内核笔记

段武杰

September 14, 2016

目录

1 VFS 1

1.1 sys_mount 流程 1

1.2 do_mount 流程 1

1.3 注册一个文件系统 2

1.4 file_system_type 2

1.5 super_block 4

1.6 inode 6

1.7 inode_operations 8

1.8 dentry 9

VFS

1.1 sys_mount 流程

```
//linux-4.3.3/fs/namespace.c

/**
 * sys_mount - 挂载文件系统
 * @dev_name 设备名称:`
 * @dir_name 挂载点路径:
 * @type 文件系统类型:
 * @flags 标志位供: do_mount 调用。
 * @data 选项信息:
 */
SYSCALL_DEFINE5(mount, char __user *, dev_name, char __user *, dir_name,
                char __user *, type, unsigned long, flags, void __user *, data);
```

- 将用户态参数拷贝至内核态
- 调用 do_mount 完成主要挂载工作

1.2 do_mount 流程

其中 get_fs_type: 用于判断使用那个 file_system_type

1.3 注册一个文件系统

```
#include <linux/fs.h>
extern int register_filesystem(struct file_system_type *);
extern int unregister_filesystem(struct file_system_type *);
```

file_systems: 文件系统链表，后续对 VFS 的操作将围绕该链表展开。

- register_filesystem: 通过文件系统的名字在 file_systems 链表中查找对应的文件系统，没有找到，则将文件新的文件系统加入链表。
- unregister_filesystem: 将文件系统从 file_systems 链表中删除
- /proc/filesystems 下显示了所有已经注册的文件系统。

1.4 file_system_type

file_system_type 用于描述一个文件系统：

```
struct file_system_type {
        const char *name;/* 文件系统的名字 */
        int fs_flags;   /* 文件系统类型标志 */
        struct dentry *(*mount) (struct file_system_type *, int,
                        const char *, void *);/* 当挂在一个文件系统时调用*/
        void (*kill_sb) (struct super_block *);/*  */
        struct module *owner;/* 文件系统模块，初始化为THIS_MODULE */
        struct file_system_type * next;/*内部使用，初始化为空VFS*/
        struct hlist_head fs_supers;/* supperblock list */
        /* 相关锁 */
        /* ... */
};
```

The mount() method must return the root dentry of the tree requested by caller. An active reference to its superblock must be grabbed and the superblock must be locked. On failure it should return ERR_PTR(error).

The arguments match those of mount(2) and their interpretation depends on filesystem type. E.g. for block filesystems, dev_name is interpreted as block

device name, that device is opened and if it contains a suitable filesystem image the method creates and initializes struct super_block accordingly, returning its root dentry to caller.

->mount() may choose to return a subtree of existing filesystem - it doesn't have to create a new one. The main result from the caller's point of view is a reference to dentry at the root of (sub)tree to be attached; creation of new superblock is a common side effect.

The most interesting member of the superblock structure that the mount() method fills in is the "s_op" field. This is a pointer to a "struct super_operations" which describes the next level of the filesystem implementation.

Usually, a filesystem uses one of the generic mount() implementations and provides a fill_super() callback instead. The generic variants are:

- mount_bdev: mount a filesystem residing on a block device
- mount_nodev: mount a filesystem that is not backed by a device
- mount_single: mount a filesystem which shares the instance between all mounts

A fill_super() callback implementation has the following arguments:

- struct super_block *sb: the superblock structure. The callback must initialize this properly.
- void *data: arbitrary mount options, usually comes as an ASCII string (see "Mount Options" section)
- int silent: whether or not to be silent on error

VFS 使用面向对象的设计思路，VFS 中有 4 个主要的对象类型：

- 超级块对象 (super_block): 它表示一个具体的已安装的文件系统
- 索引节点对象 (inode): 它表示一个具体的文件
- 目录项对象 (dentry): 它表示一个目录项，是路径的一个组成部分。
- 文件对象 (file): 它表示进程打开的文件。

VFS 将目录当作文件来处理，所以不存在目录对象，目录项代表的是路径中的一个组成部分。

1.5 super_block

A superblock object represents a mounted filesystem.

```
struct super_block {
        struct list_head        s_list;         /* 指向的链表super_block */
        dev_t                   s_dev;          /* 设备标识符 */
        unsigned char           s_blocksize_bits;/* 以位为单位的块大小 */
        unsigned long           s_blocksize;/* 以字节为单位的块大小 */
        loff_t                  s_maxbytes;     /* Max file size */
        struct file_system_type *s_type;/* Filesystem type */
        const struct super_operations   *s_op;/*超级块方法*/
        const struct dquot_operations   *dq_op;/*磁盘限额方法 */
        const struct quotactl_ops       *s_qcop;/* 限额控制方法 */
        const struct export_operations *s_export_op;/* 导出方法 */
        unsigned long           s_flags;/* 挂载标志 */
        unsigned long           s_iflags;       /* internal SB_I_* flags */
        unsigned long           s_magic;/* 文件系统魔数 */
        struct dentry           *s_root;/* 目录挂载点 */
        struct rw_semaphore     s_umount;/* 卸载信号量 */
        int                     s_count;/* 超级块引用计数 */
        atomic_t                s_active;/* 活动引用计数 */
#ifdef CONFIG_SECURITY
        void                    *s_security;/* 安全模块 */
#endif
        const struct xattr_handler **s_xattr;/*扩展的属性操作*/

        struct hlist_bl_head    s_anon;         /* anonymous dentries for (nfs) exporting */
        struct list_head        s_mounts;       /* list of mounts; _not_ for fs use */
```

```
struct block_device     *s_bdev;/*相关的块设备*/
struct backing_dev_info *s_bdi;
struct mtd_info         *s_mtd;
struct hlist_node       s_instances;
unsigned int            s_quota_types;  /* Bitmask of supported quota types */
struct quota_info       s_dquot;        /* Diskquota specific options */

struct sb_writers       s_writers;

char s_id[32];                          /* Informational name */
u8 s_uuid[16];                          /* UUID */

void                    *s_fs_info;     /* 文件系统私有数据 */
unsigned int            s_max_links;
fmode_t                 s_mode;

/* Granularity of c/m/atime in ns.
   Cannot be worse than a second */
u32                s_time_gran;

/*
 * The next field is for VFS *only*. No filesystems have any business
 * even looking at it. You had been warned.
 */
struct mutex s_vfs_rename_mutex;        /* Kludge */

/*
 * Filesystem subtype.  If non-empty the filesystem type field
 * in /proc/mounts will be "type.subtype"
 */
char *s_subtype;

/*
 * Saved mount options for lazy filesystems using
 * generic_show_options()
 */
char __rcu *s_options;
const struct dentry_operations *s_d_op; /* default d_op for dentries */

/*
 * Saved pool identifier for cleancache (-1 means none)
 */
int cleancache_poolid;

struct shrinker s_shrink;       /* per-sb shrinker handle */

/* Number of inodes with nlink == 0 but still referenced */
atomic_long_t s_remove_count;

/* Being remounted read-only */
int s_readonly_remount;
```

```
        /* AIO completions deferred from interrupt context */
        struct workqueue_struct *s_dio_done_wq;
        struct hlist_head s_pins;

        /*
         * Keep the lru lists last in the structure so they always sit on their
         * own individual cachelines.
         */
        struct list_lru         s_dentry_lru ____cacheline_aligned_in_smp;
        struct list_lru         s_inode_lru ____cacheline_aligned_in_smp;
        struct rcu_head         rcu;
        struct work_struct      destroy_work;

        struct mutex            s_sync_lock;    /* sync serialisation lock */

        /*
         * Indicates how deep in a filesystem stack this SB is
         */
        int s_stack_depth;

        /* s_inode_list_lock protects s_inodes */
        spinlock_t              s_inode_list_lock ____cacheline_aligned_in_smp;
        struct list_head        s_inodes;       /* all inodes */
};
```

1.6 inode

内核处理文件的关键是 inode，每个文件（和目录）都有且只有一个对应的 inode, 其中包含元数据（如访问权限，上次修改的日期，等等）和指向文件数据的指针。

```


/*
 * Keep mostly read-only and often accessed (especially for
 * the RCU path lookup and 'stat' data) fields at the beginning
 * of the 'struct inode'
 */
struct inode {/* fs.h */
        umode_t                 i_mode;/* 文件访问权限和所有权 */
        unsigned short          i_opflags;
        kuid_t                  i_uid;/* uid about the file */
        kgid_t                  i_gid;/* gid about the file */
        unsigned int            i_flags;

#ifdef CONFIG_FS_POSIX_ACL
```

```
        struct posix_acl        *i_acl;
        struct posix_acl        *i_default_acl;
#endif
        /* 负责管理结构性操作（如删除一个文件）和文件相关的元数据例如属性() */
        const struct inode_operations   *i_op;
        struct super_block      *i_sb;
        struct address_space    *i_mapping;

#ifdef CONFIG_SECURITY
        void                    *i_security;
#endif

        /* Stat data, not accessed from path walking */
        /* 对给定的文件系统，唯一的编号标识 */
        unsigned long           i_ino;
        /*
         * Filesystems may only read i_nlink directly.  They shall use the
         * following functions for modification:
         *
         *    (set|clear|inc|drop)_nlink
         *    inode_(inc|dec)_link_count
         */
        union {
                /* 记录使用该 inode 的硬链接总数 */
                const unsigned int i_nlink;
                unsigned int __i_nlink;
        };
        dev_t                   i_rdev;

        loff_t                  i_size;/* 文件大小 */
        struct timespec         i_atime;/* 最后访问时间 */
        struct timespec         i_mtime;/* 最后修改时间*/
        struct timespec         i_ctime;/* inode 最后修改时间 */
        spinlock_t              i_lock; /* i_blocks, i_bytes, maybe i_size */
        unsigned short          i_bytes;
        unsigned int            i_blkbits;
        blkcnt_t                i_blocks;/*指定了按块存放的长度*/

#ifdef __NEED_I_SIZE_ORDERED
        seqcount_t              i_size_seqcount;
#endif

        /* Misc */
        unsigned long           i_state;
        struct mutex            i_mutex;

        unsigned long           dirtied_when;   /* jiffies of first dirtying */
        unsigned long           dirtied_time_when;

        struct hlist_node       i_hash;
        struct list_head        i_io_list;      /* backing dev IO list */
#ifdef CONFIG_CGROUP_WRITEBACK
```

```
        struct bdi_writeback    *i_wb;          /* the associated cgroup wb */

        /* foreign inode detection, see wbc_detach_inode() */
        int                     i_wb_frn_winner;
        u16                     i_wb_frn_avg_time;
        u16                     i_wb_frn_history;
#endif
        struct list_head        i_lru;          /* inode LRU list */
        struct list_head        i_sb_list;
        union {
                struct hlist_head       i_dentry;
                struct rcu_head         i_rcu;
        };
        u64                     i_version;
        atomic_t                i_count;/* 访问该的进程数目inode */
        atomic_t                i_dio_count;
        atomic_t                i_writecount;
#ifdef CONFIG_IMA
        atomic_t                i_readcount; /* struct files open RO */
#endif

        const struct file_operations    *i_fop; /* 用于操作文件中包含的数据 */
        struct file_lock_context        *i_flctx;
        struct address_space    i_data;
        struct list_head        i_devices;
        union {
                struct pipe_inode_info  *i_pipe;
                struct block_device     *i_bdev;
                struct cdev             *i_cdev;
                char                    *i_link;
        };

        __u32                   i_generation;

#ifdef CONFIG_FSNOTIFY
        __u32                   i_fsnotify_mask; /* all events this inode cares about */
        struct hlist_head       i_fsnotify_marks;
#endif

        void                    *i_private; /* fs or device private pointer */
};
```

1.7 inode_operations

大多数请况下，各个函数指针成员的意义可以根据其名称推断。它们与对应的系统调用和用户空间工具在名称方面非常相似。

```
struct inode_operations {
        /* lookup 根据文件系统对象的名称表示为字符串）查找其( inode 实例*/
        struct dentry * (*lookup) (struct inode *,struct dentry *, unsigned int);
        const char * (*follow_link) (struct dentry *, void **);
        int (*permission) (struct inode *, int);
        struct posix_acl * (*get_acl)(struct inode *, int);

        int (*readlink) (struct dentry *, char __user *,int);
        void (*put_link) (struct inode *, void *);

        int (*create) (struct inode *,struct dentry *, umode_t, bool);
        int (*link) (struct dentry *,struct inode *,struct dentry *);
        int (*unlink) (struct inode *,struct dentry *);
        int (*symlink) (struct inode *,struct dentry *,const char *);
        int (*mkdir) (struct inode *,struct dentry *,umode_t);
        int (*rmdir) (struct inode *,struct dentry *);
        int (*mknod) (struct inode *,struct dentry *,umode_t,dev_t);
        int (*rename) (struct inode *, struct dentry *,
                       struct inode *, struct dentry *);
        int (*rename2) (struct inode *, struct dentry *,
                        struct inode *, struct dentry *, unsigned int);
        int (*setattr) (struct dentry *, struct iattr *);
        int (*getattr) (struct vfsmount *mnt, struct dentry *, struct kstat *);
        int (*setxattr) (struct dentry *, const char *,const void *,size_t,int);
        ssize_t (*getxattr) (struct dentry *, const char *, void *, size_t);
        ssize_t (*listxattr) (struct dentry *, char *, size_t);
        int (*removexattr) (struct dentry *, const char *);
        int (*fiemap)(struct inode *, struct fiemap_extent_info *, u64 start,
                      u64 len);
        int (*update_time)(struct inode *, struct timespec *, int);
        int (*atomic_open)(struct inode *, struct dentry *,
                           struct file *, unsigned open_flag,
                           umode_t create_mode, int *opened);
        int (*tmpfile) (struct inode *, struct dentry *, umode_t);
        int (*set_acl)(struct inode *, struct posix_acl *, int);

        /* WARNING: probably going away soon, do not use! */
} ____cacheline_aligned;
```

1.8 dentry

```
struct dentry {
        /* RCU lookup touched fields */
        unsigned int d_flags;           /* protected by d_lock */
        seqcount_t d_seq;               /* per dentry seqlock */
        struct hlist_bl_node d_hash;    /* lookup hash list */
        struct dentry *d_parent;        /* parent directory */
```

```
        struct qstr d_name;
        struct inode *d_inode;          /* Where the name belongs to - NULL is
                                         * negative */
        unsigned char d_iname[DNAME_INLINE_LEN];        /* small names */

        /* Ref lookup also touches following */
        struct lockref d_lockref;       /* per-dentry lock and refcount */
        const struct dentry_operations *d_op;
        struct super_block *d_sb;       /* The root of the dentry tree */
        unsigned long d_time;           /* used by d_revalidate */
        void *d_fsdata;                 /* fs-specific data */

        struct list_head d_lru;         /* LRU list */
        struct list_head d_child;       /* child of parent list */
        struct list_head d_subdirs;     /* our children */
        /*
         * d_alias and d_rcu can share memory
         */
        union {
                struct hlist_node d_alias;      /* inode alias list */
                struct rcu_head d_rcu;
        } d_u;
};
```

参考文献

[1] http://linux-mm.org/PageTableStructure.