File Systems

ICT 41203 Operating Systems

Nimal Skandhakumar

Faculty of Technology
University of Sri Jayewardenepura

2018/10/22

Need for long-term storage

- Computer applications need to store and retrieve information
- There is need for long-term storage, because
  - Limited amount of space in main memory or virtual memory
  - Information must be retained beyond the life of a process
  - Multiple processes may need to access (parts of) same information

Requirements for long-term storage

- Essential requirements for long-term information storage:
  1. It must be possible to store a very large amount of information.
  2. The information must survive the termination of the process using it.
  3. Multiple processes must be able to access the information at once.

History of long-term storage

Why File Systems?

- Think of a disk as a linear sequence of fixed-size blocks and supporting two operations:
  - Read block *k*
  - Write block *k*
- For large systems, few of the questions that quickly arise:
  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?

File Systems

- Files are logical units of information created by processes.
- Processes can read existing files and create new ones if need be.
- Files are managed by the operating system.
- The operating system dealing with files is known as the **file system**.

File Systems

- Implement an abstraction (files) for secondary storage
- Organize files logically (directories)
- Permit sharing of data between processes, people, and machines
- Protect data from unwanted access (security)

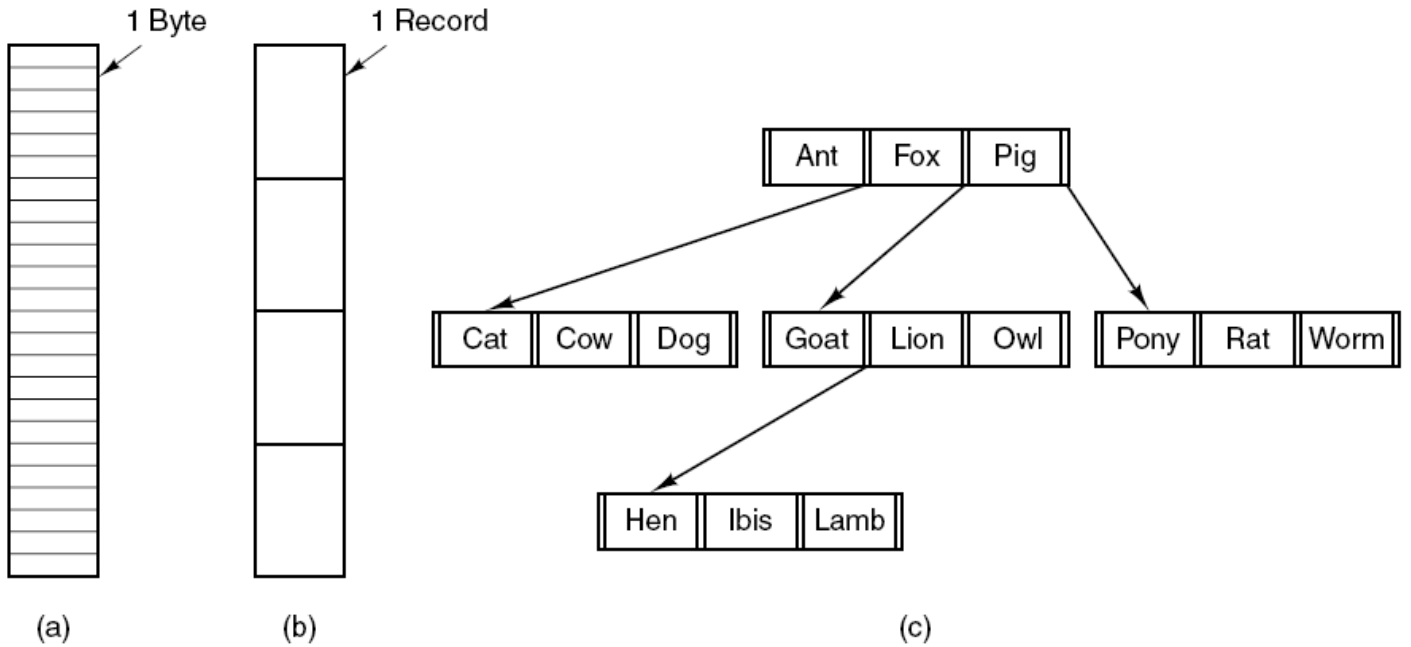
Files

- A file is data with some properties
  - Contents, size, owner, last read/write time, protection, etc.
- A file can also have a type
  - Understood by other parts of the OS or runtime libraries
    - Executable, dll, souce, object, text, etc.
  - Understood by the file system
    - Block/character device, directory, link, etc.
- A file's type can be encoded in its name or contents
  - Windows encodes type in name - .com, .exe, .bat, .dll, .jpg, etc.
  - Unix encodes type in contents - Magic numbers, initial characters
    - e.g., #! for shell scripts

File Naming

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 Structure



a) A file is an unstructured sequence of bytes. The operating system does not know or care what is in the file. Used in Unix and Windows.

b) A file is a sequence of fixed-length records, generally matching the 80-column punched cards and 132-character line printer. This was a common model on mainframe computers.

c) A file consists of a tree of records, each containing a key field in a fixed position in the record. This is used on some large mainframe computers for commercial data processing.

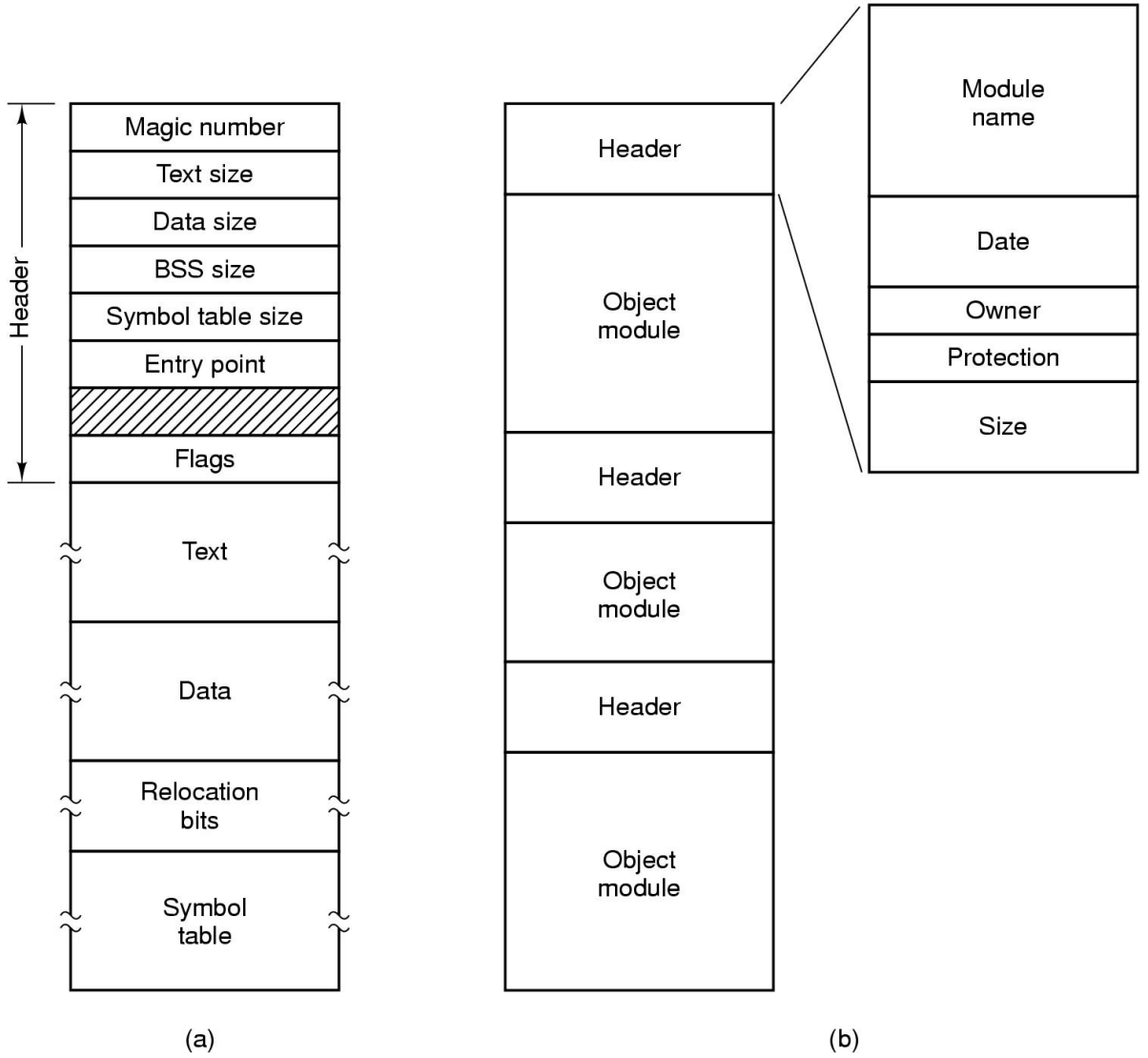
File Types

- **Regular files** are the ones that contain user information
- **Directories** are system files for maintaining the file system structure
- **Character special files** are related to input/output
- **Block special files** are used to model disks

- Regular files are generally either ASCII files or binary files.
  - ASCII files can be displayed and printed as is, and they can be edited with any text editor.
  - Binary files have some internal structure known to programs that use them.

File Types

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
  - Used with magnetic tapes
- **Random-access**
  - Files whose bytes or records can be read in any order
  - Used with discs
  - Essential for many applications, such as database systems

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

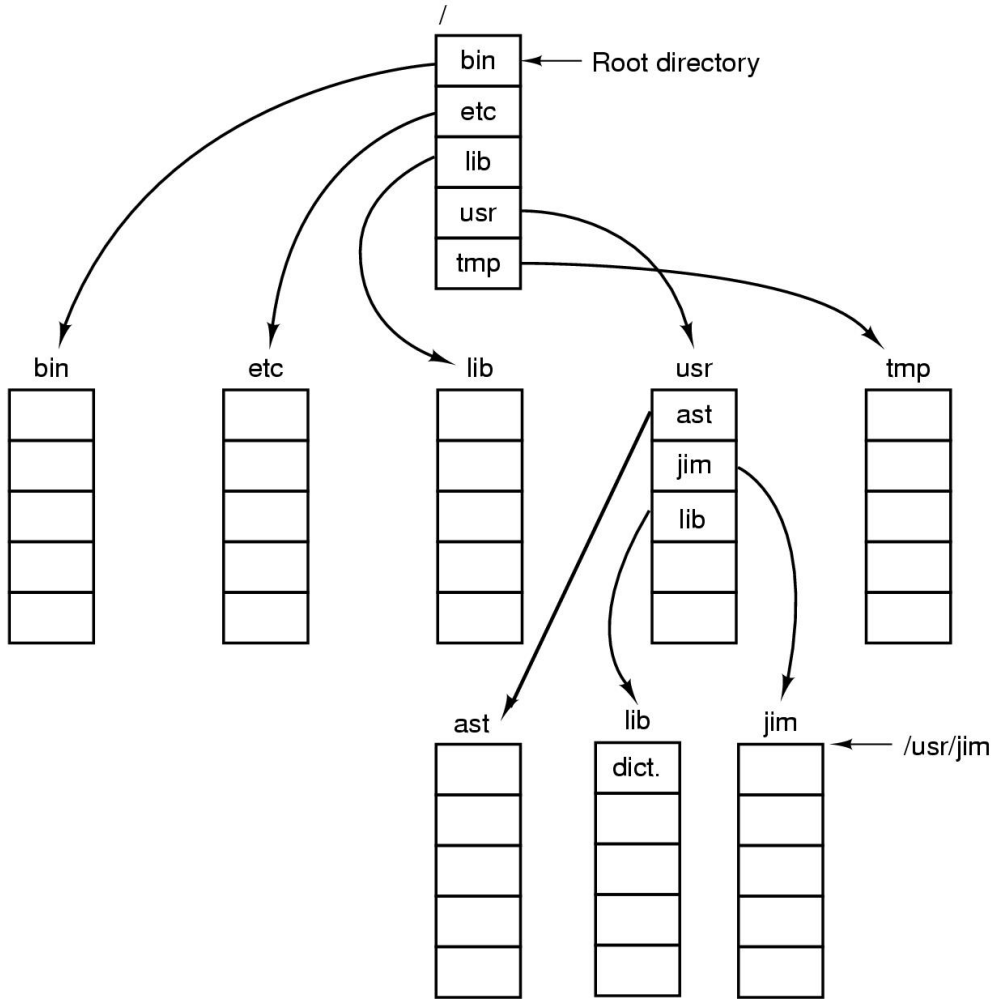
- The most common system calls relating to files:
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write
  - Append
  - Seek
  - Get Attributes
  - Set Attributes
  - Rename

Directories

- Directories serve two purposes
  - For users, they provide a structured way to organize files
  - For the file system, they provide a convenient naming interface that allows the implementation to separate logical file organization from physical file placement on the disk
- Most file systems support multi-level directories
  - Naming hierarchies (/, /usr, /usr/local/, …)
- Most file systems support the notion of a current directory
  - Relative names specified with respect to current directory
  - Absolute names start from the root of directory tree

Directory Internals

- A directory is a list of entries
  - <name, location>
  - Name is just the name of the file or directory
  - Location depends upon how file is represented on disk
- List is usually unordered (effectively random)
  - Entries usually sorted by program that reads directory
- Directories typically stored in files



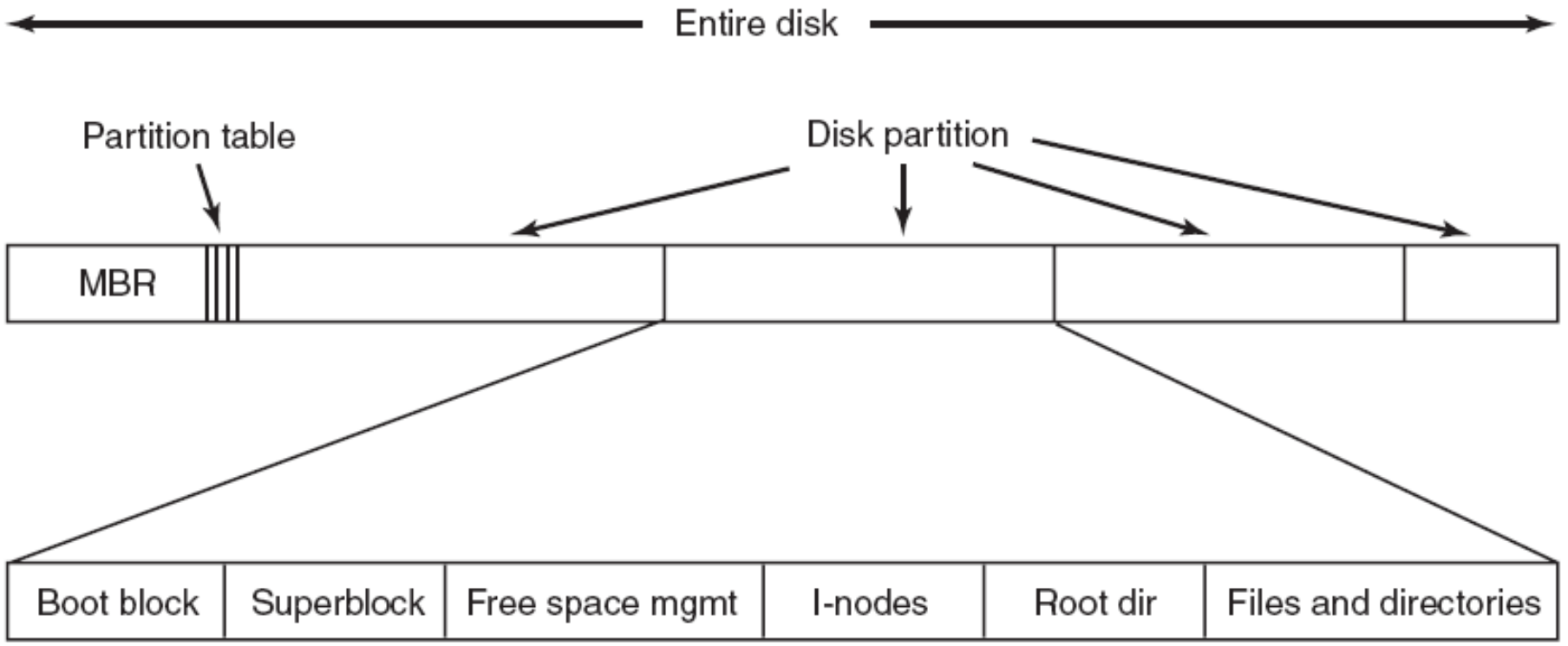
Directory Operations

- System calls for managing directories:
  - Create
  - Delete
  - Opendir
  - Closedir
  - Readdir
  - Rename
  - Link
  - Uplink

File System Layout

- File systems define a block size (e.g., 4KB)
  - Disk space is allocated in granularity of blocks
- A “Master Block” determines location of root directory
  - Always at a well-known disk location
  - Often replicated across disk for reliability
- A free map determines which blocks are free, allocated
  - Usually a bitmap, one bit per block on the disk
  - Also stored on disk, cached in memory for performance
- Remaining disk blocks used to store files (and dirs)
  - There are many ways to do this

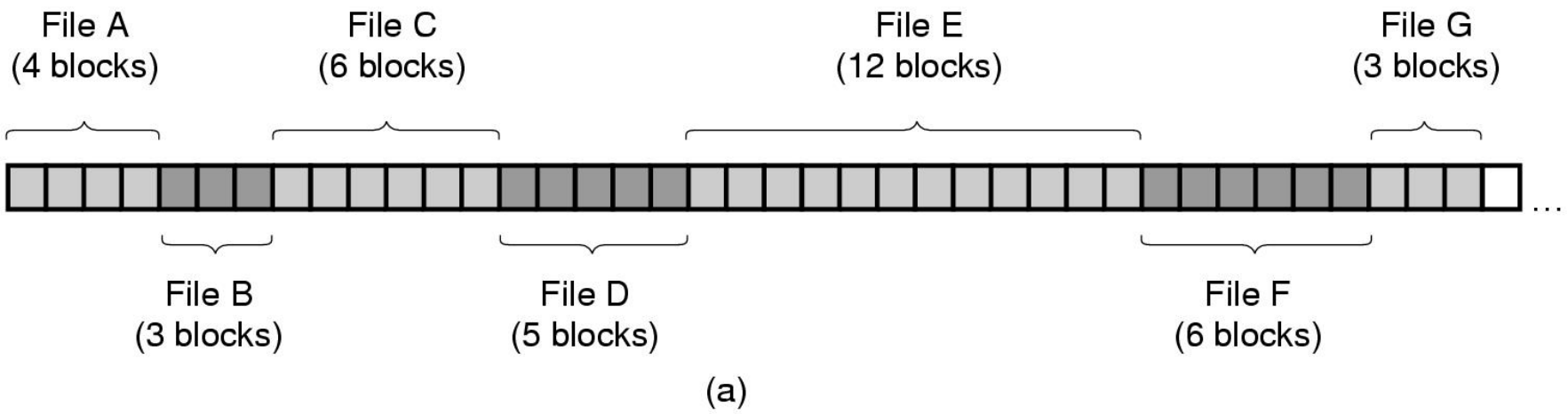
File System Layout



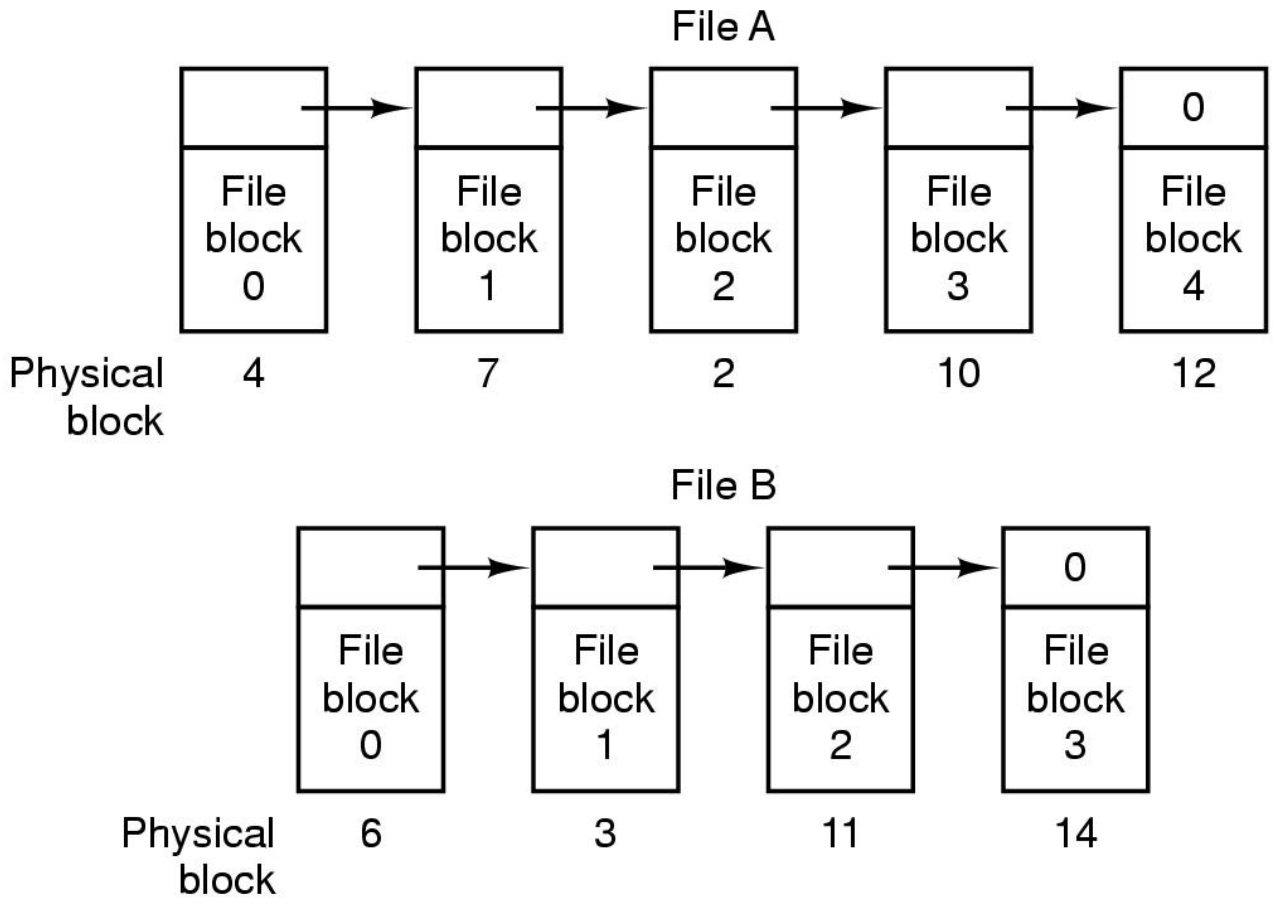
Disk Layout Strategies

- Files can span multiple disk blocks
- How do you find all of the blocks for a file?
  1. Contiguous allocation
     - Fast, simplifies directory access
     - Inflexible, causes fragmentation, needs compaction
  2. Linked structure
     - Each block points to the next, directory points to the first
     - Good for sequential access, bad for all others
  3. Indexed structure (indirection, hierarchy)
     - An “index block” contains pointers to many other blocks
     - Handles random better, still good for sequential
     - May need multiple index blocks (linked together)

Contiguous Allocation

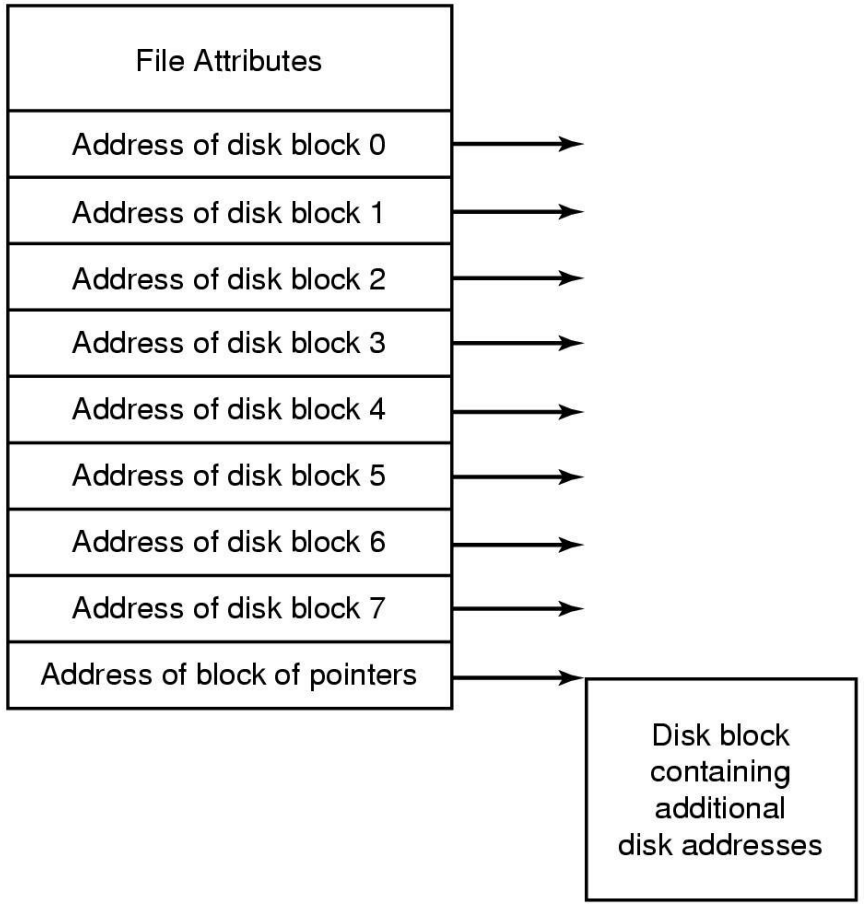


Linked List Allocation



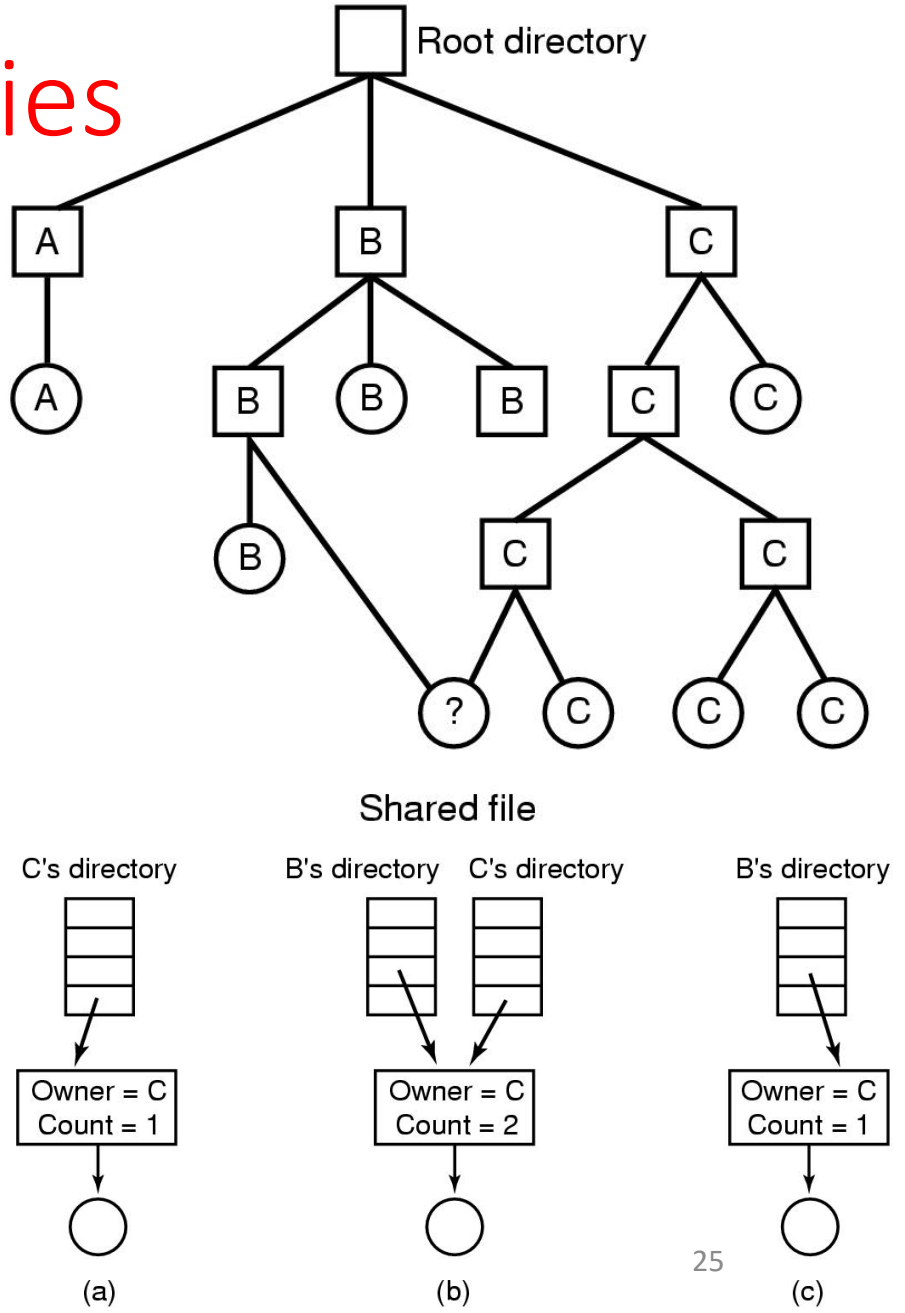
Unix i-nodes

- Unix i-nodes implement an indexed structure for files
- Each i-node contains number of file block pointers
- Last block points to a block containing more disk-block addresses



Sharing Files Between Directories

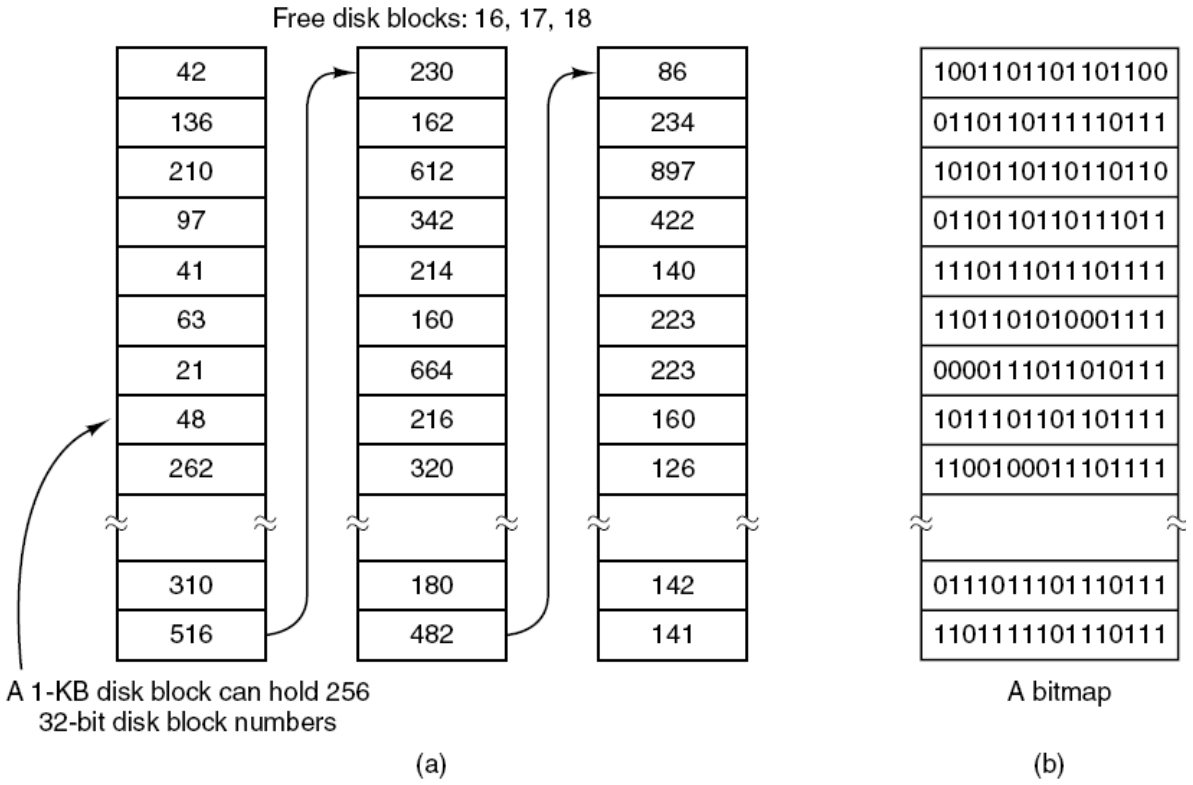
- Links (or hard links) in source_file target_dir
  - Simply create another link from target_dir to the **inode** of source_file (the inode is not duplicated)
  - Now two directories have links to source_file
  - What if we remove one?
  - Now you understand why the system call to remove a file is named “unlink”?



File Buffer Cache

- Applications exhibit significant locality for reading and writing files
- Idea: Cache file blocks in memory to capture locality
  - This is called the file buffer cache
  - Cache is system wide, used and shared by all processes
  - Reading from the cache makes a disk perform like memory
  - Even a 4 MB cache can be very effective
- Issues
  - The file buffer cache competes with VM (tradeoff here)
  - Like VM, it has limited size
  - Need replacement algorithms again (LRU usually used)

Keeping Track of Free Blocks

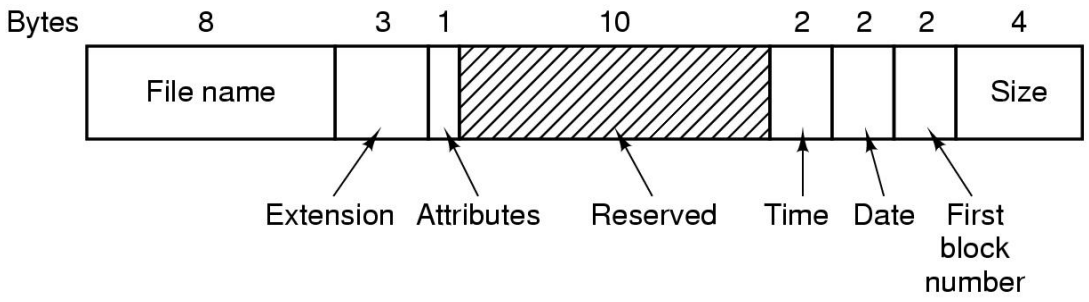


a) Storing the free list on a linked list

b) A bitmap

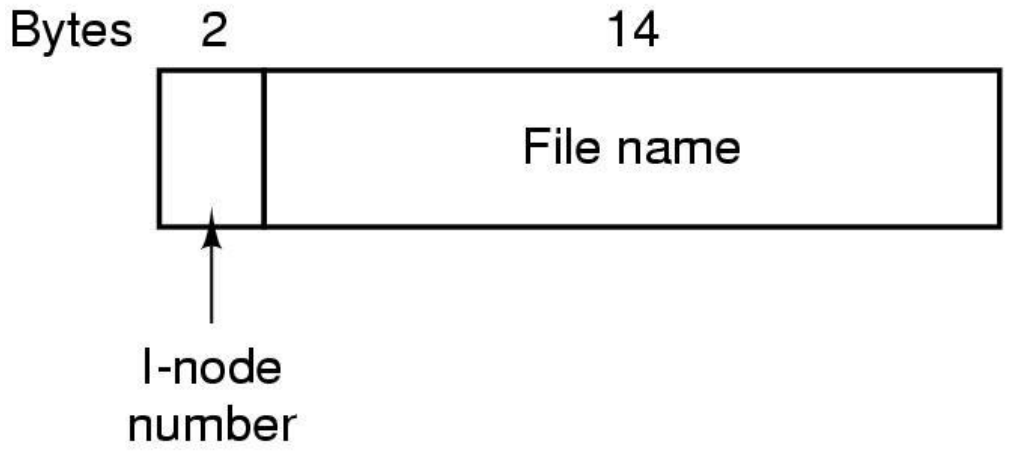
- A disk with n blocks requires a bitmap with n bits.
- Free blocks are represented by 1s in the map, allocated blocks by 0s (or vice versa).

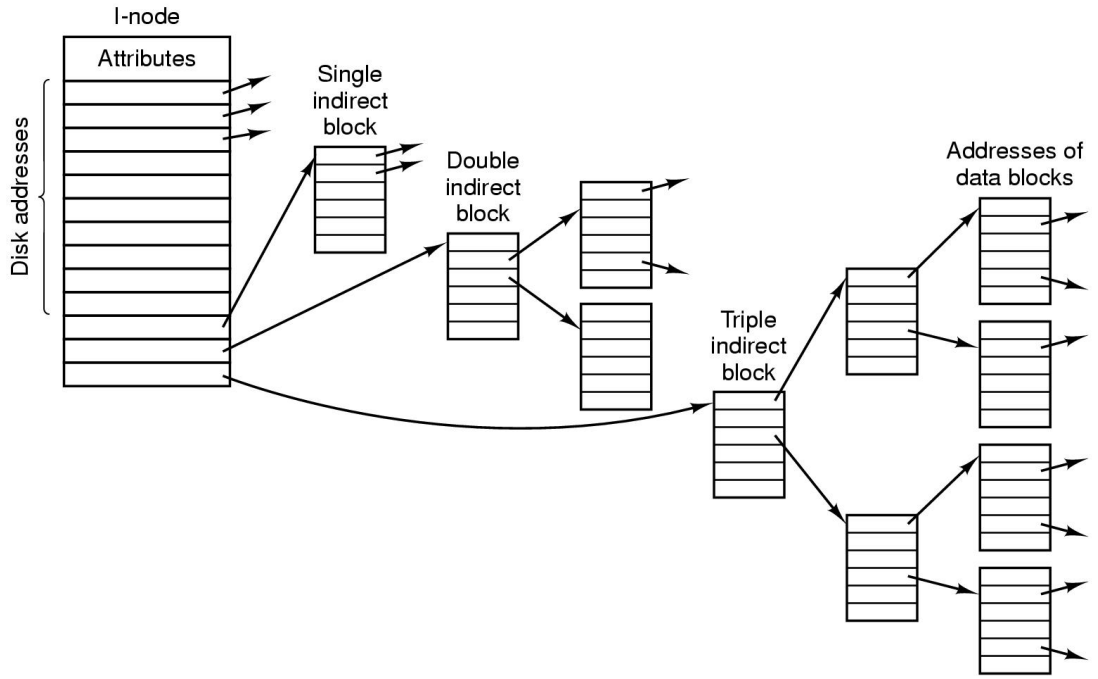
The MS-DOS File System



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

The UNIX V7 File System





CD-ROM File System – ISO 9660