Operating Systems
File System

dr. Tomasz Jordan Kruk

T.Kruk@ia.pw.edu.pl

Institute of Control & Computation Engineering
Warsaw University of Technology

File System

File management:

- √ 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.

- √ **field**, basic data unit, contains single value characterized by a size and a type,
- √ **record**, collection of related to each other fields treated as a whole,
- √ **file**, collection of similar records treated as a whole, identified by a unique name, with an access restricted by given access rights.

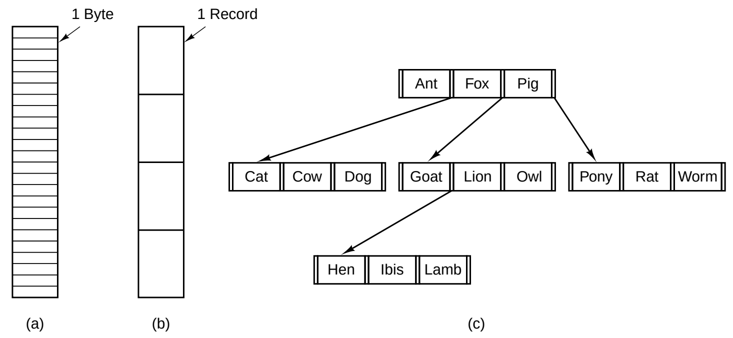
File System Functions

- √ identification and location of files,
- √ usage of directories,
- √ user to files access control,
- √ blocking of files during access to files,
- √ free blocks allocation management,
- √ free blocks space management.

Criteria of files organization:

- √ access performance,
- √ flexibility,
- √ storage efficiency,
- √ manageability,
- √ fault tolerance.

Types of File Structure



Three kinds of files:

a. byte sequence,
b. record sequence,
c. tree.

Possible 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 has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File Operations (I)

```
/* 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 */
```

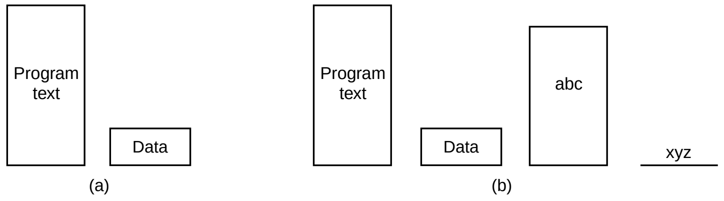
File Operations (II)

```
    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 */
}
```

Memory-Mapped Files



Process segments:

a. A segmented process before mapping files into its address space.

b. The process after mapping an existing file *abc* into one segment and creating a new segment for file *xyz.*

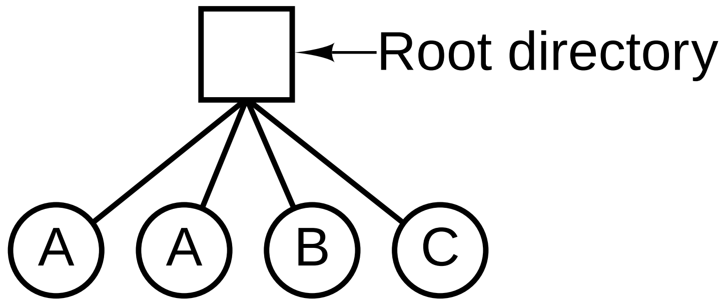
Directories

- √ directories contain information about files: attributes, address, owner information,
- √ directory may be a file itself (like under Unix),
- √ the main purpose of directories is to keep and enable translation between files and their names.

Structures of directories organization:

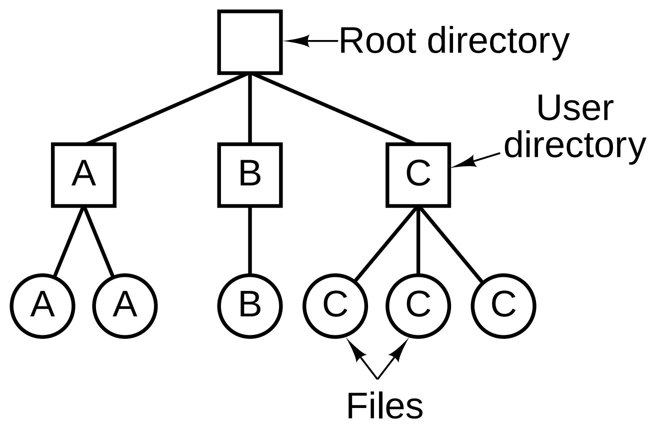
- √ single-level structure,
- √ two-levels directory systems (one directory per each user),
- √ hierarchical directory systems.
  - ★ files are identified by paths,
  - ★ it is possible to have more than one name for the same file,
  - ★ current directory (working directory) idea, absolute and relative path names.

Single-level Directory Systems

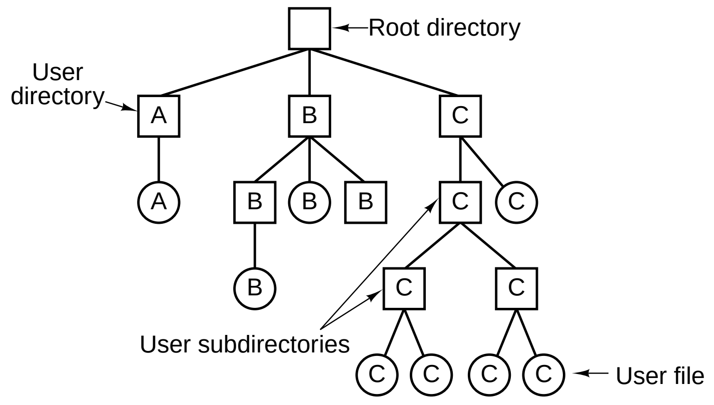


A single-level directory system containing four files, owned by three different people, A, B and C.

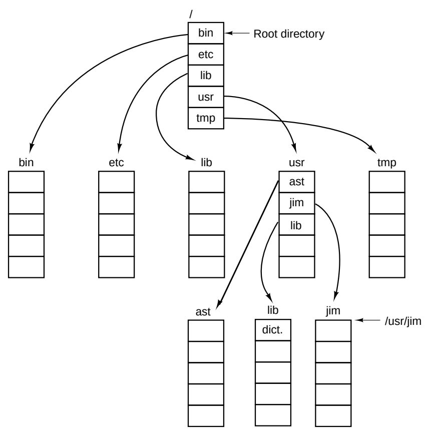
Two-level Directory Systems



Hierarchical Directory Systems



A Unix Directory Tree

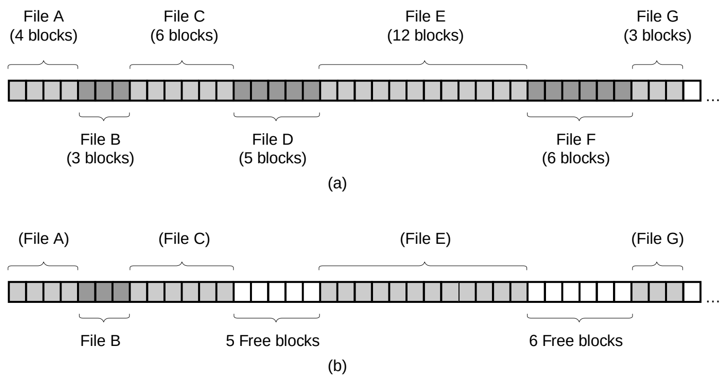


Disk Space Management

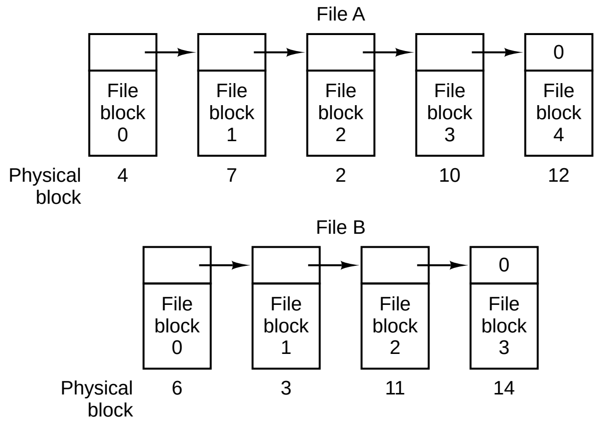
Methods of file allocation:

- √ contiguous allocation,
  - ★ FAT entry = name, start block, size,
- √ linked list allocation,
  - ★ FAT entry = name, start block, size,
  - ★ in each block field with the reference to the next data block.
- √ i-nodes (index nodes),
  - ★ FAT entry = name, reference to the block with indexes,
  - ★ i-node block contains references to data blocks.
  - ★ possible extensions with introduction of areas with local continuity, i-node block entry would have reference to data block and the count of blocks located there,

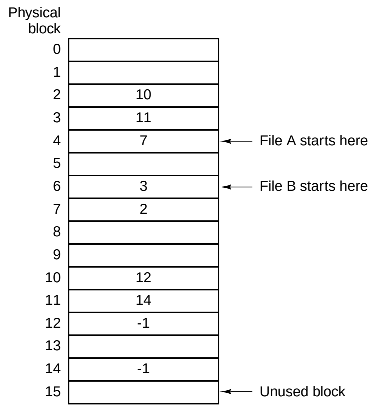
Contiguos Allocation



Linked List Allocation (I)

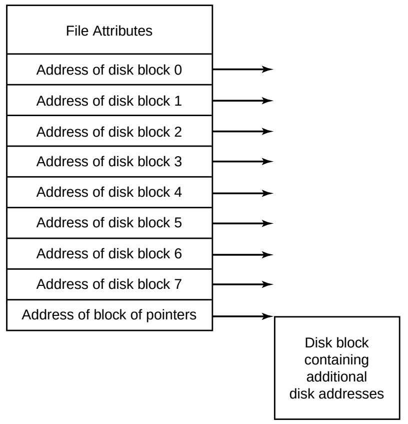


Linked List Allocation (II)

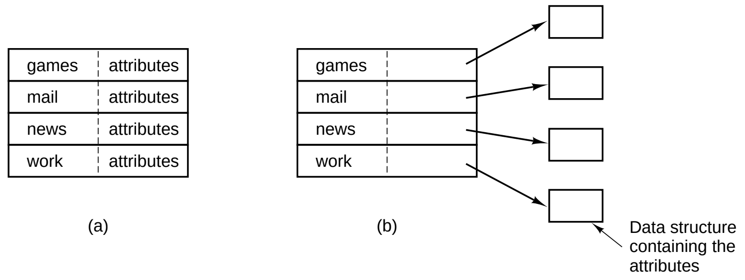


Linked list allocation using a file allocation table in main memory.

An Example i-node



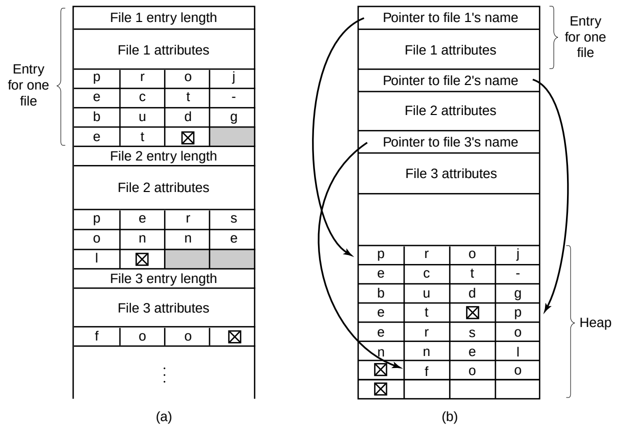
Implementing Directories



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.

Long File Names Handling

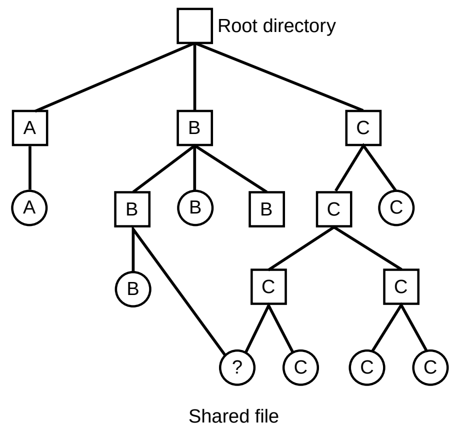


Two ways of handling long file names in a directory:

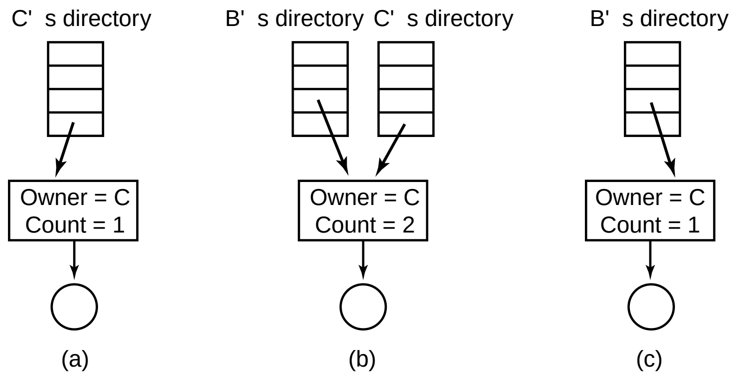
a. in-line,

b. in a heap.

File System Containing Shared Files



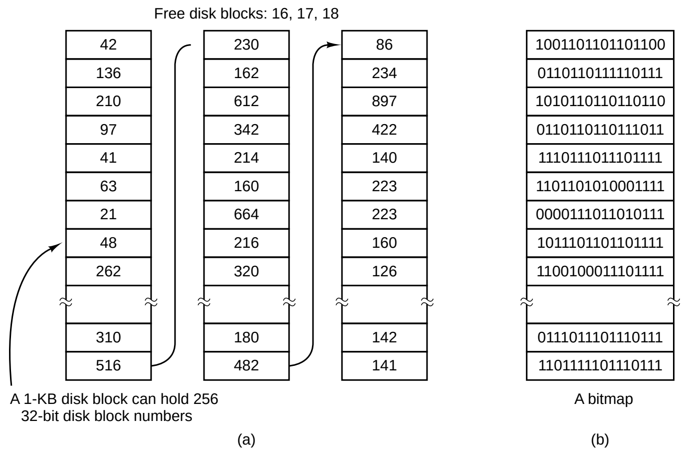
Directed Acyclic Graphs



Symbolic linking

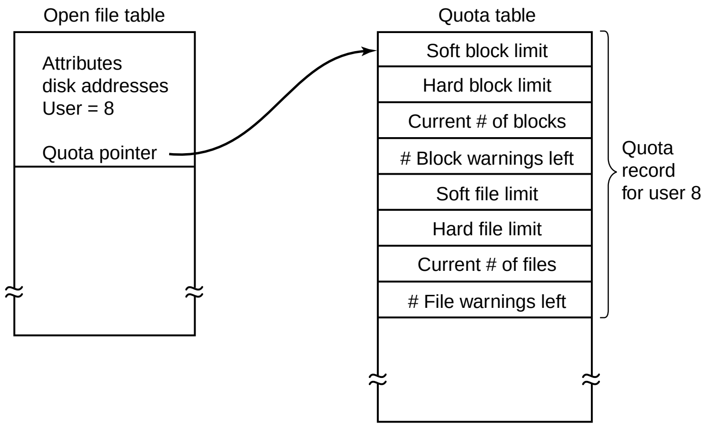
a. situation prior to linking,
b. after the link is created,
c. after the original owner removes the file.

Free Disk Space Management



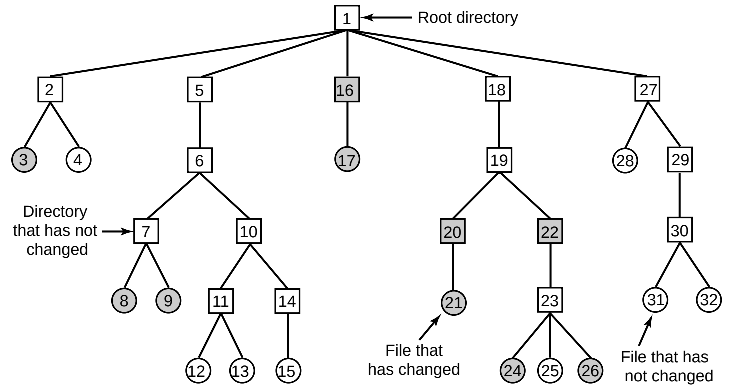
a. storing the free list on a linked list,
b. a bitmap.

Disk Quota



Quotas controlled on a per-user basis in a quota table.

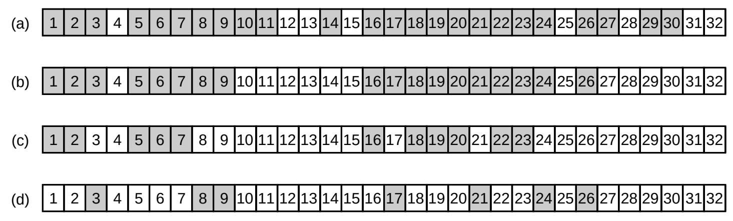
Disk Backups



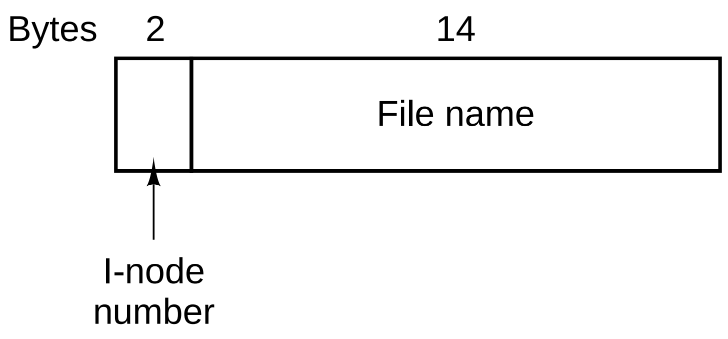
A file system to be dumped.

- √ each file and directory labeled by its i-node number,
- √ the shaded items has been modified since the last dump.

Usage of Bitmaps for Backup

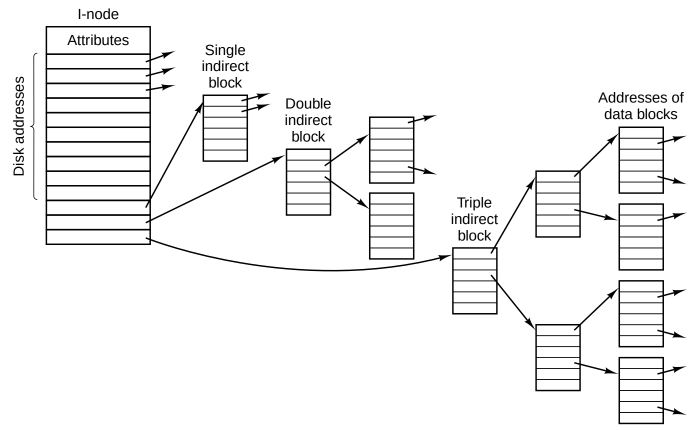


The Unix V7 File System (I)



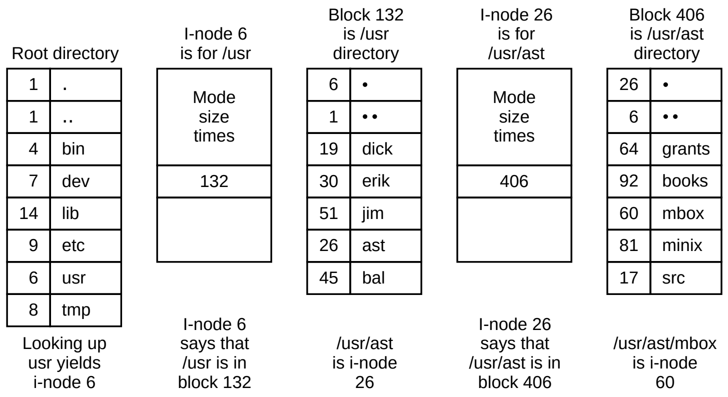
A Unix V7 directory entry.

The Unix V7 File System (II)



A Unix i-node.

The Unix V7 File System (III)


Root directory
1 .
1 ..
4 bin
7 dev
14 lib
9 etc
6 usr
8 tmp
Looking up usr yields i-node 6
I-node 6 is for /usr
Mode size times
132
I-node 6 says that /usr is in block 132
Block 132 is /usr directory
6 •
1 ••
19 dick
30 erik
51 jim
26 ast
45 bal
/usr/ast is i-node 26
I-node 26 is for /usr/ast
Mode size times
406
I-node 26 says that /usr/ast is in block 406
Block 406 is /usr/ast directory
26 •
6 ••
64 grants
92 books
60 mbox
81 minix
17 src
/usr/ast/mbox is i-node 60

The steps in looking up *usr/ast/mbox*.

The Unix V7 File System (IV)

Under Unix the file system is usually written to the disk in four separate sections:

- √ block 0 (**boot block**), used for booting the operating system,
- √ block 1, (**superblock**), contains information about the structure of the file system,
- √ blocks 2 – m, (**i-nodes**), **i-nodes** lists,
- √ blocks m+1 – n, data blocks.

I-nodes and special files:

- √ in case of special files in i-nodes, instead of the first data pointer, number of device driver handling procedure is written,
- √ that number consists of two parts: **major** number and **minor** number,
- √ special file may be created with the **mknod** command.

The Unix V7 File System (V)

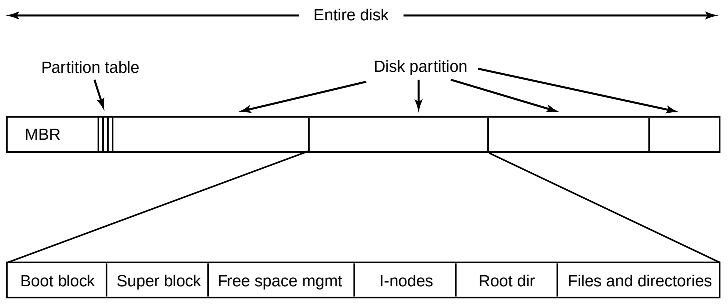
Superblock contains:

- √ size in blocks of i-nodes list,
- √ sieze in blocks of the file system,
- √ number of free blocks in a file system,
- √ index of the next free block on the free blocks list,
- √ number of free i-nodes in the file system,
- √ index of the next free i-node on the free i-node lists,
- √ modification marker of the superblock,
- √ time of modification and the name of the file system.

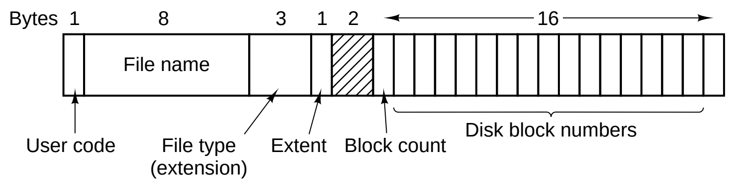
Modern operating systems may have additional features:

- √ many copies of the superblock,
- √ journaling,
- √ snapshot awareness.

Example File System Structure

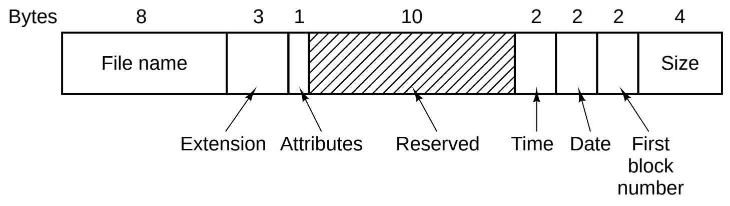


The CP/M File System



The CP/M directory entry format.

The MS-DOS File System (I)



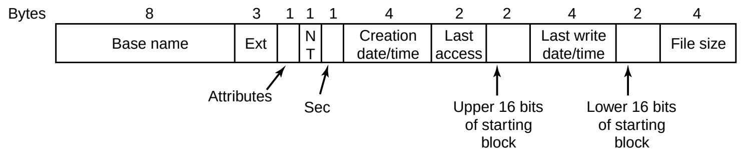
The MS-DOS directory entry.

The MS-DOS File System (II)

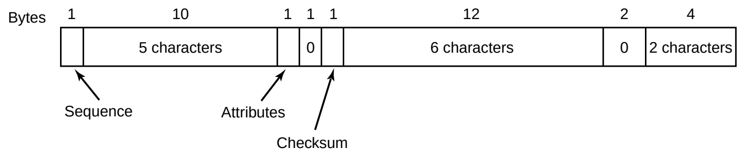
Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

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

The File System under Windows 98 (I)



The extended MS-DOS directory entry used in Windows 98.



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

The File System under Windows 98 (II)

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

Bytes

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