现 代 操 作 系 统

Modern Operating Systems

宋虹 songhong@csu. edu. cn

中南大学计算机学院网络空间安全系

Chapter 4 File Systems

- **4.1 Files**
- **4.2 Directories**
- **4.3 File System Implementation**
- **4.4 File System Management and Optimization**
- **4.5 Example File System**
- **4.6 Reading Materials**

4.1 Files

- **Some Concepts**
  - File: logical units of information created by processes
  - File System: the part of OS dealing with files
  - File Naming
    - The exact rules for file naming vary somewhat from system to system
  - File extension
- **File structure**
  - Three structure ways: byte sequence, record sequence, tree

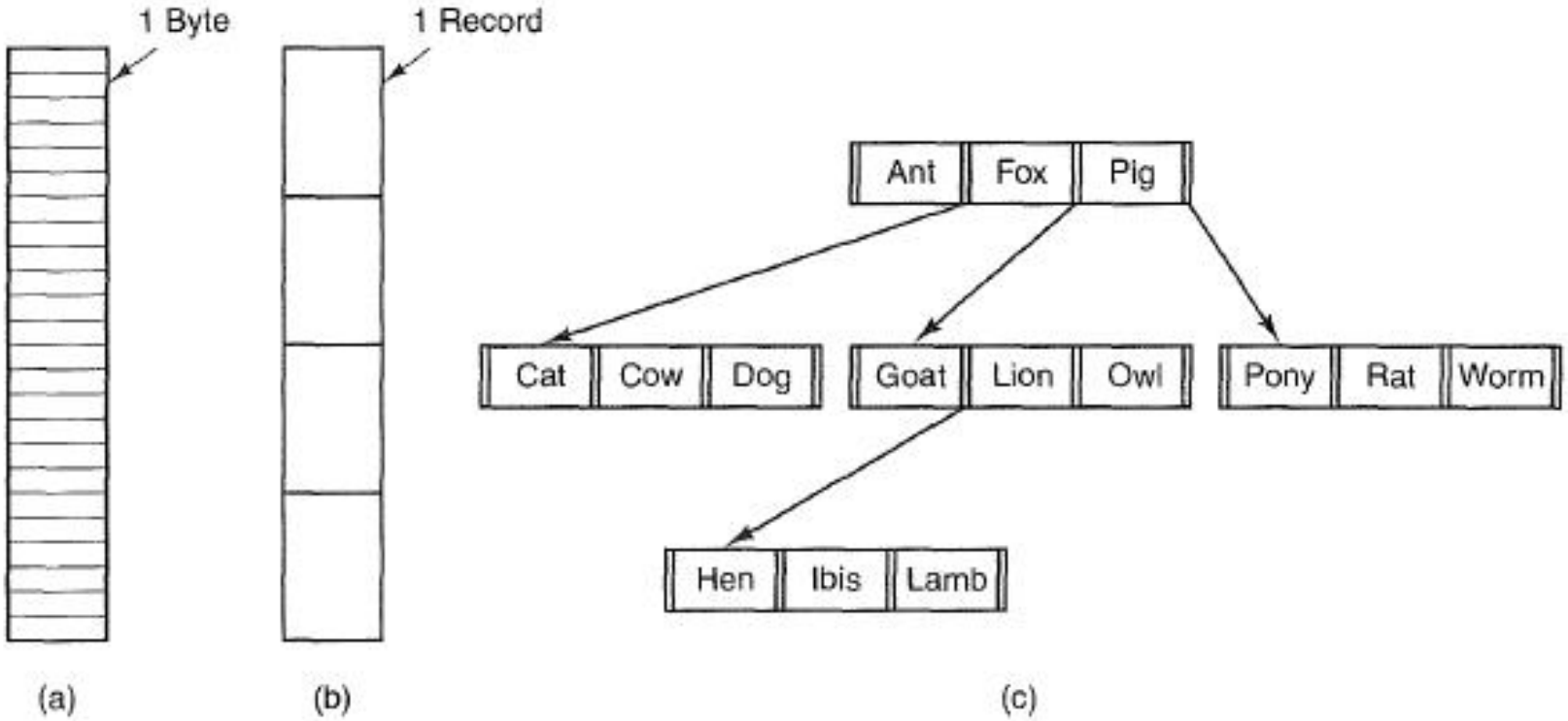


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

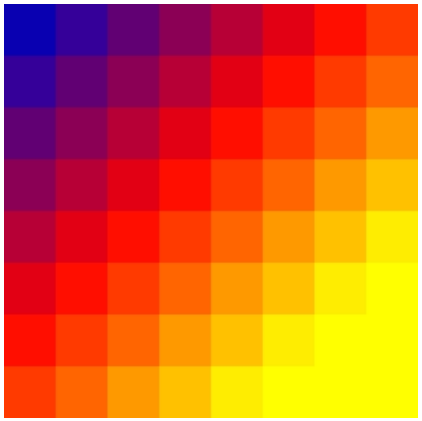
4.1 Files

- **File Types**
  - Regular file
    - ASCII file
    - Binary file
  - Character special file
  - Block special file
- **File Access**
  - Sequential access
  - Random access file
  - read & seek

4.1 Files

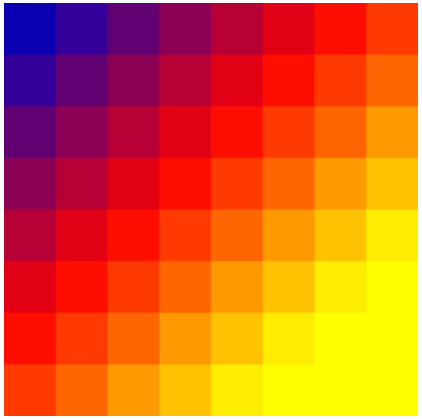
- **File Attributes**

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



4.1 Files

- **File Attributes**



```
00000000h: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 ; 塒NG........IHDR
00000010h: 00 00 00 08 00 00 00 08 04 03 00 00 00 36 21 A3 ; .............6!?
00000020h: B8 00 00 00 03 73 42 49 54 08 08 08 DB E1 4F E0 ; ?...sBIT...坩O?
00000030h: 00 00 00 27 50 4C 54 45 FF FF 00 FF ED 00 FF C1 ; ...'PLTE    .  ?  ?
00000040h: 00 FF 99 00 FF 66 00 FF 3B 00 FF 0F 00 E2 00 15 ; .  ?  f.  ;.  ..?.
00000050h: B7 00 34 8B 00 54 60 00 73 33 00 99 09 00 B2 5F ; ?4?T`.s3.?.瞋
00000060h: F5 BB DD 00 00 00 09 70 48 59 73 00 00 0B 12 00 ; 冟?...pHYs.....
00000070h: 00 0B 12 01 D2 DD 7E FC 00 00 00 20 74 45 58 74 ; ....逸~?.. tEXt
00000080h: 53 6F 66 74 77 61 72 65 00 4D 61 63 72 6F 6D 65 ; Software.Macrome
00000090h: 64 69 61 20 46 69 72 65 77 6F 72 6B 73 20 4D 58 ; dia Fireworks MX
000000a0h: BB 91 2A 24 00 00 00 16 74 45 58 74 43 72 65 61 ; 粪*$....tEXtCrea
000000b0h: 74 69 6F 6E 20 54 69 6D 65 00 30 34 2F 30 31 2F ; tion Time.04/01/
000000c0h: 30 35 22 44 50 99 00 00 00 27 49 44 41 54 78 9C ; 05"DP?..'IDATx?
000000d0h: 63 38 BD B2 3D 95 61 D7 8C B2 10 06 20 C3 99 01 ; c8讲=昇譲?. 腟.
000000e0h: C8 30 62 00 32 14 19 80 0C 01 06 10 83 01 C4 00 ; ?b.2..€....??
000000f0h: 00 24 A7 0B A4 DA 12 06 A5 00 00 00 00 49 45 4E ; .$?~..?...IEN
00000100h: 44 AE 42 60 82                                 ; D頤`□
```

  - 00 00 00 0D：IHDR头块长
  - 00 00 00 08宽，00 00 00 08高
  - 04色深， 03 颜色类型
  - 00 00 00表示压缩方法为非隔行扫描
  - 36 21 A3 B8 CRC校验

File	Type	ExtensionsHeader
JPEG	jpg;jpeg	0xFFD8FF
PNG	png	0x89504E470D0A1A0A
GIF	gif	GIF8
TIFF	tif;tiff	0x49492A00
TIFF	tif;tiff	0x4D4D002A
Bit map	bmp	BM
Lotus WordPro v9	lwp	0x576F726450726F
Lotus 123 v9	123	0x00001A00051004
Lotus 123 v5	wk4	0x00001A0002100400
Lotus 123 v3	wk3	0x00001A0000100400
Lotus 123 v1	wk1	0x2000604060
Windows Password	pwl	0xE3828596
ZIP Archive	zip;jar	0x504B0304
ZIP Archive (outdated)	zip	0x504B3030
RAR Archive	rar	Rar!
MPEG	mpg;mpeg	0x000001BA
MPEG	mpg;mpeg	0x000001B3
Windows Media	asf	0x3026B2758E66CF11

压缩包：

a、文件头结构

组成	长度
文件头标记	4 bytes
解压文件所需 pkware 版本	2 bytes
全局方式位标记	2 bytes
压缩方式	2 bytes
最后修改文件时间	2 bytes
最后修改文件日期	2 bytes
CRC-32校验	4 bytes
压缩后尺寸	4 bytes
未压缩尺寸	4 bytes
文件名长度	2 bytes
扩展记录长度	2 bytes
文件名	(不定长度)
扩展字段	(不定长度)

b、文件数据

c、数据描述符

组成	长度
CRC-32校验	4 bytes
压缩后尺寸	4 bytes
未压缩尺寸	4 bytes

4.1 Files

- **File Operations**
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write
  - Append
  - Seek
  - Get attributes
  - Set attributes
  - Rename

4.1 Files

- **Example Program**

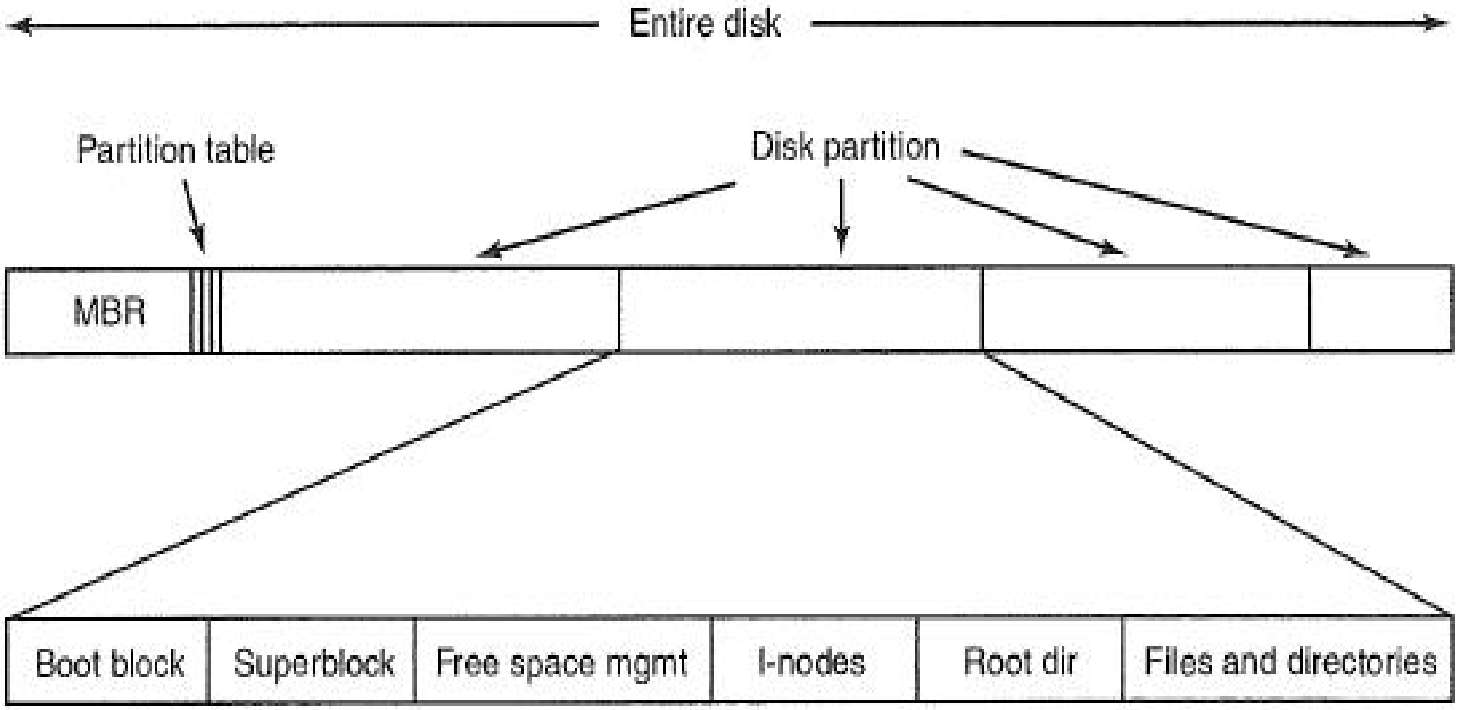
```
/* File copy program. Error checking and reporting is minimal. */

#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 */

    /* 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 */
}
```

4.2 Directories

- **Single-level Directory Systems**
- **Hierarchical Directory Systems**
- **Path names**
  - Relative path name: Working directory…current directory
  - Absolute path name
- **Directory operations**
  - Create
  - Delete
  - Opendir
  - Closedir
  - Readdir
  - Rename
  - Link: hard link & symbolic link
  - Unlink

4.3 File System Implementation

- **From the implementor's view**
  - How files and directories are stored
  - How disk space is managed
  - How to make everything work efficiently and reliably
- **File system layout**
  - Master Boot Record: Sector 0
  - Partition table
  - Boot block: the first block of active partition
  - Super block



A possible file system layout

4.3 File System Implementation

- **Implementing Files**
  - File storage
    - Contiguous allocation
    - Linked list allocation
    - Linked List Allocation using a table in memory——FAT
    - i-node——index-node
- **Implementing Directories**
  - File attributes must be stored
    - in directory entry
    - in the i-node

问题：

- 一种在磁盘上连续分配并且可以避免空洞的方案是，每次删除一个文件后就紧缩一下磁盘。由于所有的文件都是连续的，复制文件时需要寻道和旋转延迟以便读取文件，然后全速传送。在写回文件时要做同样的工作。假设寻道时间为5ms，旋转延迟为4ms，传送速率为8MB/s，而文件平均长度是8KB，把一个文件读入内存并写回到磁盘上的一个新位置需要多长时间？紧缩16GB磁盘的一半需要多少时间？

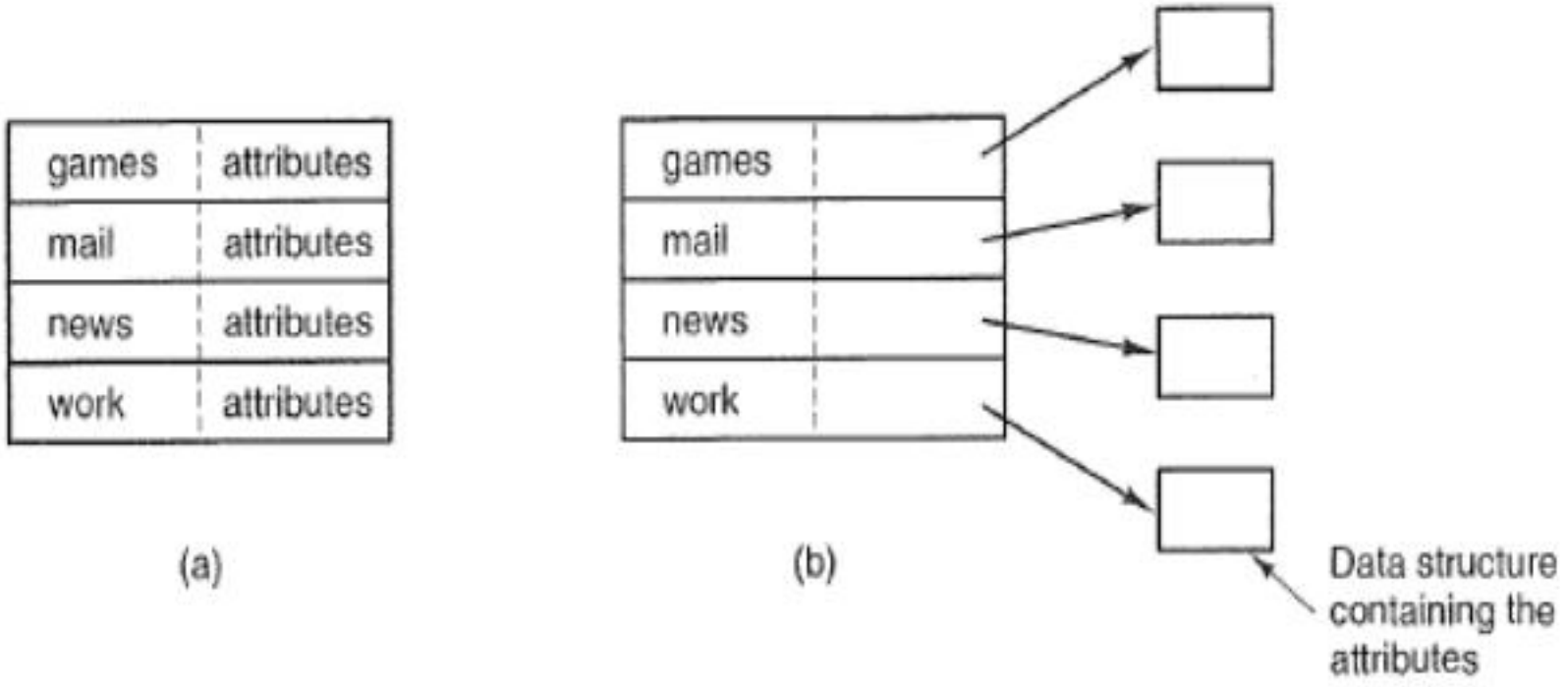


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

4.3 File System Implementation

- **Implementing Directories**
  - Implementing variable-length file names
    - Fixed file name length——255 characters
    - Fixed portion of directory entries
- **Shared Files**
  - Directed Acyclic Graph——DAG
  - Problems
    - If directories really do contain disk addresses, new blocks appended to the file will not be visible to the other users but the user doing the append.
  - Two solutions
    - i-node
    - Symbolic linking

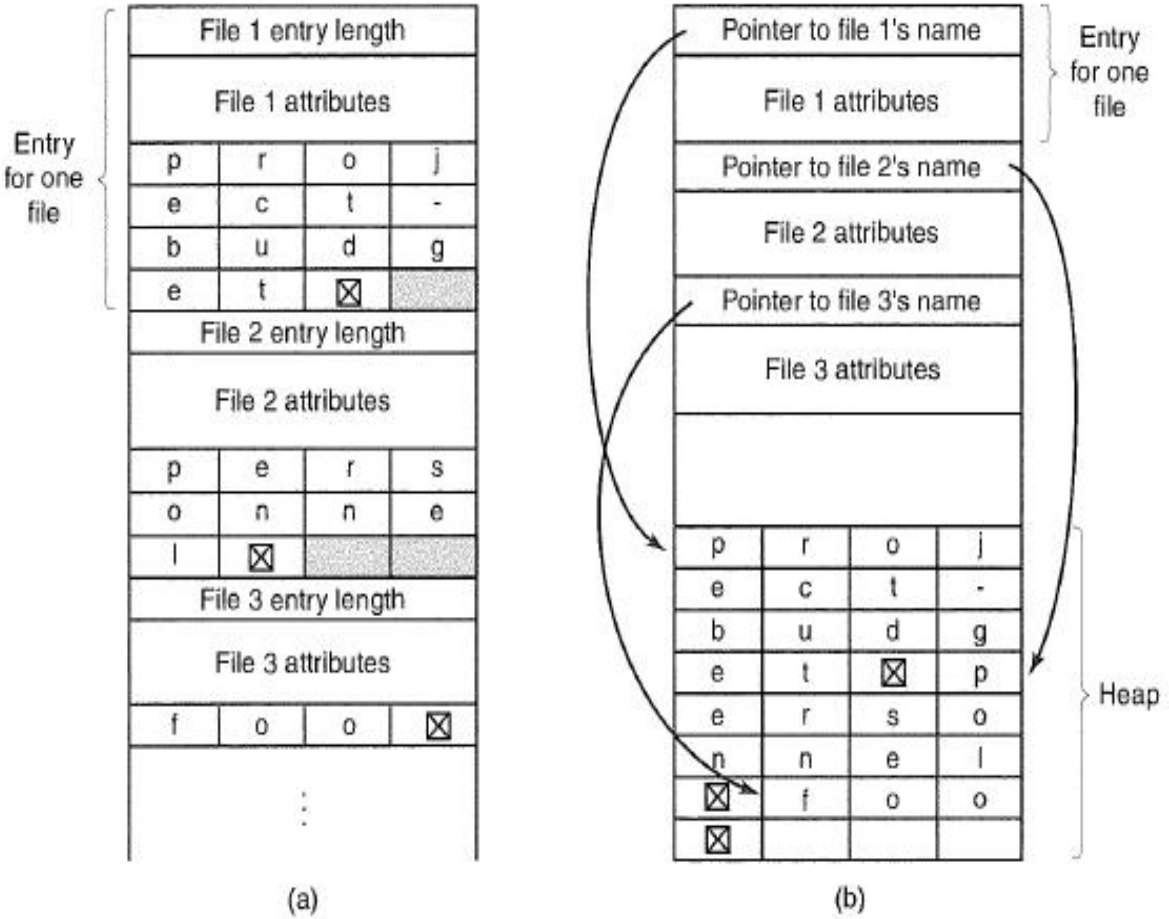


Figure 4-15. Two ways of handling long file names in a directory. (a) In-line. (b) In a heap.

4.3 File System Implementation

- **Log-structured file systems**
  - A new kind of file system designed by Berkeley in 1991.
  - in order to achieve the full bandwidth of the disk, even in the face of a workload consisting in large part of small random writes.
  - Main idea: structure the entire disk as a log, all the pending writes being buffered in memory are collected into a single segment and written to the disk as a single contiguous segment at the end of the log periodically.
  - The average segment can be made to be about 1MB.
  - Cleaner thread in LFS to scan the log circularly to compact it.

4.3 File System Implementation

- **Journaling file systems**
  - Used in NTFS, Linux ext3, ReiserFS and etc.
  - Basic idea: to keep a log of what the file system is going to do before it does it, so that if the system crashes before it can do its planned work, upon rebooting the system can look in the log to see what was going on at the time of the crash and finish the job.
  - Atomic transaction

4.3 File System Implementation

- **File System in NVMM(Non-Volatile Main Memory)**
  - BPFS: using shadow paging.
  - PMFS, Ext4-DAX: based on DAX, using journaling technology
  - SCMFS: using virtual main memory management.
  - Aerie: journaling technology
  - NOVA: using log-structuring FS

4.3 File System Implementation

- **Virtual file systems**
  - To integrate multiple file systems into a single structure.
  - POSIX interface to user processes: upper interface
  - VFS interface to the concrete file system: lower interface
  - Object-oriented
  - key object types: superblock, v-node, directory, mount table, array of file descriptors

Virtual File Systems (1)

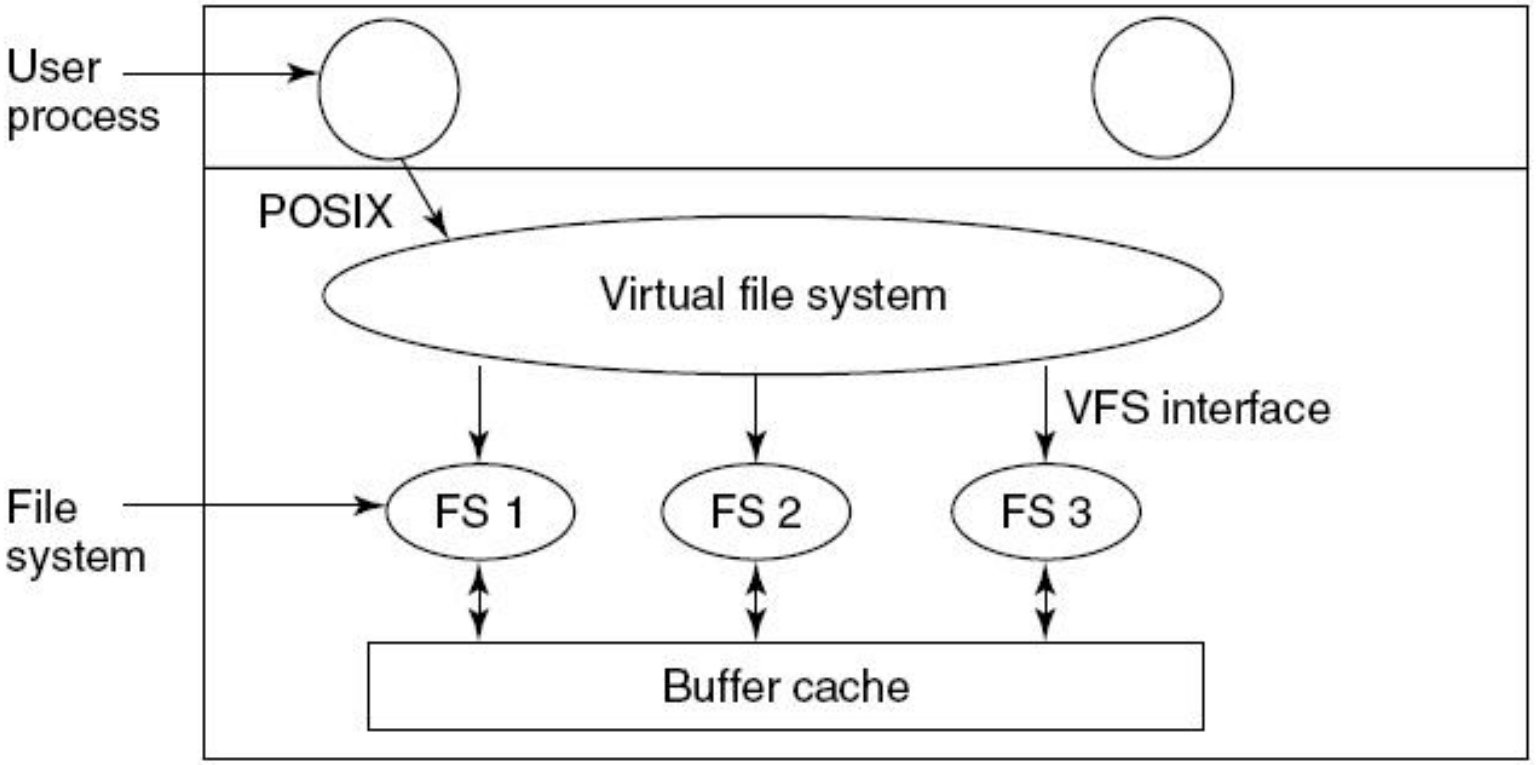


Figure 4-18. Position of the virtual file system.

4.3 File System Implementation

- **Virtual file systems**
  - How VFS works:
    - When system is booted, root file system is registered to the VFS——provide a list of the addresses of the functions the VFS requires.
    - When a process uses system call for file system, VFS sees the new file system has been mounted and locates its superblock by searching the list of superblocks.
    - VFS finds the root directory of the mounted file system and look up the path.
    - VFS creates a v-node and makes a call to the concrete file system to return all the information in the file's i-node. The information is copied into the v-node in RAM.
    - VFS makes an entry in the file descriptor table for the calling process and sets it to point to the new v-node. Return the file descriptor and execute the proper operations.
    - The data structures involved are shown in next slides.

Virtual File Systems (2)

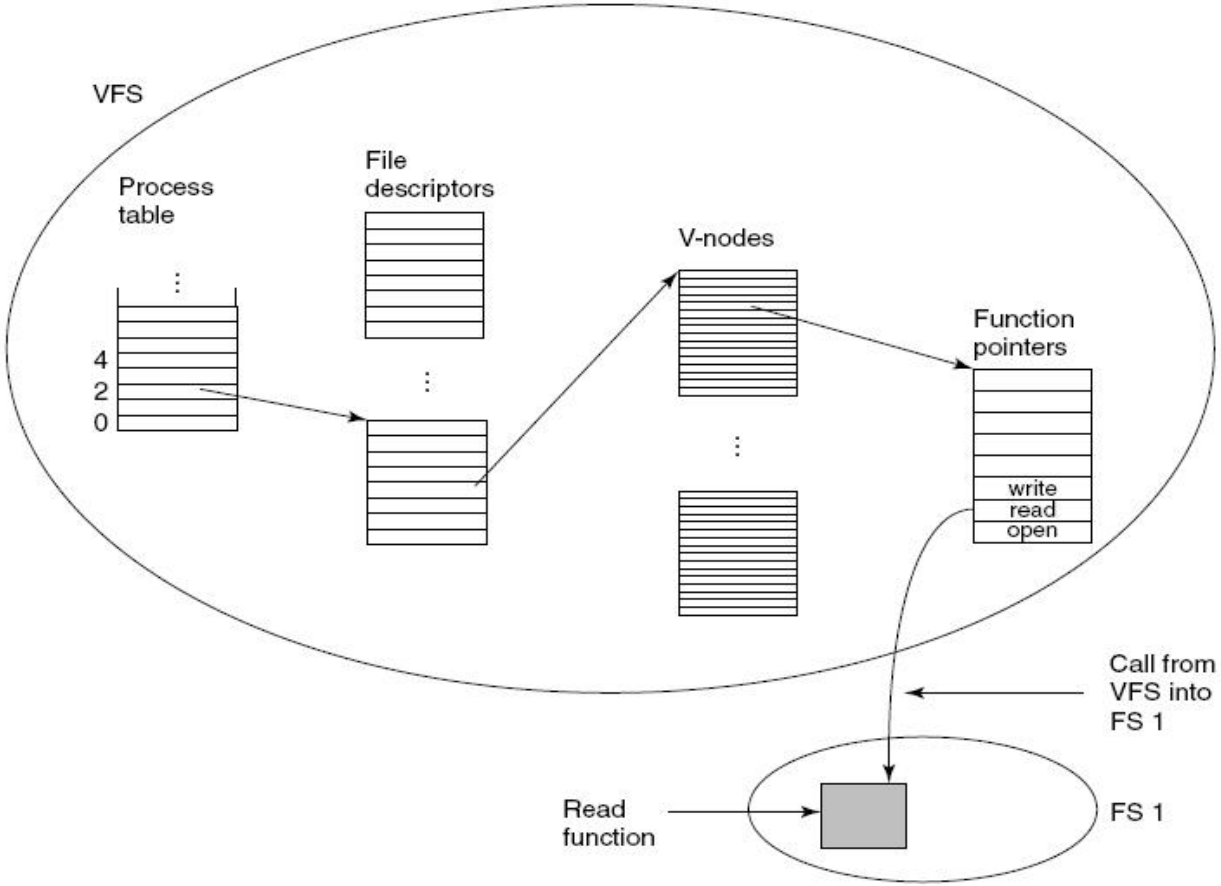


Figure 4-19. A simplified view of the data structures and code used by the VFS and concrete file system to do a read.

4.4 File System management and optimization

- **Disk Space Management**
  - Block Size: Tanenbaum et al. (2006) studied the file size distribution in 1984 and in 2005, and the result are shown in the next slide.
  - The time to read a block of k bytes is the sum of the seek, rotational delay, and transfer times.

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

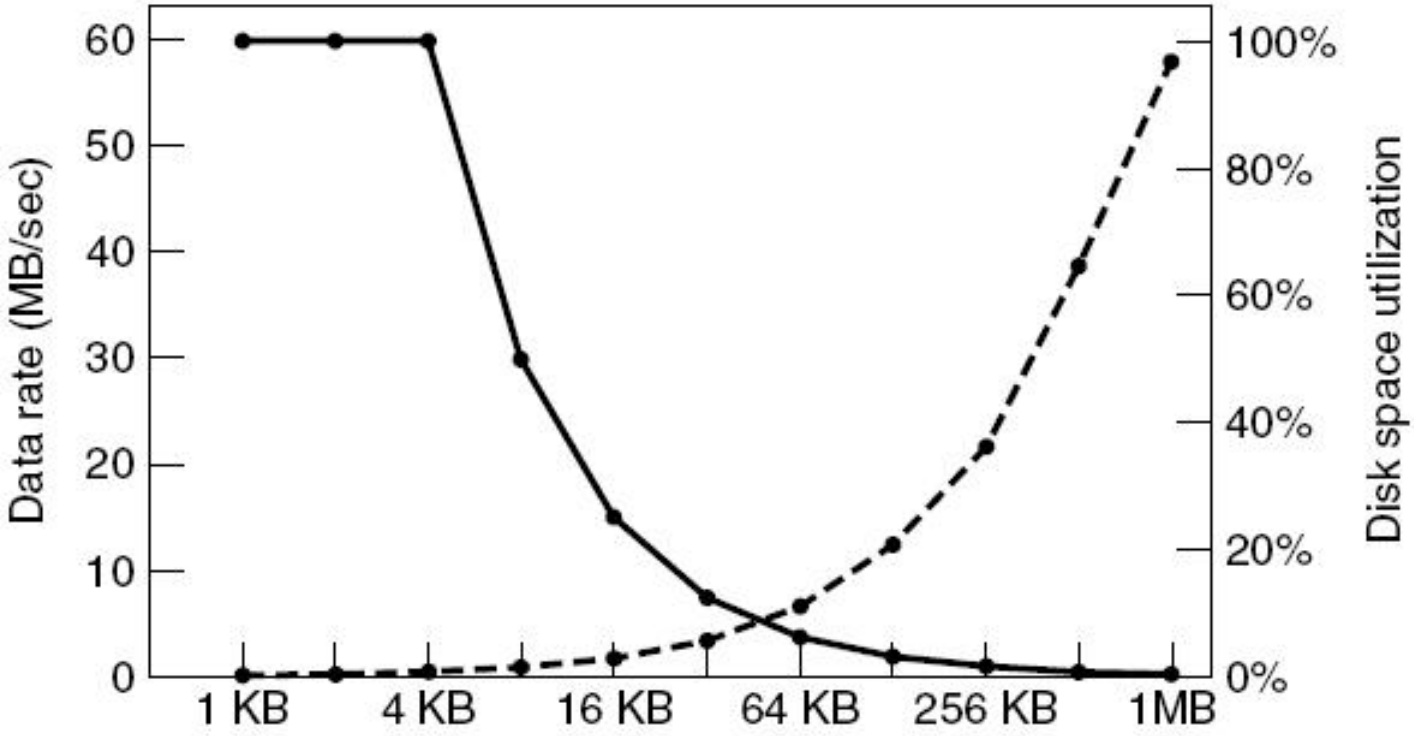
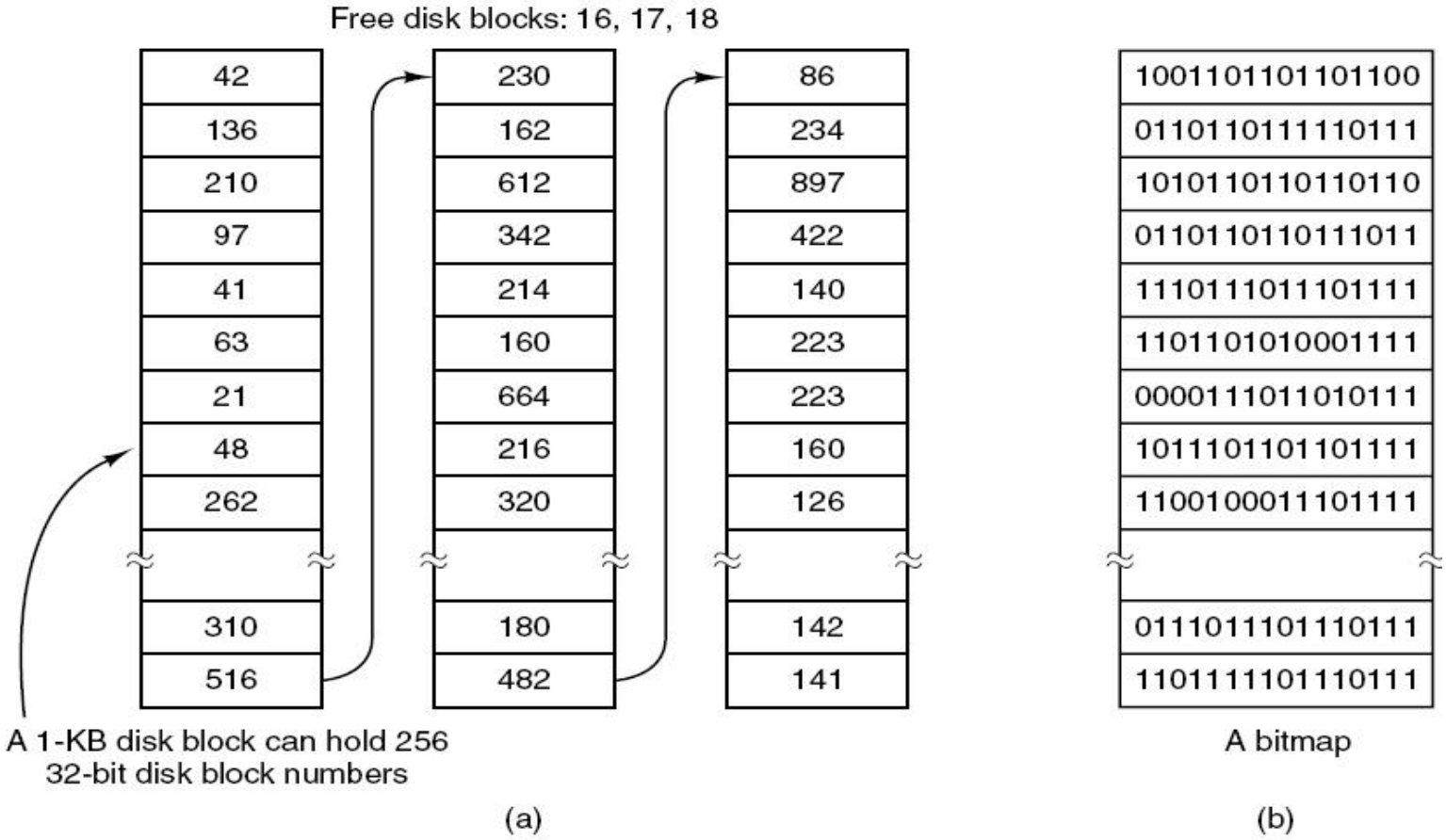


Figure 4-21. The solid curve (left-hand scale) gives the data rate of a disk. The dashed curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

4.4 File System management and optimization

- **Keeping Track of Free Blocks**



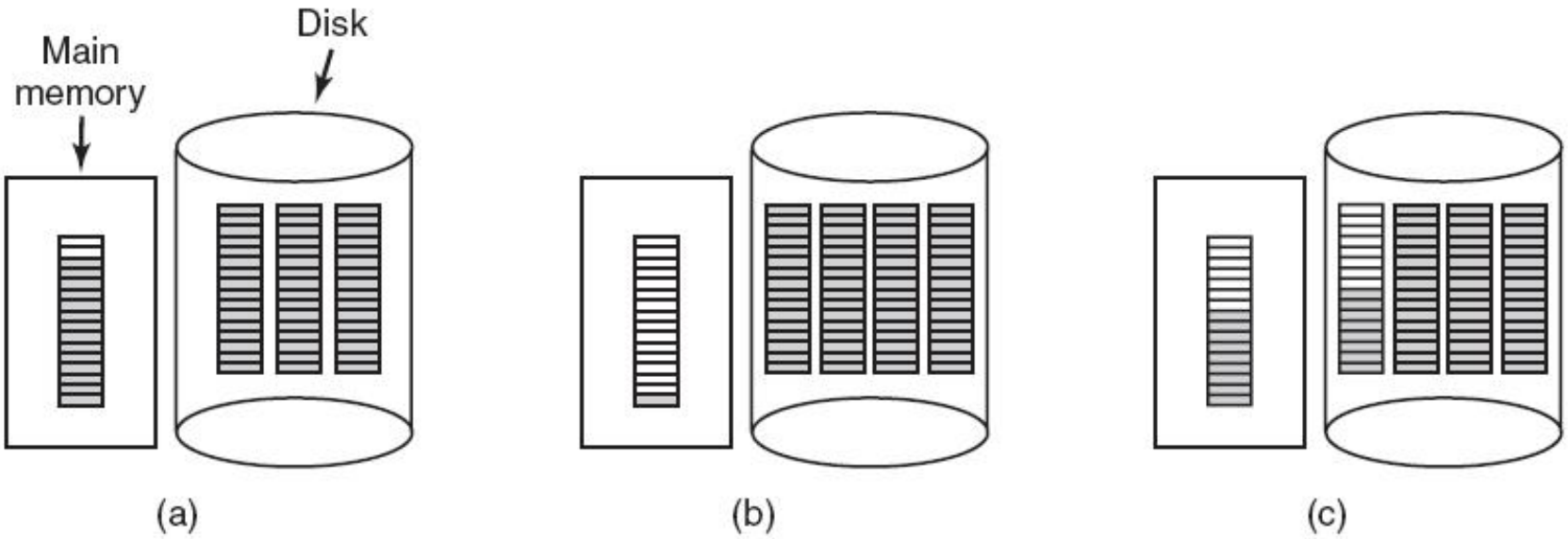
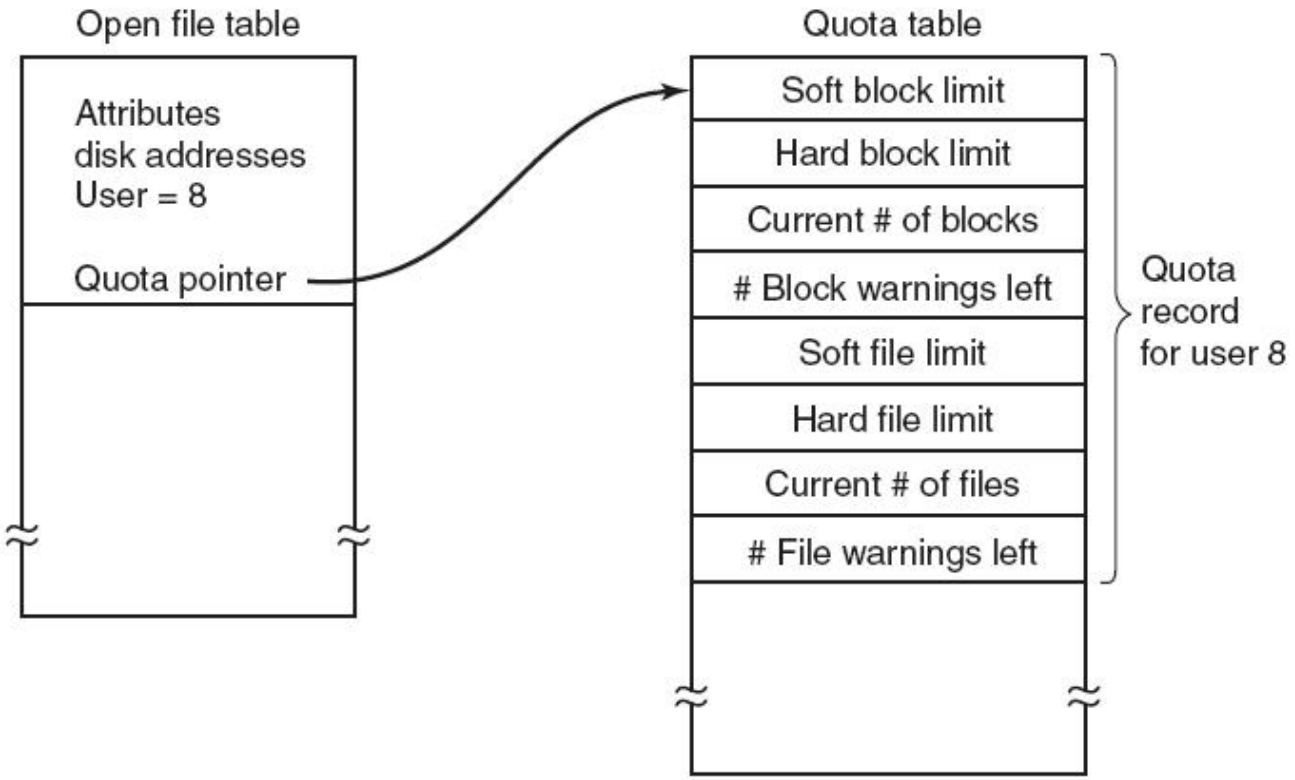


Figure 4-23. (a) An almost-full block of pointers to free disk blocks in memory and three blocks of pointers on disk. (b) Result of freeing a three-block file. (c) An alternative strategy for handling the three free blocks. The shaded entries represent pointers to free disk blocks.

4.4 File System management and optimization

- **Disk Quotas**



4.4 File System management and optimization

- **File System backups**
  - Backups to tape are generally made to handle one of two potential problems:
    - Recover from disaster
    - Recover from stupidity

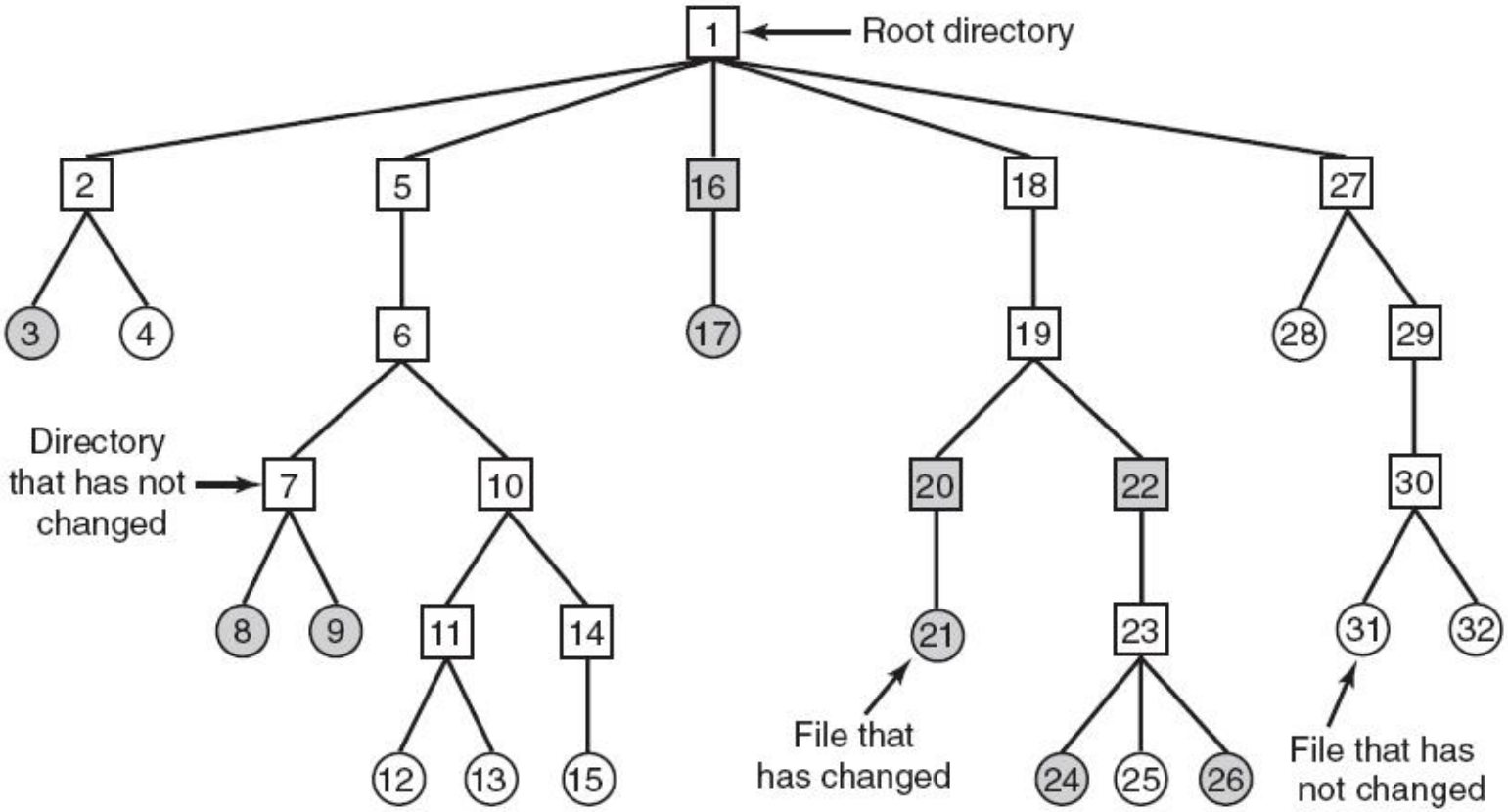


Figure 4-25. A file system to be dumped. Squares are directories, circles are files. Shaded items have been modified since last dump. Each directory and file is labeled by its i-node number.

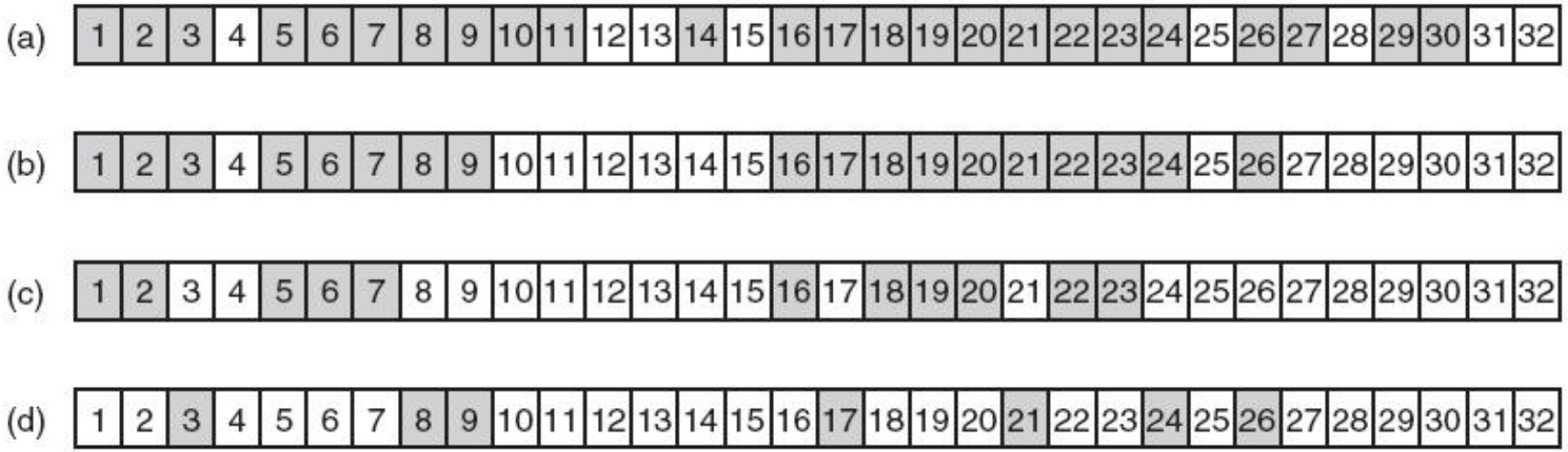


Figure 4-26. Bitmaps used by the logical dumping algorithm.

4.4 File System management and optimization

- **File System Consistency**

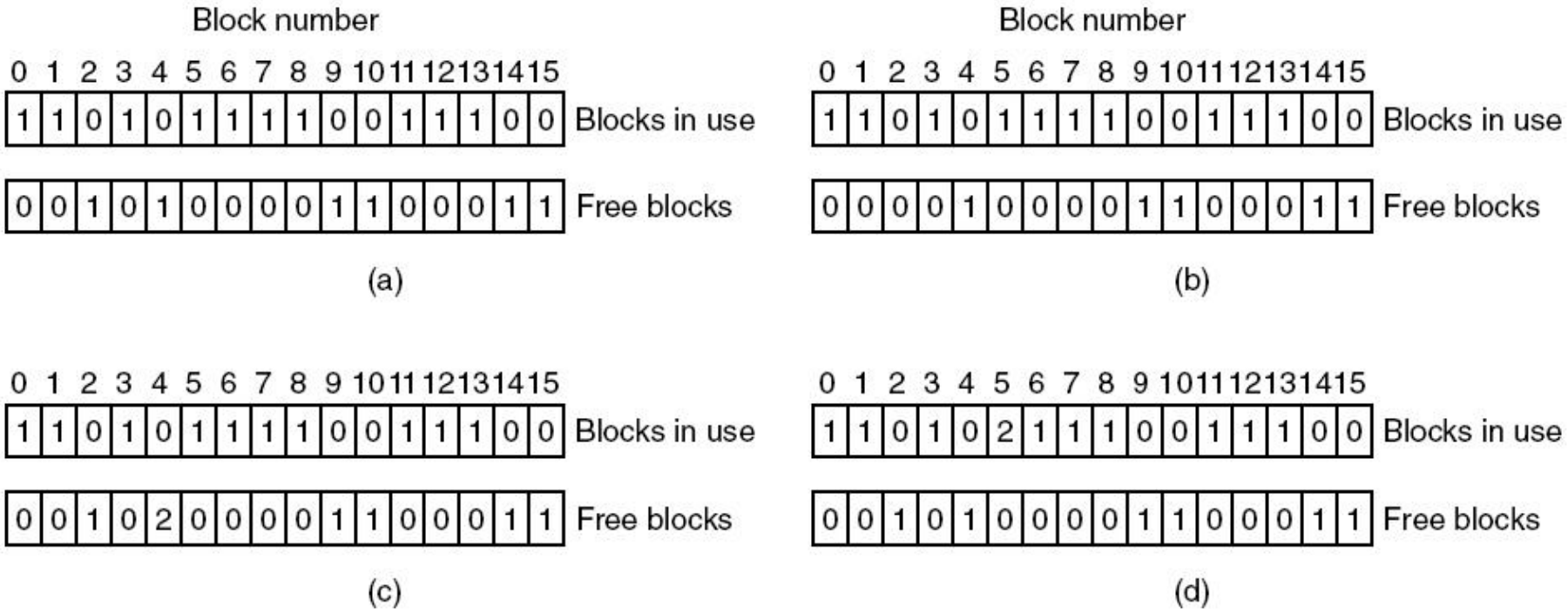


Figure 4-27. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

4.4 File System management and optimization

- **File System Performance**
  - Caching
    - Some blocks, such as i-node blocks, are rarely referenced two times within a short interval.
    - Consider a modified LRU scheme, taking two factors into account:
      - Is the block likely to be needed again soon?
      - Is the block essential to the consistency of the file system?
  - Block Read Ahead
  - Reducing Disk Arm Motion
- **Defragmenting Disks**

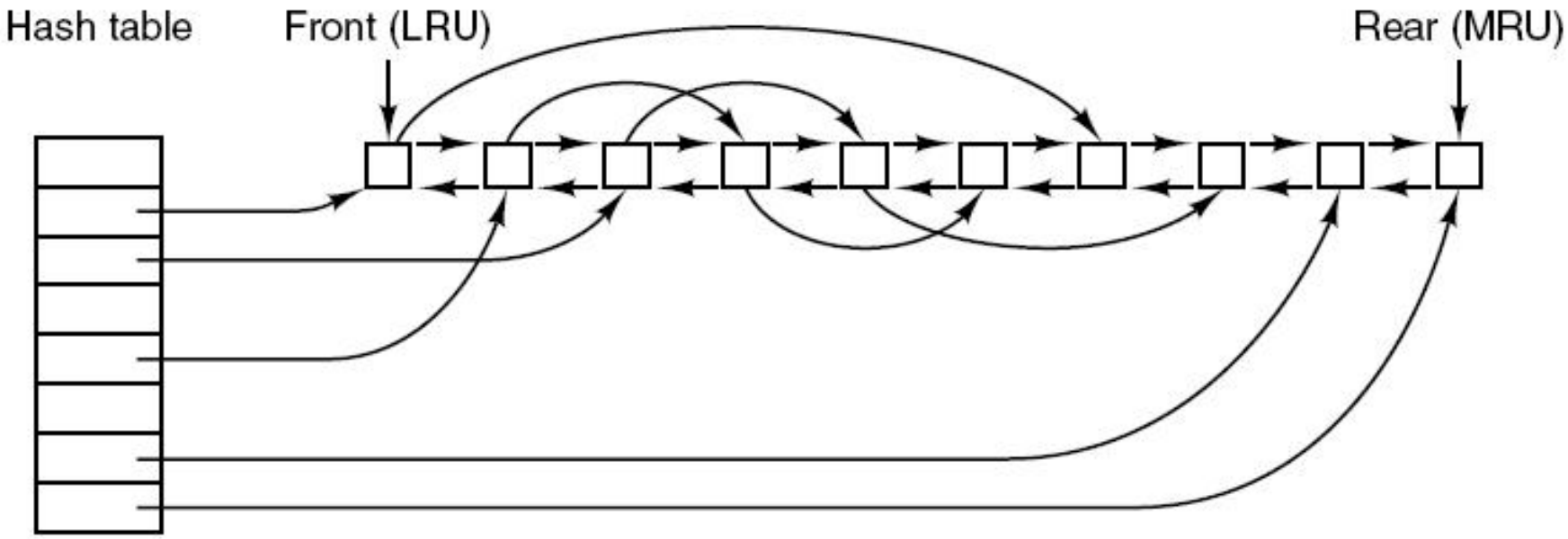


Figure 4-28. The buffer cache data structures.

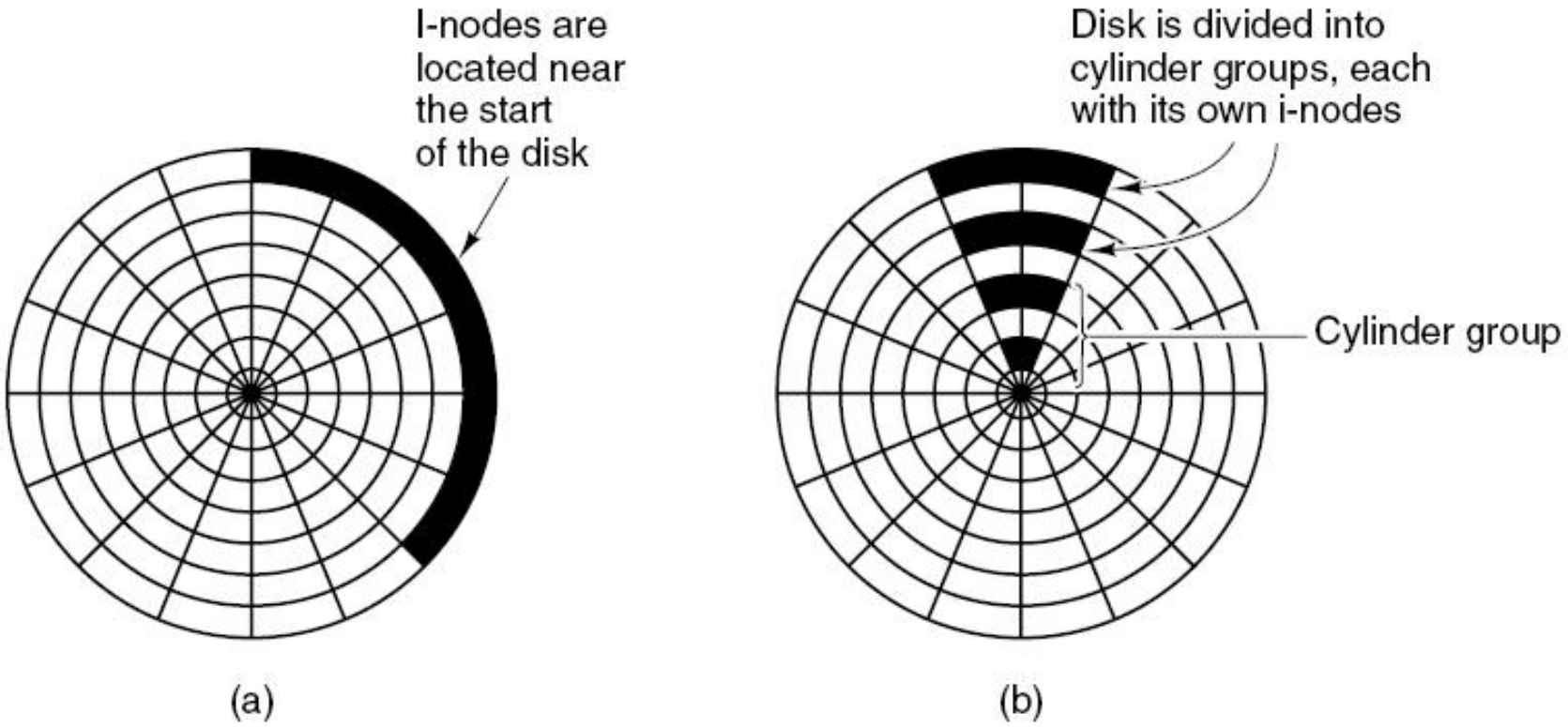


Figure 4-29. (a) I-nodes placed at the start of the disk.
(b) Disk divided into cylinder groups, each with its own blocks and i-nodes.

4.5 Example File Systems

- **CD-ROM file system**
- **ISO file system**
  - Rock Ridge extension fields:
    - PX - POSIX attributes
    - PN - Major and minor device numbers
    - SL - Symbolic link
    - NM - Alternative name
    - CL - Child location
    - PL - Parent location
    - RE – Relocation
    - TF - Time stamps
  - Joliet Extensions fields:
    - Long file names
    - Unicode character set
    - Directory nesting deeper than eight levels
    - Directory names with extensions

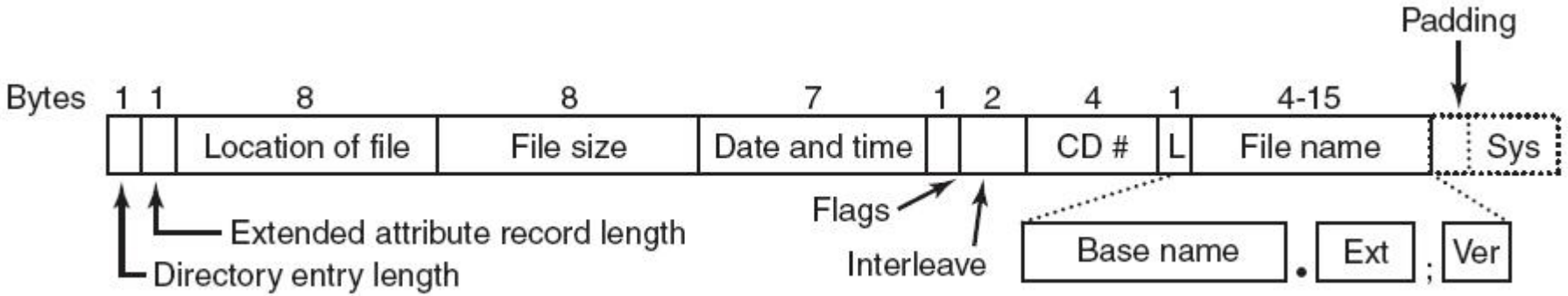


Figure 4-30. The ISO 9660 directory entry.

4.5 Example File Systems

- **MS-DOS file system**

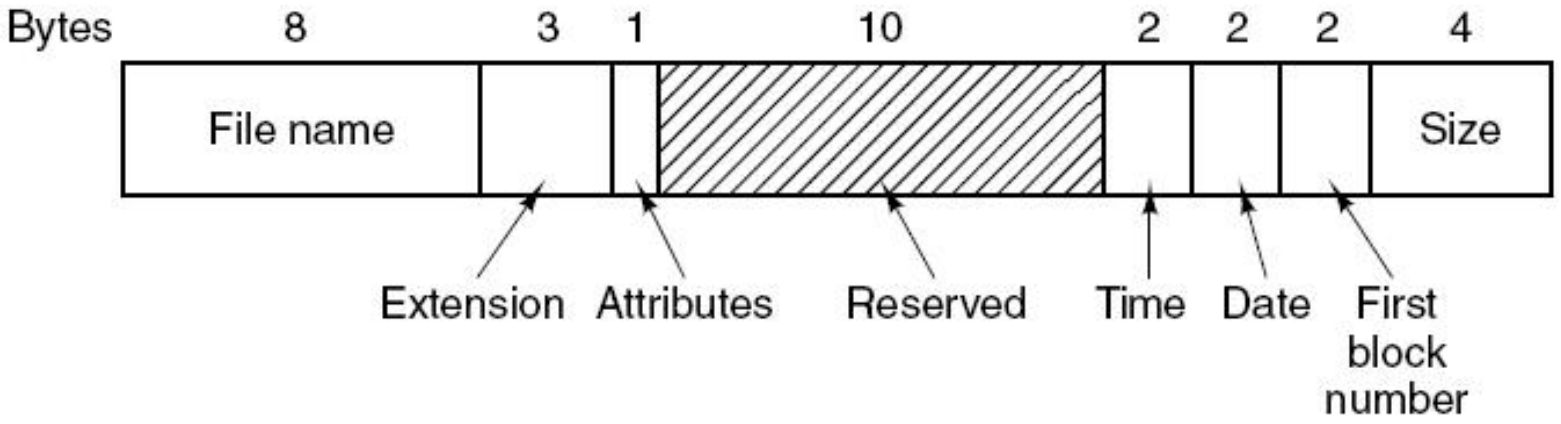


Figure 4-31. The MS-DOS directory entry.

4.5 Example File Systems

- **MS-DOS file system**

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

Figure 4-32. Maximum partition size for different block sizes. The empty boxes represent forbidden combinations.

4.5 Example File Systems

- **UNIX V7 File System**

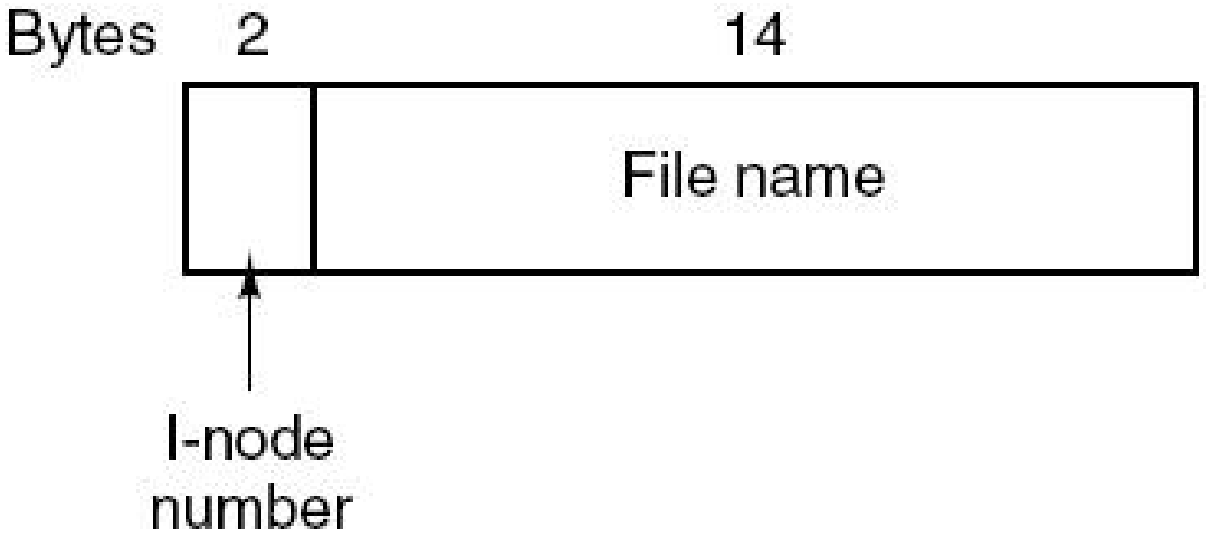


Figure 4-33. A UNIX V7 directory entry.

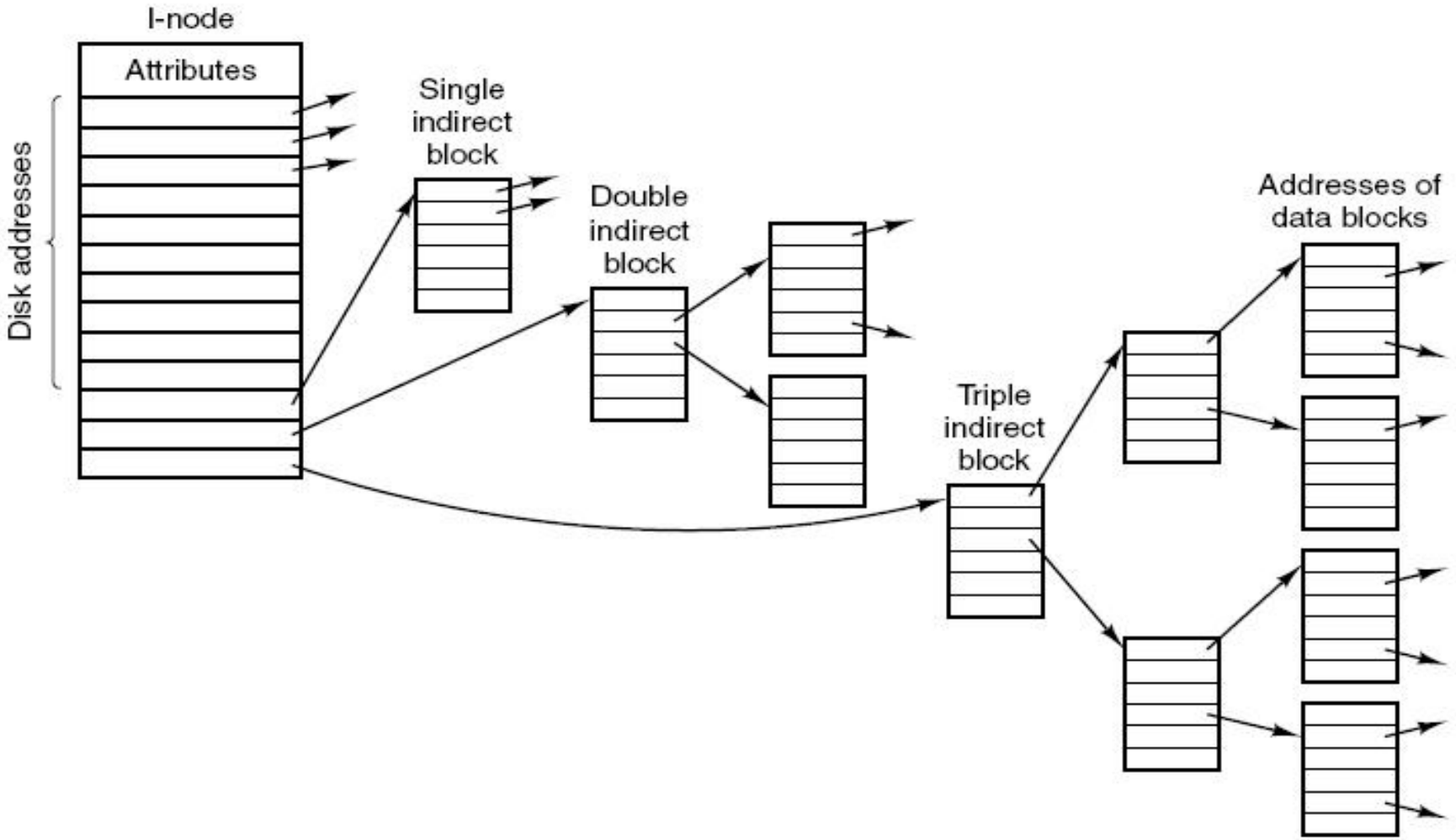


Figure 4-34. A UNIX i-node.

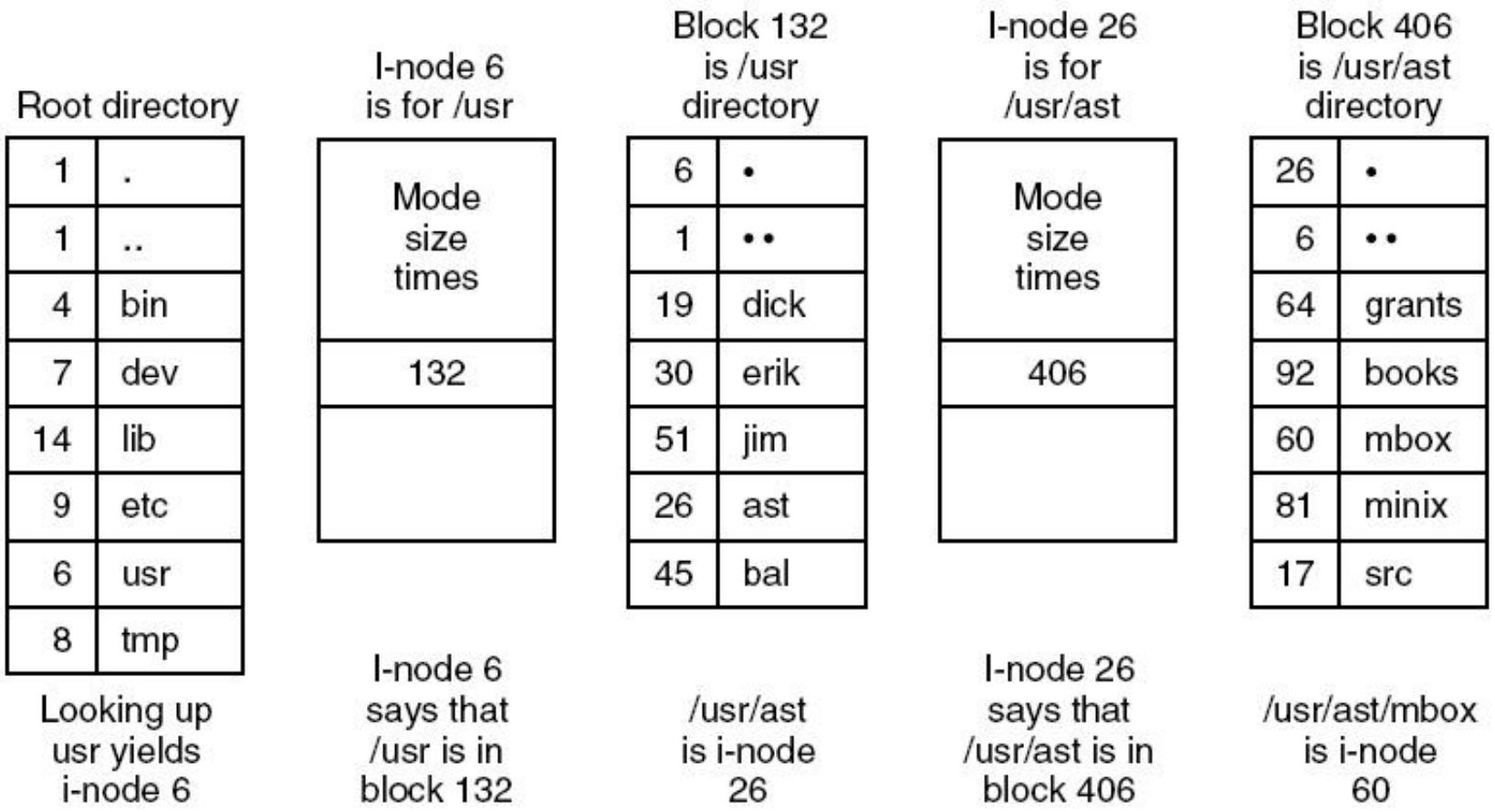


Figure 4-35. The steps in looking up */usr/ast/mbox*.

Reading Materials

- Abhirup Chakraborty · Ajit Singh. cost-aware caching schemes in heterogeneous storage systems. J Supercomputer, 2011, 56: 56-78
- Yongseok Oh, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. Proceedings of the 10th USENIX conference on File and Storage Technologies, p.25-25, February 14-17, 2012, San Jose, CA

谢 谢！