File System

File

- A collection of related information defined by its creator
- The abstraction used by the kernel to represent and organize the system's non-volatile storage resources, including hard disks, floppy disks, CD-ROM, and optical disks.

Why need File System?

- Storage capacity of a system (i.e. Main Memory) restricted to size of available virtual memory
- May not be enough for applications involving large data (face expt.)
- Virtual memory is volatile
- May not be good for long term storage
- Information need not be dependent upon process
- passed file may need to be modified by different processes
- Essential requirements of long-term information storage
  - Store very large amount of information
  - Information must survive termination of processes (be persistent)
  - Multiple processes must be able to access information concurrently
- Store information in files
- File system – Part of the OS that deals with file management

Files Naming

- Most visible aspect of an OS
- Mechanism to store and retrieve information from the disk
- Represent programs (both source and object) and data
- Data files may be numeric, alphanumeric, alphabetic, or binary
- May be free form or formatted rigidly
- Accessed by a name
- Created by a process and continues to exist after the process has terminated
- Information in the file defined by creator
- Naming conventions
  - Set of a fixed number of characters (letters, digits, special characters)
  - Case sensitivity

File structure

- **Byte sequence**

  Fig: refer from class note

  - Unix and ms-dos use byte sequence
  - File is the unstructured sequence of bytes.
  - OS doesn't care or know that is in the file. All it sees are bytes.
  - Meaning on the bytes is imposed by user programs
  - Provides maximum flexibility but minimal support
  - Advantages to users who want to define their own semantics on files
- **Record sequence**

  Fig: refer from class note

A file is a sequence of fixed-length records, each with some internal structure.
Each read operation returns one records, and write operation overwrites or append one record.
Many old mainframe systems use this structure.

- **Tree**

Fig: refer from class note

File consists of tree of records, not necessarily all the same length.
Each containing a key field in a fixed position in the record, sorted on the key to allow the rapid searching.
The operation is to get the record with the specific key.
Used in large mainframe for commercial data processing.

File Types

- Regular files
- Most common types of files
- May contain ASCII characters, binary data, executable program binaries, program input or output
- no kernel level support to structure and the contents of these files
- Both sequential and random access are supported

There are following types of regular file

- Text files (ASCII)
  - Lines of text
  - Lines may be terminated by carriage return
  - File itself has an end-of-file character
  - Useful for inter-process communication via pipes in Unix
- Binary files
  - Not readily readable
  - Has internal structure depending upon the type of file (executable or archive)
  - Executable file (a.out format)
    - Header: Magic number, Text size, Data size, bss size, Symbol table size, Entry point, Flags
    - Text
    - Data
    - Relocation bits
    - Symbol table
- BSS or Block Started by Symbol
  - Uninitialized data for which kernel should allocate space

- ❖ Used by an obsolete IBM assembler, BSS was an assembler pseudo-opcode that filled an area of memory with zero's
- Library archive: compiled but not linked modules
  - ❖ Header: Module name, Date, Owner, Protection, Size
  - ❖ Object module

File types - name.extension

File Type	*Possible extension*	*Function*
Executable	Exe,com,bin	Machine language program
Object	Obj, o	Compiled machine lang., not linked
Source code	c, CC, p, java, asm…	Source code in various languages
Batch	Bat, sh	Commands to command interpreter
text	Txt, doc	Textual data, documents
Print, view	ps, dvi, gif	ASCII or binary file
archive	Arc, zip, tar	Group of files, sometimes compressed
Library	Lib, a	Libraries of routines

Directories

- System files for maintaining the structure of the file system
- Binary file containing a list of files contained in it (including other directories)
- May contain any kind of files, in any combination and refer to directory itself and its parent directory
- Created by mkdir and deleted by rmdir, if empty Non-empty directories can be deleted by rm -r
- Each entry made up of a file-inode pair
- Used to associate inodes and directory locations
- Character-special files and Block-special files
- Character-special files
- Allow the device drivers to perform their own i/o buffering
- Used for unbuffered data transfer to and from a device
- Generally have names beginning with r(for raw), such as /dev/rsd0a
- Block-special devices
- Expect the kernel to perform buffering for them
- Used for devices that handle i/o in large chunks, known as blocks

File access

- Sequential access
  - ❖ Bytes or records can be read only sequentially
  - ❖ Like magnetic tape
- Random access

- Bytes or records can be read out of order
- Access based on key rather than position
- Like magnetic disk

- Distinction more apparent in older OS.

File attributes

- Name
  - symbolic file-name, only information in human-readable form
- Type -
  - for systems that support multiple types
- Location -
  - pointer to a device and to file location on device
- Size -
  - current file size, maximal possible size
- Protection -
  - controls who can read, write, execute
- Time, Date and user identification
  - data for protection, security and usage monitoring
- Information about files are kept in the directory structure, maintained on disk

File operations:

File treated as abstract data type so there are following file operations

- Create
  - Create a new file of size zero (no data)
  - The attributes are set by the environment in which the file is created.
- Delete
  - Delete an existing file
  - Some systems may automatically delete a file that has not been used in n days
- Open
  - Establish a logical connection between process and file
  - Fetch the attributes and list of disk addresses into main memory for rapid access during subsequent calls
  - I/O devices attached instead of opened
  - * System call open (2)
- Close
  - Disconnects file from the current process
  - File not accessible to the process after close
  - I/O devices detached instead of closed
- Read
  - Transfer the logical record starting at current position in file to memory starting at buffer (buffer known as input buffer)
  - Older systems have a read sequence to achieve the same effect and read direct for random access files
- Write
  - Transfer the memory starting at buf to logical record starting at current position in file
  - If current position is the end of file, file size may increase
  - If current position is in the middle of file, some records may be overwritten and lost
  - Older systems have a write seq to achieve the same effect and write direct for random

access files

- Append.
  - Restrictive form of write
- Seek.
  - Primarily used for random access files
  - Repositions the pointer to a specific place in file
  - Unix system call lseek(2)
- Get attributes.
  - Get information about the file
  - In Unix, system calls stat(2), fstat(2), lstat(2)
  - Device file resides on
  - File serial number, i-node number
  - File mode
  - Number of hard links to the file
  - uid/gid of owner
  - Device identifier (for special files)
  - Size of file
  - Last time of access, modification, status change
  - Preferred block size of file system
  - Actual number of blocks allocated
- Set attributes.
  - Set the attributes of a file, e.g., protection, access time
  - Unix system calls: utimes(2), chmod(2)
  - Rename.
  - Change the name of a file

Memory-Mapped files

Many versions of UNIX support **memory-mapped files**. This feature makes it possible to map a file onto a portion of a process' address space so the file can be read and written as if it were a byte array in memory. Mapping a file in makes random access to it much easier than using I/O system calls such as read and write. Shared libraries are accessed by mapping them in using this mechanism. In Fig. 10-14 we see a file that is mapped into two processes at the same time, at different virtual addresses.

An additional advantage of mapping a file in is that two or more processes can map in the same file at the same time. Writes to the file by any one of them are them instantly visible to the others. In fact, by mapping in a scratch file (which will be discarded after all the processes exit), this mechanism provides a

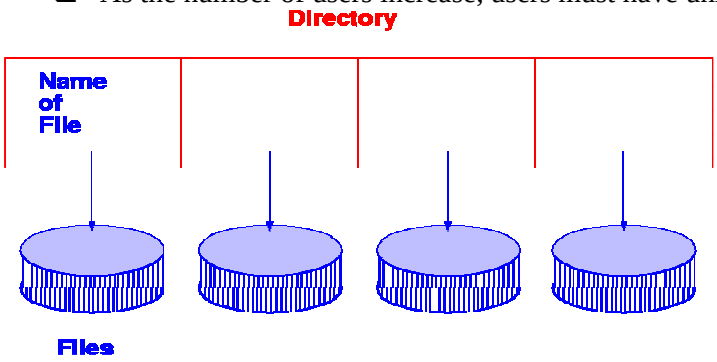
high-bandwidth way for multiple processes to share memory. In the most extreme case, two or more processes could map in a file that covers the entire address space, giving a form of sharing that is partway between separate processes and threads. Here the address space is shared (like threads), but each process maintains its own open files and signals, for example, which is not like threads. In practice, making two address spaces exactly correspond is never done, however.

Directories

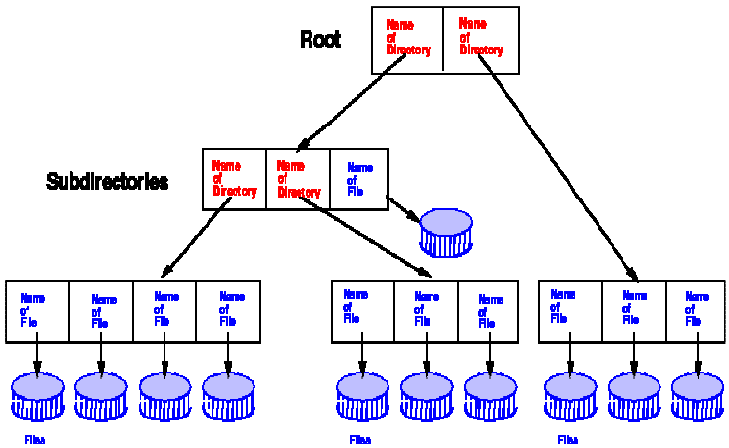
- Provided by the file system to keep track of files
- Hierarchical directory systems
- Directory contains a number of entries – one for each file
- Directory may keep the attributes of a file within itself, like a table, or may keep them elsewhere and access them through a pointer
- In opening a file, the os puts all the attributes in main memory for subsequent usage

Single Level Directory

- A single directory for all users
- Naming Problem and Grouping Problem
  - As the number of files increases, difficult to remember unique names
  - As the number of users increase, users must have unique names



Tree structured Directory



- Arbitrary depth of directories
  - Leaf nodes are files, interior nodes are directories.
- Efficient Searching
- Grouping Capability
- Current Directory (working directory)
  - cd /spell/mail/prog
  - type list
- MS-DOS uses a tree structured directory
- Absolute or relative path name
- Absolute from root
- Relative paths from current working directory pointer.
- Creating a new file is done in current directory
- Creating a new subdirectory is done in current directory, e.g. mkdir <dir-name>
- Delete a file , e.g. rm file-name
- Deletion of directory
- Option 1 : Only delete if directory is empty
- Option 2: delete all files and subdirectories under directory

- **Information in directory entry**
  - File name: Symbolic file name, only information kept in human readable form
  - File type: Needed for those systems that support different file types
  - Location: Pointer to the device and location of file on that device
  - Size: Current size of the file May also include the maximum possible size for the file
  - Current position: Pointer to the current read/write position in the file
  - Protection: Access control information
  - Usage count: Number of processes currently using the file
  - Time, date, and process identification: May be kept for creation, last modification, and last use, Useful for protection and usage monitoring
- Directory entry may use from 16 to over 1000 bytes for each file
- Size of directory may itself become very large
- Directory can be brought piecemeal into memory as needed

- **Data structures for a directory**
  - Linked list
    - Requires linear search to find an entry
    - Simple to program but time consuming in execution
    - To create a new file, must search entire directory to avoid name duplication
    - To delete a file, search the directory and release the space allocated to it
    - Entry can be marked as unused or attached to a list of free entries
  - Sorted list
    - Allows binary search and decrease average search time
    - Search algorithm is more complicated to program
    - May complicate creation and deletion as large amount of directory information may be moved to keep list sorted

- Hash table
  - Greatly improves directory search time
  - Insertion and deletion straightforward
  - Problem because of fixed size of hash tables

Path names

- Convention to specify a file in the tree-like hierarchy of directories
- Hierarchy starts at the directory known as the root directory
- listing of directories crossed to reach the file, followed by the file name itself
- The name of each directory must be less than 256 characters
- No pathname can be longer than 1023 characters
- Absolute path name
  - Path from root directory to the file
  - Always start at the root directory and is unique
  - /usr/bin/X11/xdvi
- Relative path name
  - Used in conjunction with the concept of "current working directory"
  - Specified relative to the current directory
  - More convenient than the absolute form and achieves the same effect
  - Unix allows arbitrary depth for the file tree

Directory operations

- Create directory.
  - Create a directory and put entries for . and .. in there
  - mkdir command in Unix and ms-dos and int mkdir(path, mode) unix system call
- Delete directory.
  - Delete an existing directory
  - Only empty directory can be deleted
  - Directory containing just . and .. is considered empty
  - rmdir command in Unix and ms-dos and rmdir(2) system call in Uni
  - Can remove a directory only if its link count is zero and no process has the directory open
  - If one or more processes have the directory open when the last link is removed, the . and.. entries,
  - If present, are removed before rmdir() returns and no new entries may be created in the directory, but the directory is not removed until all references to the directory have been closed
- Open Directory
  - Open a directory to read/write
  - Used by many applications, e.g. ls
  - C library function opendir(3)
    - DIR *opendir (dirname)
    - char *dirname
  - Open the directory named by dirname and associate a directory stream with it
  - Returns a pointer to identify the directory stream in subsequent operations
  - If directory cannot be accessed, a null pointer is return close Directory
- Close directory.
  - Close the directory that was opened for reading
  - C library function closedir(3)

- int closedir (dirp)
- DIR *dirp
- Closes the named directory stream and frees the structure associated with the DIR pointer

❖ Read Directory
- ➢ Returns the next entry in an open directory
- ➢ C library function readdir(3)
  - Returns a pointer to the next directory entry
  - Returns null upon reaching the end of the directory or detecting an invalid seekdir(3) operation

❖ Rename.
- ➢ Operates the same way as renaming a file

❖ Link.
- ➢ Allows creation of aliases (links) for the files/directories
- ➢ Same file can appear in multiple directories
- ➢ User command ln(1)
- ➢ Creates hard or symbolic links to files
- ➢ A file may have any number of links
- ➢ Links do not affect other attributes of a file
- ➢ Hard links
  - – Can only be made to existing files
  - – Cannot be made across file systems (disk partitions, mounted file systems) To remove a file, all hard links to it must be removed
  - – System call link(2)
    - · Used to make a hard link to a file
    - · Increments the link count of the file by one
- ➢ Symbolic links
  - – Points to another named file
  - – Can span file systems and point to directories
  - – Removing the original file does not affect or alter the symbolic link itself
  - – symbolic link puts you in the pointed-to location within the file system; changing to the parent of the symbolic link puts you in the parent of the original directory
  - – System call symlink(2)
    - Used to make a symbolic link to a file
    -

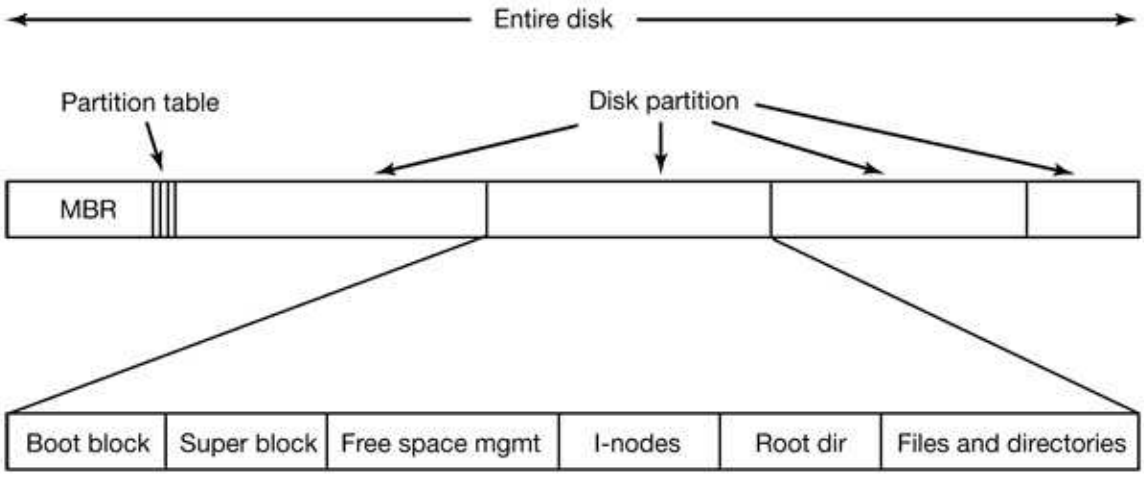
❖ Unlink.
- Remove a directory entry
- If the file being removed is present in one directory, it is removed from the file system
- If the file being removed is present in multiple directories, only the path name specified is removed; others remain
- User commands rm(1) and rmdir(1)
- System call unlink(2)
  - o Removes the directory entry

.

File system implementation

File System Layout

- File systems are stored on disks.
- Most disks can be divided up into one or more partitions, with independent file systems on each partition.
- 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 which gives the starting and ending addresses of each partition and one of the partitions in the table is marked as active.
- MBR program locate the active partition, read in its first block (boot block) and execute it.
- The program in the boot block loads the operating system contained in that partition



- **Superblock** contains all the key parameters (Magic number to identify the file system type, the number of blocks in the file system, and other key administrative information) about the file system and is read into memory when the computer is booted or the file system is first touched
- Boot block
  - Occupies the beginning of the file system, typically the first sector
  - May contain the bootstrap code
  - Only one boot block needed for initialization but every file system has a possibly empty boot block
- Super block
  - Describes the state of the file system
  - Size of the file system
  - Maximum number of files it can store
- Location of the free space
- Inode list
  - List of inodes following the superblock
  - Kernel references inodes by indexing into the inode list
  - Root inode
  - Inode by which the directory structure of the file system is accessible after execution of the mount
  - system call

- Data blocks
  - Start at the end of the inode list
  - Contain file data and administrative data
  - An allocated data block can belong to one and only one file in the file system
- File system manages files, allocating files space, administering free space, controlling access to files, and retrieving data for users
- Processes interact with the file system using a set of system calls
- File data is accessed using a buffering mechanism that regulates data flow between the kernel and secondary storage devices
- Buffering mechanism interacts with the block I/O device drivers to initiate data transfer to and from kernel
- Device drivers are kernel modules that control the operation of peripheral devices
- It is the device driver that makes the kernel treat a device as a block-special device (or random access storage device)
- Yes, the UNIX kernel can allow a tape drive to be treated as a random access storage device
- File system also interacts directly with the character-special devices

Implementing files

- Support of primitives for manipulating files and directories
- Make an association between disk blocks and files
- Contiguous allocation
  - Simplest allocation technique
  - Simple to implement; files can be accessed by knowing the first block of the file on the disk
  - Improves performance as the entire file can be read in one operation

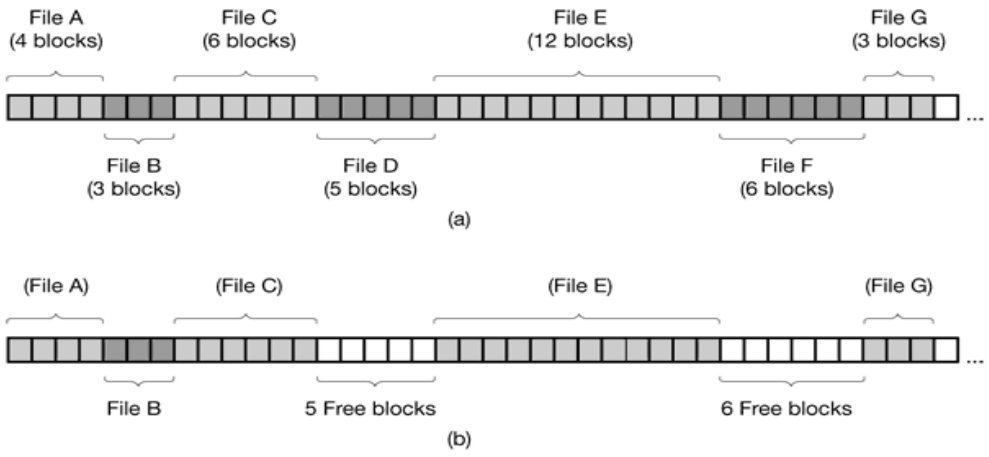


Figure .(a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Problem 1 – File size may not be known in advance
Problem 2 – Disk fragmentation; can be partially solved by compaction

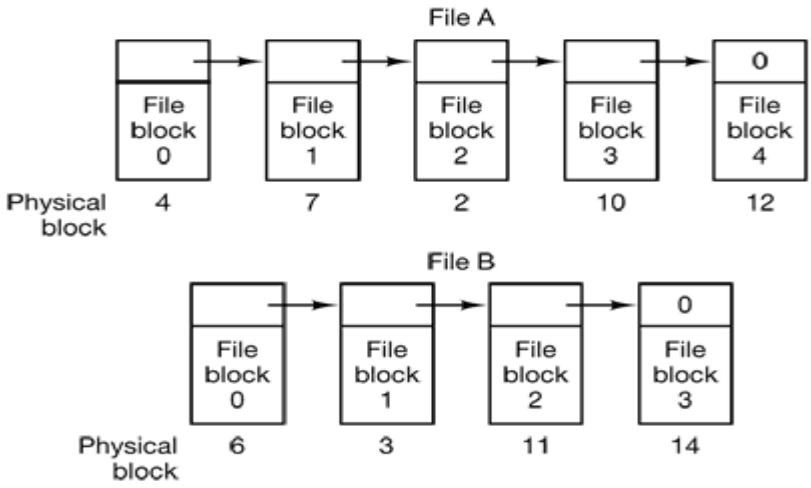
– Linked list allocation

- Files kept as a linked list of disk blocks

- First word of each block points to the next block
- only the address of the first block appears in the directory entry
- No disk fragmentation

Disadvantage

- Random access is extremely slow
- Data in a block is not a power of 2 (to accommodate the link to next block)



– **Linked list allocation using an index**

- Memory contains a table pointing to each disk block called a **FAT** (**File Allocation Table**).
- The entire block is available for data (solution of 2nd problem).
- Random access is easier because the chain must still be followed to find without making any disk references.
- Large file can be access easily

Disadvantage

- The entire table must be in memory all the time to make it work.
- With a 20-GB disk and a 1-KB block size, the table needs 20 million entries, one for each of the 20 million disk blocks. Each entry has to be a minimum of 3 bytes. For speed in lookup, they should be 4 bytes. Thus the table will take up 60 MB or 80 MB of main memory all the time, depending on whether the system is optimized for space or time.

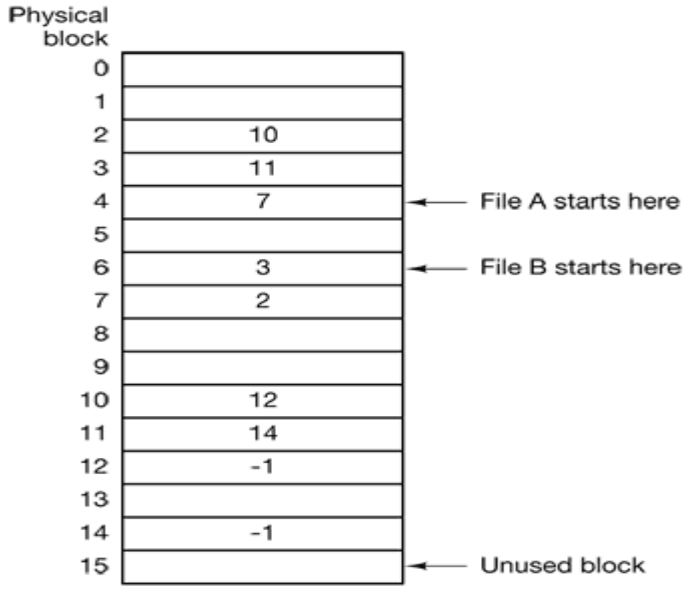
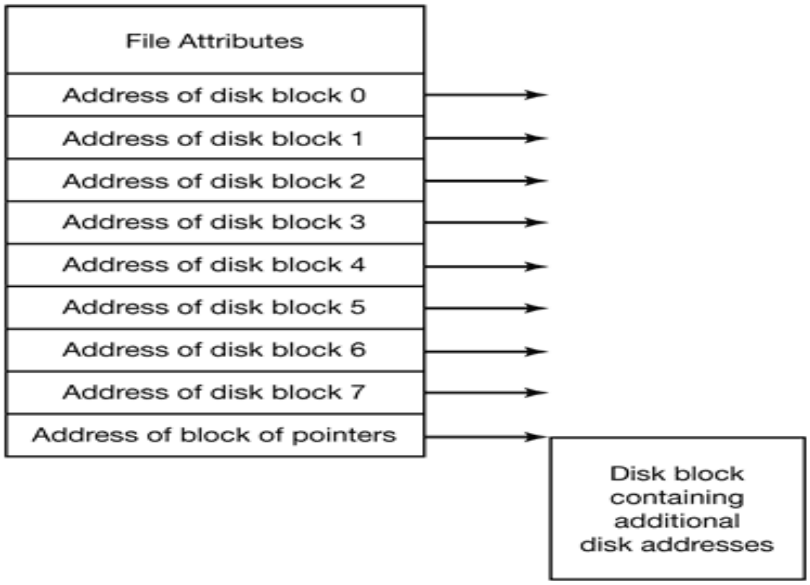


Figure. Linked list allocation using a file allocation table in main memory.

Inodes (Index node)

- Structure to keep information about each file, including its attributes and location
- Inodes need only in memory the file is opened,
- Contain about 40 separate pieces of information useful to the kernel
  - Uid(User id) and gid (Group ID)
  - File type
  - File creation, access, and modification time
  - Inode modification time
  - Number of links to the file
  - Size of the file
  - Disk addresses, specifying or leading to the actual disk locations for disk blocks that make up the file
- Inode table
  - Created at the time of creation of file system (disk partition)
  - Always in the same position on the file system
  - Inode table size determines the maximum number of files, including directories, special files, and links, that can be stored into the file system
  - Typically, one Inode for every 2 to 8 K of file storage

- Opening a file
  - Process refers to the file by name
  - Kernel parses the filename one component at a time, checking that the process has permission to search the directories in the path
  - Kernel eventually retrieves the inode for the file
  - Upon creation of a new file, kernel assigns it an unused inode
  - Inodes stored in the file system but kernel reads them into an in-core inode table when manipulating files



Implementing Directories

- The main function of the directory system is to map the ASCII name of the file onto the information needed to locate the data.
- When a file is opened, the operating system uses the path name supplied by the user to locate the directory entry.
- The directory entry provides the information needed to find the disk blocks.
- Two methods are used to keep the information of attributes of the file
  - Store at Directory itself
  - Store file name at directory and attribute using pointer

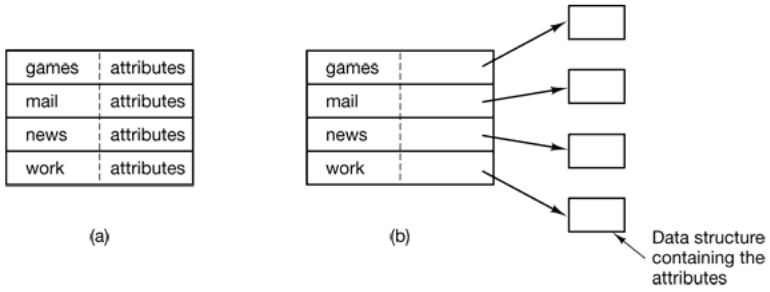
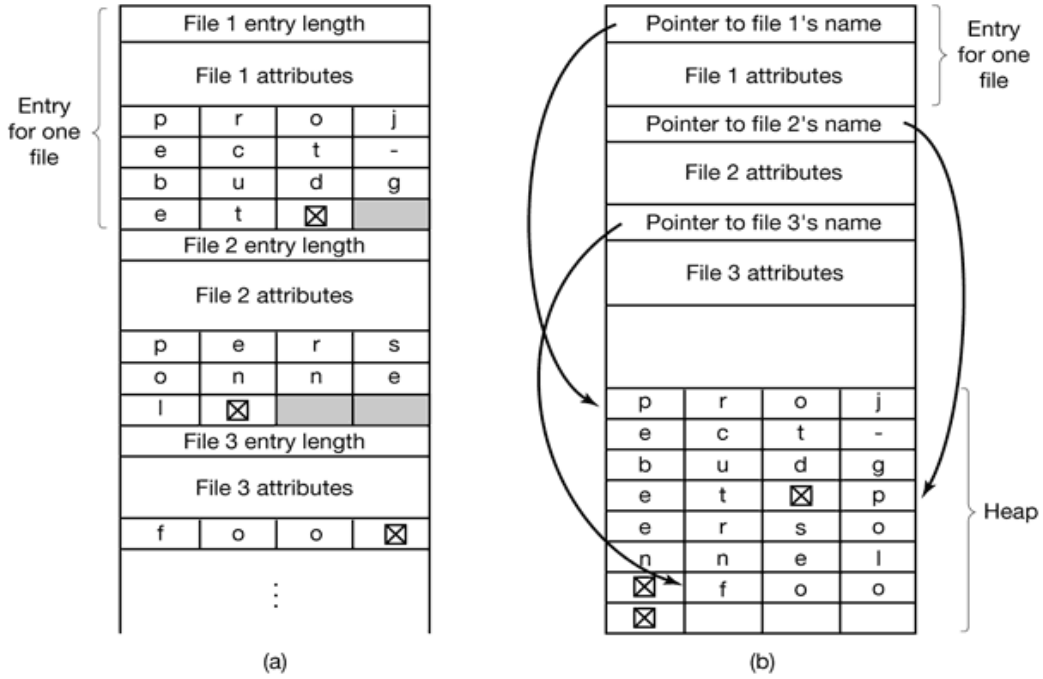


Figure (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

- **Implementation**
  - Fixed Number Character in each Entry.
    - The simplest approach is to set a limit on file name length, typically 255 characters, and then use one of the designs with 255 characters reserved for each file name.
    - Wastes a great deal of directory space, since few files have such long names. For efficiency reasons, a different structure is desirable. Above example

- All directory entries are the same size



  - Each directory entry contains a fixed portion, typically starting with the length of the entry, and then followed by data with a fixed format, usually including the owner, creation time, protection information, and other attributes.
  - This fixed-length header is followed by the actual file name, however long it may be, as shown in figure.
  - Disadvantage
    - When a file is removed, a variable-sized gap is introduced into the directory into which the next file to be entered may not fit
    - Single directory entry may span multiple pages, so a page fault may occur while reading a file name.
  - Another way to handle variable-length names is to make the directory entries themselves all fixed length and keep the file names together in a heap at the end of the directory, as shown in figure.
  - This method has the advantage that when an entry is removed, the next file entered will always fit there. Of course, the heap must be managed and page faults can still occur while processing file names. One minor win here is that there is no longer any real need for file names to begin at word boundaries.

- All Method gives slow searching problem
  - Solution
    - Make Hashed table but it is complex in administration
    - Use of cache

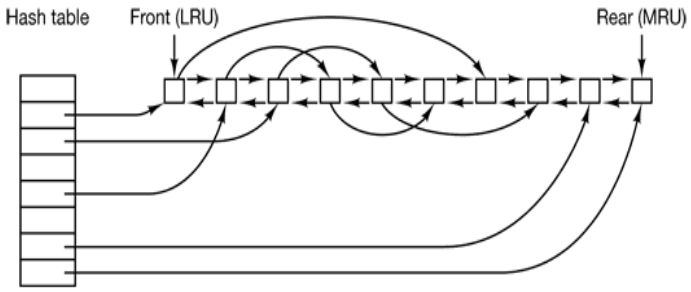
Disk Management

1. Sequence of Byes but problem if file size growth
2. Block Size

a. Size of block?
  i. If large wastage of memory for small file.
  ii. If Small then slow because of seek time and rotational time

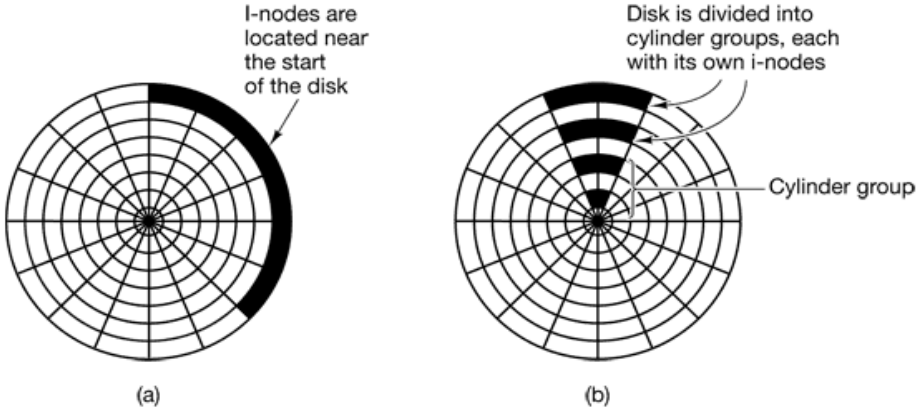
File system Performance

- Disk access is slower then memory access because of seek time and latency time (Rotational) so disk performance is worst then memory performance.
- Disk performance can be increase using following concept
  - *Block Cache or Buffer Cache*
  - *Block Read Ahead*
  - *Reduce Disk Arm Motion*
- Block Cache
  - Cache is a collection of blocks that logically belong on the disk but are being kept in memory for performance reasons.
  - First check whether the required block is in cache or not. If it is, the read request can be satisfied without a disk access. If the block is not in the cache, it is first read into the cache, and then copied to wherever it is needed. Subsequent requests for the same block can be satisfied from the cache.



  - There are many blocks on cache so we have to manage it so it is managed as shown in figure above as similar concept as page replacement algorithm in memory management.

- Block Read Ahead
  - Get blocks into the cache before they are needed to increase the hit rate.
  - File system ask to produce block *k* in a file, it does that, but when it is finished, it makes a sneaky check in the cache to see if block *k* + 1 is already there. If it is not, it schedules a read for block *k* + 1 in the hope that when it is needed, it will have already arrived in the cache. At the very least, it will be on the way
  - It is used in sequential file access but we need random access so sequential and random access attribute are attached in file attribute so that we can perform both access.
- Reduce Disk Arm motion
  - Keep blocks that are likely to be accessed in sequence close to each other, preferably in the same cylinder.
  - When an output file is written, the file system has to allocate the blocks one at a time, as they are needed. If the free blocks are recorded in a bitmap, and the whole bitmap is in main memory, it is easy enough to choose a free block as close as possible to the previous block.



- Another performance bottleneck in systems that use i-nodes or anything equivalent to i-nodes is that reading even a short file requires two disk accesses: one for the i-node and one for the block.
- From the above figure a it is shown that all the i-nodes are near the beginning of the disk, so the average distance between an i-node and its blocks will be about half the number of cylinders, requiring long seeks.
- Solution 1: Put the i-nodes in the middle of the disk, rather than at the start, thus reducing the average seeks between the i-node and the first block by a factor of two.
- Solution 2: divide the disk into cylinder groups, each with its own i-nodes, blocks, and free list. When creating a new file, any i-node can be chosen, but an attempt is made to find a block in the same cylinder group as the i-node. If none is available, then a block in a nearby cylinder group is used.

Example File System

- CD- ROM File System
- MS-DOS File System
- UNIX File System

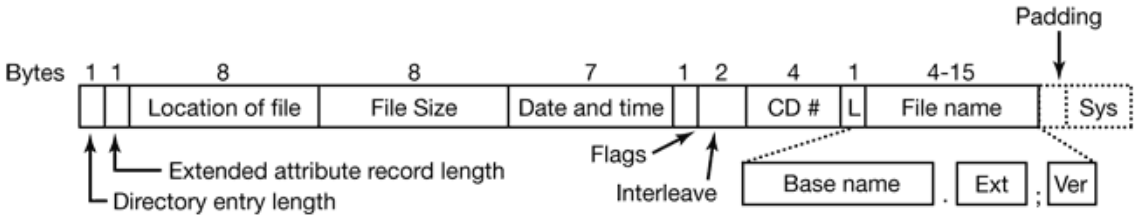
CD-ROM File System

The ISO 9660 File System

- First CD-ROM files system Standard in 1988 under the name **ISO 9660**.
- Do not have concentric cylinders the way magnetic disks do.
- Contain single continuous spiral containing the bits in a linear sequence, bits along the spiral are divided into logical blocks (also called logical sectors) of 2352 bytes. Some of these are for preambles, error correction, and other overhead.
- The payload portion of each logical block is 2048 bytes.
- It can be converted to a linear block number using the conversion factor of 1 sec = 75 blocks.
- ISO 9660 supports CD-ROM sets with as many as $2^{16} - 1$ CDs in the set. The individual CD-ROMs may also be partitioned into logical volumes (partitions). However, below we will concentrate on ISO 9660 for a single unpartitioned CD-ROM.
- Every CD-ROM have
  - Starting 16 blocks used by manufacturer to provide a bootstrap program to allow the

computer to be booted from the CD-ROM, or for some other purpose.

- One block for **primary volume descriptor**, Which contains
  - The system identifier (32 bytes), volume identifier (32 bytes), publisher identifier (128 bytes), and data preparer identifier (128 bytes).
  - Three file abstract, copyright notice, and bibliographic information, respectively.
  - Key numbers for logical block size (normally 2048, but 4096, 8192, and larger powers of two are allowed in certain cases), the number of blocks on the CD-ROM, and the creation and expiration dates of the CD-ROM.
  - A directory entry for the root directory, telling where to find it on the CD-ROM.
- CD-ROM may contain a supplementary volume descriptor that contains similar information to the primary
- The root directory contains



Shared Files

A tree structure prohibits the sharing of file or directory. A Directed Acyclic Graph (DAG) provides the sharing. It is a graph with no Circle, allowing directories to have shared directories and file.

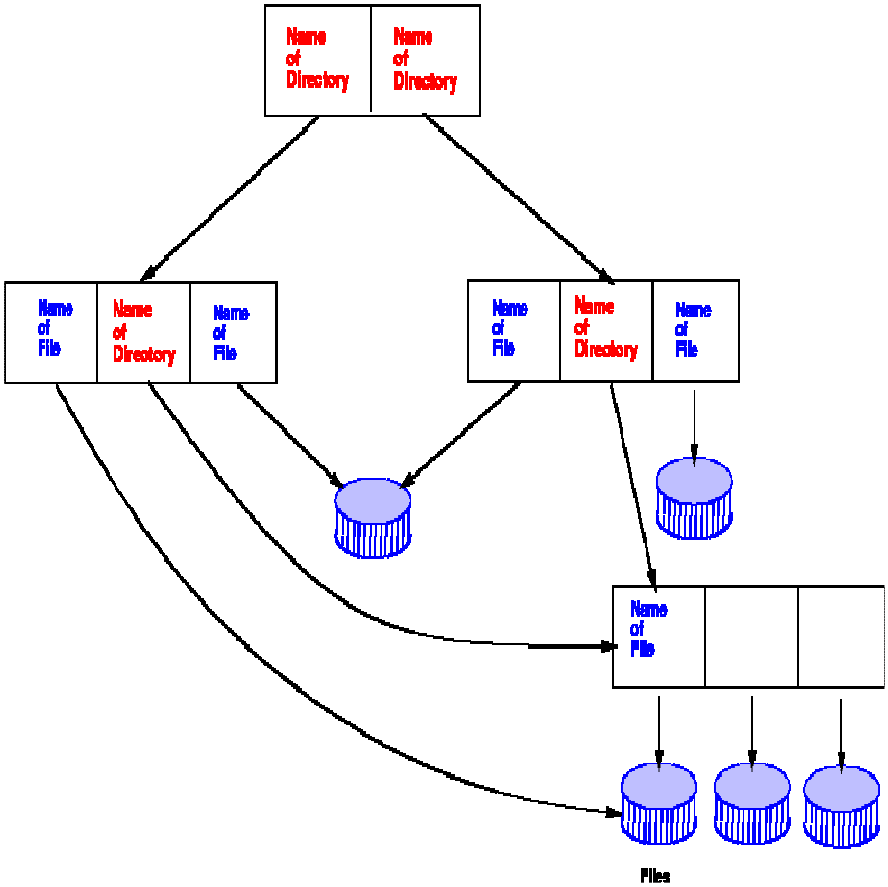


Fig: Directed Acyclic Graph

Sharing file is convenient but it also introduces some problem.

- Sharing file keep a copy of a disk address when file is linked. When thing is appended in the file, it doesn't appear in the other file.
- The above stated problem can be solved by the process where disk blocks are not listed the directories, but in the little data structure (I node) associated with the file itself. The directory would then point just to the little data structure. Problem: When file is removed and I node is cleared; the pointer in the other file is not deleted, so it may point wrong I node
- To solve the above stated problem symbolic linking is used. New file contains just the path name of the file to which it is linked. Problem: Extra overhead required.

Some points about DAG

- Naming : File may have multiple absolute path names
  - Two different names for the same file
- Traversal
  - ensure that shared data structures are traversed only once.
- Deletion
  - Removing file when someone deletes it may leave dangling pointers.
  - Preserve file until all references to it are deleted
    - Keep a list of all references to a file or
    - Keep a count of the number of references - *reference count.*
    - When count = 0, file can be deleted.

File system Reliability

Loss of data in a file system can have catastrophic effect (much worse than failure of hardware.) Need to ensure reasonable level of safety against data loss in the event of system failures. Three threats:

- accidental or malicious deletion of data by users
- media (disk) failure
- system crash during file system modifications, leaving data on disk in an inconsistent state

Backup: copy entire file system onto low-cost media (tape) at regular intervals (e.g., once a day). In the event of a failure of the primary media (disk), can replace media and restore data from backup media. Amount of data loss is limited to modifications that occurred since the last backup.

Mirrored disks: multiple copies of the files system are maintained on independent disks. Disk writes update all the redundant disks in parallel. Used in applications that cannot tolerate any data loss (e.g., banking).
RAID disks: use multiple parallel disk drives for higher throughput and increased reliability. For example: each bit of a data byte is stored on one of eight disks. A ninth disk stores a parity bit for each data byte. Parity bit is the result of an XOR among the eight data bits of the corresponding data byte. As a result, can still recover the data if one of the nine disks fails.

Versioning file system: file system created a new version every time a file is modified. Old versions are kept until explicitly deleted. Can use copy-on-write to reduce storage requirements.

After a system crash in the middle of a file system operation, file system metadata may be in an inconsistent state. (Invariants of the on-disk data structures may not hold.) Example: a file was deleted, but its disk blocks have not yet been added to the free list.

Solution 1: run a program during system startup that examines the entire file system, detects inconsistencies, and restores the invariants. In Unix, that programs is called fsck. Some of the conditions fsck checks and repairs:

- correct i-node reference counts
- missing blocks in the free list
- blocks that are both in the free list and part of a file
- incorrect information in the superblock
- out-of-range block and i-node numbers
- disconnected (unreachable) files and directories

Solution 2: keep a separate log

- Write log entry describing operations about to be performed
- perform the operation on the file system
- delete the log entry
- after a crash:
- check the log
- if non-empty, perform operations described in the log