Das VFS und ext2fs

Tanja Pfeiff und Matthias Riese

15. Mai 1998

Inhaltsverzeichnis

1 Das VFS

1.1 Funktion des VFS

Das VFS (virtual filesystem switch) ist ein generisches filesystem: Es ist ein Rahmen, in den jedes konkerte filesystem hineinpaßt. Einige Anfragen an ein filesystem, die für alle filesystems gleich bearbeitet werden, beantwortetet das VFS selbst. Ansonsten hat das VFS für alle Operationen auf einem konkreten filesystem Zeiger auf Funktionen dieses filesystems. Über diese Zeiger werden die entsprechenden Funktionen dann aufgerufen. Wegen diesem Mechanismus des Leitens von Anfragen an die Funktion des richtigen filesystem oder device driver wird das VFS VF switch genannt. Für manche Operationen, die durch ein filesystem implementiert werden können, stellt das VFS Default Funktionen zur Verfügung. Diese werden aufgerufen, wenn ein Funktionszeiger NULL ist.

1.2 block device driver

Derselbe Mechanismus existiert, um Anfragen an den richtigen block device driver weiterzuleiten. Die block device driver sind in drivers/block. Die Organisation der block device driver ist ähnlich zu den filesystems: Auch hier gibt es einen generischer driver, der vom VFS und dem konkreten filesystem benutzt wird. Dieser leitet Anfragen je nach major und minor des device an den richtigen Treiber weiter. Diese eigentlichen Treiber sind in Verzeichnissen unter drivers/block. Die Funktion des device driver, die am häufigsten benutzt wird, ist `ll_rw_block`.

1.3 Quellen von VFS

Die Sourcen des VFS sind in `fs/`, das zugehörige include file ist `include/linux/fs.h`. Die jeweiligen filesystems sind in `fs/ext2/`, `fs/minix/` usw. implementiert und haben ein include file namens `fs_ext2.h` usw. in `include/linux`. Die Aufgaben und Quellen des VFS sind im einzelnen:

- Erkennen und Ausführen von Binärformaten. Sourcen: `binfmt_name.c`, wobei name aout, elf, script, ... sein kann.
- In `fs/block_dev.c` werden die generischen Funktionen `read()`, `write()`, und `fsync()` für den blockweisen Zugriff auf block devices implementiert. Diese Funktionen werden nur von den device drivers selbst aufgerufen! Lese - Anfragen werden, wenn möglich, durch den

Block Buffer (`fs/buffer.c`) befriedigt. Ansonsten wird die Anfrage über `ll_rw_block` an den device driver zurückgegeben. Bei Schreib - Anfragen wird, wenn mglich, ein Read Ahead gemacht und die gelesenen Blöcke im Block Buffer gespeichert. Anschliessend werden die modifizierten Blcke mit `ll_rw_block` geschriebenen.

- In `fs/buffer.c` wird ein Block Buffer implementiert. Dieser nimmt Anfragen, Blöcke zu schreiben oder zu lesen entgegen, oder den Buffer zu leeren (sync). Ist ein Block nicht direkt verfügbar oder soll der Buffe geleert werden, werden entsprechende Anfragen an den device driver gequeued. Die Funktion `ll_rw_block` für Anfragen an block devices wird in `drivers/block/ll_rw_blk.c` implementiert. In diesen Verzeichnissen sind auch die übrigen Quellen zu (block) device drivers.

- In `fs/dcache.c` wird ein Cache für das Suchen von Verzeichnissen implementiert. Dieser greift nicht auf das filesystem durch, sondern ist rein generisch: Man kann gefunden Einträge hinzufügen und nachschauen, ob ein Eintrag schon vorhanden ist.

- In `fs/devices.c` sind die Funktionen, um device driver zu registrieren, owie einige generische Funktionen zum Zugriff auf devices äquivalent zu `fs/filesystems.c`. Die Anfragen werden, wenn nötig, an den entsprechenden Treiber weitergereicht.

- In `fs/dquot.c` wird generisch disk quota unterstützt. Es werden Funktionen konkreter filesystems aufgerufen.

- In `fs/exec.c` sind Funktionen, um ausführbare Programme zu laden, Argumente und Umgebungsvariablen anzulegen und die Programme zu starten. Dazu werden die Funktionen in `binfmt_name.c` aufgerufen, aber auch Funktionen des konkreten filesystem. Die Funktion zum Reistrieren der Binärformate (`register_binfmt`) ist ebenfalls hier.

- ...

- In `fs/filesystems.c` ist nur die Funktion `sys_setup`, die sämtliche filesystems registriert und das root filesystem mountet.

- ...

- In `fs/open.c` sind Funktionen, um Dateien auf zu beeinflussen: statfs, fstatfs, truncate, ftruncate, utime, utimes, utime, utimes, access, chdir, fchdir, chroot, fchmod, chmod, fchown, chown, open, creat, close, vhangup.

- In `fs/super.c` sind die Funktionen `sys_mount` und `sys_umount` sowie `register_filesystem` definiert (siehe unten).

2 Das ext2

2.1 Beispiel: Registrierung und Mount von filesystems

Die konkreten filesystems (ext2, minix, xiafs, ...) werden beim kernel startup durch die vfs Funktion `sys_setup` in `fs/filesystems.c` registriert. Für jedes filesystem wird `init_name_fs` aufgerufen, für ext2 also `init_ext2_fs` in `fs/ext2/super.c`. `init_name_fs` ruft meist nur `register_filesystem` in `fs/super.c` auf mit folgender Struktur als Parameter:

```
struct file_system_type {
  struct super_block *(*read_super) (struct super_block *, void *, int);
  const char *name;
  int requires_dev;
  struct file_system_type * next;
};
```

Der erste Parameter ist ein Zeiger auf die Funktion, um den superblock zu lesen. Der zweite Paramter ist der Name des filesystems. `requires_dev` gibt an, ob das filesystem auf einem device gemountet werden muß – ob unter dem filesystem ein device liegen muß. Der letzte Zeiger zeigt auf die Beschreibung des nächsten filesystems und wird im Kernel verwendet, um die Liste der filesystems aufzubauen.

Die obige Struktur ist für das ext2 ebenfalls in `fs/ext2/super.c` definiert:

```
static struct file_system_type ext2_fs_type = {
  ext2_read_super,
  "ext2",
  1,
  NULL
};
```

D.h. die Funktion, um den superblock von ext2 zu lesen, ist `ext2_read_super`, die auch in `fs/ext2/super.c` definiert ist.

Nun kann man ein Beispiel für das *switchen* des VFS angeben: Ein User ruft `mount -t ext2 /dev/xxxx /mnt` auf. Dabei wird system call 21 – mount per interrupt aufgerufen. Der interrupt handler ruft `sys_mount` in `fs/super.c` auf. In `sys_mount` wird (über `do_mount`) `read_super` aufgerufen. Diese Funktion geht nun die registrierten file systems durch, bis der Eintrag für ext2 gefunden wird. Dann wird die Funktion in dieser Struktur (`ext2_read_super`) aufgerufen, um den superblock zu lesen.

2.2 Die Quellen von ext2

In `fs/ext2` liegen die Sourcen des filesystem ext2. Dazu gehört `include/linux/ext2_fs.h`.

- In `fs/ext2/acl.c` kann man den Check der Rechte durch ACLs einbauen. Die Datei enthält nur folgende Funktion:

```
/*
 * This file will contain the Access Control Lists management for the
 * second extended file system.
 */

int ext2_permission (struct inode * inode, int mask)
```

  Diese testet, ob die Rechte, gegeben durch mask, auf den inode durch current gegeben sind. current ist eine globale Struktur, die uid gid usw. angibt. Noch nicht abgetestet werden ACLs.

- In `balloc.c` wird die Verwaltung freier Blöcke gemacht. Dazu:

```
/*
 * The free blocks are managed by bitmaps.  A file system contains several
 * blocks groups.  Each group contains 1 bitmap block for blocks, 1 bitmap
 * block for inodes, N blocks for the inode table and data blocks.
 *
 * The file system contains group descriptors which are located after the
 * super block.  Each descriptor contains the number of the bitmap block an
 * the free blocks count in the block.  The descriptors are loaded in memor
 * when a file system is mounted (see \verb+ext2_read_super+).
 */
```

- `bitmap.c` enthält eine Hilfsfunktion für balloc.c.

- In `dir.c` sind die beiden folgenden, wichtigen Strukturen definiert:

```
static struct file_operations ext2_dir_operations = {
        NULL,                   /* lseek - default */
        ext2_dir_read,          /* read */
        NULL,                   /* write - bad */
        ext2_readdir,           /* readdir */
        NULL,                   /* select - default */
        ext2_ioctl,             /* ioctl */
        NULL,                   /* mmap */
        NULL,                   /* no special open code */
        NULL,                   /* no special release code */
        file_fsync,             /* fsync */
        NULL,                   /* fasync */
        NULL,                   /* check_media_change */
        NULL                    /* revalidate */
};

/*
 * directories can handle most operations...
 */
struct inode_operations ext2_dir_inode_operations = {
        &ext2_dir_operations,   /* default directory file-ops */
        ext2_create,            /* create */
        ext2_lookup,            /* lookup */
        ext2_link,              /* link */
        ext2_unlink,            /* unlink */
        ext2_symlink,           /* symlink */
        ext2_mkdir,             /* mkdir */
        ext2_rmdir,             /* rmdir */
        ext2_mknod,             /* mknod */
        ext2_rename,            /* rename */
        NULL,                   /* readlink */
        NULL,                   /* follow_link */
        NULL,                   /* readpage */
        NULL,                   /* writepage */
        NULL,                   /* bmap */
        ext2_truncate,          /* truncate */
        ext2_permission,        /* permission */
        NULL                    /* smap */
};
```

`ext2_dir_operations` wird an jeden file descriptor gehängt, der auf ein geöffnetes Verzeichnis zeigt. `ext2_dir_inode_operations` wird an jeden Inode (im Speicher) gehängt, in dem ein Verzeichnis abgelegt ist. D.h. wenn das VFS einen inode oder ein directory über das ext2 liest, werden diese Zeiger auf Funktionen mitgeliefert. Da, wo NULL angegeben wurde, werden die Default Funktionen des VFS benutzt. Der Mechanismus ist also der gleiche wie am Beispiel des Lesens des superblocks gezeigt.

- In `file.c` sind nochmal zwei entsprechende Sätze von Funktionszeigern für Datei Operationen definiert. Die Strukturen werden an File Deskriptoren bzw. Inodes gehängt, die Dateien referenzieren. Wenn man versucht ene Datei zu öffnen, wird der entsprechende Inode gelesen (siehe namei.c). Der Inode hat dann den Funktionssatz für Dateien. In diesem Satz ist auch die Funktion zum Öffnen der Datei, die schließlich aufgerufen wird.

- In `fsync.c` sind Funktionen, um Blöcke, die noch im Speicher sind, mit `ll_rw_block` auszuschreiben.

- In `ialloc.c` wird die Verwaltung freier Inodes gemacht. Dazu:

```
/*
* The free inodes are managed by bitmaps.  A file system contains several
* blocks groups.  Each group contains 1 bitmap block for blocks, 1 bitmap
* block for inodes, N blocks for the inode table and data blocks.
*
* The file system contains group descriptors which are located after the
* super block.  Each descriptor contains the number of the bitmap block an
* the free blocks count in the block.  The descriptors are loaded in memor
* when a file system is mounted (see \verb+ext2_read_super+).
*/
```

- In `inode.c` werden Inodes gelesen und geschrieben. Vor allem werden hier die richtigen Funktionssätze an die Inodes gehängt.

- `ioctl.c`

- In `namei.c` sind Funktionen, um einem Dateinamen einen Inode zuzuordnen, und Dateinamen in Verzeichnissen anzulegen und zu löschen.

- Zu `super.c` siehe oben.

- In symlink.c sind die inode_operations für symbolische Links definiert. Das sind die Funktionen ext2_readlink und ext2_follow_link.

- In truncate.c sind Funktionen, um Dateien abzuschneiden.

3 Systemaufrufe

Jeder Systemaufruf hat eine Nummer, mount z.B. die 21. Ein Systemaufruf wird gemacht, indem man die Paramter des Aufrufs mitsamt der Nummer auf den Stack legt und dann Interrupt 0x80 auslöst und damit in den kernel springt. Der Interrupthandler befindet sich in den Kernel Quellen für Intel in arch/i386/kernel/entry.S. Dort wird der Nummer 21 der Aufruf der Funktion sys_mount zugeordnet.

Allerdings kann man Funktionen definieren, die einem diese Arbeit abnehmen. Dazu benutzt man Makros aus /usr/include/syscall.h. Diese Datei inkludiert eine maschinenspezifisches Datei, für Intel z.B. /usr/include/i386/syscall.h. In dieser Datei sind Makros namens _systcallX definiert, um für systemcalls mit X verschieden Parametern eine Funktion mit ebensovielen Parametern zu definieren.

Im allgemeinen definiert die libc bereits eine Funktion für jeden systemcall. Die Funktion mount ist beispielsweise mit obigen Makros in den libc Quellen in sysdeps/linux/mount.S definiert.