale, care poate fi un director sau un fişier. Spre exemplu, pentru calea /bin/vi, vor fi create obiecte dentry pentru /, bin şi vi (un total de 3 obiecte dentry).

- dentry are un corespondent pe disc, dar corespondenţa nu este directă deoarece fiecare sistem de fişiere păstrează dentry-urile într-un mod specific
- în VFS, entitatea dentry este reprezentată de structura struct dentry şi de operaţiile cu aceasta definite în structura struct dentry_operations.

Înregistrarea şi deînregistrarea sistemelor de fişiere

În versiunea actuală, kernel-ul Linux are suport pentru un număr în jur de 50 de sisteme de fişiere, dintre care:

- ext2/ ext3
- reiserfs
- xfs
- fat
- ntfs
- iso9660
- udf pentru cd-uri şi dvd-uri
- hpfs

Pe un singur sistem, însă, este puţin probabil să existe mai mult de 5-6 sisteme de fişiere. Din acest motiv, sistemele de fişiere (sau, mai corect, tipurile de sisteme de fişiere) sunt implementate ca module şi pot fi încărcate sau descărcate oricând.

Pentru a putea încărca / descărca în mod dinamic un modul de sistem de fişiere este necesar un API de înregistrare / deînregistrare a tipului sistemului de fişiere în / din sistem. Structura care descrie un anumit sistem de fişiere este struct file_system_type:

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

struct file_system_type {
        const char *name;
        int fs_flags;
        int (*get_sb) (struct file_system_type *, int,
                       const char *, void *, struct vfsmount *);
        void (*kill_sb) (struct super_block *);
        struct module *owner;
        struct file_system_type * next;
        struct list_head fs_supers;
        struct lock_class_key s_lock_key;
        struct lock_class_key s_umount_key;
        //...
};
```

- `name` este şirul de caractere prin care numele va identifica un sistem de fişiere (argumentul dat la `mount -t`).

- `owner` este `THIS_MODULE` pentru sisteme de fişiere implementate în module, şi `NULL` dacă sunt scrise direct în kernel.
- Funcţia `get_sb` citeşte superblocul de pe disc în memorie la încărcarea sistemului de fişiere. Funcţia este proprie fiecărui sistem de fişiere. Pentru mai multe detalii, citiţi Secţiunea Funcţiile get_sb kill_sb .
- Funcţia `kill_sb` eliberează superblocul din memorie, citiţi Secţiunea Funcţiile get_sb kill_sb .
- `fs_flags` precizează flag-urile cu care trebuie montat sistemul de fişiere. Un exemplu de flag este FS_REQUIRES_DEV care precizează VFS-ului că sistemul de fişiere are nevoie de un disc (nu este un sistem de fişiere virtual).
- `fs_supers` este o listă ce conţine toate superblocurile asociate acestui sistem de fişiere. Dat fiind că acelaşi tip de sistem de fişiere poate fi montat de mai multe ori, pentru fiecare mount va exista un superbloc separat.

Înregistrarea unui sistem de fişiere în sistem se realizează, în general, în funcţia de iniţializare a modulului. Pentru înregistrare, programatorul va trebui să

1. iniţializeze o structură de tipul struct file_system_type cu numele, flag-urile, funcţia care implementează operaţia de citire a superblocului şi referinţa la structura ce identifică modulul curent
2. apeleze funcţia register_filesystem.

La descărcarea modulului trebuie să se deînregistreze sistemul de fişiere prin apelarea funcţiei unregister_filesystem.

Un exemplu de înregistrare / deînregistrare a unui sistem de fişiere virtual se găseşte în codul pentru ramfs:

```
static struct file_system_type ramfs_fs_type = {
        .name           = "ramfs",
        .get_sb         = ramfs_get_sb,
        .kill_sb        = kill_litter_super,
};

static int __init init_ramfs_fs(void)
{
        return register_filesystem(&ramfs_fs_type);
}

static void __exit exit_ramfs_fs(void)
{
        unregister_filesystem(&ramfs_fs_type);
}
```

Funcţiile get_sb, kill_sb

La montarea sistemului de fişiere, nucleul apelează funcţia get_sb definită în cadrul structurii struct file_system_type. Funcţia face un set de iniţializări şi returnează un superbloc (structura struct super_block). De obicei, `get_sb` este o funcţie simplă care apelează una din funcţiile:

- get_sb_bdev, care montează un sistem de fişiere aflat pe un device de tip bloc
- get_sb_single, care montează un sistem de fişiere care partajează o instanţă între toate operaţiile de montare
- get_sb_nodev, care montează un sistem de fişiere ce nu se afla pe un device fizic
- get_sb_pseudo, o funcţie ajutătoare pentru pseudo-sistemele de fişiere (`sockfs`, `pipefs`, în general sisteme de fişiere care nu pot fi montate)

Aceste funcţii primesc ca parametru un pointer spre o funcţie fill_super care va fi apelată după iniţializarea superblocului pentru terminarea iniţializării acestuia de către driver.

La demontare nucleul apelează kill_sb, care face operaţii de tip cleanup şi apelează una din funcţiile:

- kill_block_super, care demontează un sistem de fişiere aflat pe un device de tip bloc
- kill_anon_super, care demontează un sistem de fişiere virtual (informaţia este generată în momentul în care este cerută)
- kill_litter_super, care demontează un sistem de fişiere ce nu se afla pe un device fizic (menţine informaţia în memorie)

Un exemplu pentru un sistem de fişiere fără suport pe disc este funcţia ramfs_get_sb din sistemul de fişiere ramfs:

```
int ramfs_get_sb(struct file_system_type *fs_type,
        int flags, const char *dev_name, void *data, struct vfsmount *mnt)
{
        return get_sb_nodev(fs_type, flags, data, ramfs_fill_super, mnt);
}
```

Un exemplu pentru un sistem de fişiere pentru disc este funcţia minix_get_sb din sistemul de fişiere minix:

```
static int minix_get_sb(struct file_system_type *fs_type,
         int flags, const char *dev_name, void *data, struct vfsmount *mnt)
{
```

```
        return get_sb_bdev(fs_type, flags, dev_name, data, minix_fill_super, mnt);
}
```

Superblocul în VFS

Superblocul există atât ca entitate fizică (entitate pe disc) cât şi ca entitate VFS (în cadrul structurii struct super_block). Superblocul conţine numai metainformaţie şi este folosit pentru scrierea şi citirea de metainformaţii de pe disc (inode-uri, directory entries). Un superbloc (şi implicit structura struct super_block) va conţine informaţii despre dispozitivul utilizat, lista de inode-uri, pointer-ul la inode-ul rădăcină al sistemului de fişiere şi un pointer la operaţiile de superbloc.

Structura struct super_block

O parte din definiţia structurii struct super_block este prezentată mai jos:

```
struct super_block {
        //...
        dev_t                   s_dev;              /* identifier */
        unsigned long           s_blocksize;        /* block size in bytes */
        unsigned char           s_blocksize_bits;   /* block size in bits */
        unsigned char           s_dirt;             /* dirty flag */
        unsigned long long      s_maxbytes;         /* max file size */
        struct file_system_type *s_type;            /* filesystem type */
        struct super_operations *s_op;              /* superblock methods */
        //...
        unsigned long           s_flags;            /* mount flags */
        unsigned long           s_magic;            /* filesystem?s magic number */
        struct dentry           *s_root;            /* directory mount point */
        //...
        char                    s_id[32];           /* informational name */
        void                    *s_fs_info;         /* filesystem private info */
};
```

Superblocul memorează informaţia globală pentru o instanţă a unui sistem de fişiere:

- dispozitivul fizic pe care rezidă
- dimensiunea blocului
- dimensiunea maximă a unui fişier
- tipul sistemului de fişiere
- operaţiile pe care le suportă
- numărul magic (identifică sistemul de fişiere)
- `dentry`-ul directorului rădăcină

În plus, un pointer generic (`void *`) memorează date private sistemului de fişiere. Superblocul poate fi privit ca un obiect abstract căruia îi sunt adăugate date proprii în momentul în care există o implementare concretă.

Operaţiile pe superbloc

Operaţiile pe superbloc sunt descrise de structura struct super_operations:

```
struct super_operations {
       //...
       void (*read_inode) (struct inode *);
       void (*write_inode) (struct inode *, int);
       void (*delete_inode) (struct inode *);
       void (*clear_inode)(struct inode *);
       void (*put_super) (struct super_block *);
       void (*write_super) (struct super_block *);
       int (*statfs) (struct dentry *, struct kstatfs *);
       int (*remount_fs) (struct super_block *sb, int *flags, char *data);
       //...
};
```

Câmpurile structurii sunt pointeri de funcţii cu următoarele semnificaţii:

- `read_inode, write_inode, delete_inode, clear_inode` citeşte, scrie, dealocă, respectiv eliberează resurse asociate unui inode şi sunt descrise în Laboratorul 10;
- `put_super` este apelată în momentul eliberării superblocului, la `umount`; în cadrul acestei funcţii trebuie eliberate orice resurse (în general memorie) din datele private ale sistemului de fişiere, dacă există memorie alocată;
- `write_super` este apelată atunci când nucleul decide să scrie superblocul pe disc (după ce, în prealabil, a verificat câmpul `s_dirt`)

- `remount_fs` este apelată atunci când nucleul a detectat că se încearcă o remontare (flag-ul de montare `MS_REMOUNTM`); cel mai adesea aici trebuie să se detecteze dacă se încearcă o trecere read-only read-write sau viceversa; acest lucru se poate face simplu pentru că se pot accesa flag-urile vechi (în `sb->s_flags`) cât şi cele noi (în `flags`); `data` e un pointer către datele trimise de `mount` ce reprezintă opţiuni specifice sistemului de fişiere;
- `statfs` se apelează atunci când se face un apel de sistem `statfs` (încercaţi `stat ?f` sau `df`); în cazul acestui apel trebuie completate câmpurile structurii struct kstatfs

După cum s-a specificat, funcţia `fill_super` este apelată pentru terminarea iniţializării superblocului. Această iniţializare presupune completarea câmpurilor structurii struct super_block şi iniţializarea inode-ului rădăcină.

Un exemplu de implementare este funcţia ramfs apelată pentru iniţializarea câmpurile din superbloc care au mai rămas de iniţializat:

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

#define RAMFS_MAGIC     0x858458f6

static const struct super_operations ramfs_ops = {
        .statfs         = simple_statfs,
        .drop_inode     = generic_delete_inode,
};

static int ramfs_fill_super(struct super_block * sb, void * data, int silent)
{
        struct inode * inode;
        struct dentry * root;

        sb->s_maxbytes = MAX_LFS_FILESIZE;
        sb->s_blocksize = PAGE_CACHE_SIZE;
        sb->s_blocksize_bits = PAGE_CACHE_SHIFT;
        sb->s_magic = RAMFS_MAGIC;
        sb->s_op = &ramfs_ops;
        sb->s_time_gran = 1;
        inode = ramfs_get_inode(sb, S_IFDIR | 0755, 0);
        if (!inode)
                return -ENOMEM;

        root = d_alloc_root(inode);
        if (!root) {
                iput(inode);
                return -ENOMEM;
        }
        sb->s_root = root;
        return 0;
}
```

În kernel sunt disponibile funcţii generice pentru implementarea operaţiilor cu structurile sistemelor de fişiere. Funcţiile generic_delete_inode şi simple_statfs, folosite în codul de mai sus, sunt astfel de funcţii şi pot fi folosite în implementarea driverelor, daca funcţionalitatea acestora este suficientă.

În mare, funcţia ramfs_fill_super din codul de mai sus setează câteva câmpuri din superbloc, apoi citeşte inode-ul rădăcină şi alocă dentry-ul rădăcină. Citirea inode-ului radăcină se face în funcţia ramfs_get_inode, şi constă din alocarea unui nou inode folosind new_inode şi iniţializarea acestuia. Pentru eliberarea inode-ului se foloseşte iput, iar pentru alocarea dentry-ului rădăcină d_alloc_root.

Un exemplu de implementare pentru un sistem de fişiere pentru disc este funcţia minix_fill_super din sistemul de fişiere minix.
Funcţionalitatea pentru sistemul de fişiere pentru disc este similară cu cea a sistemului de fişiere virtual, cu deosebirea folosirii buffer cache-ului. De asemenea, sistemul de fişiere `minix` păstrează date private de forma struct minix_sb_info. Mare parte a acestei funcţii se ocupa cu iniţializarea acestor date private (care nu sunt incluse în extrasul de cod de mai sus, penru claritate). Datele private sunt alocate folosind funcţia kzalloc şi sunt păstrate în câmpul `s_fs_info` al superblocului.

Funcţiile VFS-ului primesc de obicei ca parametru superblocul, un inode sau/şi un dentry, care conţin un pointer către superbloc, astfel încât aceste date private pot fi accesate uşor.

O scurtă digresiune în buffer cache

Buffer cache-ul este un subsistem în kernel care se ocupă cu citirea şi scrierea blocurilor de pe dispozitivele de tip bloc. Entitatea de bază cu care lucrează buffer cache-ul este struct buffer_head. Câmpurile mai importante din această

structură sunt:

- `b_data`, pointer către o zonă din memorie de unde au fost citite sau unde trebuie scrise date
- `b_size`, dimensiunea buffer-ului
- `b_bdev`, device-ul cu care se lucrează
- `b_blocknr`, numărul blocului de pe device care a fost încărcat sau trebuie să fie salvat pe disc
- `b_state`, starea buffer-ului

Există câteva funcţii importante ce lucrează cu astfel de structuri:

- __bread: citeşte un bloc cu numărul dat şi de dimensiune dată într-o structură `buffer_head`; în caz de succes întoarce un pointer către `buffer_head`, altfel întoarce `NULL`;
- sb_bread: face acelaşi lucru ca şi funcţia precedentă, dimensiunea blocului de citit fiind luată din superbloc, la fel şi dispozitivul de pe care se face citirea;
- mark_buffer_dirty: marchează buffer-ul dirty (setează bitul `BH_Dirty`); buffer-ul va fi scris pe disc la un moment ulterior de timp (din când în când kernel thread-ul `bdflush` se trezeşte şi scrie buffere pe disc);
- brelse: eliberează memoria folosită de buffer, după ce în prealabil a scris buffer-ul pe disc daca era nevoie;
- map_bh: asociază buffer-head-ul cu sectorul corespunzător.

Funcţii şi macro-uri folositoare

Superblocul conţine de obicei o hartă a blocurilor ocupate (de inode-uri, dentry, date) sub forma unui bitmap (vector de biţi). Pentru lucrul cu aceste hărţi se recomandă folosirea următoarelor funcţii:

- find_first_zero_bit, pentru găsirea primului bit zero dintr-o zonă de memorie. Parametrul size semnifică numărul de biţi al zonei în care se face căutarea;
- test_and_set_bit, pentru setarea unui bit şi obţinerea vechii valori;
- test_and_clear_bit, pentru ştergerea unui bit şi obţinerea vechii valori;
- test_and_change_bit, pentru inversarea valorii unui bit şi obţinerea vechii valori.

Pentru verificarea tipului unui inode se pot folosi următoarele macrodefiniţii:

- S_ISDIR(inodei_mode), pentru a verifica dacă inode-ul este un director;
- S_ISREG(inodei_mode), pentru a verifica dacă inode-ul este un fişier clasic (nu de tip link sau device).

Exerciţii

- Folosiţi arhiva de sarcini a laboratorului.
- Punctaj total: **11 puncte**

myfs

- Intraţi în directorul `myfs/` din arhiva de sarcini a laboratorului. Creaţi un modul de kernel care să implementeze un sistem de fişiere simplu. Sistemul de fişiere va fi unul virtual (nu va avea suport fizic). Implementaţi doar operaţiile pe superbloc referitoare la lucrul cu superblocul.
- Punctaj: **4.5 puncte**
- **Hint**:
  - Parcurgeţi TODO-urile asociate fiecărui exerciţiu

1. (**1.5 puncte**) Implementaţi înregistrarea şi deînregistrarea sistemului de fişiere.
   - **Hints:**
     - Definiţi o structură de tip `struct file_system_type`.
     - Folosiţi `"myfs"` pentru numele sistemului de fişiere.
     - Pentru completarea unui superbloc folosiţi `get_sb_nodev`
     - Consultaţi secţiunea Inregistrarea si deinregistrarea sistemelor de fisiere.
   - Inseraţi modulul în kernel.
   - Verificaţi prezenţa modulului inspectând `/proc/filesystems`
   - Descărcaţi modulul din kernel.
2. (**1 punct**) Completaţi câmpurile necesare ale structurii `struct super_block`.

- Nu iniţializaţi inode-ul rădăcină.
- Definiţi o structură `struct super_operations`.
  - Iniţializaţi câmpul drop_inode şi câmpul statfs.
- **Hint:**
  - Consultaţi secţiunea Operaţiile pe superbloc.

3. (**2 puncte**) Creaţi şi iniţializaţi inode-ul rădăcină.
   - **Hints:**
     - Completaţi funcţia `myfs_get_inode`.
     - La montare se verifică dacă inode-ul rădăcină e director, aşa că va trebui să marcaţi inode-ul rădăcină ca director ca să vă reuşească montarea. Asta se face setând câmpul `i_mode` din inode la `S_ISDIR`, **folosiţi** parametru `mode` al funcţiei `myfs_get_inode`. De asemenea, va trebui să iniţializaţi operaţiile pentru inode-ul de tip director; puteţi folosi funcţiile deja implementate în kernel simple_dir_operations şi simple_dir_inode_operations.
     - Pentru completarea câmpurilor modified, create şi access time folosiţi macro-ul `CURRENT_TIME`.
   - Inseraţi modulul în kernel şi montaţi-l. mount -t myfs none /mnt/dir
     - `myfs` este numele sistemului de fişiere
     - `none` specifică faptul că nu se foloseşte nici un device
     - `/mnt/dir` este punctul unde va fi montat.
   - Verificaţi că sistemul de fişiere a fost montat folosind comanda: `cat /proc/mounts`
   - Ce inode are directorul `/mnt/dir`? De ce?
   - Verificaţi statisticile sistemului de fişiere `myfs` cu ajutorul comenzii: `stat -f /mnt/dir`.
   - Intraţi în directorul `/mnt/dir`. Afişaţi conţinutul acestuia cu ajutorul comenzii `ls -la`. Ce intrări aveţi în acest director?
   - Folosiţi `touch` pentru a crea fişierul `a.txt`. Ce observaţi?
   - Demontaţi sistemul de fişiere şi descărcaţi modulul din kernel. umount /mnt/dir
   - Pentru testarea întregii funcţionalităţi puteţi folosi scriptul `test-myfs.sh`. Rulaţi `./test-myfs.sh`.

minfs

- Intraţi în directorul `minfs/` din arhiva de sarcini a laboratorului. Creaţi un modul de kernel care să implementeze un sistem de fişiere simplu. Sistemul de fişiere va avea suport fizic. Implementaţi doar operaţiile pe superbloc referitoare la lucrul cu superblocul.
- Punctaj: **6.5 puncte**
- **Hint:**
  - Parcurgeţi TODO-urile asociate fiecărui exerciţiu

1. (**0.5 puncte**) Implementaţi înregistrarea şi deînregistrarea sistemului de fişiere.
   - Inseraţi modulul în kernel.
   - Verificaţi prezenţa modulului inspectând `/proc/filesystems`
   - Descărcaţi modulul din kernel.
   - **Hints:**
     - Completaţi structura de tip `struct file_system_type`.
     - Pentru completarea unui superbloc folosiţi `get_sb_bdev`
     - Ce valoare trebuie să aibă câmpul `.kill_sb`?
     - Consultaţi secţiunea Înregistrarea şi deînregistrarea sistemelor de fişiere
2. (**2.5 puncte**) Completaţi superblocul şi implementaţi operaţiile pe superbloc.
   - Va trebui să citiţi primul bloc de pe disc (cel cu indexul 0).
   - Folositi structura de tipul `superbloc_info` definită în fişierul sursă. Structura deţine informaţii specifice sistemului de fişiere, care nu se regăsesc în `struct super_block` (în cazul de faţă doar versiunea).
   - **Hints:**
     - Pentru a citi superbloc-ul, folosiţi `sb_bread`. Parcurgeţi secţiunea O scurtă digresiune în buffer cache.
     - Folosiţi macrourile şi structurile din `minfs.h`.
   - **Pentru verificarea funcţionalităţii, pentru citirea inode-ului rădăcină folosiţi funcţia de citire a inode-ului de la exerciţiul cu sistem de fişiere virtual** (`myfs_get_inode`).
   - Formataţi partiţia `/dev/sdb` folosind utilitarul `mk_minfs` compilabil din fişierul `mk_minfs.c` şi fişierul `Makefile.format`. make -f Makefile.format
     ./mk_minfs /dev/sdb
   - Inseraţi modulul, montaţi partiţia `/dev/sdb`, demontaţi-o şi descărcaţi modulul. mount -t minfs /dev/sdb /mnt/dir
3. (**1.5 puncte**) Completaţi funcţiile `alloc_inode` şi `destroy_inode`.
   - Veţi aloca inode-uri de tip `minfs_inode_info`, dar veţi returna doar `vfs_inode` (ca să puteţi accesa informaţiile suplimentare apoi cu macro-urile `list_entry` sau `container_of`.)
   - **Hints:**
     - Funcţia `alloc_inode` este apelată de iget_locked.
     - În funcţia de tipul `alloc_inode` trebuie apelată funcţia inode_init_once pentru iniţializarea inode-ului.

- Recomandăm folosirea `kzalloc` şi `kfree` respectiv în cadrul funcţiilor `alloc_inode` şi `destroy_inode`.
- Pentru a obţine structura `struct minfs_inode_info` în cazul în care cunoaşteţi inode-ul asociat, folosiţi macro-ul `list_entry` sau `container_of`.

4. (**2 puncte**) Citiţi de pe disc inode-ul rădăcină (cel cu indexul 0).
   - iget_locked obţine un inode din inode cache, iar în cazul în care acesta nu există, alocă un nou inode şi pune un lock pe el, lock ce trebuie eliberat după iniţializarea inode-ului cu unlock_new_inode.
   - Trebuie completate informaţiile din inode, citindu-le din al doilea bloc de pe disc (cel cu indexul 1).
   - Trebuie completate câmpurile `alloc_inode` şi `destroy_inode` ale structurii `struct super_operations` cu funcţiile implementate de voi.
   - Completaţi câmpurile corespunzătoare ale inode-ului cu adresele variabilelor simple_dir_operations şi simple_dir_inode_operations pentru inode-urile ce corespund unui director.
   - Pentru testarea întregii funcţionalităţii puteţi folosi script-ul `test-minfs.sh`: `./test-minfs.sh`.
   - **Hints:**
     - Folosiţi sb_bread.

Soluţii

- Soluţii exerciţii laborator 9

Resurse utile

1. Robert Love ? Linux Kernel Development, Second Edition ? Chapter 12. The Virtual Filesystem
2. Understanding the Linux Kernel, 3rd edition - Chapter 12. The Virtual Filesystem
3. Linux Virtual File System (presentation)
4. Understanding Unix/Linux Filesystem
5. Creating Linux virtual filesystems
6. The Linux Documentation Project - VFS
7. The "Virtual File System" in Linux
8. A Linux Filesystem Tutorial
9. The Linux Virtual File System
10. File Systems
11. Documentation/filesystems/vfs.txt
12. Sursele sistemelor de fişiere
    - Sistemul de fişiere procfs (*Process File System*) - sursele sistemului de fişiere procfs şi documentaţia
    - Sistemul de fişiere ramfs (*Random Access Memory Filing System*) - sursele sistemului de fişiere ramfs şi documentaţia
    - Sistemul de fişiere romfs (*Read-Only Memory File System*) - sursele sistemului de fişiere romfs şi documentaţia
    - Sistemul de fişiere minix - sursele sistemului de fişiere minix şi informaţii: Minix Homepage, Operating Systems Design and Implementation, Third Edition - Chapter 5. File Systems
    - Sistemul de fişiere bfs (*UnixWare Boot FileSystem*)- sursele sistemului de fişiere bfs, documentaţia şi informaţii: The BFS filesystem structure, Linux Kernel 2.4 Internals - 3.7. Example Disk Filesystem: BFS