Linux VFS虚拟文件系统初探

2019-12-02 | 技术

“关于Linux内核的VFS文件系统的一些基础”

记录一下关于VFS文件系统的一些基础学习，和一些理解，如有错误望指正！

> 参考资料：
>
> 《深入理解Linux内核》（第三版）
>
> Linux 文件系统剖析
>
> VFS中的file、dentry和inode
>
> 源码：Linux-4.19.65

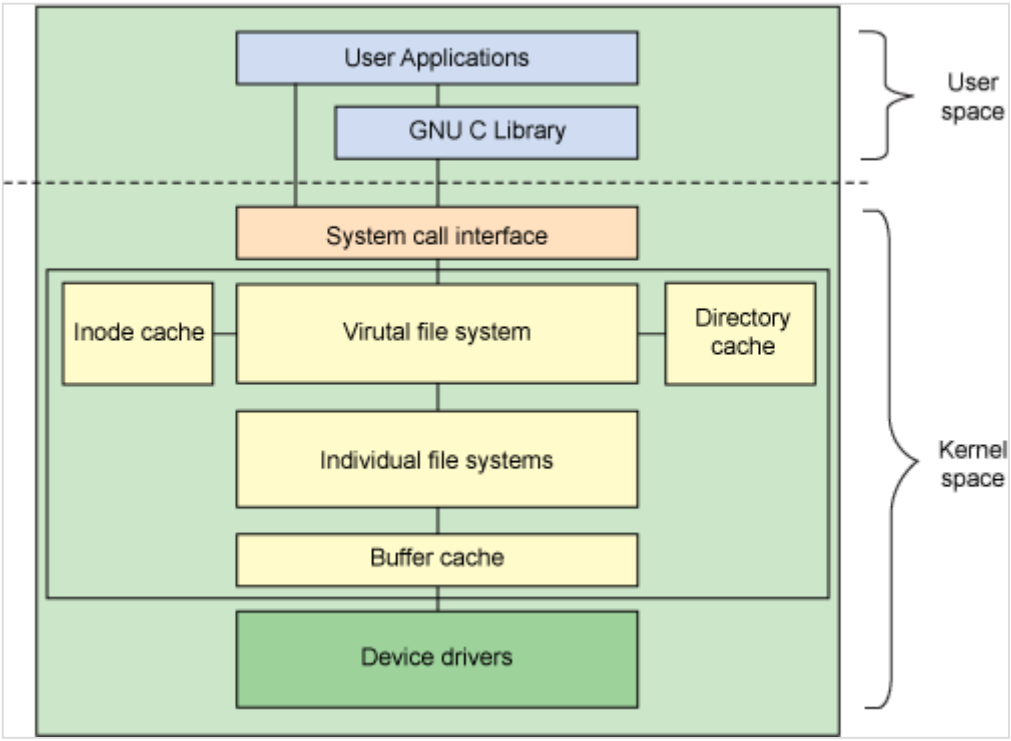
一些概念、数据结构

VFS

虚拟文件系统（Virtual Filesystem），是一个内核软件层，用来处理与Unix标准文件系统相关的所有系统调用。其健壮性表现在能为各种文件系统提供一个通用的接口。VFS是用户的应用程序与文件系统实现之间的抽象层，应用程序不需要知道操作的文件是什么文件系统类型，直接与VFS交互，VFS再与不同的文件系统交互。

VFS引入了一个通用的文件模型（common file model），这个模型能够表示所有支持的文件系统。而要实现每个具体的文件系统，必须将其物理组织结构转换为虚拟文件系统的通用文件模型。

下图所示的体系结构显示了用户空间和内核中与文件系统相关的组件之间的关系：



file_system_type

> 可以使用一组注册函数在 Linux 中动态地添加或删除文件系统。内核保存当前支持的文件系统的列表，可以通过 /proc 文件系统在用户空间中查看这个列表。这个虚拟文件还显示当前与这些文件系统相关联的设备。在 Linux 中添加新文件系统的方法是调用 `register_filesystem()` 。这个函数的参数定义一个文件系统结构（ `file_system_type` ）的引用，这个结构定义文件系统的名称、一组属性和两个超级块函数。也可以注销文件系统。
>
> 在注册新的文件系统时，会把这个文件系统和它的相关信息添加到 file_systems 列表中（见图 2 和 linux/include/linux/mount.h ）。这个列表定义可以支持的文件系统。在命令行上输入 `cat /proc/filesystems` ，就可以查看这个列表。
>
> from https://www.ibm.com/developerworks/cn/linux/l-linux-filesystem/index.html

源码中对 `file_system_type` 结构的定义如下：

```
1  /* include/linux/fs.h */
2
3  struct file_system_type {
4          const char *name;
5          int fs_flags;
6  #define FS_REQUIRES_DEV         1
7  #define FS_BINARY_MOUNTDATA     2
8  #define FS_HAS_SUBTYPE          4
9  #define FS_USERNS_MOUNT         8       /* Can be mounted by userns root */
```

```
#define FS_RENAME_DOES_D_MOVE   32768   /* FS will handle d_move() during rename()
        struct dentry *(*mount) (struct file_system_type *, int,
                        const char *, void *);
        void (*kill_sb) (struct super_block *);
        struct module *owner;
        struct file_system_type * next;  //下一个file_system_type，可以看到所有文件系
        struct hlist_head fs_supers;

        struct lock_class_key s_lock_key;
        struct lock_class_key s_umount_key;
        struct lock_class_key s_vfs_rename_key;
        struct lock_class_key s_writers_key[SB_FREEZE_LEVELS];

        struct lock_class_key i_lock_key;
        struct lock_class_key i_mutex_key;
        struct lock_class_key i_mutex_dir_key;
};
```

vfsmount

这个结构提供当前挂装的文件系统 。

```
/* /include/linux/mount.h */
struct vfsmount {
        struct dentry *mnt_root;        /* root of the mounted tree */
        struct super_block *mnt_sb;     /* pointer to superblock */
        int mnt_flags;
} __randomize_layout;
```

super_block 超级块

存放已安装文件系统的有关信息。 它包含管理文件系统所需的信息，包括文件系统名称（比如 ext2）、文件系统的大小和状态、块设备的引用和元数据信息（比如空闲列表等等）。超级块通常存储在存储媒体上，但是如果超级块不存在，也可以实时创建它。可以在/include/linux/fs.h 中找到超级块结构 。

```
/* /include/linux/fs.h */
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;
```

```
 9          const struct super_operations   *s_op;
10          const struct dquot_operations   *dq_op;
11          const struct quotactl_ops       *s_qcop;
12          const struct export_operations *s_export_op;
13          unsigned long           s_flags;
14          unsigned long           s_iflags;       /* internal SB_I_* flags */
15          unsigned long           s_magic;
16          struct dentry           *s_root;
17          struct rw_semaphore     s_umount;
18          int                     s_count;
19          atomic_t                s_active;
20  #ifdef CONFIG_SECURITY
21          void                    *s_security;
22  #endif
23          const struct xattr_handler **s_xattr;
24  #if IS_ENABLED(CONFIG_FS_ENCRYPTION)
25          const struct fscrypt_operations *s_cop;
26  #endif
27          struct hlist_bl_head    s_roots;        /* alternate root dentries for NFS
28          struct list_head        s_mounts;       /* list of mounts; _not_ for fs us
29          struct block_device     *s_bdev;
30          struct backing_dev_info *s_bdi;
31          struct mtd_info         *s_mtd;
32          struct hlist_node       s_instances;
33          unsigned int            s_quota_types;  /* Bitmask of supported quota type
34          struct quota_info       s_dquot;        /* Diskquota specific options */
35
36          struct sb_writers       s_writers;
37
38          char                    s_id[32];       /* Informational name */
39          uuid_t                  s_uuid;         /* UUID */
40
41          void                    *s_fs_info;     /* Filesystem private info */
42          unsigned int            s_max_links;
43          fmode_t                 s_mode;
44
45          /* Granularity of c/m/atime in ns.
46             Cannot be worse than a second */
47          u32                s_time_gran;
48
49          /*
50           * The next field is for VFS *only*. No filesystems have any business
51           * even looking at it. You had been warned.
52           */
53          struct mutex s_vfs_rename_mutex;        /* Kludge */
54
55          /*
```

```
56          * Filesystem subtype.  If non-empty the filesystem type field
57          * in /proc/mounts will be "type.subtype"
58          */
59         char *s_subtype;
60
61         const struct dentry_operations *s_d_op; /* default d_op for dentries */
62
63         /*
64          * Saved pool identifier for cleancache (-1 means none)
65          */
66         int cleancache_poolid;
67
68         struct shrinker s_shrink;       /* per-sb shrinker handle */
69
70         /* Number of inodes with nlink == 0 but still referenced */
71         atomic_long_t s_remove_count;
72
73         /* Pending fsnotify inode refs */
74         atomic_long_t s_fsnotify_inode_refs;
75
76         /* Being remounted read-only */
77         int s_readonly_remount;
78
79         /* AIO completions deferred from interrupt context */
80         struct workqueue_struct *s_dio_done_wq;
81         struct hlist_head s_pins;
82
83         /*
84          * Owning user namespace and default context in which to
85          * interpret filesystem uids, gids, quotas, device nodes,
86          * xattrs and security labels.
87          */
88         struct user_namespace *s_user_ns;
89
90         /*
91          * Keep the lru lists last in the structure so they always sit on their
92          * own individual cachelines.
93          */
94         struct list_lru         s_dentry_lru ____cacheline_aligned_in_smp;
95         struct list_lru         s_inode_lru ____cacheline_aligned_in_smp;
96         struct rcu_head         rcu;
97         struct work_struct      destroy_work;
98
99         struct mutex            s_sync_lock;    /* sync serialisation lock */
100
101         /*
102          * Indicates how deep in a filesystem stack this SB is
```

```
103           */
104          int s_stack_depth;
105
106          /* s_inode_list_lock protects s_inodes */
107          spinlock_t              s_inode_list_lock ____cacheline_aligned_in_smp;
108          struct list_head        s_inodes;       /* all inodes */
109
110          spinlock_t              s_inode_wblist_lock;
111          struct list_head        s_inodes_wb;    /* writeback inodes */
112  } __randomize_layout;
```

super_operations

super_block中的一个重要字段是 `const struct super_operations *s_op` ，这个结构定义一组用来管理这个文件系统中inode的函数，其中实现了一些高级操作，比如删除文件或安装磁盘。

每个具体的文件系统都可以定义自己的超级块操作。当VFS需要调用其中一个操作时，比如 `write_inode()` ，它执行下列操作：

```
1  sb->s_op->write_inode();
```

`super_operations` 结构的定义如下：

```
1  /* /include/linux/fs.h */
2
3  struct super_operations {
4          struct inode *(*alloc_inode)(struct super_block *sb);
5          void (*destroy_inode)(struct inode *);
6
7          void (*dirty_inode) (struct inode *, int flags);
8          int (*write_inode) (struct inode *, struct writeback_control *wbc);
9          int (*drop_inode) (struct inode *);
10         void (*evict_inode) (struct inode *);
11         void (*put_super) (struct super_block *);
12         int (*sync_fs)(struct super_block *sb, int wait);
13         int (*freeze_super) (struct super_block *);
14         int (*freeze_fs) (struct super_block *);
15         int (*thaw_super) (struct super_block *);
16         int (*unfreeze_fs) (struct super_block *);
17         int (*statfs) (struct dentry *, struct kstatfs *);
18         int (*remount_fs) (struct super_block *, int *, char *);
19         void (*umount_begin) (struct super_block *);
20
```

```
        int (*show_options)(struct seq_file *, struct dentry *);
        int (*show_devname)(struct seq_file *, struct dentry *);
        int (*show_path)(struct seq_file *, struct dentry *);
        int (*show_stats)(struct seq_file *, struct dentry *);
#ifdef CONFIG_QUOTA
        ssize_t (*quota_read)(struct super_block *, int, char *, size_t, loff_t);
        ssize_t (*quota_write)(struct super_block *, int, const char *, size_t, lof
        struct dquot **(*get_dquots)(struct inode *);
#endif
        int (*bdev_try_to_free_page)(struct super_block*, struct page*, gfp_t);
        long (*nr_cached_objects)(struct super_block *,
                                  struct shrink_control *);
        long (*free_cached_objects)(struct super_block *,
                                    struct shrink_control *);
};
```

inode 索引节点

inode（索引节点对象）表示文件系统中的一个对象，它具有惟一标识符。各个文件系统提供将文件名映射为惟一 inode 标识符和 inode 引用的方法 ，文件名可以随时更改，但是索引节点对文件是唯一的，并且随文件的存在而存在。

inode 结构的定义如下：

```
/* /include/linux/fs.h */
/*
 * 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 {
        umode_t                 i_mode;
        unsigned short          i_opflags;
        kuid_t                  i_uid;
        kgid_t                  i_gid;
        unsigned int            i_flags;

#ifdef CONFIG_FS_POSIX_ACL
        struct posix_acl        *i_acl;
        struct posix_acl        *i_default_acl;
#endif

        const struct inode_operations   *i_op;
        struct super_block      *i_sb;  //对应的超级块对象
```

```
21          struct address_space    *i_mapping;
22
23  #ifdef CONFIG_SECURITY
24          void                    *i_security;
25  #endif
26
27          /* Stat data, not accessed from path walking */
28          unsigned long           i_ino;  //索引节点号
29          /*
30           * Filesystems may only read i_nlink directly.  They shall use the
31           * following functions for modification:
32           *
33           *    (set|clear|inc|drop)_nlink
34           *    inode_(inc|dec)_link_count
35           */
36          union {
37                  const unsigned int i_nlink;
38                  unsigned int __i_nlink;
39          };
40          dev_t                   i_rdev;
41          loff_t                  i_size;
42          struct timespec64       i_atime;
43          struct timespec64       i_mtime;
44          struct timespec64       i_ctime;
45          spinlock_t              i_lock; /* i_blocks, i_bytes, maybe i_size */
46          unsigned short          i_bytes;
47          u8                      i_blkbits;
48          u8                      i_write_hint;
49          blkcnt_t                i_blocks;
50
51  #ifdef __NEED_I_SIZE_ORDERED
52          seqcount_t              i_size_seqcount;
53  #endif
54
55          /* Misc */
56          unsigned long           i_state;
57          struct rw_semaphore     i_rwsem;
58
59          unsigned long           dirtied_when;   /* jiffies of first dirtying */
60          unsigned long           dirtied_time_when;
61
62          struct hlist_node       i_hash;
63          struct list_head        i_io_list;      /* backing dev IO list */
64  #ifdef CONFIG_CGROUP_WRITEBACK
65          struct bdi_writeback    *i_wb;          /* the associated cgroup wb */
66
67          /* foreign inode detection, see wbc_detach_inode() */
```

```
68          int                     i_wb_frn_winner;
69          u16                     i_wb_frn_avg_time;
70          u16                     i_wb_frn_history;
71  #endif
72          struct list_head        i_lru;          /* inode LRU list */
73          struct list_head        i_sb_list;
74          struct list_head        i_wb_list;      /* backing dev writeback list */
75          union {
76                  struct hlist_head       i_dentry;
77                  struct rcu_head         i_rcu;
78          };
79          atomic64_t              i_version;
80          atomic_t                i_count;  //引用计数器
81          atomic_t                i_dio_count;
82          atomic_t                i_writecount;
83  #ifdef CONFIG_IMA
84          atomic_t                i_readcount; /* struct files open RO */
85  #endif
86          const struct file_operations    *i_fop; /* former ->i_op->default_file_ops
87          struct file_lock_context        *i_flctx;
88          struct address_space    i_data;
89          struct list_head        i_devices;
90          union {
91                  struct pipe_inode_info  *i_pipe;
92                  struct block_device     *i_bdev;
93                  struct cdev             *i_cdev;
94                  char                    *i_link;
95                  unsigned                i_dir_seq;
96          };
97
98          __u32                   i_generation;
99
100 #ifdef CONFIG_FSNOTIFY
101         __u32                   i_fsnotify_mask; /* all events this inode cares ab
102         struct fsnotify_mark_connector __rcu    *i_fsnotify_marks;
103 #endif
104
105 #if IS_ENABLED(CONFIG_FS_ENCRYPTION)
106         struct fscrypt_info     *i_crypt_info;
107 #endif
108
109         void                    *i_private; /* fs or device private pointer */
110 } __randomize_layout;
```

inode_operations

inode_operations 定义直接在 inode 上执行的操作

```
/* /include/linux/fs.h */
struct inode_operations {
        struct dentry * (*lookup) (struct inode *,struct dentry *, unsigned int);
        const char * (*get_link) (struct dentry *, struct inode *, struct delayed_c
        int (*permission) (struct inode *, int);
        struct posix_acl * (*get_acl)(struct inode *, int);

        int (*readlink) (struct dentry *, char __user *,int);

        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 *, unsigned int);
        int (*setattr) (struct dentry *, struct iattr *);
        int (*getattr) (const struct path *, struct kstat *, u32, unsigned int);
        ssize_t (*listxattr) (struct dentry *, char *, size_t);
        int (*fiemap)(struct inode *, struct fiemap_extent_info *, u64 start,
                      u64 len);
        int (*update_time)(struct inode *, struct timespec64 *, int);
        int (*atomic_open)(struct inode *, struct dentry *,
                           struct file *, unsigned open_flag,
                           umode_t create_mode);
        int (*tmpfile) (struct inode *, struct dentry *, umode_t);
        int (*set_acl)(struct inode *, struct posix_acl *, int);
} ____cacheline_aligned;
```

file_operations

file_operations 定义与文件和目录相关的方法（标准系统调用）。

```
/* /include/linux.h */
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
```

```
8           ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
9           int (*iterate) (struct file *, struct dir_context *);
10          int (*iterate_shared) (struct file *, struct dir_context *);
11          __poll_t (*poll) (struct file *, struct poll_table_struct *);
12          long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
13          long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
14          int (*mmap) (struct file *, struct vm_area_struct *);
15          unsigned long mmap_supported_flags;
16          int (*open) (struct inode *, struct file *);
17          int (*flush) (struct file *, fl_owner_t id);
18          int (*release) (struct inode *, struct file *);
19          int (*fsync) (struct file *, loff_t, loff_t, int datasync);
20          int (*fasync) (int, struct file *, int);
21          int (*lock) (struct file *, int, struct file_lock *);
22          ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, i
23          unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned l
24          int (*check_flags)(int);
25          int (*flock) (struct file *, int, struct file_lock *);
26          ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *,
27          ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, s
28          int (*setlease)(struct file *, long, struct file_lock **, void **);
29          long (*fallocate)(struct file *file, int mode, loff_t offset,
30                            loff_t len);
31          void (*show_fdinfo)(struct seq_file *m, struct file *f);
32  #ifndef CONFIG_MMU
33          unsigned (*mmap_capabilities)(struct file *);
34  #endif
35          ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
36                          loff_t, size_t, unsigned int);
37          int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t,
38                          u64);
39          int (*dedupe_file_range)(struct file *, loff_t, struct file *, loff_t,
40                          u64);
41          int (*fadvise)(struct file *, loff_t, loff_t, int);
42  } __randomize_layout;
```

file 文件对象

存放打开文件与进程之间进行交互的有关信息。当进程打开一个文件时，会创建一个 file 文件对象，它描述进程怎样与一个打开的文件进行交互。

> 严格来讲，这个结构体不仅仅和文件系统相关，它也和进程相关联。对于虚拟文件系统而言，它的重要性一般，而相反，这个结构体对进程管理更重要，它记录了进程打开文件的上下文。该结构体，浮于表面，更贴近用户，更贴近进程。

> from https://bean-li.github.io/vfs-inode-dentry/

```
/* /include/linux/fs.h */
struct file {
        union {
                struct llist_node       fu_llist;
                struct rcu_head         fu_rcuhead;
        } f_u;
        struct path             f_path;
        struct inode            *f_inode;       /* cached value */
        const struct file_operations    *f_op;

        /*
         * Protects f_ep_links, f_flags.
         * Must not be taken from IRQ context.
         */
        spinlock_t              f_lock;
        enum rw_hint            f_write_hint;
        atomic_long_t           f_count;
        unsigned int            f_flags;
        fmode_t                 f_mode;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;

        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        /* needed for tty driver, and maybe others */
        void                    *private_data;

#ifdef CONFIG_EPOLL
        /* Used by fs/eventpoll.c to link all the hooks to this file */
        struct list_head        f_ep_links;
        struct list_head        f_tfile_llink;
#endif /* #ifdef CONFIG_EPOLL */
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
} __randomize_layout
```

文件指针

有一个重要信息是文件指针 `loff_t f_pos` ，即文件中当前的位置，下一个操作将在该位置发生。由于几个进程可能同时访问同一文件，因此文件指针必须存放在文件对象而不是索引节点对象中。

file结构体中的file_operations

0%

每个文件系统都有其自己的文件操作集合，执行诸如读写文件这样的操作。当进程打开一个文件时，VFS会用对应的 `inode->i_fop` 中的 `file_operations`结构体 来初始化新文件对象的 `file->fop` ，使得对文件操作的后续调用能够使用这些函数。如果需要VFS也可以通过修改这个字段而修改文件操作的集合。

```
/* /include/linux/fs.h */
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iterate) (struct file *, struct dir_context *);
        int (*iterate_shared) (struct file *, struct dir_context *);
        __poll_t (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        unsigned long mmap_supported_flags;
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, i
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned l
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *,
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, s
        int (*setlease)(struct file *, long, struct file_lock **, void **);
        long (*fallocate)(struct file *file, int mode, loff_t offset,
                          loff_t len);
        void (*show_fdinfo)(struct seq_file *m, struct file *f);
#ifndef CONFIG_MMU
        unsigned (*mmap_capabilities)(struct file *);
#endif
        ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
                        loff_t, size_t, unsigned int);
```

```
37          int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t,
38                          u64);
39          int (*dedupe_file_range)(struct file *, loff_t, struct file *, loff_t,
40                          u64);
41          int (*fadvise)(struct file *, loff_t, loff_t, int);
42  } __randomize_layout;
```

0%

path

file 结构体中的 struct path f_path 字段，记录了这个文件对象对应的目录项（dentry）和文件系统（vfsmount）

```
1  /* /include/linux/path.h */
2  struct path {
3          struct vfsmount *mnt;  //含有该文件的已安装文件系统
4          struct dentry *dentry;  //与文件相关的目录项对象
5  } __randomize_layout;
```

dentry 目录项对象

dentry是目录项缓存，是一个存放在内存里的缩略版的磁盘文件系统目录树结构,他是directory entry的缩写。它存放目录项（也就是文件的特定名称）与对应文件进行链接的有关信息。一旦目录项被读入内存，VFS就把它转换成基于 dentry 结构的一个目录项对象。对于进程查找的路径名中的每个分量，内核都为其创建一个目录项对象；目录对象将每个分量与其对应的索引节点相联系。例如在查找路径名 /tmp/test 时，内核为根目录 / 创建一个目录对象，为根目录下的 tmp 项创建一个第二级目录项对象，对 /tmp 目录下的 test 项创建一个第三级目录项对象。

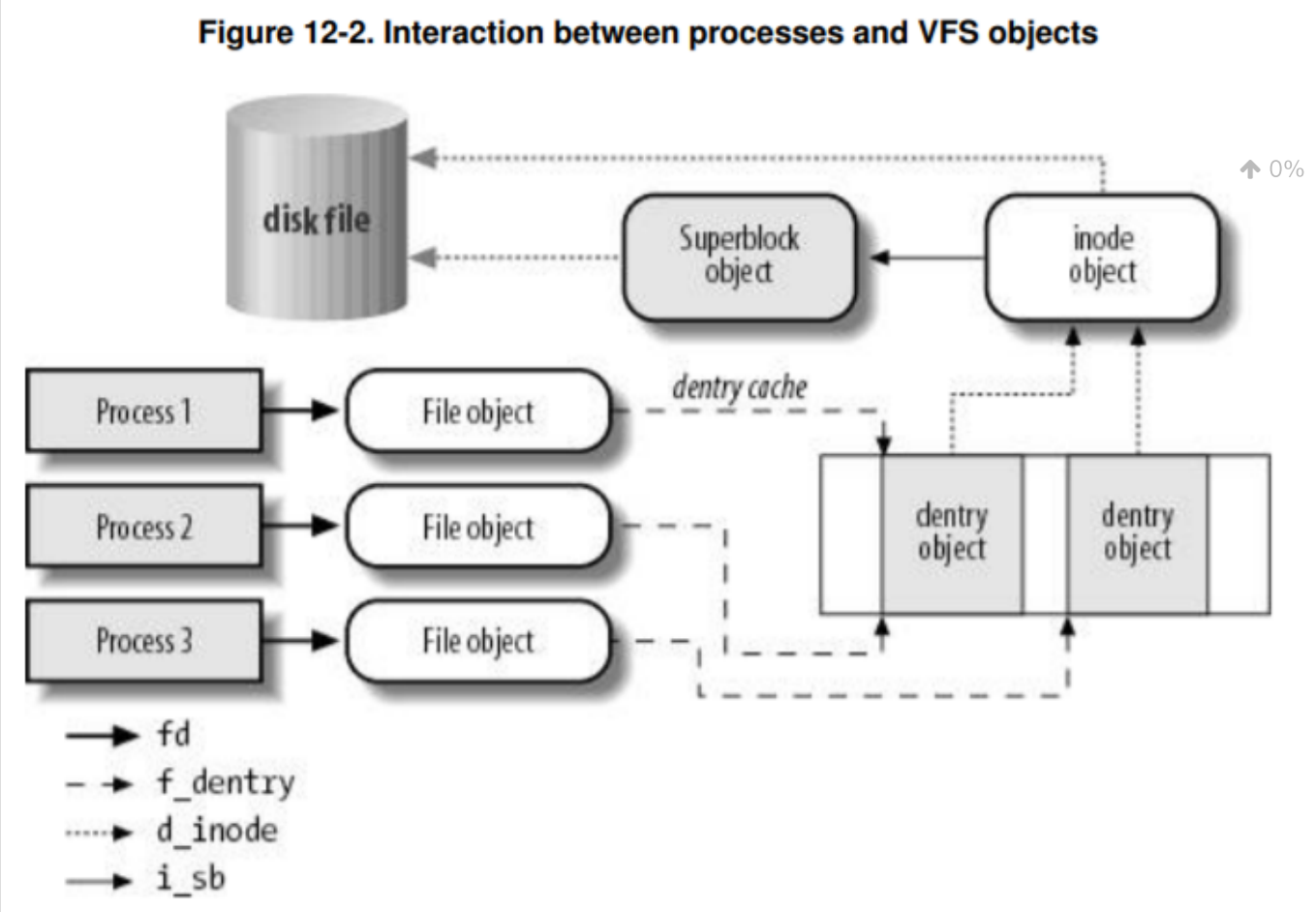
目录项对象存放在名为 dentry_cache 的slab分配器高速缓存中。因此，目录项对象的创建和删除是通过 kmem_cache_alloc() 和 kmem_cache_free() 实现的。

```
1  /* /include/linux/dcache.h */
2  struct dentry {
3          /* RCU lookup touched fields */
4          unsigned int d_flags;           /* protected by d_lock */
5          seqcount_t d_seq;               /* per dentry seqlock */
6          struct hlist_bl_node d_hash;    /* lookup hash list */
7          struct dentry *d_parent;        /* parent directory */
8          struct qstr d_name;
9          struct inode *d_inode;          /* Where the name belongs to - NULL is
10                                          * negative */
11         unsigned char d_iname[DNAME_INLINE_LEN];        /* small names */
```

```
12
13          /* Ref lookup also touches following */
14          struct lockref d_lockref;       /* per-dentry lock and refcount */
15          const struct dentry_operations *d_op;
16          struct super_block *d_sb;       /* The root of the dentry tree */
17          unsigned long d_time;           /* used by d_revalidate */
18          void *d_fsdata;                 /* fs-specific data */
19
20          union {
21                  struct list_head d_lru;         /* LRU list */
22                  wait_queue_head_t *d_wait;      /* in-lookup ones only */
23          };
24          struct list_head d_child;       /* child of parent list */
25          struct list_head d_subdirs;     /* our children */
26          /*
27           * d_alias and d_rcu can share memory
28           */
29          union {
30                  struct hlist_node d_alias;      /* inode alias list */
31                  struct hlist_bl_node d_in_lookup_hash;  /* only for in-lookup ones
32                  struct rcu_head d_rcu;
33          } d_u;
34  } __randomize_layout;
```

inode dentry file 对象之间的关系

下图是一个简单的实例。三个不同的进程打开同一个文件，其中两个进程使用同一个硬链接，还有一个进程使用另一个硬链接，但是都是指向同一个文件的。在这种情况下，其中的每个进程都使用自己的文件对象，但只需要两个目录项对象，每个硬链接对应一个目录项对象。这两个目录项对象都指向同一个索引节点对象，该索引节点对象标识超级块对象。



与进程相关的文件

fs_struct

文件描述符 task_struct 中有一个字段 struct fs_struct *fs ，它记录了进程当前的工作目录和根目录等信息，这是内核用来表示进程与文件系统相互作用所必须维护的数据。

```
/* /include/linux/fs_struct.h */
struct fs_struct {
        int users;
        spinlock_t lock;
        seqcount_t seq;
        int umask;
        int in_exec;
        struct path root, pwd; //root为根目录，pwd为当前工作的目录
} __randomize_layout;

/* /include/linux/path.h */
struct path {
        struct vfsmount *mnt;
        struct dentry *dentry;
} __randomize_layout;
```

files_struct

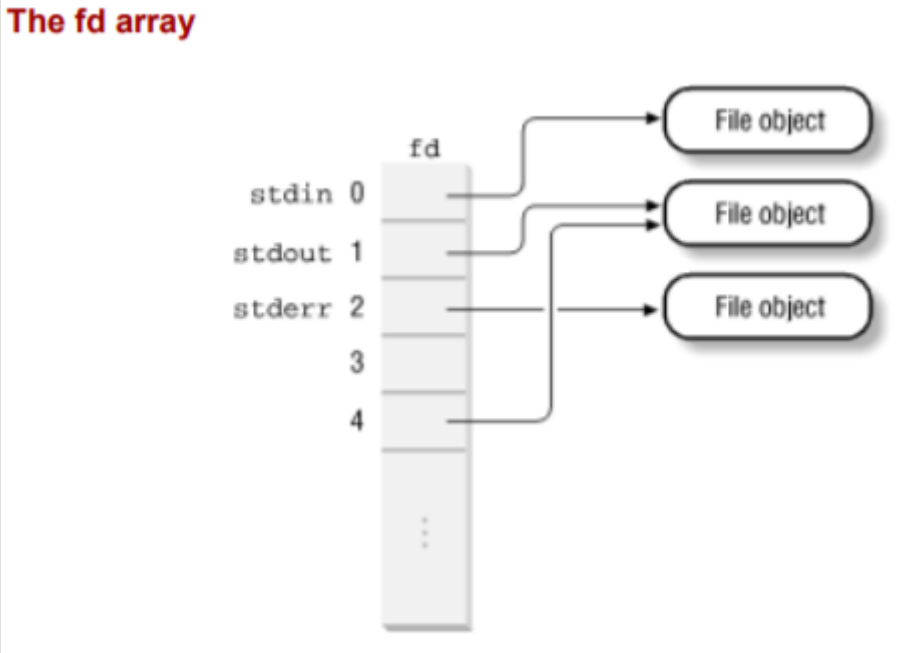
文件描述符 task_struct 中有一个字段 struct files_struct *files ，它记录了进程当前打开的文件的信息。


```
/* /include/linux/fdtable.h */
/*
 * Open file table structure
 */
struct files_struct {
  /*
   * read mostly part
   */
        atomic_t count;
        bool resize_in_progress;
        wait_queue_head_t resize_wait;

        struct fdtable __rcu *fdt;
        struct fdtable fdtab;
  /*
   * written part on a separate cache line in SMP
   */
        spinlock_t file_lock ____cacheline_aligned_in_smp;
        unsigned int next_fd;
        unsigned long close_on_exec_init[1];
        unsigned long open_fds_init[1];
        unsigned long full_fds_bits_init[1];
        struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};

struct fdtable {
        unsigned int max_fds;
        struct file __rcu **fd;      /* current fd array */
        unsigned long *close_on_exec;
        unsigned long *open_fds;
        unsigned long *full_fds_bits;
        struct rcu_head rcu;
};
```

fdtable 中的 fd 是一个指针，指向一个数组， 数组中的每一个元素都是一个 struct file 类型的指针 ， 对于在fd数组中有相应对象指针的文件来说，数组的索引下标就是文件描述符(file descriptor)。通常，数组的第一个元素（索引为0）是进程的标准输入文件，数组的第二个元素（索引为1）是进程的标准输出文件，数组的第三个元素（索引为2）是进程的标准错误文件。



0%

请注意，借助于dup()、dup2()和fcntl()系统调用，两个文件描述符可以指向同一个打开的文件，也就是说，数组的两个元素可以指向同一个文件对象。例如，当用户将标准错误文件重定向到标准输出文件上时。

相关文章

- Linux mmap与DMA 初探 【未完成】
- Linux内核学习参考资料汇总
- Linux内核漏洞利用 Bypass SMEP
- QEMU+busybox 搭建Linux内核运行环境
- Linux内核-进程、线程初探
- 初探Linux内核UAF
- 初探Linux内核栈溢出

Linux内核

Linux内核-进程、线程初探
Linux中进行glibc源码调试的一种方法