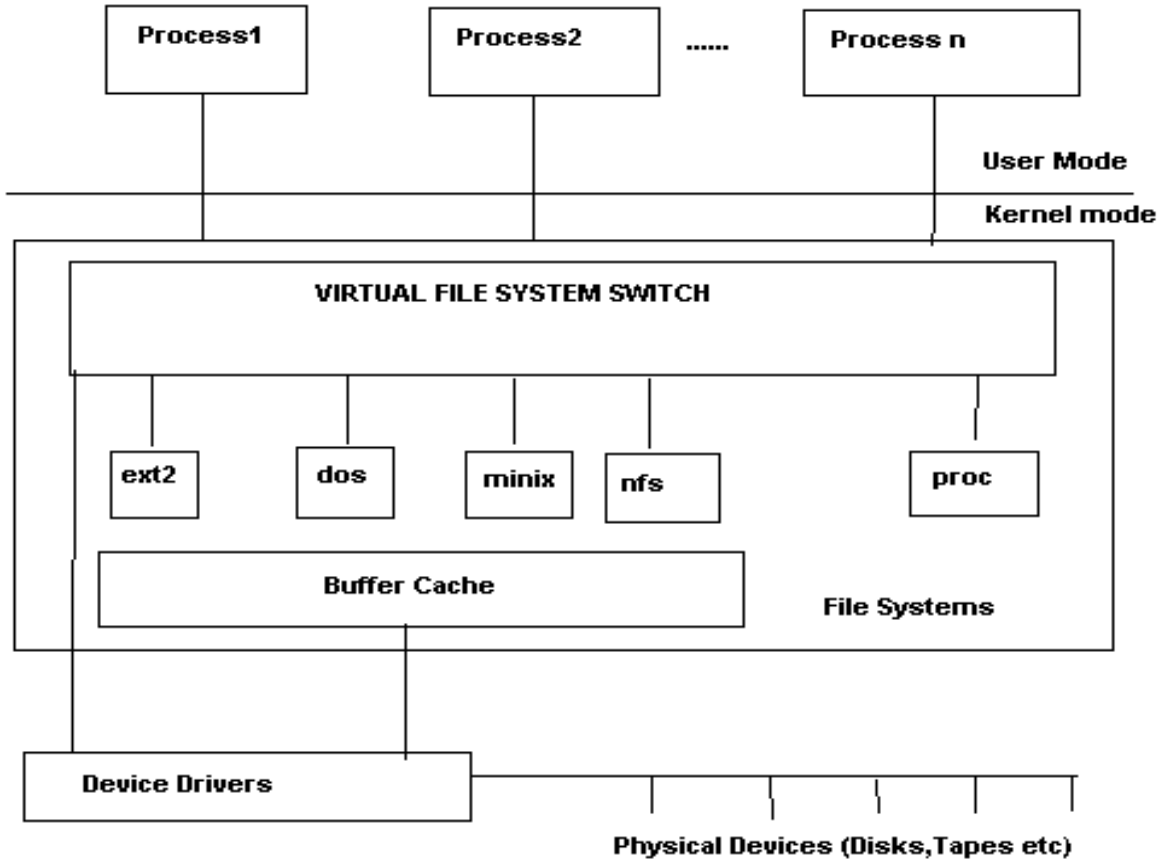
LINUX FILE SYSTEM

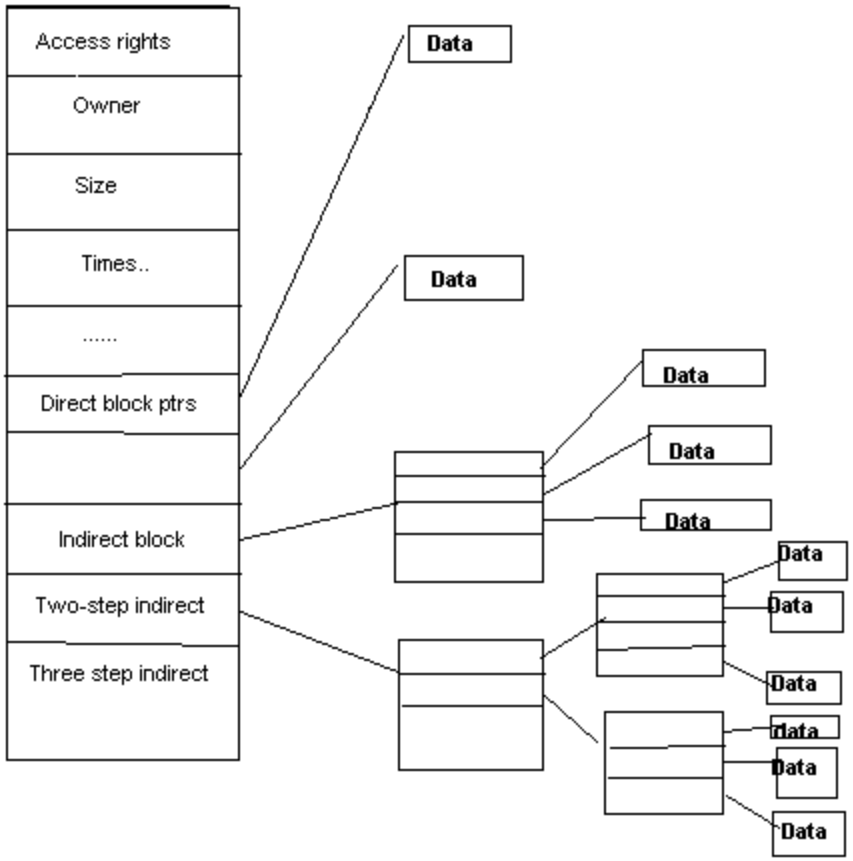
Virtual File System Switch

- Supplies applications with the system calls for file system management.
- Specific focus on ext2fs and procfs of LINUX.
- Different types of files are possible:regular files,directories,device files,fifo's,pipes,symbolic links,sockets.
- Meta data information kept separately."inodes" are used for describing files.

Virtual File System Switch



Structure of Inode



Unix File System

FEATURES OF UNIX FILE SYSTEMS

- Boot Block : Will it always be present?
- SuperBlock:Information for managing file systems
- InodeBlocks:inode structures for the file system.
- DataBlocks:blocks for files,directories..etc
- Mounting,Unmounting file systems

Representation of File systems

- Management structures for file systems is similar to logical structure of UNIX file system.
- VFS responsible for calling file-system specific implementations. How does VFS know which filesystems are configured?
- Void register_filesystem(struct file_system_type *);
- Configured at boot-time and possibly loaded as modules as well.
- struct file_system_type {
  - Const char *name;
  - Int fs_flags;
  - Struct super_block *(*read_super)(struct super_block*,void *,int);
  - #ifdef VERSION > 2.4
  - Struct module *owner;
  - Struct vfsmount *kern_mnt;
  - #endif
  - Struct file_system_type *next;
  - };

Mounting

- Any file system in UNIX must be mounted before it can be accessed,either through mount() or mount_root().
- What is mount_root() ?
- Representation of a mounted file system : struct super_block.Limited to NR_SUPER.
- It is initialized by the function “read_super” in the VFS.It interrogates floppy disks,CD-ROM for a change of media,tests if superblock is present and if so returns it.else it calls the fs-specific function to create a superblock.
- File system specific Read_super() reads its data from the block device using the LINUX cache functions,bread,brw_page.Processes may be put to sleep when reading/writing or mounting file systems.

Superblock

- struct super_block {
  - kdev_t s_dev;/* device */
  - unsigned long s_blocksize;/* block size */
  - unsigned char s_blocksize_bits;/* ld(block size) */
  - unsigned char s_lock;/* superblock lock */
  - unsigned char s_rd_only;
  - Unsigned char s_dirt;
  - Struct file_system_type *s_type;
  - Struct super_operations *s_op;
  - Unsigned long s_flags;
  - Unsigned long s_magic;
  - Unsigned long s_time;
  - Struct inode *s_covered;/* mount point */
  - Struct inode *s_mounted; /* root inode */
  - Struct wait_queue *s_wait;/* s_lock wait queue */
  - Union {
    - Struct minix_sb_info minix_sb;
    - Struct ext2_sb_info ext2_sb;
    - ….
    - Void *generic_sb;
    - }u;

Locking/Unlocking superblock

- Lock_super(struct super_block *sb)
- {
  - If(sb->s_lock) __wait_on_super(sb);
  - Sb->s_lock=1;
  - **}**

  - **unlock_super(struct super_block *sb)**
  - **{**
  - sb->s_lock=0;
  - wake_up(&sb->s_wait);
  - **}**
  - The read_super( ) function in the file system specific implementation should also make the root inode available using the function iget( );

Superblock operations

- s_op points to a vector of functions for accessing the file system.
- struct super_operations {
- void (*read_inode)(struct inode *);
- Int (*notify_change)(struct inode *,struct iattr *);
- Void (*write_inode)(struct inode *);
- Void (*put_inode)(struct inode *);
- Void (*put_super)(struct super_block *);
- Void (*write_super)(struct super_block *);
- Void (*statfs)(struct super_block *,struct statfs *);
- Void (*remount_fs)(struct super_block *,int *,char *);
- };
- These functions serve to translate the specific representation of the superblock and inode on data media to their general form in memory and vice-versa.Does every file system have a superblock and inode?
- Elaborating a little more…

- **Write_super(sb)** : saves info on super block.Used in synchronising the device.dirty bit is set in buffer.
- **Put_super(sb) :** Called when unmounting file systems.Releases super_block and buffers used.restores consistency if necessary.super_block structure can be reused.
- **Statfs(sb,statfsbuf)** : fills the statfs structure with file system information.
- **Remount_fs(sb,flags,options)** : new attributes for file system and restoring consistency.
- **Read_inode(inode)** : fills in the inode structure.Marks different file types by using different inode_operations struct.for example
- If(S_ISREG(inode->I_mode)
- Inode->I_op=&ext2_file_inode_operations;
- Else if(S_ISDIR(inode->I_mode)
- Inode->I_op=&ext2_dir_inode_operations;
- Else if(S_ISLNK(inode->I-mode)
- Inode->I_op=&ext2_symlink_inode_operations;
- …….Similiarly for character devices,block devices,fifos,etc

- Notify_change(inode,attr) : changes made to inode are acknowledged by this function.It is of interest only in NFS/Coda etc.Why?Each file on such a system has a local and a remote inode.The remote file system is informed of the change using iattr.
- Write_inode(inode) : saves inode structure.
- Put_inode(inode) : if inode is no longer required.Called when deleting file and release its blocks.

INODE

- Struct inode {
- Kdev_t I_dev;
- Unsigned long I_ino;
- Umode_t I_mode;
- Nlink_t I_nlinkl;
- Uid,gid etc….
- Dev_t I_rdev; /* only if device special file */
- Offset,times of modification,access,creation etc
- Struct semaphore *I_sem;
- Struct inode_operations *I_op;
- Struct super_block *I_sb;
- Struct wait_queue (I_wait;
- Struct file_lock *I_flock;
- Struct vm_area_struct *I_mmap;
- Struct inode *I_next,*I_prev;
- Struct inode *I_hash_next,*I_hash_prev;
- Struct inode *I_boundto;

INODE…contd

- Struct inode *I_mount;
- Struct socket *I_socket;
- Reference_counter,flags,lock,bits that denote if this is a pipe,socket.
- Union {
- Struct pipe_node_info pipe_i;
- Struct minix_inode_info minix_i;
- ….
- Void *generic_ip;
- }u;
- };
- Inodes are managed as doubly linked lists as well as a hash table.starts with first_inode(/ inode).iget() function can be used to get the inode specified by the superblock.It uses hints to resolve cross mounted file systems as well.Any access to inode increments a usage counter.Analogous function iput( ).

Inode Operations

- Struct inode_operations {
- Struct file_operations *default_file_ops;
- Int (*create)(struct inode *,const char *,int,int,struct inode **);
- Int (*lookup)(struct inode *,const char *,int,struct inode **);
- Int (*link)(struct inode *,struct inode *,const char *,int);
- Int (*unlink)(struct inode *,const char *,int);
- Int (*symlink)(struct inode *,const char *,int);
- Int (*mkdir)(struct inode *,const char *,int);
- Similiarly we have rmdir,mknod,rename,readlink,follow_link,bmap,truncate,permission,smap.
- NOTE :All these functions are directly called from the implementation of the corresponding system call.

File System Calls.

- Create(),lookup(),link(),unlink(),symlink(),mkdir(),rmdir(),
- mknod(),rename(),readlink(),follow_link()
- Bmap() : used for mmap'ing files .This function is called by the function generic_mmap() to map a block of data from file to an address in the user space.
- Smap():allows swap files to be created on a UMSDOS file system.
- To allow for shared accesses to files,UNIX introduced an extra structure.
- Struct file {
- Mode_t f_mode;
- Loff_t f_pos;
- Unsigned short f_flags;
- Unsigned short f_count;
- Off_t f_reada;/* readahead flag */
- Struct file *f_next,*f_prev;
- Struct inode *f_inode;
- Struct file_operations *f_op;
- Void *private_data;
- };

File Operations .

- General interface to work on files,Has functions to open,close,read,write etc.Why are these isolated from inode_operations?These operations also need to modify struct file.Hence.
- Struct file_operations {
- Struct module *owner;
- Int (*lseek)(…)
- Read,write,readdir,select,ioctl,mmap,open,release,fsync,fasync,check_media_change,revalidate..
- These are also used by sockets/drivers etc..The inode operations only use the representation of the socket/device .
- If any of these functions are not implemented ,it tries to call default file operations function,failing which an error is returned.Certain functions only make sense for drivers,like release,open

Opening Files

- Open : most complicated system call. Authorization to open the file is checked,an empty file structure is allocated and entered in file decsriptor table of process and open_namei() is used to retreive the inode for the file.Performs a lot of tests.
- Name ends in a /.What should be done?
- If O_CREAT is set,What function should this call?
- What about symbolic links,directories?
- Permission is checked with the inode permission list also.
- What about character devices and block devices ?
- Directory cache originated in ext2fs.Directory names are cached.It is a 2-level cache with LRU at both levels.Hash key is formed by the dev number and the inode number and the name.Exports dcache_add,dcache_lookup functions

PROC FILE SYSTEM

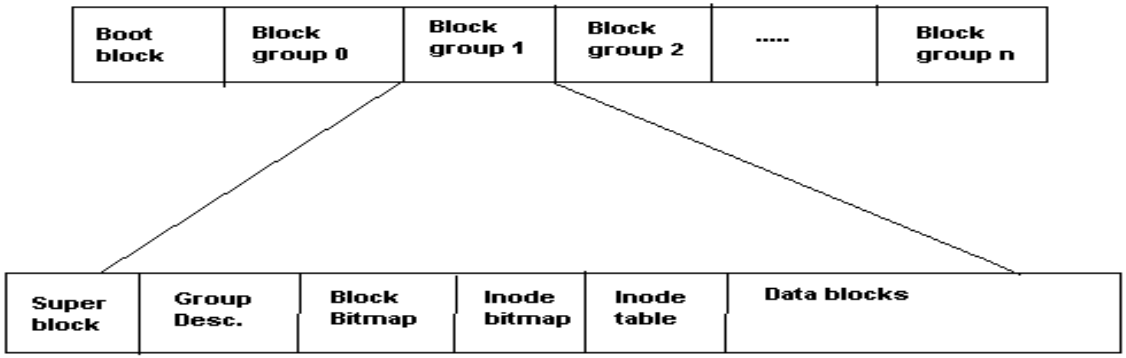
- Every process has a directory in /proc/pid which contains files relevant to the process.
- VFS function read_super( ) is called by do_mount( ) and in turn calls proc_read_super( ) for the proc file system.
- Struct super_block *proc_read_super(struct super_block *s,void *data, int silent)
- {
- Lock_super(s);
- Sets some fields in s.
- S->s_op=&proc_sops;
- Unlock_super(s);
- If(!(s->s_mounted=iget(s,PROC_ROOT_INO))) {
- S->s_dev=0;/* free the super block */
- Return NULL;
- }
- Parse_options(data,&s->s_mounted->I_uid,s->s_mounted->I_gid);
- Return s;
- }

PROC FILE SYSTEM…contd

- Some of the invariable entries in the root dircetory entry are
- . , .. , loadavg,uptime,meminfo,kmsg,version,pci,cpuinfo,self,ioports,profile.

EXT2 File System

- Originally used MINIX file system.Serious limitations like partition size,file name size etc.Proposed ext file system which was even worse.Led to excessive fragmentation.Remy Card et'al propose ext2fs which is the LINUX file system.
- Influenced by BSD FFS.Every partition is divided into a number of block groups,corresponding to the cylinder groups of FFS,with each block group holding a copy of superblock,inode and data blocks.



EXT2FS STRUCTURE

STRUCTURE OF E2FS

- What is the basic idea of block groups ?
- The physical superblock contains information on the number of inodes,blocks per block groups,time of last mount,last write to sb,mount counter,maximum mount operations.Padded to a size of 1k bytes.
- Block group descriptors:Information on block groups.Each block group is described by a 32 byte desc.Contains block numbers in the inode bitmap,block bitmap and inode table,number of free inodes,number of free blocks,number of directories in this block group.
- Number of directories is made use by allocator to spread files evenly.
- Block size=1024bytes => 8192 blocks per block group.
- Inode size=128 Bytes.

Directories in e2fs

- Directories are singly linked lists.
- Struct ext2_dir_entry {
- Unsigned long inode;
- Unsigned short rec_len;
- Unsigned short name_len;
- Char name[EXT2_NAME_LEN];
- };
- What happens on a delete?

Block allocation in e2fs

- Attempts to prevent fragmentation as much as possible.
- Target-oriented allocation.
- Always look for space in the area of target blocks.If this block is free use that,otherwise look for a block within 32 blocks from this.If not found then try to allocate from the same block group.else go to other block groups.
- Pre-allocation:
- If a free block is found then upto 8 of the following blocks are reserved if they are free.When a file is closed,the remaining blocks that are reserved are released.Also called clustering of data blocks.
- How is target block determined?Let n=logical block #,l=logical block# of the last block allocated.Then l+1 is a possible target block