

Linux Kernel Programming

The Virtual File System

Pierre Olivier

Systems Software Research Group @ Virginia Tech

April 20, 2017

Outline

- 1 General Presentation
- 2 VFS data structures
- 3 Filesystem and process data structures
- 4 Additional information

Outline

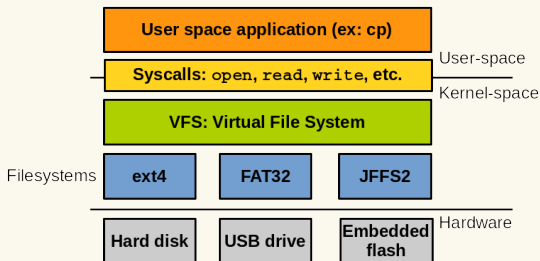
- 1 General Presentation
- 2 VFS data structures
- 3 Filesystem and process data structures
- 4 Additional information

General Presentation

Generalities

▶ The Virtual File System (VFS)

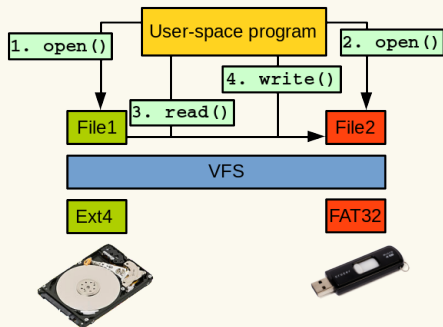
- ▶ Abstracts all the filesystems models supported by Linux
- ▶ Allow them to *coexist*
 - ▶ Example: user can have a USB drive formatted with FAT32 mounted at the same time as a HDD roots with ext4
- ▶ Allow them to *cooperate*
 - ▶ Example: user can seamlessly copy a file between the FAT32 and Ext4 partitions



General Presentation

Common filesystem interface

- ▶ VFS allows user-space to access files **independently** of the concrete filesystem they are stored on, with a **common interface**:
 - ▶ Standard system calls: `open()`, `read()`, `write()`, `lseek()`, etc.
 - ▶ "top" VFS interface (with user-space)
- ▶ Interface can work transparently between filesystems

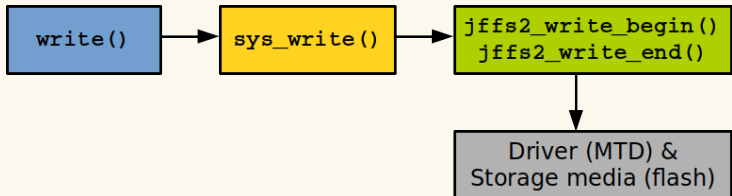


- ▶ `fd = open(path, flags)`
- ▶ `bytes_written = write(fd, buf, count)`
- ▶ `bytes_read = read(fd, buf, count)`
- ▶ `offset = lseek(fd, offset, whence)`
- ▶ `stat(path, struct_stat_ptr)`
- ▶ etc.

General Presentation

Filesystem abstraction layer

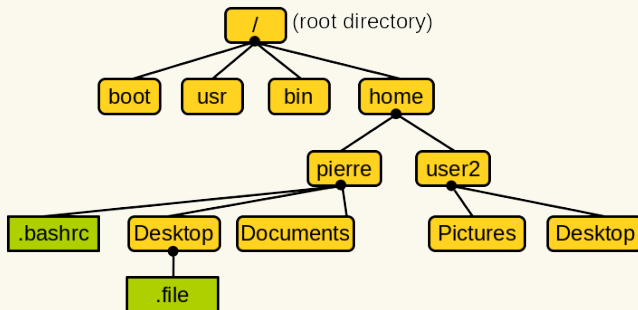
- ▶ VFS redirect user-space requests to the corresponding concrete filesystem
 - ▶ "bottom" VFS interface (with the filesystem)
 - ▶ Developing a new filesystem for Linux means **conforming** with the bottom interface



General Presentation

Unix filesystems

- ▶ The term *filesystem* can refer to a filesystem type or a partition
- ▶ Hierarchical tree of *files* organized into *directories*

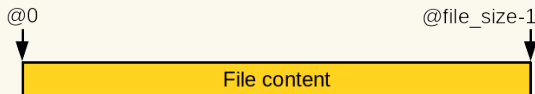


General Presentation

Unix filesystems (2)

▶ File:

- ▶ Ordered string of bytes from file address 0 to address (file size -1)



- ▶ Associated metadata: name, access permissions, modification date, etc.
 - ▶ Separated from the file data into specific objects (*inodes*, *dentries*)

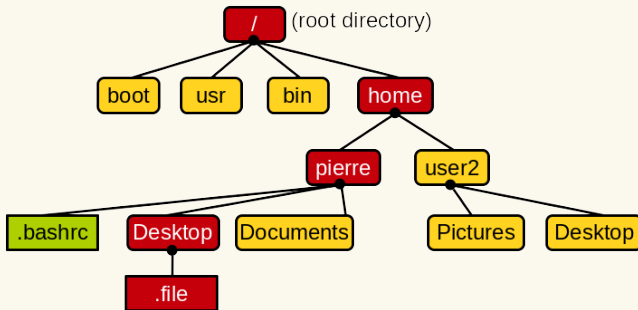
▶ Directory:

- ▶ Folder containing files or other directories (sub-directories)
- ▶ Sub-directories can be nested to create path:
`/home/pierre/Desktop/file`

General Presentation

Unix filesystems (3)

- ▶ Path example: `/home/pierre/Desktop/file:`



Outline

- 1 General Presentation
- 2 VFS data structures**
- 3 Filesystem and process data structures
- 4 Additional information

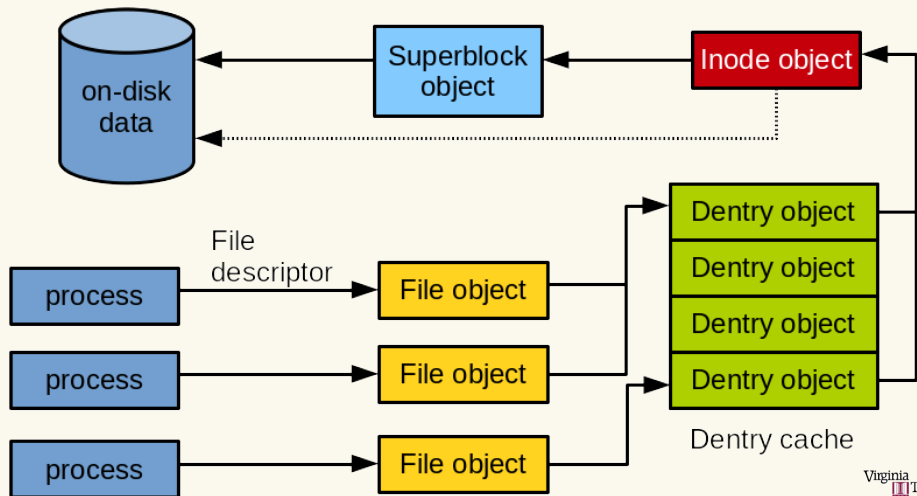
VFS data structures

Generalities

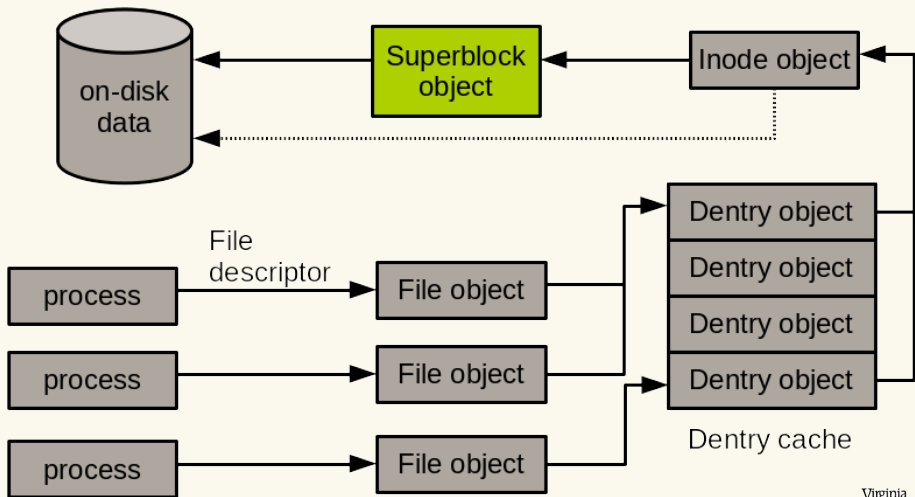
- ▶ **inode**: contains file/directory metadata
- ▶ **dentry**: contains file/directory name and hierarchical links defining the filesystem directory tree
- ▶ **superblock**: contains general information about the partition
- ▶ **file**: contains information about a file opened by a process
- ▶ **Associated operations**:
 - ▶ `super_operations`, `inode_operations`, `dentry_operations`, `file_operations`
 - ▶ Data structures containing function pointers
- ▶ **VFS implemented in a manner very close to object-oriented programming**

VFS data structures

Generalities (2)



VFS data structures



VFS data structures

Superblock object

- ▶ **Superblock**: contains global information about the filesystem (partition)
- ▶ Created by the filesystem and given to VFS at mount time:
 - ▶ Disk-based filesystem store it in a special location
 - ▶ Other filesystems have a way to generate it at mount time
- ▶ **struct super_block** defined in `include/ linux/fs.h`
 - ▶ Some fields:

```
1 struct super_block {
2     struct list_head s_list;           /* list of all superblocks */
3     dev_t             s_dev;           /* identifier */
4     unsigned long    s_blocksize;     /* block size (bytes) */
5     unsigned long    s_blocksize_bits; /* block size (bits) */
6     loff_t           s_maxbytes;      /* max file size */
7     /* ... */
```

VFS data structures

Superblock object (2)

```

1  /* ... */
2  struct file_system_type      *s_type;          /* filesystem type */
3  struct super_operations      *s_op;           /* superblock operations */
4  struct dquot_operations      *dq_op;          /* quota methods */
5  struct quotactl_ops          *s_qcop;         /* quota control methods */
6  unsigned long                s_flags;         /* mount flags */
7  unsigned long                s_magic;         /* filesystem magic number */
8  struct dentry                s_root;          /* directory mount point */
9  struct rw_semaphore          s_umount;        /* umount semaphore */
10 int                           s_count;        /* superblock reference count */
11 atomic_t                     s_active;        /* active reference count */
12 struct xattr_handler          **s_xattr;       /* extended attributes handler */
13 struct list_head              s_inodes;        /* inodes list */
14 struct hlist_bl_head          s_anon;          /* anonymous entries */
15 struct list_lru               s_dentry_lru;   /* list of unused dentries */
16 struct block_device           *s_bdev;         /* associated block device */
17 struct mtd_info               *s_mtd;          /* embedded flash information */
18 struct hlist_node             s_instances;     /* instances of this filesystem */
19 struct quota_info             s_dquot;         /* quota-specific options */
20 char                          s_id[32];       /* text name */
21 void                          *s_fs_info;     /* filesystem-specific info */
22 fmode_t                       s_mode;         /* mount permissions */
23 /* ... */
24 }

```

VFS data structures

Superblock operations

- ▶ **struct super_operations**
 - ▶ Each field is a function pointer operating on a struct `super_block`
 - ▶ Usage: `sb->s_op->write_super(sb);`
 - ▶ C++ OOP equivalent would be `sb.write_super();`
- ▶ `include/linux/fs.h:`

```
1 struct super_operations {
2     struct inode *(*alloc_inode)(struct super_block *sb);
3     void (*destroy_inode)(struct inode *);
4     void (*dirty_inode)(struct inode *, int flags);
5     int (*write_inode)(struct inode *, struct writeback_control *wbc);
6     int (*drop_inode)(struct inode *);
7     void (*evict_inode)(struct inode *);
8     void (*put_super)(struct super_block *);
9     int (*sync_fs)(struct super_block *sb, int wait);
10    /* ... */
11 }
```


VFS data structures

Superblock operations (2)

- ▶ `struct inode * alloc_inode(struct super_block *sb)`
 - ▶ **Creates and initialize a new inode**
- ▶ `void destroy_inode(struct inode *inode)`
 - ▶ **Deallocate an inode**
- ▶ `void dirty_inode(struct inode *inode)`
 - ▶ **Marks an inode as dirty (Ext filesystems)**
- ▶ `void write_inode(struct inode *inode, int wait)`
 - ▶ **Writes the inode to disk, `wait` specifies if the write should be synchronous**
- ▶ `void drop_inode(struct inode *inode)`
 - ▶ **Called by VFS when the last reference to the inode is dropped**
- ▶ `void put_super(struct super_block *sb)`
 - ▶ **Called by VFS on unmount (holding `s_lock`)**

VFS data structures

Superblock operations (3)

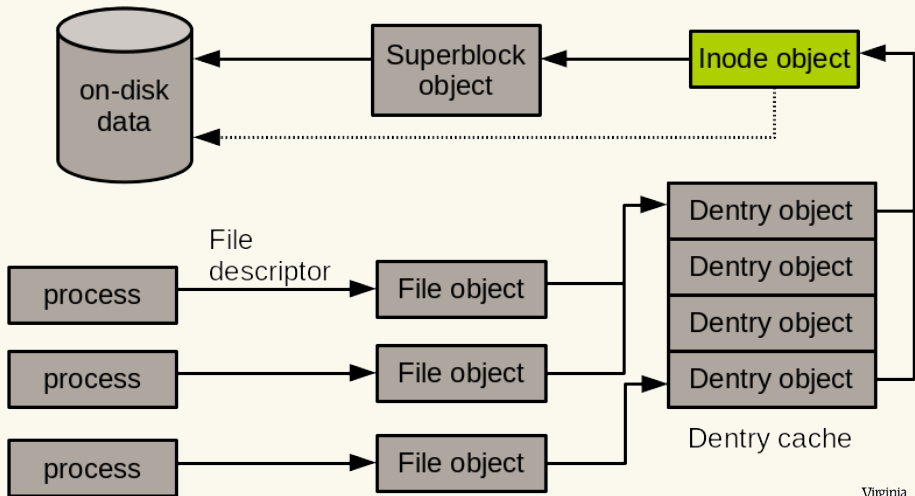
- ▶ `void write_super(struct super_block *sb)`
 - ▶ Update the on-disk superblock, caller must hold `s_lock`
- ▶ `int sync_fs(struct super_block *sb, int wait)`
 - ▶ Synchronize filesystem metadata with on-disk filesystem, `wait` specifies if the operation should be synchronous
- ▶ `void write_super_lockfs(struct super_block *sb)`
 - ▶ Prevents changes to the filesystem and update the on-disk superblock (used by the Logical Volume Manager)
- ▶ `void unlockfs(struct super_block *sb)`
 - ▶ Unlocks the filesystem locked by `write_super_lockfs()`

VFS data structures

Superblock operations (4)

- ▶ `int statfs(struct super_block *sb, struct statfs *statfs)`
 - ▶ Obtain filesystem statistics
- ▶ `int remount_fs(struct super_block *sb, int *flags, char *data)`
 - ▶ Remount the filesystem with new options, caller must hold `s_lock`
- ▶ `void clear_inode(struct inode *inode)`
 - ▶ Releases the inode and clear any page containing related data
- ▶ `void umount_begin(struct super_block *sb)`
 - ▶ Called by VFS to interrupt a mount operation (NFS)
- ▶ All of these functions are called by VFS and may block (except `dirty_inode()`)

VFS data structures



VFS data structures

Inode object

▶ Inode object

- ▶ Related to a file or directory, contains metadata plus information about how to manipulate the file/directory
- ▶ Metadata: file size, owner id/group, etc
- ▶ Must be produced by the filesystem on-demand when a file/directory is accessed:
 - ▶ Read from disk in Unix-like filesystem
 - ▶ Reconstructed from on-disk information for other filesystems
- ▶ **struct inode** (include/linux/fs.h):

```
1 struct inode {
2     struct hlist_node i_hash;    /* hash list */
3     struct list_head i_list;    /* list of inodes */
4     struct list_head i_sb_list; /* list of superblock */
5     struct list_head i_dentry;  /* list of dentries */
6     /* ... */
```



VFS data structures

Inode object (2)

```

1  /* ... */
2  unsigned long      i_ino;          /* inode number */
3  atomic_t           i_count;       /* reference counter */
4  unsigned int       i_nlink;       /* number of hard links */
5  uid_t              i_uid;         /* user id of owner */
6  gid_t              i_gid;         /* group id of owner */
7  kdev_t             i_rdev;        /* real device node */
8  u64                 i_version;    /* versioning number */
9  loff_t              i_size;        /* file size in bytes */
10 seqcount_t          i_size_seqcount /* seqlock for i_size */
11 struct timespec    i_atime;       /* last access time */
12 struct timespec    i_mtime;       /* last modify time (file content) */
13 struct timespec    i_ctime;       /* last change time (file or attributes content) */
14 unsigned int       i_blkbits;     /* block size in bits */
15 blkcnt_t           i_blocks;      /* file size in blocks */
16 unsigned short     i_bytes;       /* bytes consumed */
17 spinlock_t         i_lock;        /* inode spinlock */
18 struct rw_semaphore i_alloc_sem;  /* nests inside of i_sem */
19 struct semaphore    i_sem;        /* inode semaphore */
20 struct inode_operations *i_ops;    /* inode operations */
21 struct file_operations *i_fop;     /* file operations */
22 struct super_block  i_sb;         /* associated superblock */
23 /* ... */

```

VFS data structures

Inode object (3)

```
1  /* ... */
2  struct dquot          *i_dquot[MAXQUOTAS]; /* disk quotas for inode */
3  struct list_head      i_devices;          /* list of block device */
4  union {
5      struct pipe_inode_info *i_pipe; /* pipe information */
6      struct block_device    *i_bdev; /* block device driver */
7      struct cdev            *i_cdev; /* character device */
8  };
9  unsigned long         i_dnotify_mask; /* directory notify mask */
10 struct dnotify_struct *i_dnotify;     /* dnotify */
11 struct list_head      inotify_watches; /* inotify watches */
12 struct mutex          inotify_mutex;  /* protects inotify_watches */
13 unsigned long         i_state;        /* state flags */
14 unsigned long         dirtied_when;    /* first dirtying time */
15 unsigned int          i_flags;        /* filesystem flags */
16 atomic_t              i_writecount;    /* count of writers */
17 void *                i_private;      /* filesystem private data */
18 /* ... */
19 }
```

VFS data structures

Inode operations

- ▶ Operations that can be invoked on an inode object
- ▶ **struct inode_operations** defined in `include/linux/fs.h`

```
1 struct inode_operations {
2     int (*create) (struct inode *, struct dentry *, umode_t, bool);
3     int (*link) (struct dentry *, struct inode *, struct dentry *);
4     int (*unlink) (struct inode *, struct dentry *);
5     int (*symlink) (struct inode *, struct dentry *, const char *);
6     int (*mkdir) (struct inode *, struct dentry *, umode_t);
7     /* ... */
8 }
```


VFS data structures

Inode operations (2)

- ▶ `int create(struct inode *dir, struct dentry *dentry, int mode)`
 - ▶ **Create a new inode with access mode `mode`**
 - ▶ **Called from `creat()` and `open()` syscalls**
- ▶ `struct dentry * lookup(struct inode *dir, struct dentry *dentry)`
 - ▶ **Searches a directory (inode) for a file/directory (dentry)**
- ▶ `int link(struct dentry *old_dentry, struct inode *dir, struct dentry *dentry)`
 - ▶ **Creates a hard link with name `dentry` in the directory `dir`, pointing to `old_dentry`**
- ▶ `int unlink(struct inode *dir, struct dentry *dentry)`
 - ▶ **Remove an inode (`dentry`) from the directory `dir`**

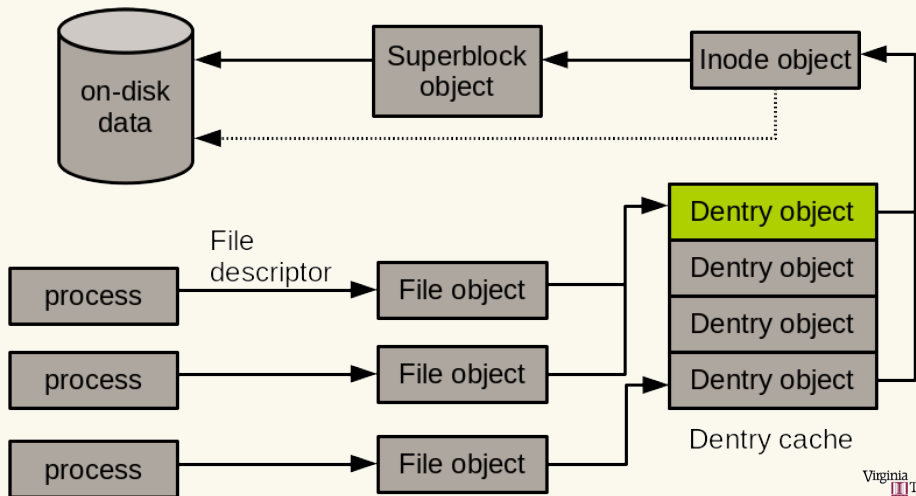
VFS data structures

Inode operations (3)

- ▶ `int symlink(struct inode *dir, struct dentry *dentry, const char *symname)`
 - ▶ **Creates a symbolic link named `symname`, to the file `dentry` in directory `dir`**
- ▶ `int mkdir(struct inode *dir, struct dentry *dentry, int mode)`
 - ▶ **Creates a directory inside `dir` with name `dentry`**
- ▶ `int rmdir(struct inode *dir, struct dentry *dentry)`
 - ▶ **Removes a directory `dentry` from `dir`**
- ▶ `int mknod(struct inode *dir, struct dentry *dentry, int mode, dev_t rdev)`
 - ▶ **Creates a special file (device file, pipe, socket)**
- ▶ `int rename(struct inode *old_dir, struct dentry *old_dentry, struct inode *new_dir, struct dentry *new_dentry)`
 - ▶ **Moves a file**

VFS data structures

Dentry object



VFS data structures

Dentry object

▶ **Dentry object**

- ▶ Associated with a file or a directory to:
 - ▶ Store the file/directory **name**
 - ▶ Store its **location in the directory tree**
 - ▶ Perform directory specific operations, for example pathname lookup
- ▶ `/home/pierre/test.txt`:
 - ▶ One dentry associated with each of: `/`, `home`, `pierre` and `test.txt`
- ▶ Constructed on the fly as files and directories are accessed:
generally no on-disk representation

VFS data structures

Dentry object

- ▶ struct dentry defined in include/linux/dcache.h

```
1 struct dentry {
2     atomic_t          d_count;    /* usage count */
3     unsigned int      d_flags;    /* dentry flags */
4     spinlock_t        d_lock;     /* per-dentry lock */
5     int               d_mounted;  /* indicate if it is a mount point */
6     struct inode       *d_inode;   /* associated inode */
7     struct hlist_node d_hash;     /* list of hash table entries */
8     struct dentry      *d_parent;  /* parent dentry */
9     struct qstr        d_name;     /* dentry name */
10    struct list_head    d_lru;      /* unused list */
11    struct list_head     d_subdirs; /* sub-directories */
12    struct list_head     d_alias;   /* list of dentries pointing to the same inode */
13    unsigned long        d_time;    /* last time validity was checked */
14    struct dentry_operations *d_op; /* operations */
15    struct super_block    *d_sb;     /* superblock */
16    void                 *d_fsdata; /* filesystem private data */
17    unsigned char        d_iname[DNAME_INLINE_LEN_MIN]; /* short name */
18    /* ... */
19 };
```

VFS data structures

Dentry states

- ▶ A dentry can be **used**, **unused** or **negative**
- ▶ **Used**: corresponds to a valid inode (pointed by `d_inode`) with one or more users (`d_count`)
 - ▶ Cannot be discarded to free memory
- ▶ **Unused**: valid inode, but no current users
 - ▶ Kept in RAM for caching
 - ▶ Can be discarded
- ▶ **Negative**: does not point to a valid inode
 - ▶ Ex: `open()` on a file that does not exist
 - ▶ Kept around for caching
 - ▶ Can be discarded

VFS data structures

The dentry cache

- ▶ Dentries are constructed on demand and **kept in RAM for quick future pathname lookups**
 - ▶ **Dentry cache** or Dcache
- ▶ Three parts:
 - ▶ Linked list of used dentries linked by the `i_dentry` field of their inode
 - ▶ One inode can have multiple links, thus multiple dentries
 - ▶ Linked list of LRU sorted unused and negative dentries
 - ▶ LRU: quick reclamation from the tail of the list
 - ▶ Hash table + hash function to quickly resolve a path into the corresponding dentry present in the dcache

VFS data structures

The dentry cache (2)

- ▶ **Hash table:** `dentry_hashtable` **array**
 - ▶ Each element is a pointer to a list of dentries hashing to the same value
- ▶ **Hashing function:** `d_hash()`
 - ▶ Filesystem can provide its own hashing function
- ▶ **Dentry lookup in the dcache:** `d_lookup()`
 - ▶ Returns dentry on success, `NULL` on failure
- ▶ **Inodes are similarly cached in RAM, in the **inode cache****
 - ▶ Dentries in the dcache are pinning inodes in the inode cache

VFS data structures

Dentry operations

- ▶ `struct dentry_operations` defined in `include/linux/dcache.h`

```
1 struct dentry_operations {
2     int (*d_revalidate)(struct dentry *, unsigned int);
3     int (*d_weak_revalidate)(struct dentry *, unsigned int);
4     int (*d_hash)(const struct dentry *, struct qstr *);
5     int (*d_compare)(const struct dentry *,
6         unsigned int, const char *, const struct qstr *);
7     int (*d_delete)(const struct dentry *);
8     int (*d_init)(struct dentry *);
9     void (*d_release)(struct dentry *);
10    void (*d_prune)(struct dentry *);
11    void (*d_iput)(struct dentry *, struct inode *);
12    char *(*d_dname)(struct dentry *, char *, int);
13    struct vfsmount *(*d_automount)(struct path *);
14    int (*d_manage)(const struct path *, bool);
15    struct dentry *(*d_real)(struct dentry *, const struct inode *,
16        unsigned int);
17 } ____cacheline_aligned;
```

VFS data structures

Dentry operations (2)

- ▶ `int d_revalidate(struct dentry *dentry, struct nameidata *)`
 - ▶ Determine if an entry to use from the dcache is valid
 - ▶ Generally set to `NULL`
- ▶ `int d_hash(struct dentry *dentry, struct qstr *name)`
 - ▶ Create a hash value for a dentry to insert in the dcache
- ▶ `int d_compare(struct dentry *dentry, struct qstr *name1, struct qstr *name2)`
 - ▶ Compare two filenames, requires `dcache_lock`
- ▶ `int d_delete (struct dentry *dentry)`
 - ▶ Called by VFS when `d_count` reaches zero, requires `dcache_lock` and `d_lock`

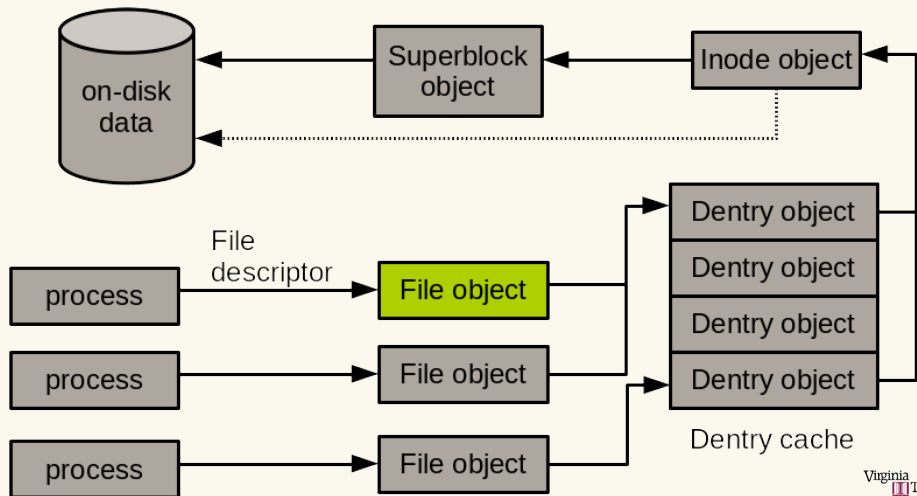
VFS data structures

Dentry operations (3)

- ▶ `void d_release(struct dentry *dentry)`
 - ▶ Called when the dentry is going to be freed
- ▶ `void d_iput(struct dentry *dentry, struct inode *inode)`
 - ▶ Called when the dentry loses its inode
 - ▶ Calls `iput()`

VFS data structures

File object



VFS data structures

File object

- ▶ The **file** object
 - ▶ Represents a file opened by a process
 - ▶ Created on `open()` and destroyed on `close()`
- ▶ 2 processes opening the same file:
 - ▶ Two file objects, pointing to the same unique dentry, that points itself on a unique inode
- ▶ No corresponding on-disk data structure

VFS data structures

File object (2)

- ▶ struct file defined in include/linux/fs.h

```
1 struct file {
2     struct path                f_path;                /* contains the dentry */
3     struct file_operations     *f_op;                /* operations */
4     spinlock_t                 f_lock;                /* lock */
5     atomic_t                   f_count;              /* usage count */
6     unsigned int               f_flags;              /* open flags */
7     mode_t                     f_mode;              /* file access mode */
8     loff_t                     f_pos;                /* file offset */
9     struct fown_struct         f_owner;              /* owner data for signals */
10    const struct cred           *f_cred;              /* file credentials */
11    struct file_ra_state        f_ra;                /* read-ahead state */
12    u64                         f_version;            /* version number */
13    void                        *private_data;        /* private data */
14    struct list_head            f_ep_link;            /* list of epoll links */
15    spinlock_t                 f_ep_lock;            /* epoll lock */
16    struct address_space        *f_mapping;           /* page cache mapping */
17    /* ... */
18 };
```

VFS data structures

File operations

- ▶ **struct file_operations** defined in `include/ linux/fs.h`

```
1 struct file_operations {
2     struct module *owner;
3     loff_t (*llseek) (struct file *, loff_t, int);
4     ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
5     ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
6     ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
7     ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
8     int (*iterate) (struct file *, struct dir_context *);
9     int (*iterate_shared) (struct file *, struct dir_context *);
10    unsigned int (*poll) (struct file *, struct poll_table_struct *);
11    /* ... */
12};
```

VFS data structures

File operations

- ▶ `loff_t llseek(struct file *file, loff_t offset, int origin)`
 - ▶ **Update file offset**
- ▶ `ssize_t read(struct file *file, char *buf, size_t count, loff_t *offset)`
 - ▶ **Read operation**
- ▶ `ssize_t aio_read(struct kiocb *iocb, char *buf, size_t count, loff_t offset)`
 - ▶ **Asynchronous read**
- ▶ `ssize_t write(struct file *file, const char *buf, size_t count, loff_t *offset)`
 - ▶ **Write operation**
- ▶ `ssize_t aio_write(struct kiocb *iocb, const char *buf, size_t count, loff_t offset)`
 - ▶ **Asynchronous write**

VFS data structures

File operations (2)

- ▶ `int readdir(struct file *file, void *dirent, filldir_t filldir)`
 - ▶ Read the next directory in a directory listing
- ▶ `unsigned int poll(struct file *file, struct poll_table_struct *poll_table)`
 - ▶ Sleeps waiting for activity on a given file
- ▶ `int ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)`
 - ▶ Sends a command and arguments to a device
 - ▶ Unlocked/compat versions
- ▶ `int mmap(struct file *file, struct vm_area_struct *vma)`
 - ▶ Maps a file into an address space

VFS data structures

File operations (3)

- ▶ `int open(struct inode *inode, struct file *file)`
 - ▶ **Opens a file**
- ▶ `int flush(struct file *file)`
 - ▶ **Called by VFS when the reference count of an open file decreases**
- ▶ `int release(struct inode *inode, struct file *file)`
 - ▶ **Called by VFS when the last reference to a file is destroyed**
(`close()/exit()`)
- ▶ `int fsync(struct file *file, struct dentry *dentry, int datasync)`
 - ▶ **Flush cached data on disk**
- ▶ `int aio_fsync(struct kiocb *iocb, int datasync)`
 - ▶ **Flush aio cached data on disk**

VFS data structures

File operations (4)

- ▶ `int lock(struct file *file, int cmd, struct file_lock *lock)`
 - ▶ **Manipulate a file lock**
- ▶ `ssize_t writev(struct file *file, const struct iovec *vector, unsigned long count, loff_t *offset)`
- ▶ `ssize_t readv(struct file *file, const struct iovec *vector, unsigned long count)`
 - ▶ **Vector read/write operations (used by the `readv` and `writev` family functions)**
- ▶ `ssize_t sendfile(struct file *file, loff_t *offset, size_t size, read_actor_t actor, void *target)`
 - ▶ **Copy data from one file to another entirely in the kernel**

VFS data structures

File operations (5)

- ▶ `ssize_t sendpage(struct file *file, struct page *page, int offset, size_t size, loff_t *pos, int more)`
 - ▶ **Send data from one file to another**
- ▶ `unsigned long get_unmapped_area(struct file *file, unsigned long addr, unsigned long len, unsigned long offset, unsigned long flags)`
 - ▶ **Get a section of unused address space to map a file**
- ▶ `int flock(struct file *filp, int cmd, struct file_lock *fl)`
 - ▶ **Used by the `flock()` syscall**

Outline

- 1 General Presentation
- 2 VFS data structures
- 3 Filesystem and process data structures**
- 4 Additional information

Filesystem and process data structures

Filesystem data structures

- ▶ **struct file_system_type**: information about a specific concrete filesystem type
- ▶ One per filesystem supported (chosen at compile time) independently of the mounted filesystem
- ▶ Defined in `include/linux/fs.h`:

Filesystem and process data structures

Filesystem data structures (2)

```
1 struct file_system_type {
2     const char *name;      /* name */
3     int fs_flags;         /* flags */
4
5     /* mount a partition */
6     struct dentry *(*mount) (struct file_system_type *, int,
7                             const char *, void *);
8
9     /* terminate access to the superblock */
10    void (*kill_sb) (struct super_block *);
11    struct module *owner;      /* module owning the fs */
12    struct file_system_type * next; /* linked list of fs types */
13    struct hlist_head fs_supers; /* linked list of superblocks */
14
15    /* runtime lock validation */
16    struct lock_class_key s_lock_key;
17    struct lock_class_key s_umount_key;
18    struct lock_class_key s_vfs_rename_key;
19    struct lock_class_key s_writers_key[SB_FREEZE_LEVELS];
20
21    struct lock_class_key i_lock_key;
22    struct lock_class_key i_mutex_key;
23    struct lock_class_key i_mutex_dir_key;
24 };
```

Filesystem and process data structures

Filesystem data structures (3)

- ▶ When a filesystem is mounted, a `vfsmount` structure is created
 - ▶ Represent a specific instance of the filesystem: a mount point
- ▶ `include/linux/mount.h`

```
1 struct vfsmount {
2     struct dentry *mnt_root; /* root of the mounted tree */
3     struct super_block *mnt_sb; /* pointer to superblock */
4     int mnt_flags;
5 };
```


Filesystem and process data structures

Process data structure (4)

- ▶ `struct files_struct`: contains per-process information about opened files and file descriptors
 - ▶ `include/linux/fdtable.h`
- ▶ `struct fs_struct`: filesystem information related to a process
 - ▶ `include/linux/fs_struct.h`
- ▶ `struct mnt_namespace`: provide processes with unique views of a mounted filesystem
 - ▶ `fs/mount.h`

Outline

- 1 General Presentation
- 2 VFS data structures
- 3 Filesystem and process data structures
- 4 Additional information**

Additional information

- ▶ Documentation/filesystems
- ▶ *Understanding the Linux Kernel*, chapter 12
- ▶ *Linux Kernel Architecture*, chapter 8