Operating Systems

13. File Systems

Paul Krzyzanowski

Rutgers University

Spring 2015

Terminology

What's a file system?

- Traditionally
  - A way to manage variable-size persistent data
    - Organize, store, retrieve, delete information
  - Random access
    - Arbitrary files can be accessed by name
    - Arbitrary parts of a file can be accessed
  - File systems are implemented on top of block devices

- More abstract
  - A way to access information by name
    - Devices
    - System configuration, process info, random numbers

Terms

- Disk
  - Non-volatile block-addressable storage.
- Block = sector
  - Smallest chunk of I/O on a disk
  - Common block sizes = 512 or 4096 (4K) bytes
    E.g., WD Black Series 4TB drive has 7,814,037,168 512-byte sectors
- Partition
  - Set of contiguous blocks on a disk. A disk has ≥ 1 partitions
- Volume
  - Disk, disks, or partition that contains a file system
  - A volume may span disks

More terms

- Track
  - Blocks are stored on concentric *tracks* on a disk
- Cylinder
  - The set of all blocks on one track
    (obsolete now since we don’t know what’s where)
- Seek
  - The movement of a disk head from track to track

File Terms

- File
  - A unit of data managed by the file system

- Data: (Contents)
  -
  - Unstructured (byte stream) or structured (records)

- Name
  - A textual name that identifies the file

File Terms

- Metadata
  - Information about the file (creation time, permissions, length of file data, location of file data, etc.)
- Attribute
  - A form of metadata – a textual name and associated value (e.g., source URL, author of document, checksum)
- Directory (folder)
  - A container for file names
  - Directories within directories provide a hierarchical name system

File System Terms

- Superblock
  - Area on the volume that contains key file system information
- inode (file control block)
  - A structure that stores a file’s metadata and location of file data
- Cluster
  - Logical block size used in the file system that is equivalent to *N* blocks
- Extent
  - Group of contiguous clusters identified by a starting block number and a block count

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?

Working with the Operating System

File System Operations

Mounting

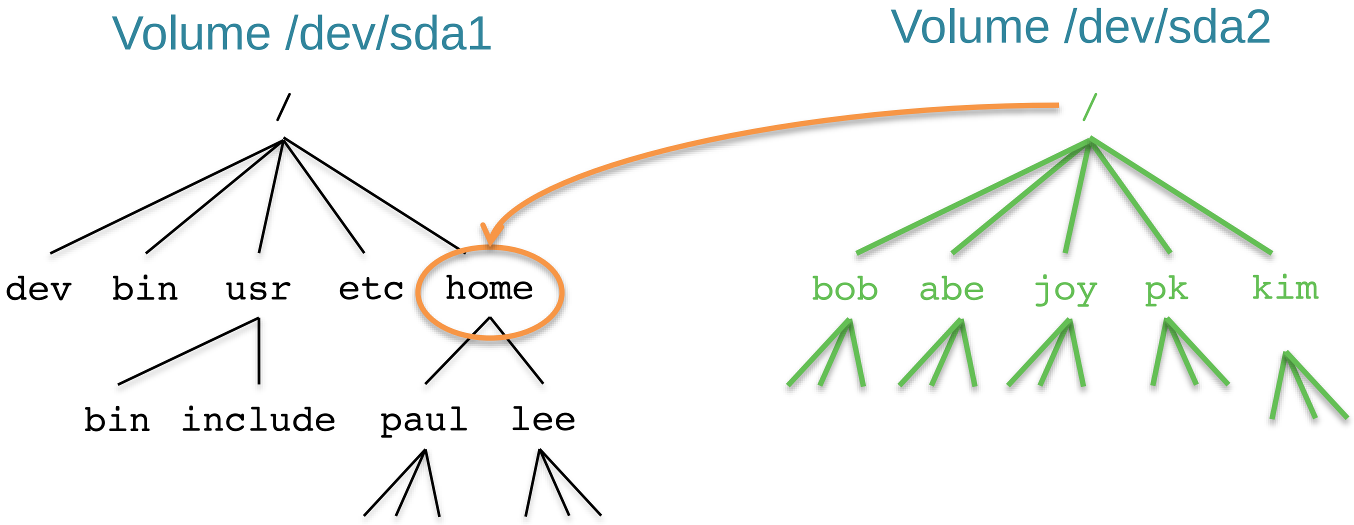
- Make file system available for use
- *mount* system call
  - Pass the file system type, block device & mount point
- Steps
  - Access the raw disk (block device)
  - Read superblock and file system metadata (free block bitmaps, root directory, etc.)
  - Check to see if the file system was properly unmounted (clean?)
    - If not, validate the structure of the file system
  - Prepare in-memory data structures to access the volume
    - In-memory version of the superblock
    - References to the root directory
    - Free block bitmaps
  - Mark the superblock as “dirty”

Unmounting

- Ensure there are no processes with open files in the file system
- Remove file system from the OS name space
- Flush all in-memory file system state to disk
- Mark the superblock as “clean” (unmount took place)

Mounting: building up a name space

- Combine multiple file systems into a single hierarchical name space
- The mounted file system overlays (& hides) anything in the file system under that mount point
- Looking up a pathname may involve traversing multiple mount points

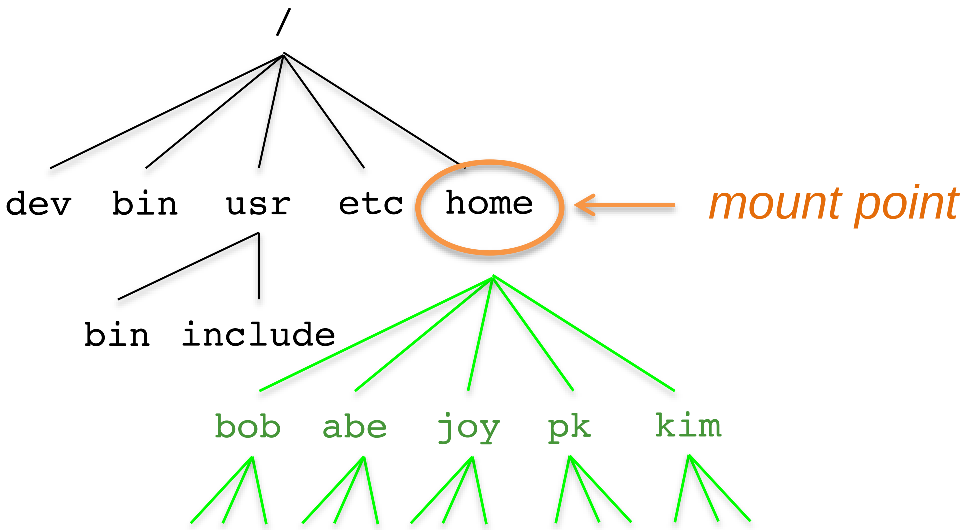


```
mount -t ext4 /dev/sda2 /home
```

`/home` becomes a mount point for `/dev/sda2`

Mounting: building up a name space

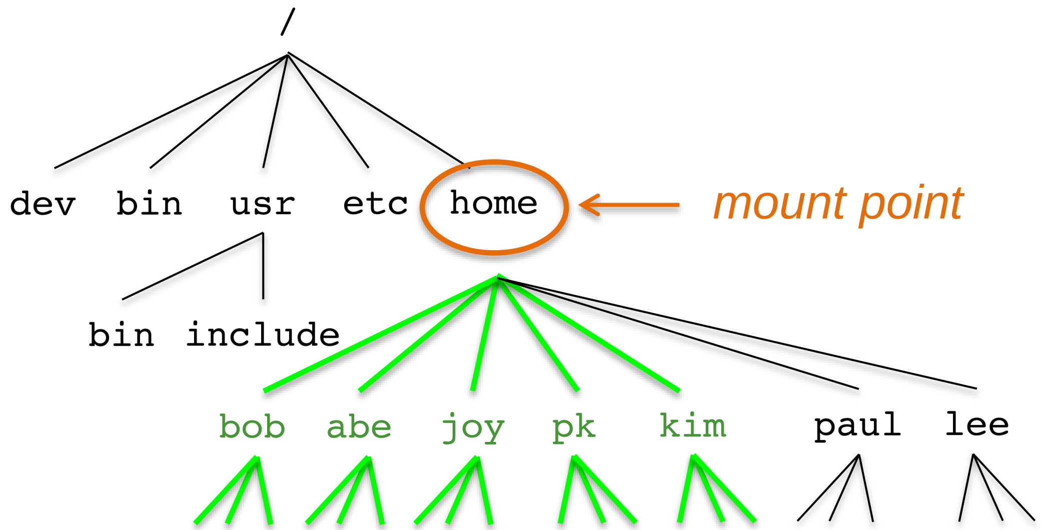
Volume /dev/sda1



/home/paul and /home/lee are no longer visible

Union mounts

Mounted file system merges the existing namespace



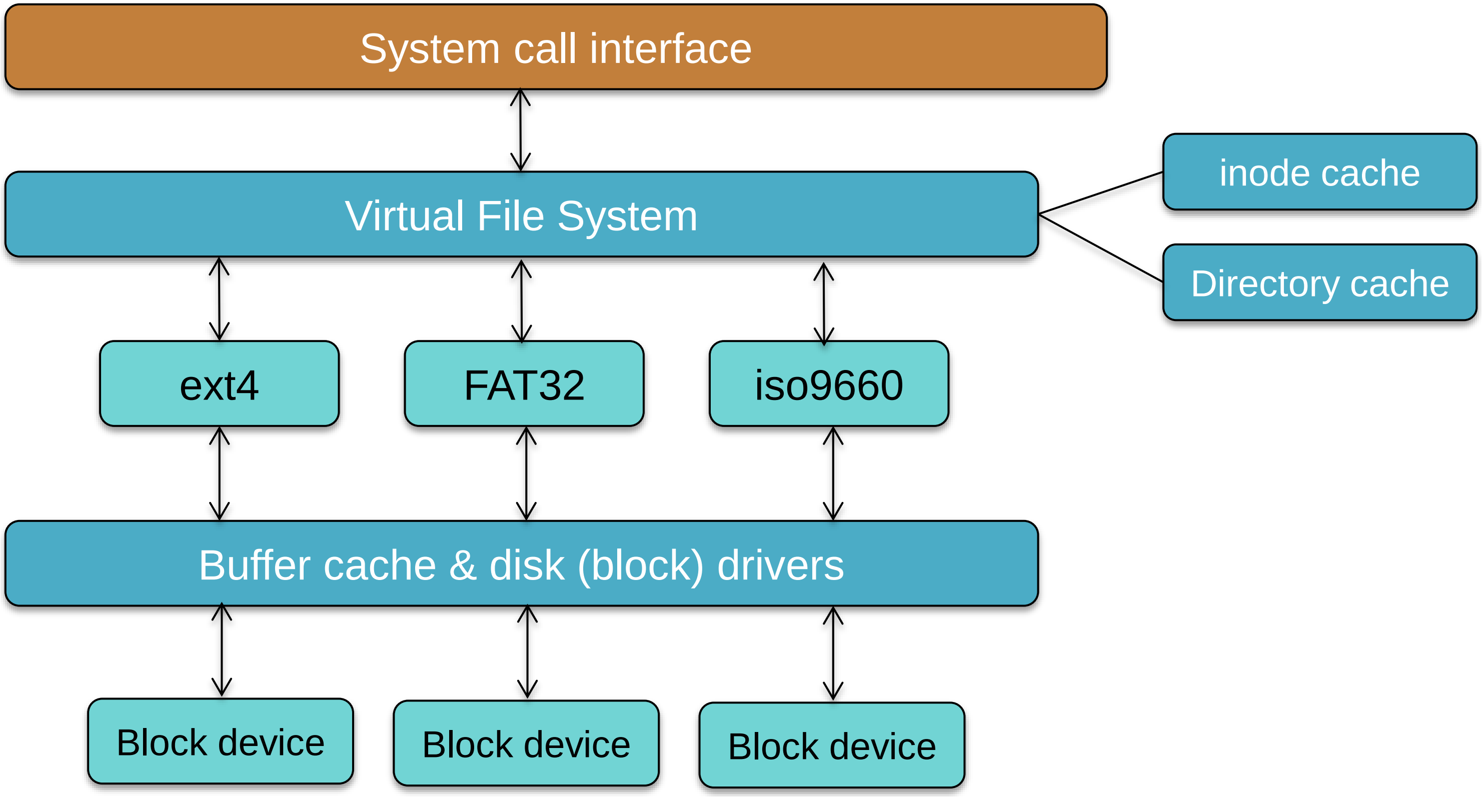
Considerations:

- Search path (what if two names are the same in the file systems)?
- Where to write?

Operating System Interfaces for File Systems

Virtual File System (VFS) Interface

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



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.

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*, …

VFS 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

VFS 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 *);
};
```

VFS 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);
};
```

VFS File operations

Not all functions need to be implemented!

Example: The same file_operations are used for a character device driver

```
struct file_operations mydriver_fops = {
    .owner = MYFS_MODULE;
    .open = myfs_open;         /* allocate resources */
    .read = myfs_read_file;
    .write = myfs_write_file;
    .release = myfs_release;  /* release resources */
    /* llseek, readdir, poll, mmap, readv, etc. not implemented */
};
```

```
register_filesystem(&myfs_type)
```

```
static struct file_system_type myfs_type = {
  .owner   = THIS_MODULE;
  .name    = “myfs”;
  .get_sb = myfs_get_super,
  .kill_sb = myfs_kill;
};
```

The End