Operating System Principles

操作系统原理

File System

李旭东

leexudong@nankai.edu.cn

Nankai University

Long-Term Information Storage

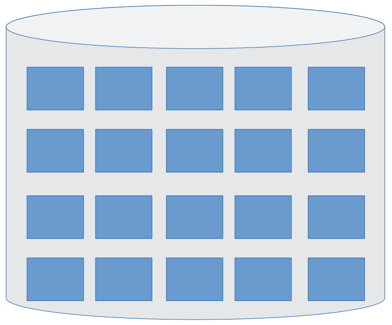
- Essential requirements
  - It must be possible to store a very large amount of information
  - The information must survive the termination of the process using it
  - Multiple processes must be able to access the information concurrently
- Solution
  - ???

Long-Term Information Storage

- Solution
  - Store information on disk and other external media in units called files.
  - Processes can then read them and write new one if need be
  - Persistent
  - File, Directory, File System

Long-Term Information Storage

- Storage with Block Device
  - Magnetic Disks
  - Magnetic Tape
  - Optical disc
  - Flash memory
- Block-Level Operations
  - Read i-th Block
  - Write i-th Block



Long-Term Information Storage

- Retrieve Information Quickly
  - 1. How do you find information?
  - 2. How do you keep one user from reading another user's data?
  - 3. How do you know which blocks are free?

Objectives

- File Concept
- Directory Concept
- File Share & Protection
- File System Implementation
- File System Reliability
- File System Performance
- File System Cases

File Concept

- File
  - A logical units of information
  - A byte stream
- File Size
- File Name
- File Logical Structure (File Content)
- File Type
- File Access
- File Attributes
- File Operations
- File Physical Structure

File Size

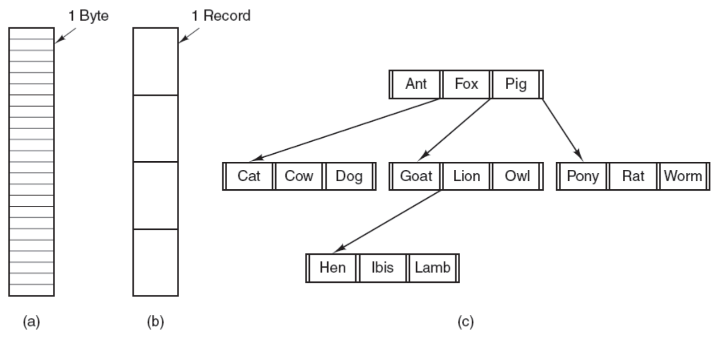
- Zero Byte
- ~~1 Bit~~
- 1 Byte
- 2 Bytes
- 1 KB (千)
- 1 MB (兆)
- ...
- GB (吉)
- TB (太)
- PB (拍)
- EB (艾)
- ZB (泽)
  - 1.2ZB
- YB (尧)
- ...

File Naming

- File extension
- File Name Charset

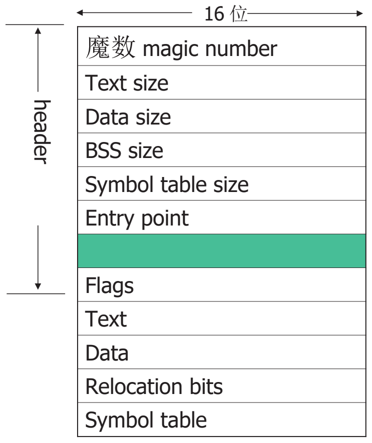
Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

File Logical Structure

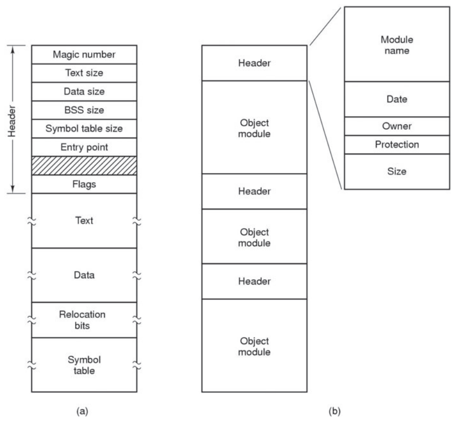


(a) Byte sequence. (b) Record sequence. (c) Tree.

File Logical Structure: Executable Program File



File Types



(a) An executable file. (b) An archive.

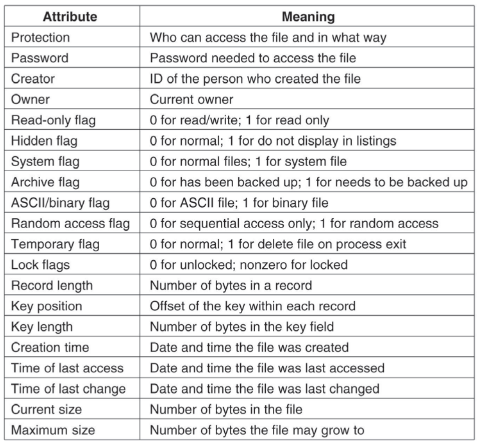
File Types

- T1
  - Regular files
  - Directory files
- T2
  - Character special files
  - Block special files
- T3
  - ASCII files
  - Binary files
- ...

File Access

- Sequential access
- Random access
- Key-value access

File Attributes



Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File Operations

The most common system calls relating to files:

- Create
- Delete
- Open
- Close
- Read
- Write
- Append
- Seek
- Get Attributes
- Set Attributes
- Rename

CP Program:Using File System Calls (1/2)

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                    /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);         /* ANSI prototype */

#define BUF_SIZE 4096                     /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                  /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);               /* syntax error if argc is not 3 */
```

CP Program:Using File System Calls (2/2)

```
    /* Open the input file and create the output file */
    in_fd = open(argv[1], O_RDONLY);     /* open the source file */
    if (in_fd < 0) exit(2);              /* if it cannot be opened, exit */
    out_fd = creat(argv[2], OUTPUT_MODE); /* create the destination file */
    if (out_fd < 0) exit(3);             /* if it cannot be created, exit */

    /* Copy loop */
    while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
    if (rd_count <= 0) break;            /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count); /* write data */
        if (wt_count <= 0) exit(4);      /* wt_count <= 0 is an error */
    }

    /* Close the files */
    close(in_fd);
    close(out_fd);
    if (rd_count == 0)                   /* no error on last read */
        exit(0);
    else
        exit(5);                         /* error on last read */
}
```

File Physical Structure

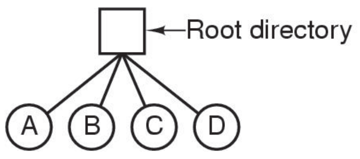
- How to store in block storage
  - ???

Directory Concept

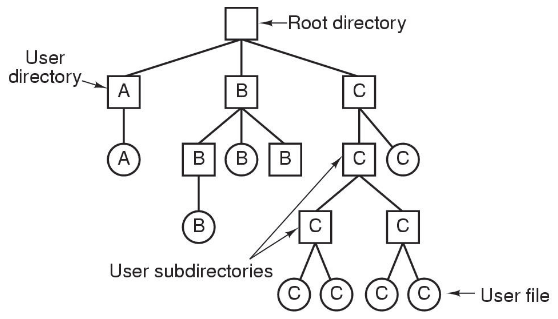
- Directory
  - Folder, Which contains files
- Single-level Directory System
- Hierarchical Directory System
- Path Names
- Directory Operations
- Implement of Directory

Single-level Directory System

Root directory

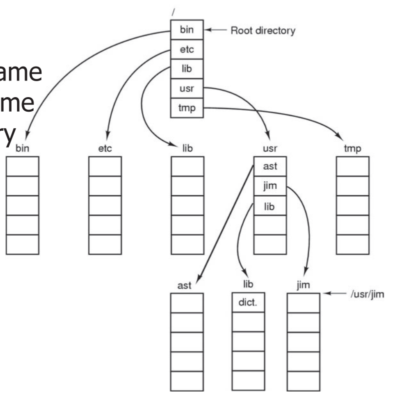


Hierarchical Directory System



Path Names

- Absolute path name
- Relative path name
- Working directory



A UNIX directory tree

Directory Operations

System calls for managing directories:

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
- Unlink

- Link
  - Hard link
  - Symbolic link

Implement of Directory

- How to store in block storage
  - ???

File System Concept

- File System
  - It contains files and directories
- File System Types
- Implement of File System

File System Types

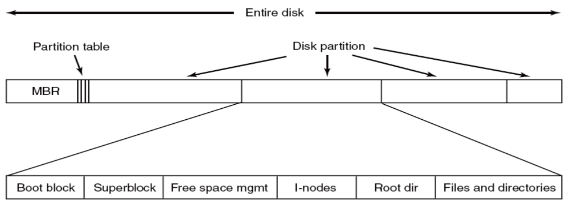
- Traditional: ext2
- Newest: ext3, ReiserFS, IBM JFS, xfs
- Other UNIX: minix, ext, xiafs
- FAT-12, FAT-16, FAT-32, VFAT, NTFS (read-only)
- HPFS (OS/2) read-only, HFS (Macintosh) read-only
- AFFS (Amiga), System V, Coherent, Xenix
- CD-ROM: **ISO 9660**
- UMSDOS (UNIX-like FS on MS-DOS)
- NFS (Network File System)
- SMBFS (Windows share), NCPFS (Novell Netware share)
- /proc (for kernel and process information)
- SHMFS (Shared Memory Filesystem)

Implement of File System

- File System Layout
- Implement of Files
- Implement of Directories
- Free Block Space Management

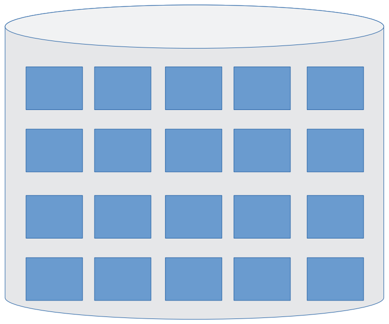
File System Layout

- Disk
- MBR (Master Boot Record)
  - Boot block
- Partitions
  - Superblock



Implement of Files

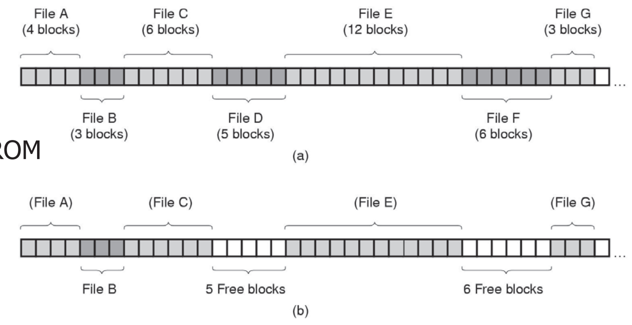
Where are File's Blocks?



Implement of Files

- Contiguous Allocation

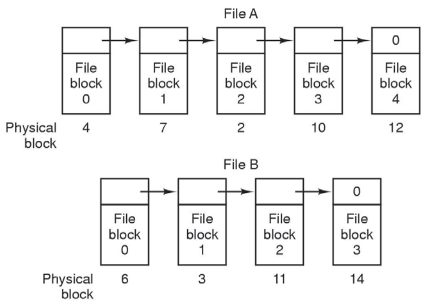
CD-ROM



(a) Contiguous allocation of disk space for 7 files.
(b) The state of the disk after files D and F have been removed.

Implement of Files

- Linked List Allocation
  - The first word of each block is used as a pointer to the next one
  - The rest of the block is for data

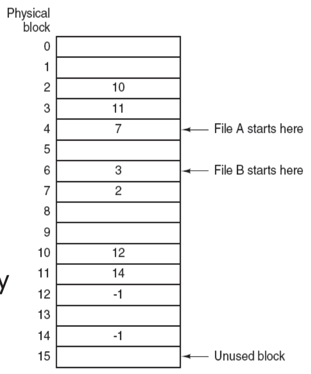


- Internal fragmentation
- Random access is extremely slow

Storing a file as a linked list of disk blocks.

Implement of Files

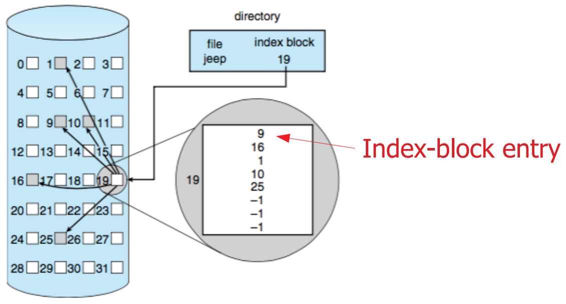
- Linked List Allocation Using a Table in Memory
  - FAT: File Allocation Table
  - FAT12
  - FAT16
  - FAT32



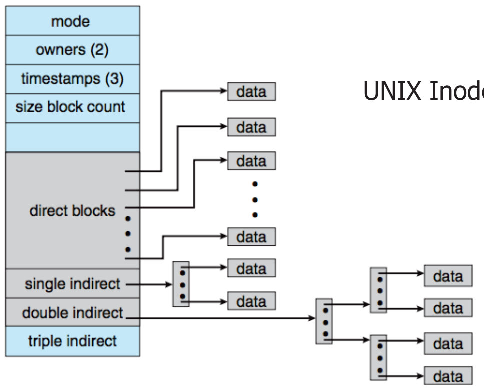
- Random access is much easier
- ?Disadvantage
  - Too many table entries in memory

Implement of Files

- I-nodes
  - Index-node, which lists the attributes and disk addresses of the files blocks

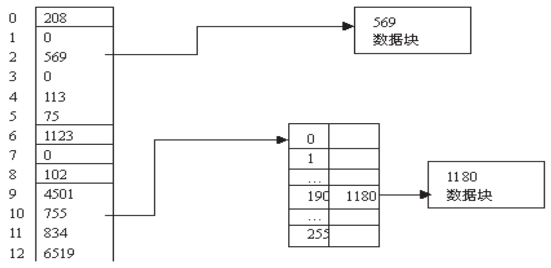


I-Nodes



UNIX Inode

Quiz

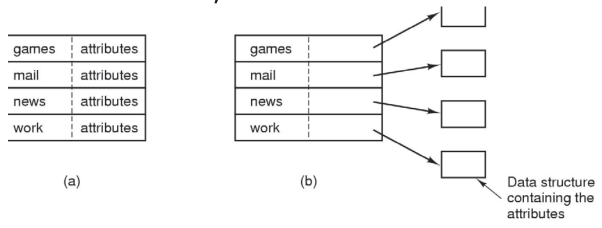


How to read the 204803th data of file A, which physical block index has 4 bytes?
Question 1. when the size of physical block is 1024?
Question 2. when the size of physical block is 2048?

Implement of Directories

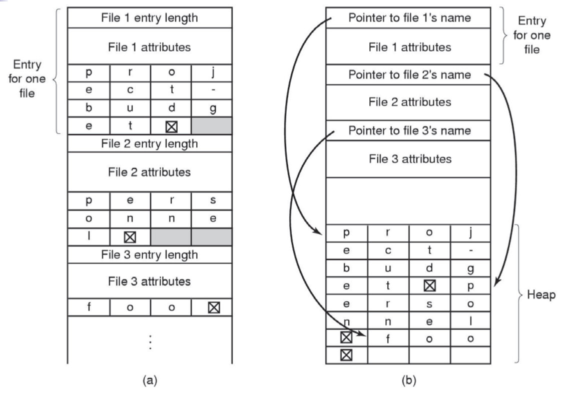
- Directory entry
  - File Control Block, FCB
  - Fixed-size, Variable-size

- File
  - FCB
  - Body



(a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry.
(b) A directory in which each entry just refers to an i-node.

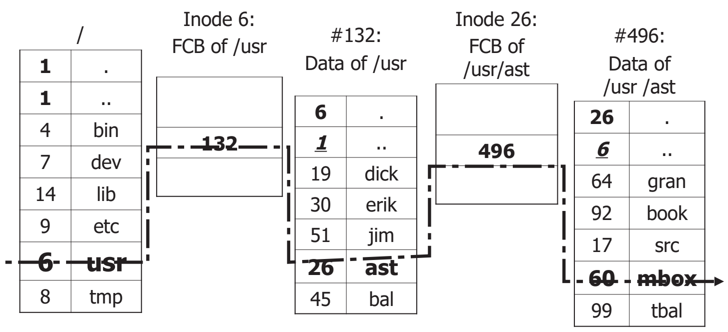
Implement of Directories



Two ways of handling long file names in a directory
(a) In-line. (b) In a heap.

Implement of Directories

- Search file by pathname
  - /usr/ast/mbox



/

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

Inode 6:
FCB of /usr

132

#132:
Data of /usr

6	.
1	..
19	dick
30	erik
51	jim
26	ast
45	bal

Inode 26:
FCB of /usr/ast

496

#496:
Data of /usr /ast

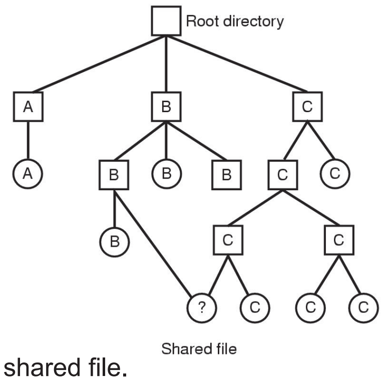
26	.
6	..
64	gran
92	book
17	src
60	mbox
99	tbal

Implement of Directories

- Directory entry
  - Linear List
  - Hash Table

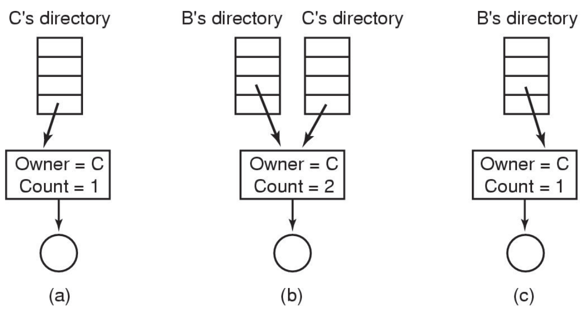
Shared Files

- Symbolic linking
- Hard linking



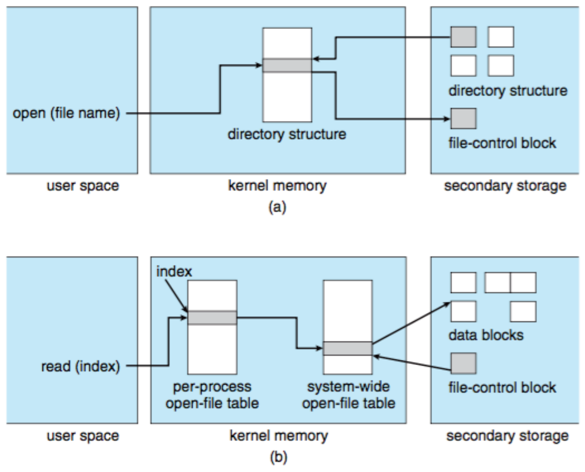
File system containing a shared file.

Shared Files



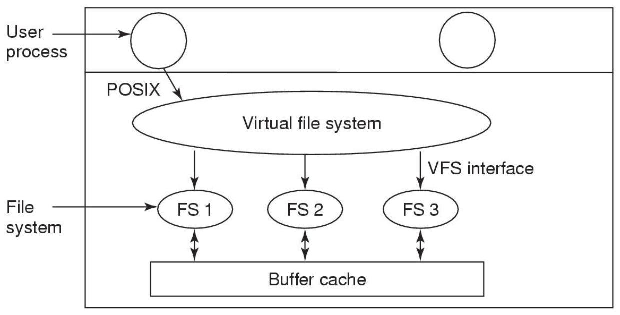
(a) Situation prior to linking. (b) After the link is created.
(c) After the original owner removes the file.

In-memory file-system structures

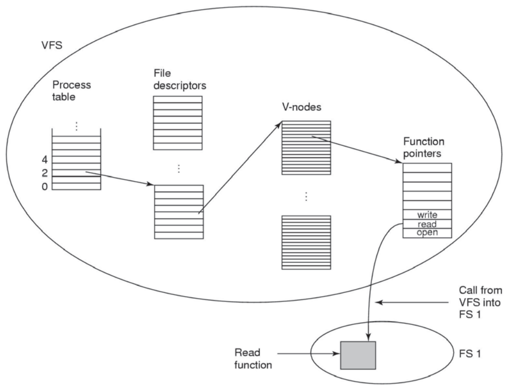


Virtual File Systems

- VFS Interface



Virtual File Systems



A simplified view of the data structures and code used by the VFS and concrete file system to do a read.

Disk Space Management

- Block Size
- Keeping Track of Free Blocks
- Disk Quotas
- File System Backups
- File System Consistency
- File System Performance
- Defragmenting Disks

Block Size

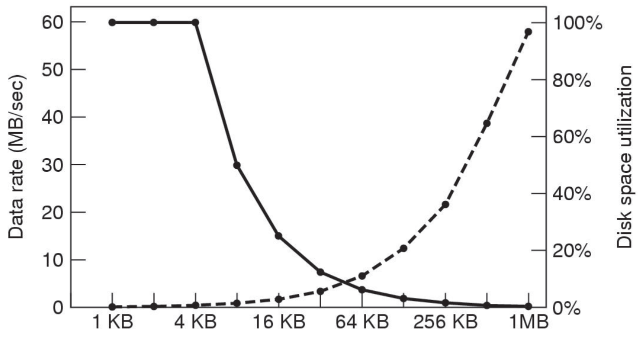
- Small ? Large?

Length	VU 1984	VU 2005	Web
1	1.79	1.38	6.67
2	1.88	1.53	7.67
4	2.01	1.65	8.33
8	2.31	1.80	11.30
16	3.32	2.15	11.46
32	5.13	3.15	12.33
64	8.71	4.98	26.10
128	14.73	8.03	28.49
256	23.09	13.29	32.10
512	34.44	20.62	39.94
1 KB	48.05	30.91	47.82
2 KB	60.87	46.09	59.44
4 KB	75.31	59.13	70.64
8 KB	84.97	69.96	79.69

Length	VU 1984	VU 2005	Web
16 KB	92.53	78.92	86.79
32 KB	97.21	85.87	91.65
64 KB	99.18	90.84	94.80
128 KB	99.84	93.73	96.93
256 KB	99.96	96.12	98.48
512 KB	100.00	97.73	98.99
1 MB	100.00	98.87	99.62
2 MB	100.00	99.44	99.80
4 MB	100.00	99.71	99.87
8 MB	100.00	99.86	99.94
16 MB	100.00	99.94	99.97
32 MB	100.00	99.97	99.99
64 MB	100.00	99.99	99.99
128 MB	100.00	99.99	100.00

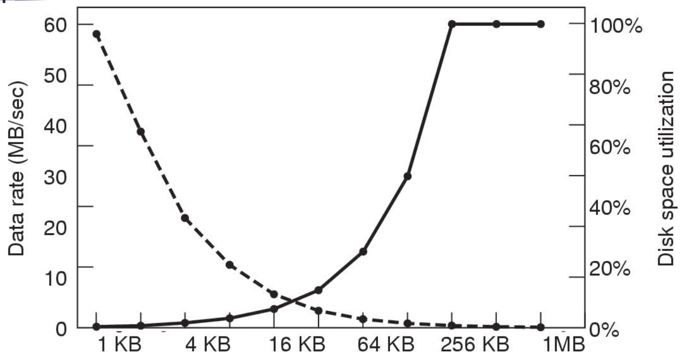
Percentage of files smaller than a given size (in bytes).

Block Size



The solid curve (left-hand scale) gives the data rate of a disk. The dashed curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

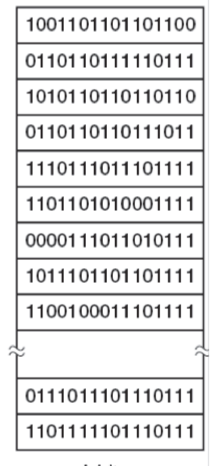
?Block Size



The solid curve (left-hand scale) gives the data rate of a disk.
The dashed curve (right-hand scale) gives the disk space efficiency.
All files are 4 KB.

Keeping Track of Free Blocks

- Bit Vector
  - Bitmap

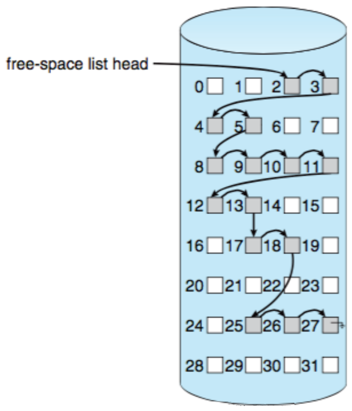


```
1001101101101100
0110110111110111
1010110110110110
0110110110111011
1110111011101111
1101101010001111
0000111011010111
1011101101101111
1100100011101111
≈
0111011101110111
1101111101110111
```

A bitmap

Keeping Track of Free Blocks

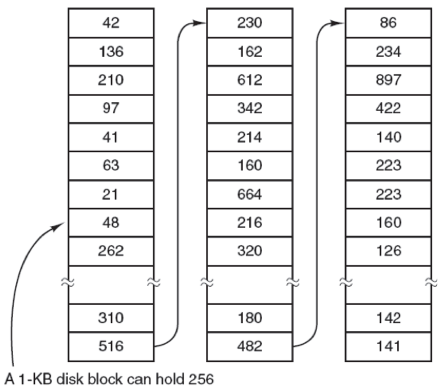
- Linked List



Keeping Track of Free Blocks

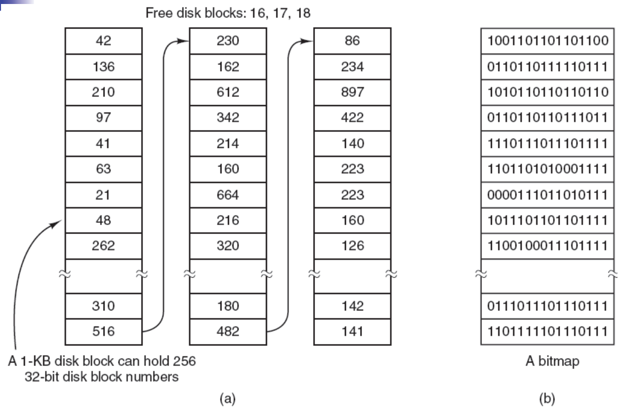
- Grouping Linked List: 成组连接法

Free disk blocks: 16, 17, 18



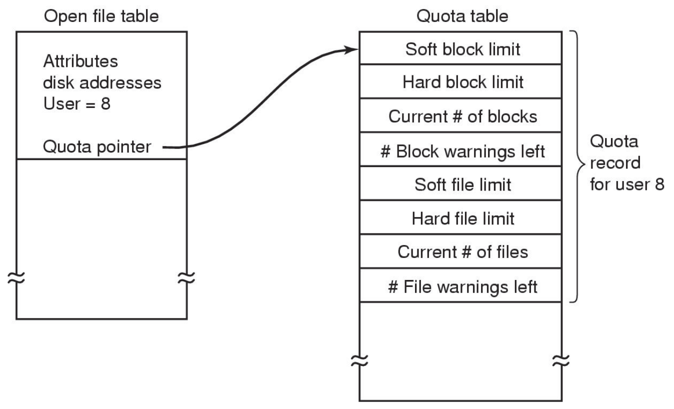
A 1-KB disk block can hold 256
32-bit disk block numbers

Keeping Track of Free Blocks



(a) Storing the free list on a linked list. (b) A bitmap.

Disk Quotas

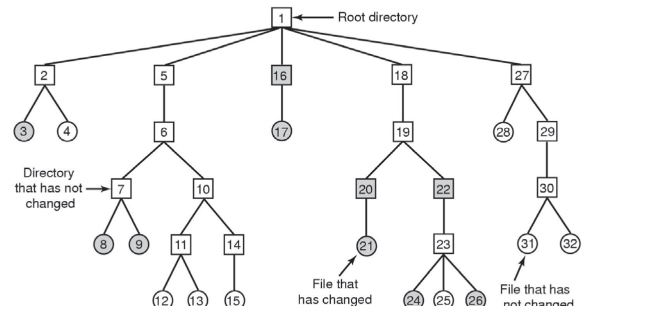


File System Backups

- Backup v.s. Recover
- Physical dump v.s. logical dump

File System Backups

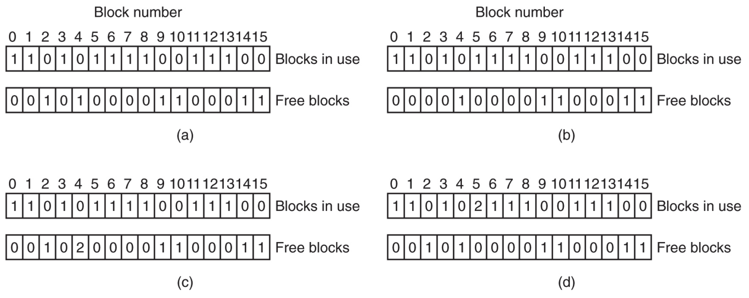
- Full dump v.s. incremental dump



A file system to be dumped. Squares are directories, circles are files. Shaded items have been modified since last dump. Each directory and file is labeled by its i-node number.

File System Consistency

- Blocks in use
- Free blocks


Block number
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 Blocks in use
0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 Free blocks
(a)
Block number
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 Blocks in use
0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 Free blocks
(b)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 Blocks in use
0 0 1 0 2 0 0 0 0 1 1 0 0 0 1 1 Free blocks
(c)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 1 0 1 0 2 1 1 1 0 0 1 1 1 0 0 Blocks in use
0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 Free blocks
(d)

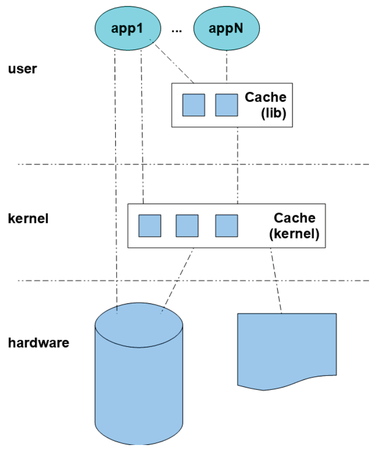
(a) Consistent. (b) Missing block.
(c) Duplicate block in free list. (d) Duplicate data block

File System Performance

- Caching
  - Block cache, buffer cache
- I-nodes layout
- Block Read Ahead
- Reducing Disk Arm Motion

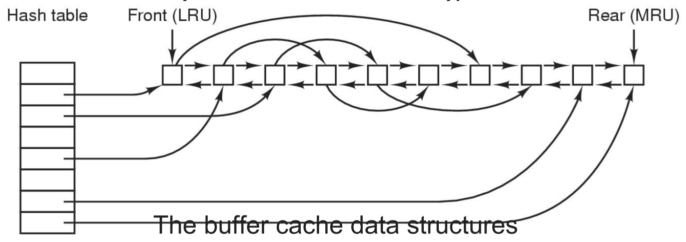
File System Performance: Cache

- Block cache
  - Buffer cache
  - Zero Copy



File System Performance: Cache

- Block cache replacement algorithms
  - LRU (最近最少使用算法),
  - FIFO,
  - MRU(最近最常使用算法), …



The buffer cache data structures

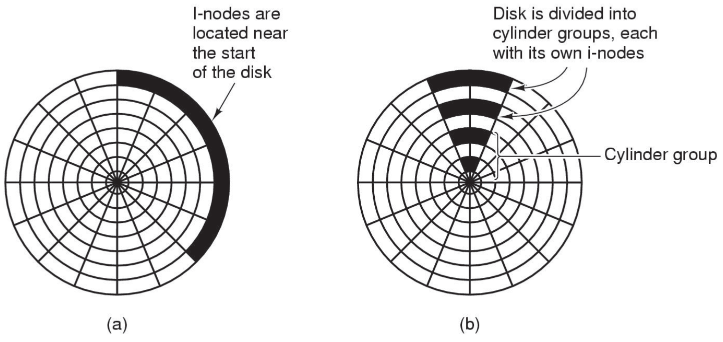
File System Performance: Cache

- Crash and File System Consistency?
- Write-back caches 回写缓存
  - Just cache
- Post Write
  - Cache and memory if modified
- Write-through caches 通写高速缓存
  - Both cache and memory
- Sync, flushXXX

File System Performance

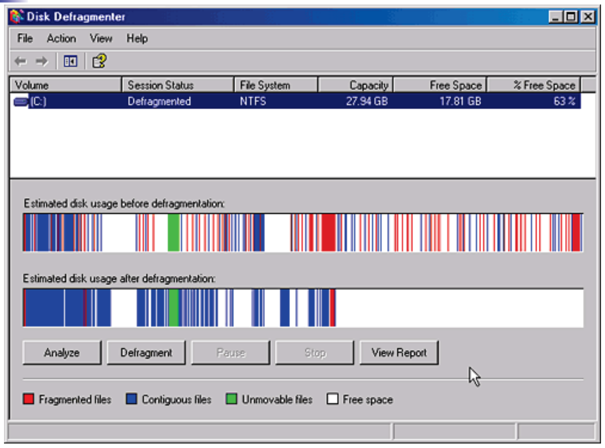
- Block Read Ahead
  - When executing a read from the disk, the disk arm moves the read/write head to (or near) the correct track, and after some settling time the read head begins to pick up bits.
  - Usually, the first sectors to be read are not the ones that have been requested by the operating system.
  - The disk's embedded computer typically saves these unrequested sectors in the disk buffer, in case the operating system requests them later.

File System Performance: I-nodes


I-nodes are located near the start of the disk
Disk is divided into cylinder groups, each with its own i-nodes
Cylinder group
(a)
(b)

- **I-nodes** placed at the start of the disk
- Disk divided into cylinder groups
  - each with its own blocks and i-nodes

Defragmenting Disks



Volume	Session Status	File System	Capacity	Free Space	% Free Space
(C:)	Defragmented	NTFS	27.94 GB	17.81 GB	63 %

Estimated disk usage before defragmentation:

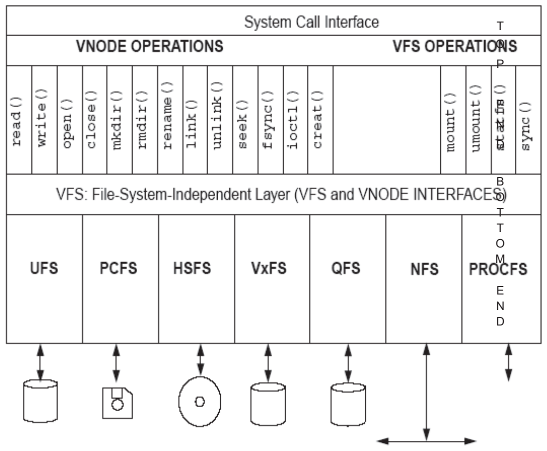
Estimated disk usage after defragmentation:

🟥 Fragmented files 🟦 Contiguous files 🟩 Unmovable files ⬜ Free space

File System Framework Facilities

- **Loadable file system modules** are dynamically loaded at the time each file system type is first mounted.
- **The vnode/vfs framework** implementes file functions and file system management functions.
- **File system caching** implements caching interface with the HAT layer of the virtual memory system to map, unmap, and manage the memory used for caching.
- **Path-name management** converts paths into vnode pointers.
- **Directory name caching** provides a mechanism to cache pathname-to-vnode mappings.

File System Layers



File Systems Supported by Linux

9p configfs freevxfs jffs2 nls smbfs
adfs cramfs fscache jfs notify squashfs
affs debugfs fuse Kconfig ntfs sysfs
afs devpts gfs2 Kconfig.binfmt ocfs2 sysv
autofs dlm hfs lockd omfs ubifs
autofs4 ecryptfs hfsplus logfs openpromfs udf
befs efs hostfs Makefile partitions ufs
bfs exofs hpfs minix proc xfs
btrfs exportfs hppfs ncpfs qnx4
cachefiles ext2 hugetlbfs nfs quota
ceph ext3 isofs nfs_common ramfs
cifs ext4 jbd nfsd reiserfs
coda fat jbd2 nilfs2 romfs

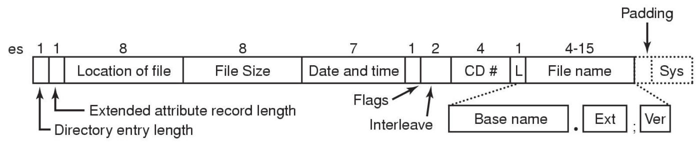
Topics of File Systems

- Log-structured File System
- Journaling File System
- Network File System: NFS

File Systems Cases

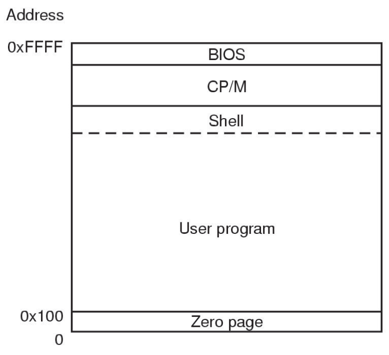
- CD-ROM: ISO9660
- CP/M File System
- FAT 16
- NTFS
- Ext4
- ...

CD-ROM File Systems



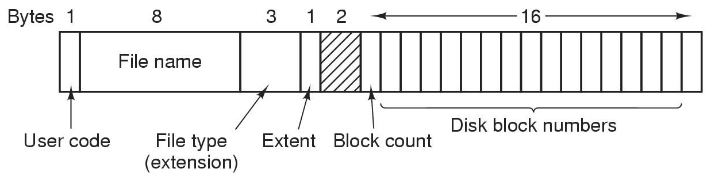
The ISO 9660 directory entry

The CP/M File System (1)



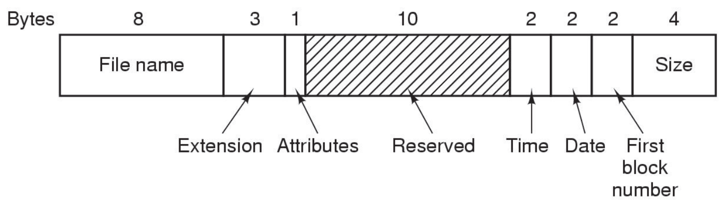
Memory layout of CP/M

The CP/M File System (2)



The CP/M directory entry format

The MS-DOS File System (1)



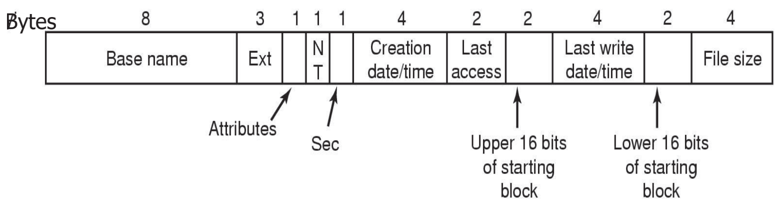
The MS-DOS directory entry

The MS-DOS File System (2)

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

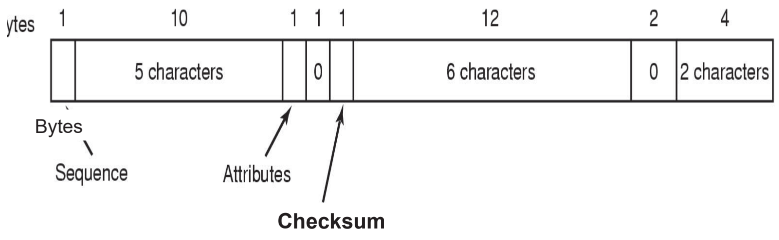
- Maximum partition for different block sizes
- The empty boxes represent forbidden combinations

The Windows 98 File System 1/3



The extended MOS-DOS directory entry used in Windows 98

The Windows 98 File System 2/3



An entry for (part of) a long file name in Windows 98

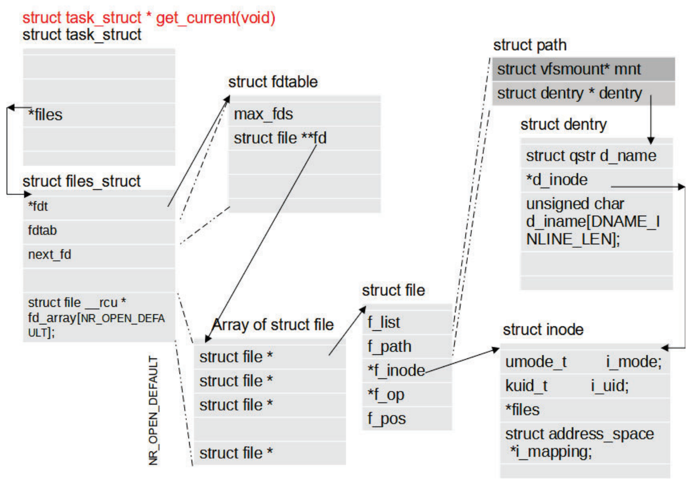
The Windows 98 File System 3/3

68	d o g	A	0	C K					0	
3	o v e	A	0	C K	t h e			l a	0	z y
2	w n f o	A	0	C K	x	j	u	m p	0	s
1	T h e q	A	0	C K	u i	c	k	b	0	r o
T	H E Q U I ~ 1	A	N T	S	Creation time	Last acc	Upp	Last write	Low	Size

Bytes

An example of how a long name is stored in Windows 98

File Structure In Linux Kernel



File Structure In Linux Kernel

- /include/linux/sched.h

```
struct task_struct{

    /* Filesystem information: */
    struct fs_struct        *fs;

    /* Open file information: */
    struct files_struct     *files;
}
```

File Structure In Linux Kernel

- /include/linux/fdtable.h

```
struct files_struct {
   atomic_t count;
   bool resize_in_progress;
   wait_queue_head_t resize_wait;
   struct fdtable __rcu *fdt;
   struct fdtable fdtab;
   spinlock_t file_lock ____cacheline_aligned_in_smp;
   unsigned int next_fd;
   unsigned long close_on_exec_init[1];
   unsigned long open_fds_init[1];
   unsigned long full_fds_bits_init[1];
   struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};
```

File Structure In Linux Kernel

- /include/linux/fdtable.h

```
struct fdtable {
    unsigned int max_fds;
    struct file __rcu **fd;       /* current fd array */
    unsigned long *close_on_exec;
    unsigned long *open_fds;
    unsigned long *full_fds_bits;
    struct rcu_head rcu;
};
```

File Structure In Linux Kernel

- /include/linux/fs.h

```
struct file {
    union {
        struct llist_node   fu_llist;
        struct rcu_head     fu_rcuhead;
    } f_u;
    struct path     f_path;
    struct inode        *f_inode;   /* cached value */
    const struct file_operations    *f_op;
    /*
     * Protects f_ep, f_flags.
     * Must not be taken from IRQ context.
     */
    spinlock_t      f_lock;
    enum rw_hint        f_write_hint;
    atomic_long_t       f_count;
    unsigned int        f_flags;
    fmode_t         f_mode;
    struct mutex        f_pos_lock;
    loff_t          f_pos;
    struct fown_struct  f_owner;
    const struct cred   *f_cred;
    struct file_ra_state    f_ra;
```

- /include/linux/fs.h

```
struct file {
    ...
    u64         f_version;
#ifdef CONFIG_SECURITY
    void            *f_security;
#endif
    /* needed for tty driver, and maybe others
*/
    void            *private_data;

#ifdef CONFIG_EPOLL
    /* Used by fs/eventpoll.c to link all the
hooks to this file */
    struct hlist_head   *f_ep;
#endif /* #ifdef CONFIG_EPOLL */
    struct address_space    *f_mapping;
    errseq_t        f_wb_err;
    errseq_t        f_sb_err; /* for syncfs */
} __randomize_layout
  __attribute__((aligned(4)));  /* lest
something weird decides that 2 is OK */
```

File Structure In Linux Kernel

- /include/linux/path.h

```
struct path {
    struct vfsmount *mnt;
    struct dentry *dentry;
} __randomize_layout;
```

File Structure In Linux Kernel

- /include/linux/dcache.h

```
struct dentry {
    /* RCU lookup touched fields */
    unsigned int d_flags;       /* protected by d_lock */
    seqcount_spinlock_t d_seq;  /* per dentry seqlock */
    struct hlist_bl_node d_hash;    /* lookup hash list */
    struct dentry *d_parent;    /* parent directory */
    struct qstr d_name;
    struct inode *d_inode;      /* Where the name belongs to -
NULL is negative */
    unsigned char d_iname[DNAME_INLINE_LEN];    /* small
names */

    /* Ref lookup also touches following */
    struct lockref d_lockref;   /* per-dentry lock and refcount */
    const struct dentry_operations *d_op;
    struct super_block *d_sb;   /* The root of the dentry tree
*/
    unsigned long d_time;       /* used by d_revalidate */
    void *d_fsdata;         /* fs-specific data */
```

- /include/linux/dcache.h

```
struct dentry {
    ...
        union {
        struct list_head d_lru;     /* LRU list */
        wait_queue_head_t *d_wait;  /* in-lookup ones
only */
    };
    struct list_head d_child;   /* child of parent list */
    struct list_head d_subdirs; /* our children */
    /*
     * d_alias and d_rcu can share memory
     */
    union {
        struct hlist_node d_alias;  /* inode alias list */
        struct hlist_bl_node d_in_lookup_hash;  /* only for
in-lookup ones */
        struct rcu_head d_rcu;
    } d_u;
} __randomize_layout;
```

File Structure In Linux Kernel

- /include/linux/dcache.h

```
struct qstr {
    union {
        struct {
            HASH_LEN_DECLARE;
        };
        u64 hash_len;
    };
    const unsigned char *name;
};
```

File Structure In Linux Kernel

- /include/linux/fs.h

```
struct inode {
    umode_t         i_mode;
    unsigned short      i_opflags;
    kuid_t          i_uid;
    kgid_t          i_gid;
    unsigned int        i_flags;

#ifdef CONFIG_FS_POSIX_ACL
    struct posix_acl    *i_acl;
    struct posix_acl    *i_default_acl;
#endif

    const struct inode_operations   *i_op;
    struct super_block  *i_sb;
    struct address_space    *i_mapping;

#ifdef CONFIG_SECURITY
    void            *i_security;
#endif

    /* Stat data, not accessed from path
walking */
    unsigned long       i_ino;
```

- /include/linux/fs.h

```
struct inode {
    ...
    /*
     * Filesystems may only read i_nlink directly.  They
shall use the
     * following functions for modification:
     *
     *    (set|clear|inc|drop)_nlink
     *    inode_(inc|dec)_link_count
     */
    union {
        const unsigned int i_nlink;
        unsigned int __i_nlink;
    };
    dev_t           i_rdev;
    loff_t          i_size;
    struct timespec64   i_atime;
    struct timespec64   i_mtime;
    struct timespec64   i_ctime;
    spinlock_t      i_lock; /* i_blocks, i_bytes, maybe
i_size */
    unsigned short          i_bytes;
    u8          i_blkbits;
    u8          i_write_hint;
    blkcnt_t        i_blocks;
#ifdef __NEED_I_SIZE_ORDERED
    seqcount_t      i_size_seqcount;
#endif
```

File Structure In Linux Kernel

- /include/linux/fs.h

```
struct inode {
...
        /* Misc */
    unsigned long       i_state;
    struct rw_semaphore i_rwsem;

    unsigned long       dirtied_when;   /* jiffies of first
dirtying */
    unsigned long       dirtied_time_when;

    struct hlist_node   i_hash;
    struct list_head    i_io_list;  /* backing dev IO list */
#ifdef CONFIG_CGROUP_WRITEBACK
    struct bdi_writeback    *i_wb;      /* the associated
cgroup wb */

    /* foreign inode detection, see wbc_detach_inode() */
    int         i_wb_frn_winner;
    u16         i_wb_frn_avg_time;
    u16         i_wb_frn_history;
#endif
    struct list_head    i_lru;      /* inode LRU list */
    struct list_head    i_sb_list;
    struct list_head    i_wb_list;  /* backing dev writeback
list */
    union {
        struct hlist_head   i_dentry;
        struct rcu_head     i_rcu;
    };
```

- /include/linux/fs.h

```
struct inode {
    ...
    atomic64_t      i_version;
    atomic64_t      i_sequence; /* see futex */
    atomic_t        i_count;
    atomic_t        i_dio_count;
    atomic_t        i_writecount;
#if defined(CONFIG_IMA) ||
defined(CONFIG_FILE_LOCKING)
    atomic_t        i_readcount; /* struct files open
RO */
#endif
    union {
        const struct file_operations    *i_fop; /*
former ->i_op->default_file_ops */
        void (*free_inode)(struct inode *);
    };
    struct file_lock_context    *i_flctx;
    struct address_space    i_data;
    struct list_head    i_devices;
    union {
        struct pipe_inode_info  *i_pipe;
       struct cdev     *i_cdev;
        char            *i_link;
        unsigned        i_dir_seq;
    };
```

File Structure In Linux Kernel

- /include/linux/fs.h

```
struct inode {
...
    __u32           i_generation;

#ifdef CONFIG_FSNOTIFY
    __u32           i_fsnotify_mask; /* all
events this inode cares about */
    struct fsnotify_mark_connector __rcu
*i_fsnotify_marks;
#endif

#ifdef CONFIG_FS_ENCRYPTION
    struct fscrypt_info *i_crypt_info;
#endif

#ifdef CONFIG_FS_VERITY
    struct fsverity_info    *i_verity_info;
#endif

    void            *i_private; /* fs or device
private pointer */
} __randomize_layout;
```

Summary

- File Concept
- Directory Concept
- File Share & Protection
- File System Implementation
- File System Reliability
- File System Performance
- File System Cases