Operating Systems Design
12. File System Design

Paul Krzyzanowski
pxk@cs.rutgers.edu

1

What's a file system

- Organization of data and metadata
- What's metadata?
  - Attributes; things that describe the data
  - Name, length, type of file, creation/modification/access times, permissions, owner, location of data
- File systems usually interact with block devices

2

Design Choices

Namespace

Flat, hierarchical, or other?

Multiple volumes

Explicit device identification (A:, B:, C:, D:)

or integrate into one namespace?

File types

Unstructured (byte streams)

or structured (e.g., indexed files)?

File system types

Support one type of file system

or multiple types (iso9660, NTFS, ext3)?

Metadata

What kind of attributes should the file system have?

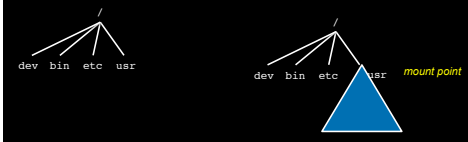
Implementation

How is the data laid out on the disk?

3

Mounting

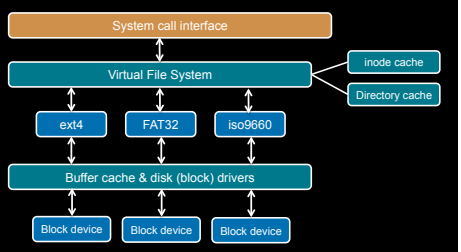
- A file system must be *mounted* before it can be used by the operating system
- The *mount* system call is given the file system type, block device & mount point
- The mounted file system overlays anything under that mount point
- Looking up a pathname may involve traversing multiple mount points



4

Virtual File System (VFS) Interface

- Abstract interface for a file system object
- Each *real* file system interface exports a common interface



5

Keeping track of file system types

- Like drivers, file systems can be built into the kernel or compiled as loadable modules (loaded at mount)
- Each file system registers itself with VFS
- Kernel maintains a list of file systems

```
struct file_system_type {
        const char *name;   name of file system type
        int fs_flags;       requires device, fs handles moves, kernel-only mount, ...
        struct super_block *(*get_sb)(struct file_system_type *,
                int, char *, void *, struct vfsmount *); set up superblock
        void (*kill_sb) (struct super_block *); clean up at unmount
        struct module *owner; module that owns this
        struct file_system_type *next; next file system type in list
        struct list_head fs_supers; list of all superblocks of this type
        struct lock_class_key s_lock_key; used for lock validation
        struct lock_class_key s_umount_key; used for lock validation
};
```

6

Keeping track of mounted file systems

- Before mounting a file system, first check if we know the file system type: look through the `file_systems` list
  - If not found, the kernel daemon will load the file system module
    - `/lib/modules/2.6.38-8-server/kernel/fs/ntfs/ntfs.ko`
    - `/lib/modules/2.6.38-11-server/kernel/fs/jffs2/jffs2.ko`
    - `/lib/modules/2.6.38-11-server/kernel/fs/minix/minix.ko`
- The kernel keeps a linked list of mounted file systems:
  `current->namespace->list`
- Check that the mount point is a directory and nothing is already mounted there

7

VFS: Common set of objects

- Superblock: Describes the file system
  - Block size, max file size, mount point
  - One per mounted file system
- inode: represents a single file
  - Unique identifier for every object (file) in a specific file system
  - File systems have methods to translate a name to an inode
  - VFS inode defines all the operations possible on it
- dentry: directory entries & contents
  - Name of file/directory, child dentries, parent
  - Directory entries: translations of names to inodes
- file: represents an open file
  - VFS keeps state: mode, read/write offset, etc.

8

VFS superblock

- Structure that represents info about the file system
- Includes
  - File system name
  - Size
  - State
  - Reference to the block device
  - List of operations for managing inodes within the file system:
    - *alloc_inode*, *destroy_inode*, *read_inode*, *write_inode*, *sync_fs*, …

9

inode

- Uniquely identifies a file in a file system
- Access metadata (attributes) of the file (except name)

```
struct inode {
        unsigned long i_ino;
        umode_t i_mode;
        uid_t i_uid;
        gid_t i_gid;
        kdev_t i_rdev;
        loff_t i_size;
        struct timespec i_atime;
        struct timespec i_ctime;
        struct timespec i_mtime;
        struct super_block *i_sb;
        struct inode_operations *i_op;
        struct address_space *i_mapping;
        struct list_head i_dentry;
        ...
}
```

inode operations

10

inode operations

Functions that operate on file & directory *names and attributes*

```
struct inode_operations {
        int (*create) (struct inode *, struct dentry *, int);
        struct dentry * (*lookup) (struct inode *, struct dentry *);
        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 *, int);
        int (*rmdir) (struct inode *, struct dentry *);
        int (*mknod) (struct inode *, struct dentry *, int, dev_t);
        int (*rename) (struct inode *, struct dentry *, struct inode *, struct dentry *);
        int (*readlink) (struct dentry *, char *,int);
        int (*follow_link) (struct dentry *, struct nameidata *);
        void (*truncate) (struct inode *);
        int (*permission) (struct inode *, 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 *);
};
```

11

File operations

Functions that operate on file & directory *data*

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, char *, size_t, loff_t);
        ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
        ssize_t (*aio_write) (struct kiocb *, const char *, size_t, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
                unsigned long, unsigned long);
};
```

12

File operations

Not all functions need to be implemented!

```
struct file_operations mydriver_fops = {
    .owner = MYDRIVER_MODULE;
    .open = mydriver_open;        /* allocate resources */
    .read = mydriver_read;
    .write = mydriver_write;
    .ioctl = mydriver_ioctl;
    .release = mydriver_release;  /* release resources */
    /* llseek, readdir, poll, mmap, readv, etc. not implemented */
};

register_chrdev(MYDRIVER_MAJOR_NUM, "mydriver", &mydriver_fops)
```

13

The End

14