More on Linux Filesystems

CSE 231

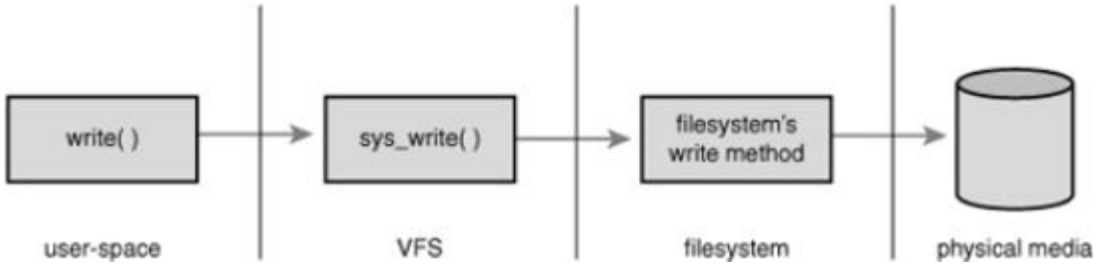
Instructor: Arani Bhattacharya

What is the goal of Linux file management?

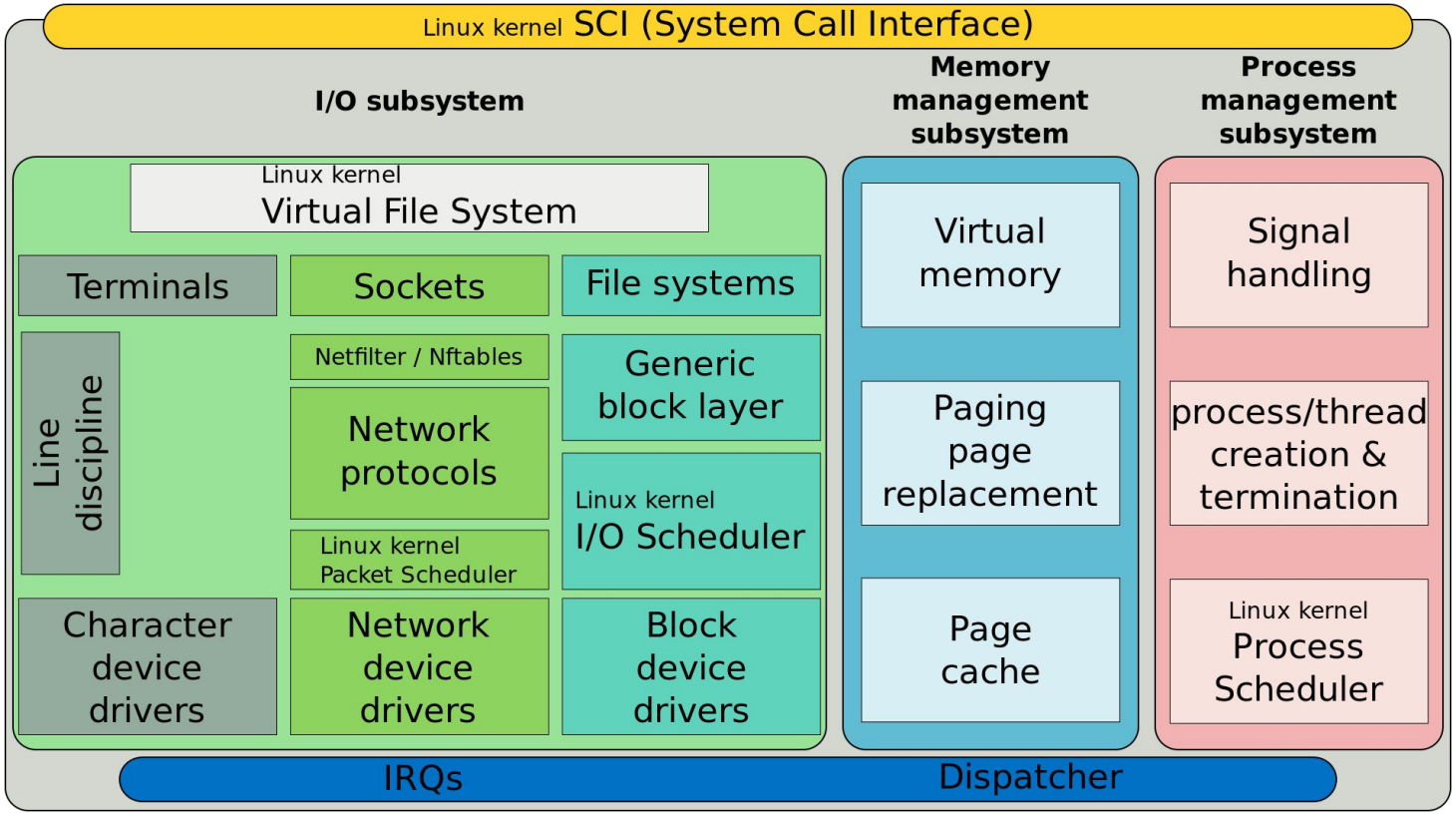
- Support a wide variety of devices
  - Hard disks -- even if Windows is installed on it
  - CD-ROM
  - Thumb drive
  - SD Cards
- Easy movement of data across these devices
- Yet the data itself might be organized physically in different ways
- Each of these filesystems must support reading and writing using exactly the same system calls

Linux uses an abstraction called Virtual File System (VFS)

- VFS provides a common file model
- Standard system calls all write only to the VFS, and not to the actual filesystem



Where does VFS fall in the picture?



Organization of Data in VFS

- VFS provides a common file model

- File Path (represented by dentry object)
  - Consider the file path /home/arani/cse231.ppt
  - There will be **four** dentry objects for this path: /, home, arani, cse231.ppt
- Open file (represented by file object)
- Inode or index node, to store file metadata
- Superblock, to store filesystem metadata

Current state of Linux filesystems

- Most Linux installations use ext4 or btrfs
  - Can support very large files
  - Does not easily lead to file fragmentation (why?)

- FAT/UbiFS for SD-cards/SSDs (why?)
- Very resistant to data loss on power/disk failure (why?)

Filesystem structure (ext2)

The entire disk consists of a set of blocks of length 1KB, 2KB or 4KB. Note that this is different from a disk sector of size 512 bytes. A block group can contain 8192-32768 blocks.

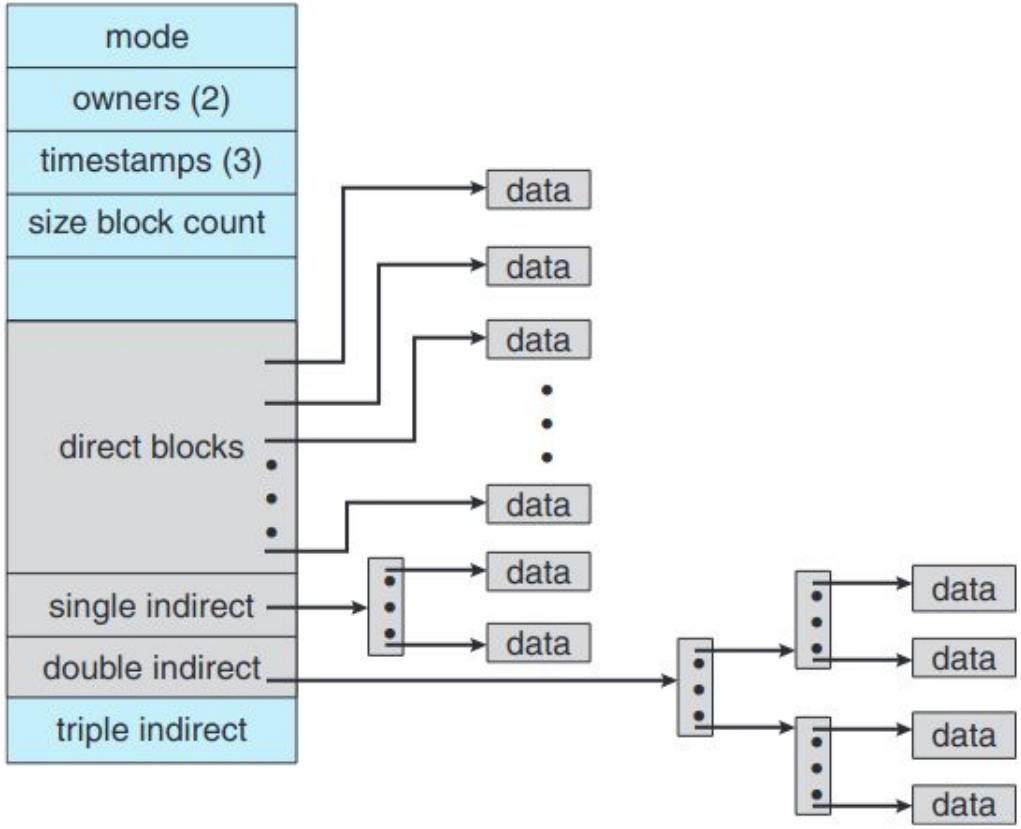
Boot Sector	Block Group 1	Block Group 2	... ...	Block Group N

Each block group contains the following information:

Super Block	FS descriptors	Block Bitmap	Inode Bitmap	Inode Table	Data Blocks

Why is so much information repeated in each block group?

Organization of an inode

- 

Some Linux commands related to inodes

- ls -il
  Shows the inode number along with other details of the files

- Showing a file with a specific inode number

```
find /var/ -inum 3634906 -exec ls -l {}
```

Advantages of Inode Structure

1. Can handle very large files
2. Very versatile

Disadvantages of Inode Structure

1. No protection from power failure -- entire disk’s inode structure can be in inconsistent state
2. Possibility of data fragmentation

Problem 1: Data Fragmentation

1. Hard disks are mechanical devices
2. The disk head needs to actually move to the point where the data is accessed, so that it can be actually read or written
3. Such movement takes time
4. If disk accesses are close to each other, then it becomes much faster
5. But inodes do not provide any such guarantee -- a single file can have data blocks located far from each other

One Possible Solution: Disk Defragmentation

1. Disk fragmentation leads to slower accesses to files

2. **Solution**: Periodically use a tool that reorganizes all the files on the disk. This process is called disk defragmentation

3. Time-consuming, but acceptable in many cases

One Possible Solution: Disk Defragmentation

1. Disk fragmentation leads to slower accesses to files

2. **Solution**: Periodically use a tool that reorganizes all the files on the disk. This process is called disk defragmentation

3. Time-consuming, but acceptable in many cases

Problem 2: Loss of Data on Power Failure

1. Notice that there is a bitmap in the group blocks

2. If power failure happens while writing to a file, the inode bitmap and inode table may be in inconsistent state

3. The entire integrity of the file system depends on the bitmap and inode table, so it is possible to lose data present on the entire disk
4. Need to fix using fsck, which can take a very long time

Solution to Data Loss: Journaled File Systems

Keep a log of what you are about to do. Effectively there are three steps in writing:

1. Mention in the log (“journal”) about where you are going to write. **Do not touch the inode table at this point.**

2. Do the actual writing on the data blocks

3. Update the log specifying that the writing has been completed

Integrating this together: ext4

1. The ext4 filesystem implements all these features, and so is considered to be a much better filesystem than ext2 and ext3
2. Supports file size of up to 16 TB
3. Supports disks of upto 1000 TB
4. Can deal with a total of 30 billion files
5. Supports file timestamp of nanosecond range
6. Not suited for SD cards/SSDs because of journal

File Superblock

```
struct super_block {
        struct list_head        s_list;         /* list of all superblocks */
        dev_t                   s_dev;          /* identifier */
        unsigned long           s_blocksize;    /* block size in bytes */
        unsigned char           s_blocksize_bits; /* block size in bits */
        unsigned char           s_dirt;         /* dirty flag */
        unsigned long long      s_maxbytes;     /* max file size */
        struct file_system_type s_type;         /* filesystem type */
        struct super_operations s_op;           /* superblock methods */
        struct dquot_operations *dq_op;         /* quota methods */
        struct quotactl_ops     *s_qcop;        /* quota control methods */
        struct export_operations *s_export_op;  /* export methods */
        unsigned long           s_flags;        /* mount flags */
        unsigned long           s_magic;        /* filesystem's magic number */
        struct dentry           *s_root;        /* directory mount point */
        struct rw_semaphore     s_umount;       /* unmount semaphore */
        struct semaphore        s_lock;         /* superblock semaphore */
        int                     s_count;        /* superblock ref count */
        int                     s_need_sync;    /* not-yet-synced flag */
        atomic_t                s_active;       /* active reference count */
        void                    *s_security;    /* security module */
        struct xattr_handler    **s_xattr;      /* extended attribute handlers */
```

File Superblock

```
        struct list_head      s_inodes;        /* list of inodes */
        struct list_head      s_dirty;         /* list of dirty inodes */
        struct list_head      s_io;            /* list of writebacks */
        struct list_head      s_more_io;       /* list of more writeback */
        struct hlist_head     s_anon;          /* anonymous dentries */
        struct list_head      s_files;         /* list of assigned files */
        struct list_head      s_dentry_lru;    /* list of unused dentries */
        int                   s_nr_dentry_unused; /* number of dentries on list */
        struct block_device   *s_bdev;         /* associated block device */
        struct mtd_info       *s_mtd;          /* memory disk information */
        struct list_head      s_instances;     /* instances of this fs */
        struct quota_info     s_dquot;         /* quota-specific options */
        int                   s_frozen;        /* frozen status */
        wait_queue_head_t     s_wait_unfrozen; /* wait queue on freeze */
        char                  s_id[32];        /* text name */
        void                  *s_fs_info;      /* filesystem-specific info */
        fmode_t               s_mode;          /* mount permissions */
        struct semaphore      s_vfs_rename_sem; /* rename semaphore */
        u32                   s_time_gran;     /* granularity of timestamps */
        char                  *s_subtype;      /* subtype name */
        char                  *s_options;      /* saved mount options */
};
```

Superblock Operations

```
struct super_operations {
  struct inode *(*alloc_inode)(struct super_block *sb);
  void (*destroy_inode)(struct inode *);
  void (*dirty_inode) (struct inode *);
  int (*write_inode) (struct inode *, int);
  void (*drop_inode) (struct inode *);
  void (*delete_inode) (struct inode *);
  void (*put_super) (struct super_block *);
  void (*write_super) (struct super_block *);
  int (*sync_fs)(struct super_block *sb, int wait);
  int (*freeze_fs) (struct super_block *);
  int (*unfreeze_fs) (struct super_block *);
  int (*statfs) (struct dentry *, struct kstatfs *);
  int (*remount_fs) (struct super_block *, int *, char *);
  void (*clear_inode) (struct inode *);
  void (*umount_begin) (struct super_block *);
  int (*show_options)(struct seq_file *, struct vfsmount *);
  int (*show_stats)(struct seq_file *, struct vfsmount *);
  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, loff_t);
  int (*bdev_try_to_free_page)(struct super_block*, struct page*, gfp_t);
};
```

Inode Structure

```
struct inode {
        struct hlist_node       i_hash;         /* hash list */
        struct list_head        i_list;         /* list of inodes */
        struct list_head        i_sb_list;      /* list of superblocks */
        struct list_head        i_dentry;       /* list of dentries */
        unsigned long           i_ino;          /* inode number */
        atomic_t                i_count;        /* reference counter */
        unsigned int            i_nlink;        /* number of hard links */
        uid_t                   i_uid;          /* user id of owner */
        gid_t                   i_gid;          /* group id of owner */
        kdev_t                  i_rdev;         /* real device node */
        u64                     i_version;      /* versioning number */
        loff_t                  i_size;         /* file size in bytes */
        seqcount_t              i_size_seqcount; /* serializer for i_size */
        struct timespec         i_atime;        /* last access time */
        struct timespec         i_mtime;        /* last modify time */
        struct timespec         i_ctime;        /* last change time */
        unsigned int            i_blkbits;      /* block size in bits */
        blkcnt_t                i_blocks;       /* file size in blocks */
        unsigned short          i_bytes;        /* bytes consumed */
        umode_t                 i_mode;         /* access permissions */
```

Inode Structure

```
        spinlock_t              i_lock;             /* spinlock */
        struct rw_semaphore     i_alloc_sem;        /* nests inside of i_sem */
        struct semaphore        i_sem;              /* inode semaphore */
        struct inode_operations *i_op;              /* inode ops table */
        struct file_operations  *i_fop;             /* default inode ops */
        struct super_block      *i_sb;              /* associated superblock */
        struct file_lock        *i_flock;           /* file lock list */
        struct address_space    *i_mapping;         /* associated mapping */
        struct address_space    i_data;             /* mapping for device */
        struct dquot            *i_dquot[MAXQUOTAS]; /* disk quotas for inode */
        struct list_head        i_devices;          /* list of block devices */
        union {
            struct pipe_inode_info  *i_pipe;        /* pipe information */
            struct block_device     *i_bdev;        /* block device driver */
            struct cdev             *i_cdev;        /* character device driver */
        };
        unsigned long           i_dnotify_mask;     /* directory notify mask */
        struct dnotify_struct   *i_dnotify;         /* dnotify */
        struct list_head        inotify_watches;    /* inotify watches */
        struct mutex            inotify_mutex;      /* protects inotify_watches */
        unsigned long           i_state;            /* state flags */
        unsigned long           dirtied_when;       /* first dirtying time */
        unsigned int            i_flags;            /* filesystem flags */
        atomic_t                i_writecount;       /* count of writers */
        void                    *i_security;        /* security module */
        void                    *i_private;         /* fs private pointer */
};
```

Inode Operations

```
struct inode_operations {
   int (*create) (struct inode *,struct dentry *,int, struct nameidata *);
   struct dentry * (*lookup) (struct inode *,struct dentry *, struct nameidata *);
   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 __user *,int);
   void * (*follow_link) (struct dentry *, struct nameidata *);
   void (*put_link) (struct dentry *, struct nameidata *, void *);
   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 *);
   void (*truncate_range)(struct inode *, loff_t, loff_t);
   long (*fallocate)(struct inode *inode, int mode, loff_t offset,
                     loff_t len);
   int (*fiemap)(struct inode *, struct fiemap_extent_info *, u64 start,
                 u64 len);
```

File structure

```
struct file {
        union {
            struct list_head    fu_list;        /* list of file objects */
            struct rcu_head     fu_rcuhead;     /* RCU list after freeing */
        } f_u;
        struct path             f_path;         /* contains the dentry */
        struct file_operations *f_op;           /* file operations table */
        spinlock_t              f_lock;         /* per-file struct lock */
        atomic_t                f_count;        /* file object's usage count */
        unsigned int            f_flags;        /* flags specified on open */
        mode_t                  f_mode;         /* file access mode */
        loff_t                  f_pos;          /* file offset (file pointer) */
        struct fown_struct      f_owner;        /* owner data for signals */
        const struct cred       *f_cred;        /* file credentials */
        struct file_ra_state    f_ra;           /* read-ahead state */
        u64                     f_version;      /* version number */
        void                    *f_security;    /* security module */
        void                    *private_data; /* tty driver hook */
        struct list_head        f_ep_links;     /* list of epoll links */
        spinlock_t              f_ep_lock;      /* epoll lock */
        struct address_space    *f_mapping;     /* page cache mapping */
        unsigned long           f_mnt_write_state; /* debugging state */
};
```

File Operations

```
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 (*aio_read) (struct kiocb *, const struct iovec *,
                             unsigned long, loff_t);
        ssize_t (*aio_write) (struct kiocb *, const struct iovec *,
                              unsigned long, 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);
        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 *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        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 (*sendpage) (struct file *, struct page *,
                             int, size_t, loff_t *, int);
```