OS invented – processes (100s of it in my computer)
– address spaces
– files

File Systems

Chapter 4

For me a file is permanent storage place where I can address it from byte number 0 to large byte number.
Appen, modify, update is handled by OS.

Authors:
Seyda Ozer and Süleyman Göl bal

File Systems (1)

Essential requirements for long-term information storage:

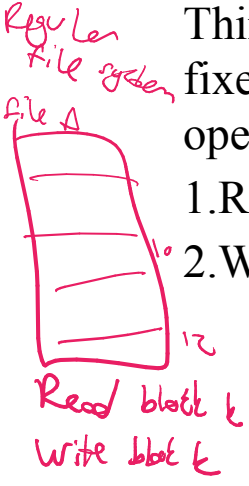
1. It must be possible to store a very large amount of information.
2. Information must survive termination of process using it.
3. Multiple processes must be able to access information concurrently.

A new abstraction solves the problem: the file.

File Systems

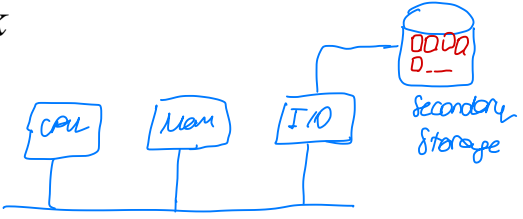
- operating system abstracted away the concept of the processor to create the abstraction of a process CH2
- It abstracted away the concept of physical memory to offer processes (virtual) address spaces, CH3
- **A new abstraction solves the problem: the file.**
- Together, the abstractions of processes (and threads), address spaces, and files are the most important concepts relating to operating systems.
- If you really understand these three concepts from beginning to end, you are well on your way to becoming an operating systems expert.

File Systems (2)



Think of a disk as a linear sequence of fixed-size blocks and supporting two operations:

1. Read block *k*.
2. Write block *k*



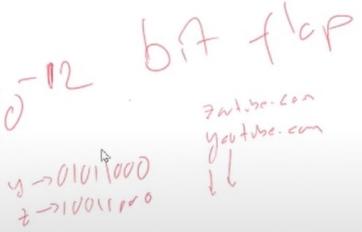
File Systems (3)

Questions that quickly arise:

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

Design Decisions

- File names
- Hierarchical or non-hierarchical: sturucture
- Types of access control: Read-only or read-write
- Versioning
- Fault recovery
- File operations
- Many more

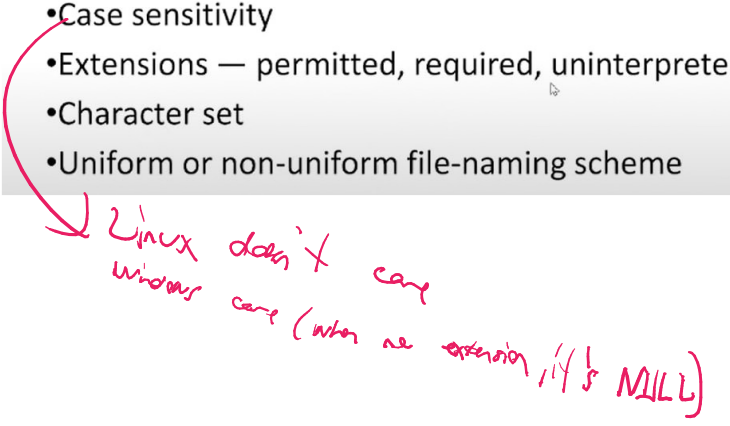


Files and File Names

- A file is an abstraction mechanism.
- It provides a way to store information on the disk and read it back later.

File names

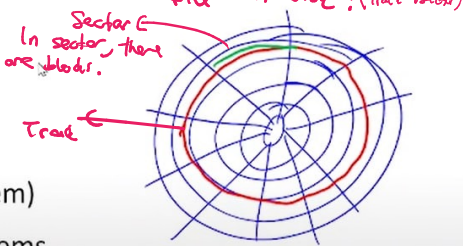
- Case sensitivity
- Extensions — permitted, required, uninterpreted, length
- Character set
- Uniform or non-uniform file-naming scheme



File systems

File systems: part of the operating system dealing with files is known as the file system and is the subject of this chapter.

- FAT-16 : MSDoS
- FAT-32: Windows 98-
- NTFS: Windows NT
- Windows 8 ReFS (or Resilient File System)
- exFAT for flash drives and large file systems



ext2 → linux file system

journaling file system

File Naming

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

Figure 4-1. Some typical file extensions.

File Structures

- Old systems: files were sequences of records, initially images of punch cards or printer lines
- Unix and windows: randomly-addressable sequences of bytes

- Page-oriented file systems: sequence of page-sized blocks
- Having the operating system regard files as nothing more than byte sequences provides the maximum amount of flexibility

80 columns of text

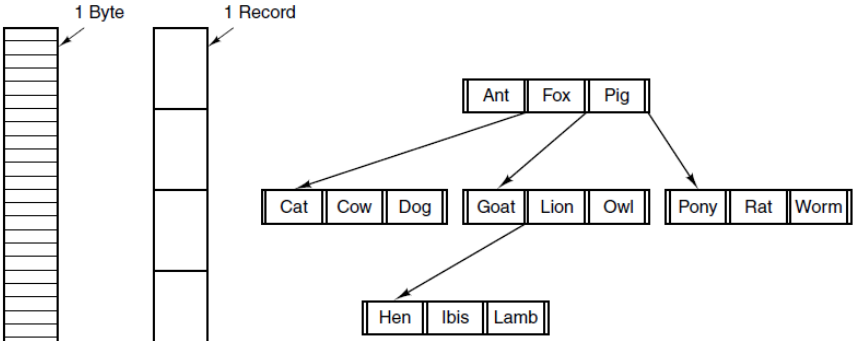
100.80

Each file is a segment,

Each file consists of record (collection of bytes)

Usually length of the record kept inside file metadata.

File Structure



For linux byte sequence

Collection of records

Collection of records with B tree

Advantage
1. search is fast
2. insert/modify is easier

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

File Types

- **Regular** files are the ones that contain user information.
- **Directories** are system files for maintaining the structure of the file system.
- **Character special** files are related to input/output and used to model serial I/O devices, such as terminals, printers, and networks.
- **Block special** files are used to model disks. In this chapter we will be primarily interested in regular files.

special file, keep another files.

for example a file (whatever you read from file, you read from keyboard) terminal

File Types

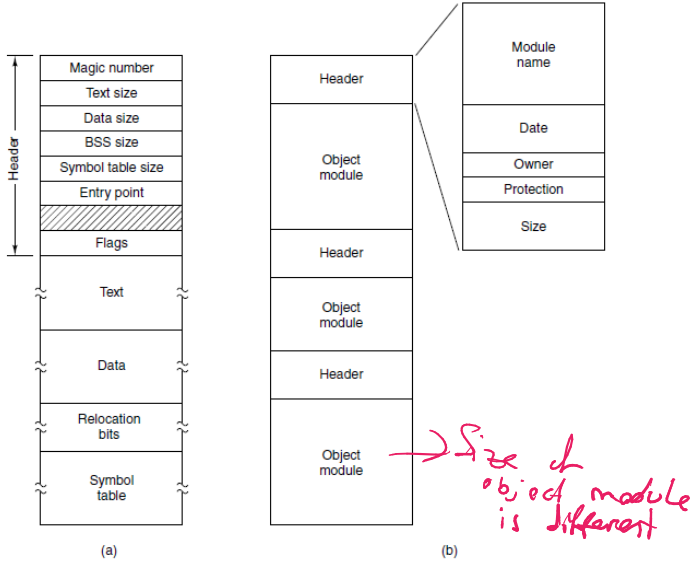


Figure 4-3. (a) An executable file. (b) An archive

File Access

- **Sequential access**. a process could read all the bytes or records in a file in order, starting at the beginning, but could not skip around and read them out of order. tape drive
- Files whose bytes or records can be read in any order are called **random-access files**. like array (Linux, windows)

File Attributes

- Files have names and data
- Other information with each file are called **attributes** = metadata
- For example, the date and time the file was last modified and the file's size.
- We will call these extra items the file's attributes.
- Some people call them metadata.

OS is responsible for keeping this information.

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

Figure 4-4. Some possible file attributes.

File Operations

Question
How did OS manage the segment in memory?

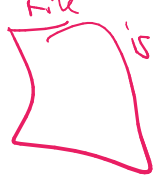
It's easy to grow segment no. 2 because there is space. But it'll be difficult to grow segment 1.
Solution: Page segments since it doesn't have to be contigious.

For each program you could have a data segment, code segment, single table, constants table, temporary heap

1.	Create	1.	Append
2.	Delete	2.	Seek
3.	Open	3.	Get attributes
4.	Close	4.	Set attributes
5.	Read	5.	Rename
6.	Write		

Because we need to load some info like current index, Load lots of attribute info

why? why?



a sequence of bytes linear address space on secondary device (a segment on the disk)

Example Program Using File System Calls (1)

```
/* 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 */
```

Figure 4-5. A simple program to copy a file.

Example Program Using File System Calls (2)

```
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 */
```

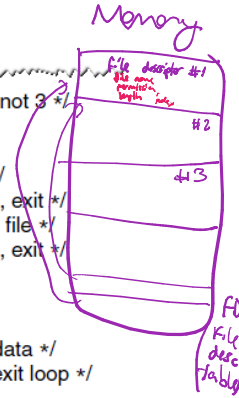


Figure 4-5. A simple program to copy a file.

Example Program Using File System Calls (3)

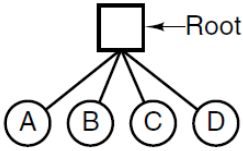
```
    /* Copy loop */
    while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
    if (rd_count <= 0) break;                     /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count); /* write data */
        if (wt_count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

Figure 4-5. A simple program to copy a file.

Directories

•To keep track of files, file systems normally have **directories or folders**, which are themselves files.

Single-Level Directory Systems

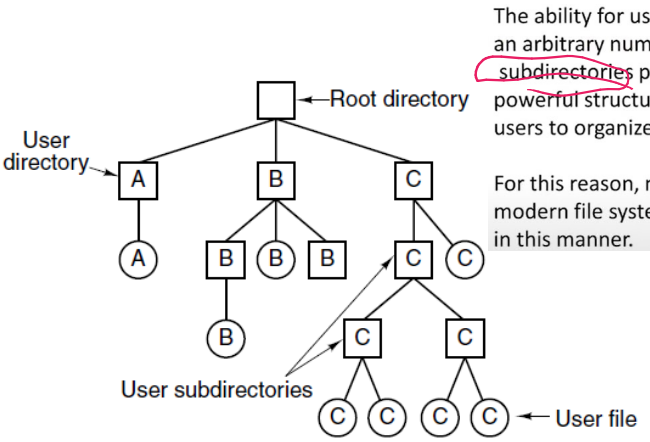


Sometimes it is called the root directory, but since it is the only one, the name does not matter much

On early personal computers, this system was common, in part because there was only one user.

Figure 4-6. A single-level directory system containing four files.

Hierarchical Directory Systems



The ability for users to create an arbitrary number of subdirectories provides a powerful structuring tool for users to organize their work.

For this reason, nearly all modern file systems are organized in this manner.

Figure 4-7. A hierarchical directory system.

Path Names

Two different methods are commonly used.

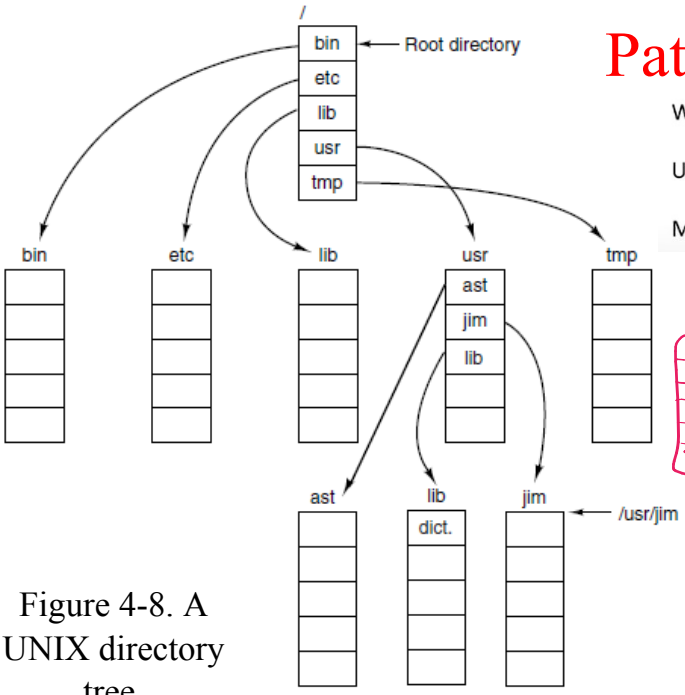
- In the first method, each file is given an **absolute** path name consisting of the path from the root directory to the file.
- The other kind of name is the **relative** path name.
- This is used in conjunction with the concept of the working directory (also called the current directory).
- A user can designate one directory as the current working directory, in which case all path names not beginning at the root directory are taken relative to the working directory.
- two special entries in every directory, "." and "..", generally pronounced "dot" and "dotdot."

Path Names

Windows \usr\ast\mailbox

UNIX /usr/ast/mailbox

MULTICS >usr>ast>mailbox



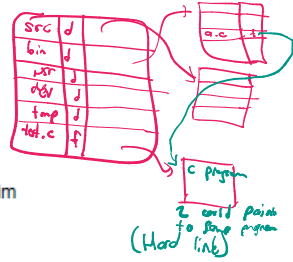
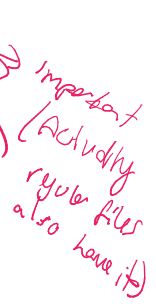


Figure 4-8. A UNIX directory tree.

Directory Operations

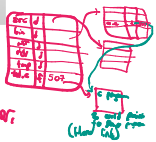
1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink



Directory operations

Linking is a technique that allows a file to appear in more than one directory

- **Hard link: C**reates a link from the existing file to the name specified by the path. instead of copying stuff, we use the OS index to point to the same ds, my file.
- **Symbolic link.** Instead, of having two names point to the same internal data structure representing a file, a name can be created that points to a tiny file naming another file.
- Symbolic links have the advantage that they can cross disk boundaries and even name files on remote computers.



We cannot use hard linking between 2 file systems.
One of the files. These 2 numbers are not valid for each other.
So there is soft linking option.

- If I remove from one place, the other still there. (Hard link)
- Symbolic link: It has performance issue. You open read first to find address of other files to open it.

File system Implementation

- Users are concerned with
  - –how files are named,
  - –what operations are allowed on them,
  - –what the directory tree looks like, and similar interface issues.
- Implementors are interested in
  - –how files and directories are stored,
  - –how disk space is managed, and
  - –how to make everything work efficiently and reliably.

File system Implementation

When computer boots up, sector 0 is the place where I need original OS. The BIOS (Basic IO System) reads this. The Apple OS

- Sector 0 of the disk is called the MBR (**Master Boot Record**) and is used to boot the computer.
- The end of the MBR contains the partition table.
- This table gives the starting and ending addresses of each partition.
- One of the partitions in the table is marked as active.
- When the computer is booted, the BIOS reads in and executes the MBR.
- The first thing the MBR program does is locate the active partition, read in its first block, which is called the **boot block**, and execute it.

File System Layout

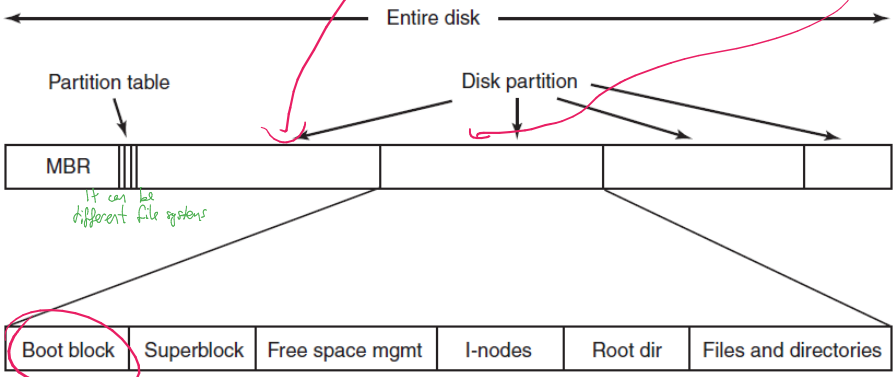
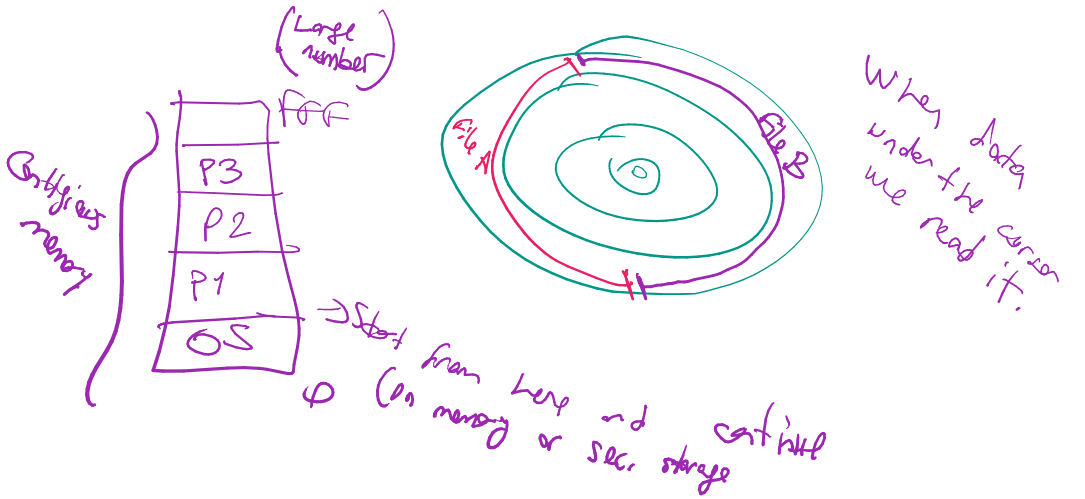


Figure 4-9. A possible file system layout.

SUPERBLOCK

File system Implementation

- **Superblock** contains all the key parameters about the file system and is read into memory when the computer is booted or the file system is first touched. .



Implementing Files: Contiguous Layout

- Each file occupies a set of contiguous blocks on the disk
- Advantage:
  - –Simple – only starting location (block #) and length (number of blocks) are required
  - –Fast sequential; also quite fast random access
- Disadvantage:
  - –External fragmentation
  - –Inflexible when appending to a file
  - However, contiguous allocation is feasible and, in fact, still used: on ***CD-ROMs.***

Perfect for CD-ROM (Bc read only, we are not gonna write again)

ISO 9660

Implementing Files
Contiguous Layout

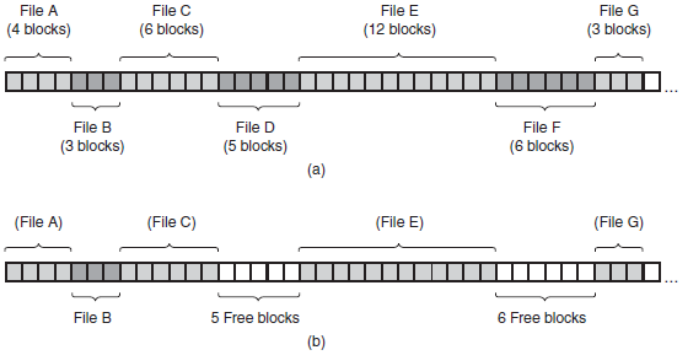
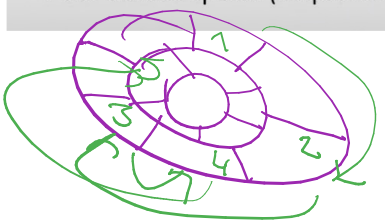


Figure 4-10. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Linked File Allocation

Solve Problem of contiguous file. (To fit)

- Each file is a linked list of disk blocks
  - –each block contains a next pointer
  - –directory only needs to store the pointer to the first block
  - –blocks may be scattered anywhere on the disk
- Advantage
  - –Space efficient
  - –Flexible in appending
- Disadvantage:
  - –Poor access speed (sequential & random)



Slow a lot of movement in cursor.

Implementing Files
Linked List Allocation

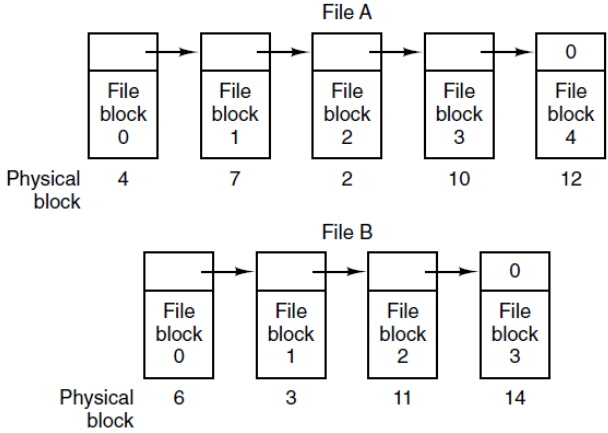
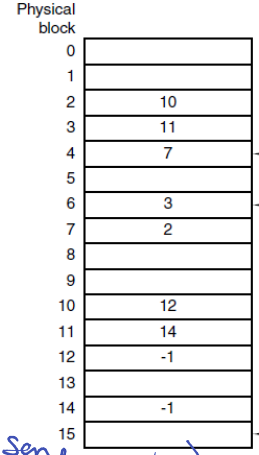


Figure 4-11. Storing a file as a linked list of disk blocks.

Implementing Files
Linked List – Table in Memory



The primary disadvantage of this method is . . . . . if we are putting it in main memory, their resources for processes will be less. It will create page faults a lot.

FAT approach (File Allocation Table)

7 2 10 12

(2 7 10 12

change order in seq trace (make sequential faster) → OPTIMIZATION

We don't have this idea in page table since it's a lot to scan page table.

Figure 4-12. Linked list allocation using a file allocation table in main memory.

Linked List Allocation Using a Table in Memory

- FAT
- Advantage
  - –Advantages of linked-list…
  - –No disk access
- Disadvantage
  - –Still need to traverse
  - –Lot's of memory required!
- Memory
  - –200GB disk with 1KB blocks
  - –Memory required?

200GB, 2^{30} → 200GB, 2^{20}

FAT-16 FAT-32

→ by No recovery but we know so still using.

Some people said it's limiting (for large systems as linux) so they took indexing idea from memory. (We don't have load all table for sca storage, we can keep only for file is open.)

INODE Indexed Allocation → Idea from money

- Brings all pointers together into the index block (i-node = index node)
- Just keep all attributes and addresses of each block belonging to the file in this structure
  - Keep in memory only when the file is open…
- Advantages:
  - Space efficiency
    - no external fragmentation
    - overhead of index blocks
  - Access speed
    - random access
    - sequential access

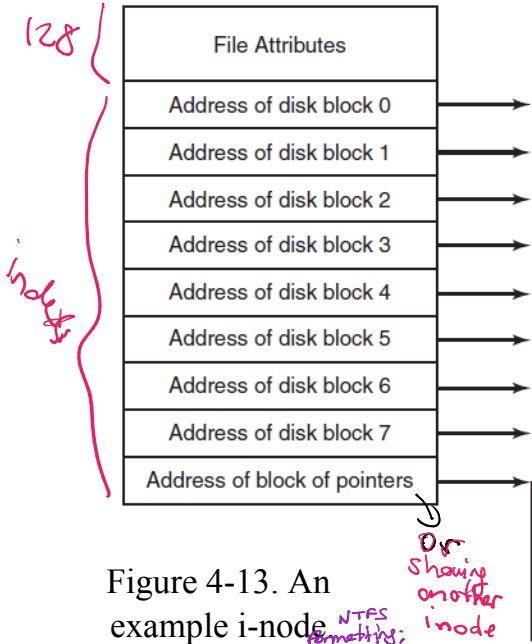


Figure 4-13. An example i-node.

Implementing Directories (1)

- The directory entry provides the information needed to find the disk blocks.
- Depending on the system,
  - –this information may be the disk address of the entire file (with contiguous allocation),
  - –the number of the first block (both linked- list schemes),
  - – or the number of the i-node.
- In all cases, the main function of the directory system is to map the ASCII name of the file onto the information needed to locate the data.

Implementing Directories (1)

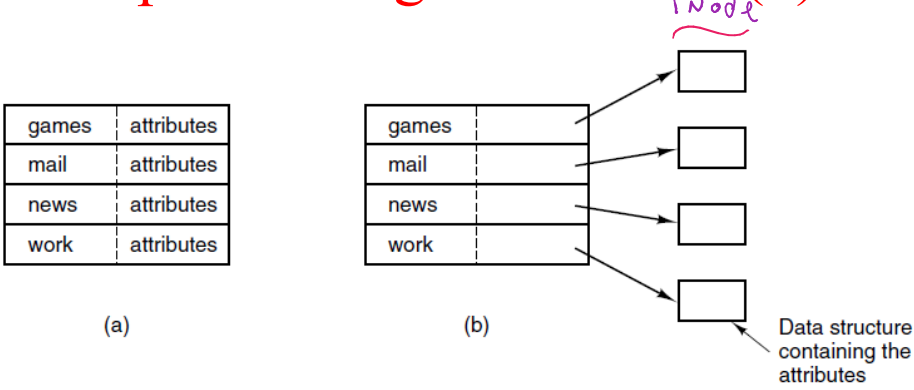


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

Implementing Directories (2)

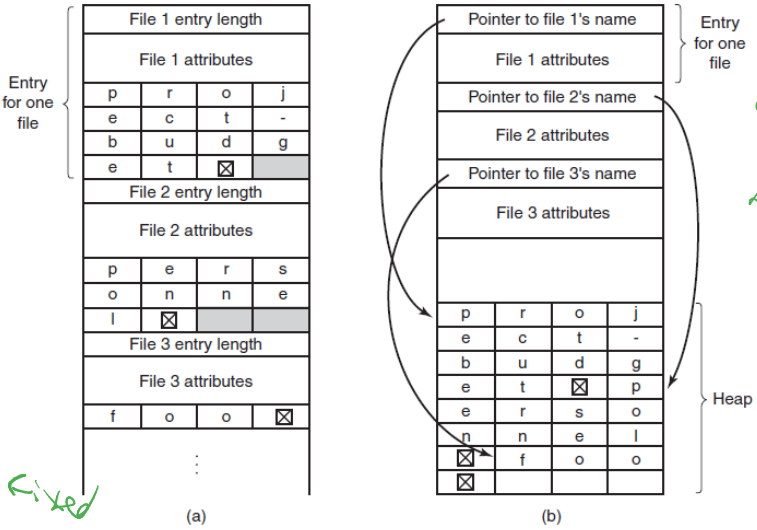


Figure 4-15. Tw o ways of handling long file names in a directory. (a) In-line. (b) In a heap.

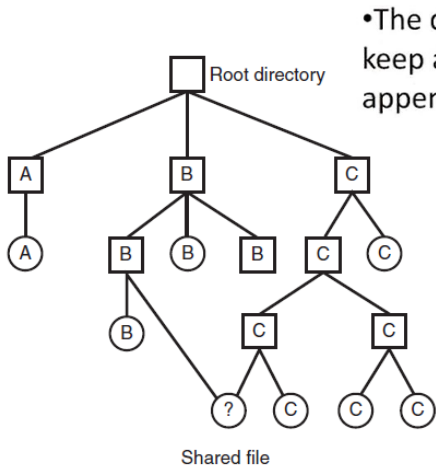
Implementing Directories

- For extremely long directories, linear searching can be slow.
- One way to speed up the search is to use a hash table in each directory.
- A different way to speed up searching large directories is to cache the results of searches.

look hash index in hash table

Add, remove, search: O(1)

Shared Files (1)



- The directories should not keep a block list, otherwise appending is a problem.

Figure 4-16. File system containing a shared file.

Shared Files (2)

- The directories should not keep a block list, otherwise appending is a problem.
- i-node solution solves the problem but ownership becomes a problem now!
- Ownership problem can be addressed using symbolic links but symbolic links need more disk access and more i-nodes

Shared Files (2)

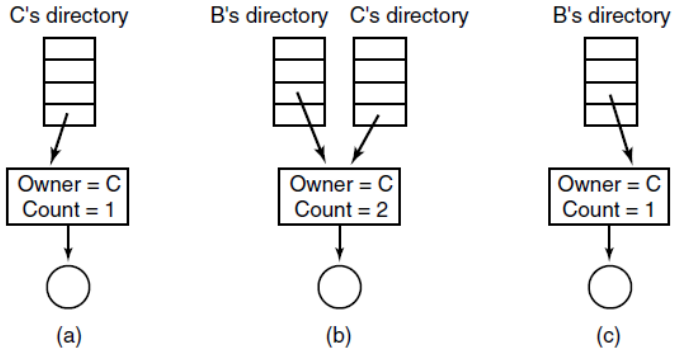


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

First write other place.

Log File Systems

- The idea that drove the LFS design is that as CPUs get faster and RAM memories get larger, disk caches are also increasing rapidly.
- A very substantial fraction of all read requests directly from the file-system cache, with no disk access needed.
- In the future, most disk accesses will be writes, so the read-ahead mechanism used in some file systems to fetch blocks.
- All writes are initially buffered in memory, and periodically all the buffered writes are written to the disk as a single segment!

On mechanical hard drives, they use big buffers. They read from disk to disk buffer. (for ex: 1 GB)
Usually you read from cache of buffer of disk.
When you write, it's harder.

Journaling File Systems

- Keep a journal of what the file system is going to do before it does it
- 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.
- Such file systems, called j**ournaling file systems**, are actually in use.
- Microsoft's NTFS file system and the Linux ext3 and ReiserFS file systems all use journaling.

Journaling File Systems

In Linux metadata (file attributes) is in i-node

Every file should have it's inode.

But you don't keep INodes in memory.

Steps to remove a file in UNIX:

1. Remove file from its directory.
2. Release i-node to the pool of free i-nodes.
3. Return all disk blocks to pool of free disk blocks.

Take inode number from that directory.

What happens if the systems crashes between steps?

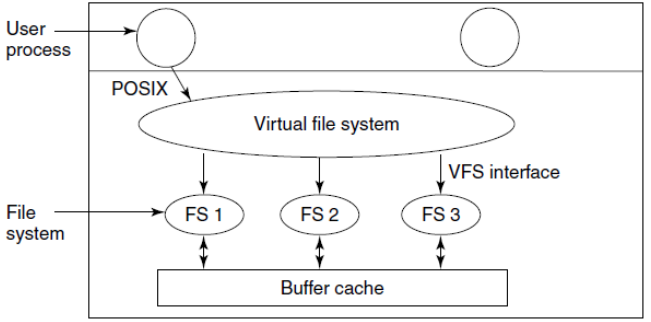
✓ ✓ ✗ → undo or finish unfinished by looking journal.

Virtual File Systems (2)

- Many different file systems are in use—often on the same computer—even for the same operating system
- Windows assigns a different drive letter for each FS: C,D, etc.
- All modern UNIX systems make a very serious attempt to integrate multiple file systems into a single structure.

I shouldn't deal with FAT based, Inode based etc.

Virtual File Systems (1)



- VFS has 2 interfaces, POSIX and lower interface

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

Virtual File Systems (2)

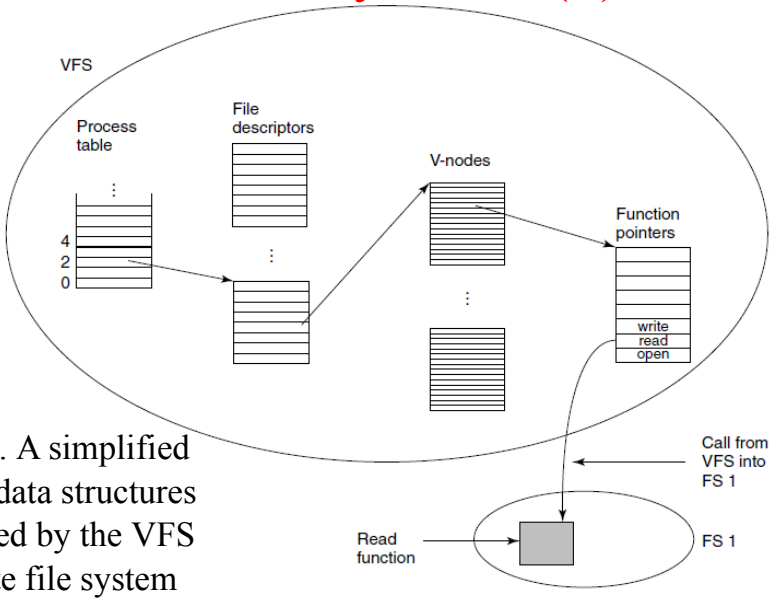


Figure 4-19. A simplified view of the data structures and code used by the VFS and concrete file system to do a *read*.

Disk Space Management- Block Size

- **Block Size:** The question arises how big the block should be.
- If the allocation unit is too large, we waste space (internal fragmentation); if it is too small, we waste time and space.

- . Two general strategies are possible for storing an n byte file:
  - n consecutive bytes of disk space are allocated, whatever n is;
  - or the file is split up into a number of (not necessarily) contiguous blocks
- The same trade-off is present in memory-management systems between pure segmentation and paging.

If I have 2^{10} blocks of all of my system each block has an address of 10 bits.
⇒ If a lot, it might be unuseful.

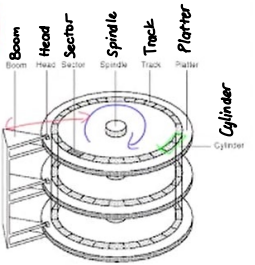
Disk Space Management (1)

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

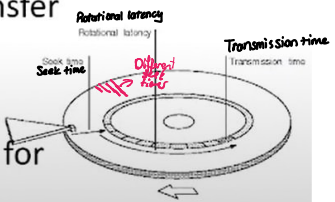
Figure 4-20. Percentage of files smaller than a given size (in bytes).

Disk Space Management (1)



5400
10000

- consider a disk with 1 MB(2^20) per track,
- a rotation time of 8.33 msec,
- an average seek time of 5 msec.
- The time in milliseconds to read a block of k bytes is then the sum of the seek, rotational delay, and transfer times:
- 5 + 4. 165 + (k/2^20) × 8. 33
- The dashed curve of Fig. 4-21 shows the data rate for such a disk as a function of block size.



Rotation is always fixed

We use average because seek changes.

Disk Space Management (2)

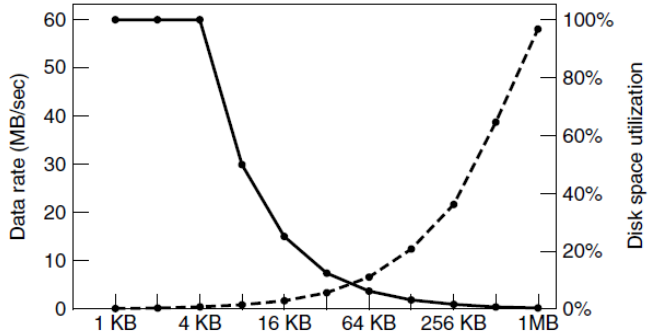


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

Keeping Track of Free Blocks (1)

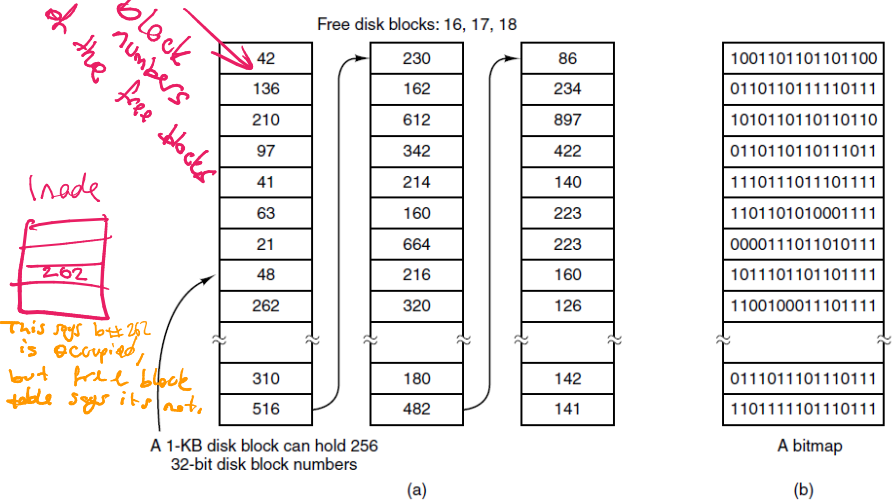


Figure 4-22. (a) Storing the free list on a linked list. (b) A bitmap.

Keeping Track of Free Blocks (2)

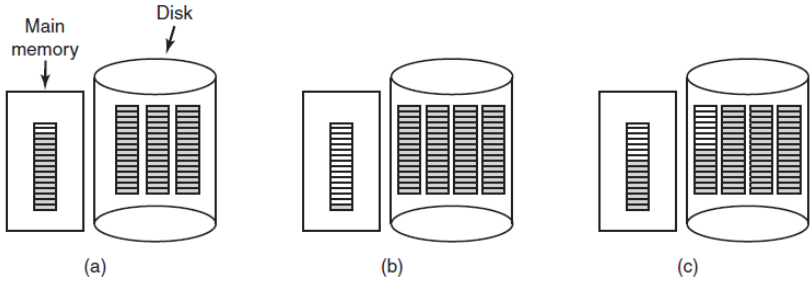


Figure 4-23. (a) An almost-full block of pointers to free disk blocks in memory and three blocks of pointers on disk. (b) Result of freeing a three-block file. (c) An alternative strategy for handling the three free blocks. The shaded entries represent pointers to free disk blocks.

Disk Quotas

- When a user opens a file, the attributes and disk addresses are located and put into an open-file table in main memory
- A second table contains the quota record for every user with a currently open file, even if the file was opened by someone else.
- The soft limit may be exceeded, but the hard limit may not. An attempt to append to a file when the hard block limit has been reached will result in an error.
- Analogous checks also exist for the number of files to prevent a user from hogging all the i-nodes.

Disk Quotas

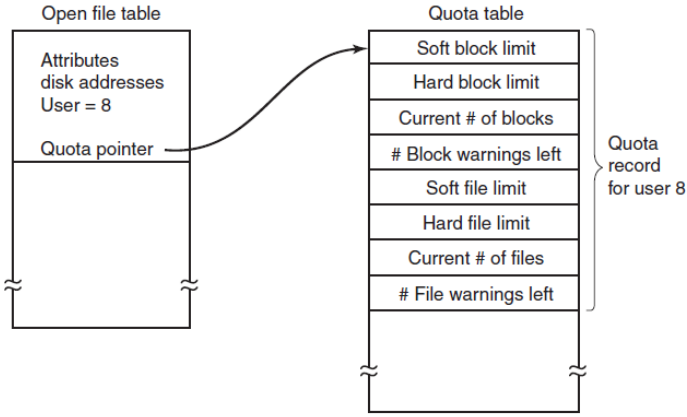


Figure 4-24. Quotas are kept track of on a per-user basis in a quota table.

File System Backups (1)

- No need to backup all the files (tmp, /dev files)
- Unchanged files need not be backedup (incremental backup)
- Compress during backup
- Difficult to backup an active system (take snapshots)
- Security problems with backups

only backup modified and append to previous

Physically copy all disk info to other disk.

File System Backups (1)

- A **physical dump** starts at block 0 of the disk, writes all the disk blocks onto the output disk in order

–no value in backing up unused disk blocks: need to write the block number

–bad blocks are always present: create a ''file'' consisting of all the bad blocks

–Physical dums are fast but are unable to skip selected directories, make incremental dumps, and restore individual files upon request.

File System Backups (1)

- A **logical dump** starts at one or more specified directories and recursively dumps all files and directories found there that have changed since some given base date
- dumps all directories (even unmodified ones) that lie on the path to a modified file or directory for two reasons.
  - –to transport entire file systems between computers.
  - –to make it possible to incrementally restore a single file

Take backup of all directories.

File System Backups (1)

Backups to tape are generally made to handle one of two potential problems:

1. Recover from disaster.
2. Recover from stupidity.

File System Backups (2)

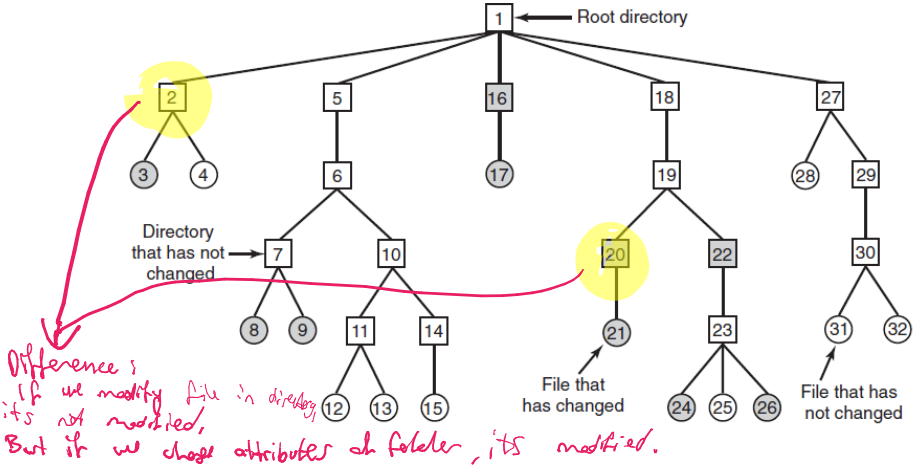
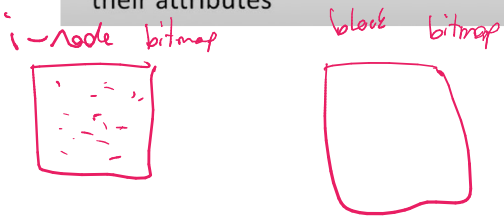


Figure 4-25. A file system to be dumped. The squares are directories and the circles are files. The shaded items have been modified since the last dump. Each directory and file is labeled by its i-node number.

File System Backups (2)

- Phase 1: For each modified file, its i-node is marked in the bitmap. Each directory is also marked whether or not it has been modified)
- Phase 2: walks the tree again, unmarking any directories that have no modified files or directories in them
- Phase 3: scan the i-nodes in numerical order and dump all the directories that are marked for dumping
- Phase 4: The files marked are also dumped, again prefixed by their attributes



File System Backups (3)

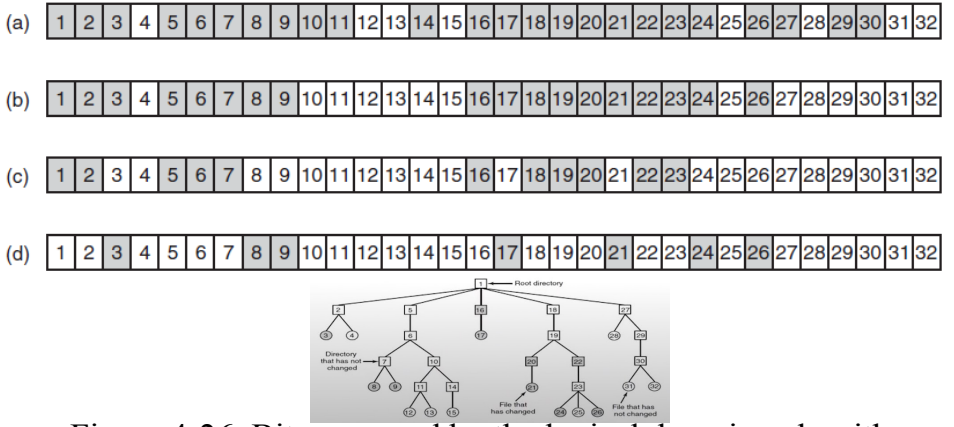


Figure 4-26. Bitmaps used by the logical dumping algorithm.

File System Consistency

- If the system crashes before all the modified blocks have been written out, the file system can become inconsistent
- Critical problem if some are i-node blocks, directory blocks, or blocks containing the free list.
- Many OSes have a utility program for system consistency check. UNIX has fsck; Windows has sfc

File System Consistency

- For block consistency two tables, each one containing a counter for each block, initially set to 0.
- The counters in the first table keep track of how many times each block is present in a file; the counters in the second table record how often each block is present in the free list
- Read all i-nodes and mark tables.
- Examine also the free list or bitmap to find all the blocks that are not in use.

File System Consistency

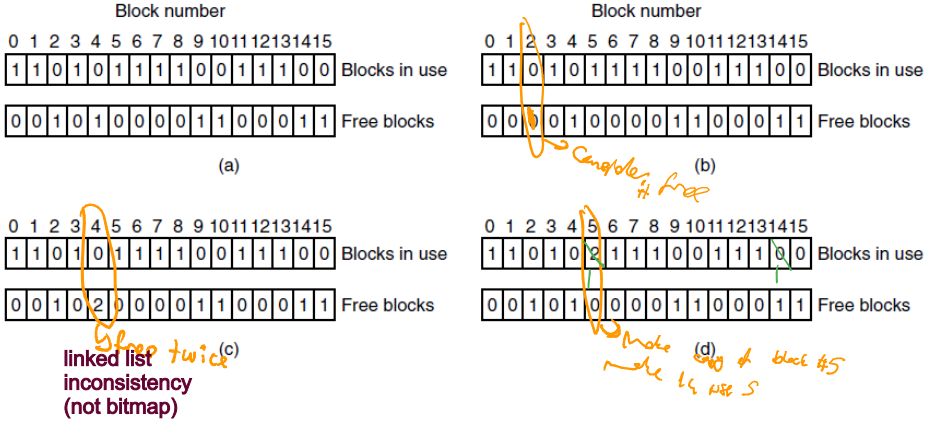


Figure 4-27. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

File System Performance (cache)

- The most common technique used to reduce disk accesses is the **block cache or buffer cache**.
- LRU algoritm can be implemented properly because cache references are relatively infrequent
- LRU is not always for file system cache: These considerations lead to a modified LRU scheme,
  - Unix –Is the block likely to be needed again soon? (i node blocks)
  - windows–Is the block essential to the consistency of the file system? (i nodes)

→ This says inodes are important for things in memory, cache.

→ This says they will be used but they are too important to be in cache.

Space problem, it uses 1 byte to keep least recently used one.

Bigger problem is whenever you reference the page, it marks with a timestamp so you know which page is least recently used. Problem; we n additional hardware in MMU to make it marking. (It takes lot of time so we use hw) Unfortunately MMU is also expensive.

When it comes to 2nd storage for caching, it's not problem anymore since I/O devices are already slow.

replace LRU block in the cache.

File System Performance (1)

we can note

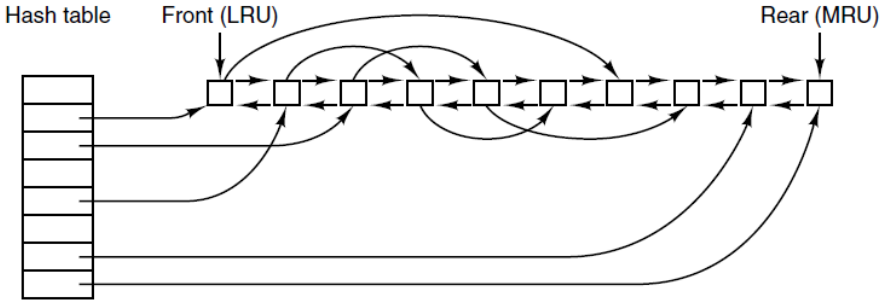


Figure 4-28. The buffer cache data structures.

File System Performance (2)

- Some blocks rarely referenced two times within a short interval.
- Leads to a modified LRU scheme, taking two factors into account:

1. Is the block likely to be needed again soon? i-blocks usually read only once!
2. Is the block essential to the consistency of the file system?

File System Performance (cache)

- In Windows all modified blocks are written back to the disk immediately are called **write-through caches**
- Removing a USB disk from a UNIX system without doing a sync will almost always result in lost data, and frequently in a corrupted file system as well

Windows keeps all modified blocks on the disk.

File System Performance (read ahead)

- A second technique for improving perceived file-system performance is to try to get blocks into the cache before they are needed to increase the hit rate.
- When the file system is asked to produce block k in a file, it does that, but when it is finished, it makes a sneaky schedule to read for block k + 1
- This only works for sequential file reads

7200 rotations per minute

Always there is data under head.

It doesn't cost reading data under.

While doing this, CPU can read.

But you cannot avoid interrupt, context switch...

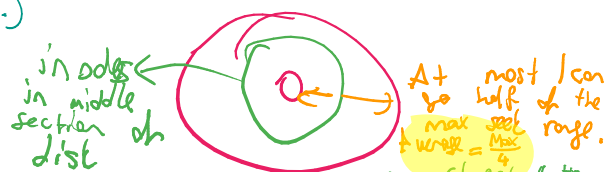
SSD

Reducing Disk Arm Motion

- Reduce the amount of disk-arm motion by putting blocks that are likely to be accessed in sequence close to each other, preferably in the same cylinder
- Performance improvement: put the i-nodes in the middle of the disk, thus reducing the average seek between the i-node and the first block by a factor of two.
- SSDs have peculiar properties when it comes to reading, writing, and deleting: each block can be written only a limited number of times, so great care is taken to spread the wear on the disk evenly

Some SSDs have this problem.

Keep inodes and blocks close to each on your disk. So you don't have to seek a lot. (If you make it small.)



Reducing Disk Arm Motion

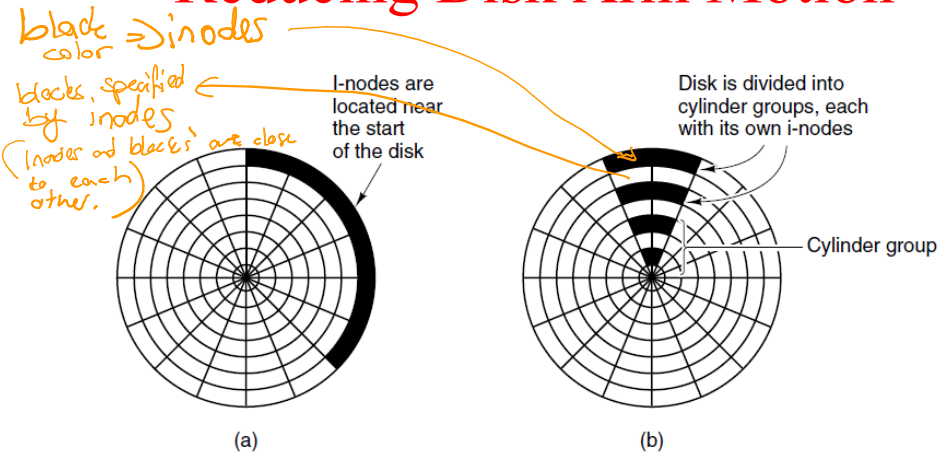
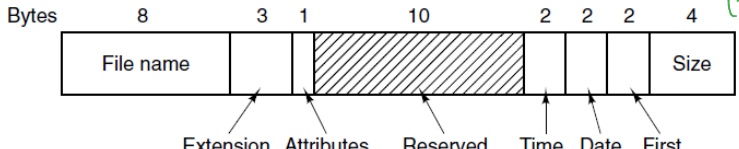


Figure 4-29. (a) I-nodes placed at the start of the disk. (b) Disk divided into cylinder groups, each with its own blocks and i-nodes.

FAT Structure

The MS-DOS File System (1)

Earlier MSDOS: Each directory entry is fixed Total size: 32 bytes 2^{32} = 4 MB at most to represent a file. (for MSDOS)



If there are 10 files size is 32 - 10 = 22 bytes

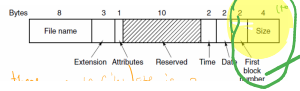
System Hidden Executable Read only

Question: - if this is first block no, where is the other block numbers? MOST IMPORTANT DIFFERENCE BETWEEN I-NODE STRUCTURE and FAT.

Figure 4-30. The MS-DOS directory entry.

For ex each block is 4 kB. So for single block it is enough (data) FAT uses linked list in memory structure. Rest block numbers are obtained from the FAT.

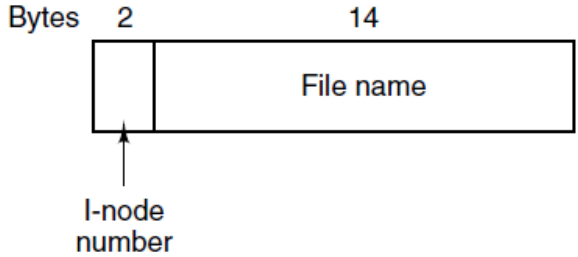
The MS-DOS File System (2)



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

Figure 4-31. Maximum partition size for different block sizes. The empty boxes represent forbidden combinations.

The UNIX V7 File System (1)



For each file, you have inode and you have all the attributes.

Figure 4-32. A UNIX V7 directory entry. why so simple?

The UNIX V7 File System (2)

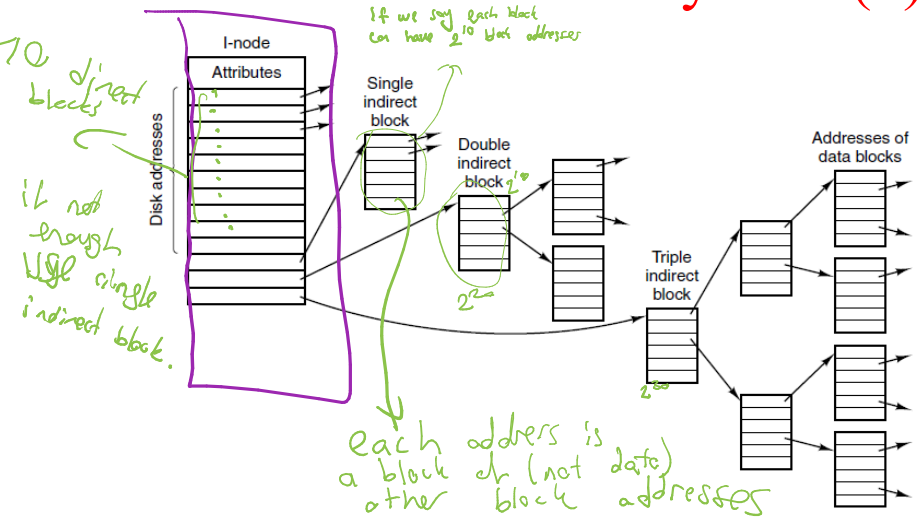


Figure 4-33. A UNIX i-node

The UNIX V7 File System (3)

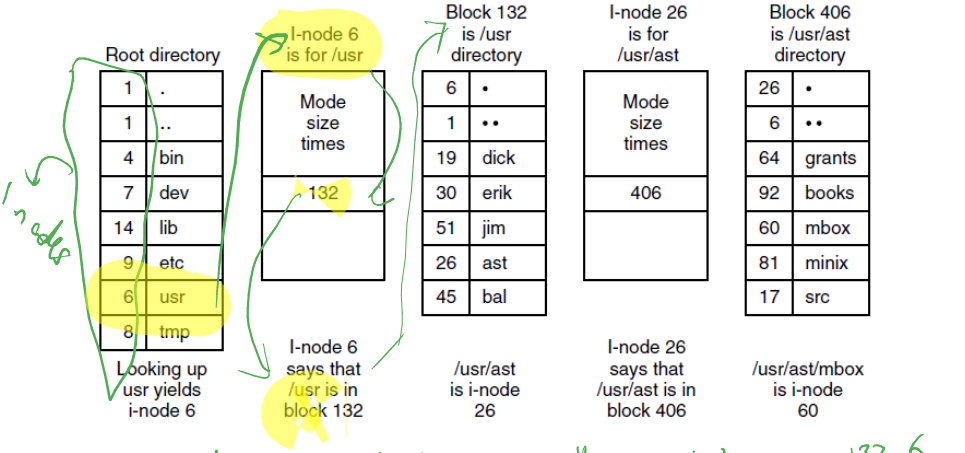


Figure 4-34. The steps in looking up */usr/ast/mbox*.

The ISO 9660 File System

- CD-ROMs do not have concentric cylinders the way magnetic disks do.
- Instead there is a single continuous spiral containing the bits in a linear sequence
- Often the position of a block along the spiral is quoted in minutes and seconds (why?)
- It can be converted to a linear block number using the conversion factor of 1 sec = 75 blocks.

They are read only.
So no need for write through,
You can optimize file system with sequential way.
No external fragmentation.

=) Not by block, by minutes/seconds.
Put this block in 7 minute 32nd mark.

The ISO 9660 File System

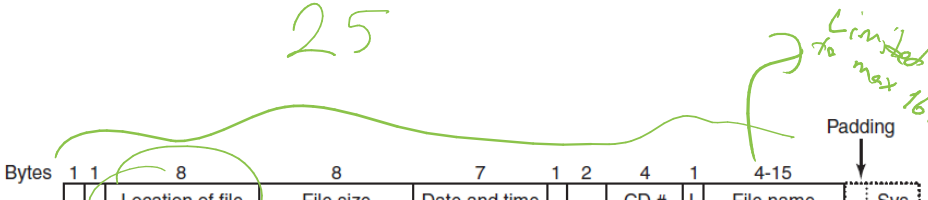


Figure 4-35. The ISO 9660 directory entry.

Rock Ridge Extensions

The extensions use system field of ISO 9660 to make CD-ROMS UNIX compatible

1. PX - POSIX attributes.
2. PN - Major and minor device numbers.
3. SL - Symbolic link.
5. NM - Alternative name.
6. CL - Child location.
7. PL - Parent location.
8. RE - Relocation.
9. TF - Time stamps.

Joliet Extensions

Microsoft invented some extensions to ISO 9660 that were called Joliet

For ex twelve characters

1. Long file names.
2. Unicode character set.
3. Directory nesting deeper than eight levels.
4. Directory names with extensions

End

Chapter 4