10 – File Systems

10 – File Systems

- File concepts
- Access methods
- Directory structures
- Mounting
- File sharing
- File protection
- Implementation
- Disk scheduling

File Concepts

- Computer may store information on several different media, e.g., disk, tape, ...
- The operating system provides a uniform logical view of information storage.
  - Define a logical unit called a *file.*
  - Abstract from physical properties of its storage.
- A file has a certain defined structure according to its type.
  - Text, source, objects, executable, database, ...

File Structures

- None: a sequence of bytes or words
- Simple record:
  - Lines
  - Fixed length records
  - Variable length records.
- Complex structure:
  - Formatted documents

File Attributes

- Vary from one operating system to another
- Typically,
  - **Name**: symbolic, only human-readable information
  - **Identifier**: unique, identify the file within a file system
  - **Type**: describe type of the file
  - **Location**: pointer to position on the device
  - **Size**: current size of the file
  - **Protection**: access-control information
  - **Time, date, user identification**: informational, may be useful for security, protection, statistic, etc.

File Operations

- Create
  - Allocation space from the file system
  - Create new entry in the directory
- Write
  - Search for the location
  - Keep point to the next place to write
- Read
  - Search for the location
  - Keep point to the next place to read
- Seek
  - Positioning the *current-file-position pointer*

(cont'd.)

- Delete
  - Search for the location
  - Release the space
  - Remove the directory entry
- Truncate
  - Delete content of the file
  - File size must be updated

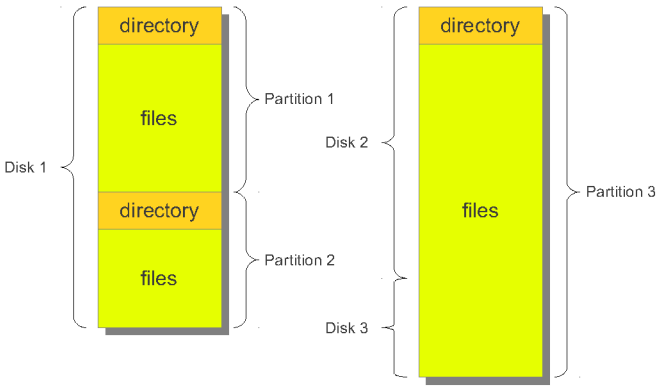
Access Methods

- Sequential access
  - Process in order
  - One unit (byte, word, record) after the other
- Direct access
  - Relative/random access
  - Based on disk media
  - No restriction on the order of access
- Indexed access
  - Index file/table to point to the location
  - e.g. Indexed Sequential Access Method (ISAM)

Directory Structures

- Disks may contain millions of files,
  - These files are needed to be organized properly.
- There are two parts:
  - Disks are split into *partitions* (or *minidisks*, or *volumes*)
    - Low-level structure in which files and directories reside.
    - Each treated as a separate storage device, whereas other systems may use
    - Also allow to group multiple disks into a single partition
  - Each partition contains information about files within a *device directory* or *volume table of contents*.
    - Efficient, locating a file quickly
    - Convenient for user to name, group, ...

(cont'd.)

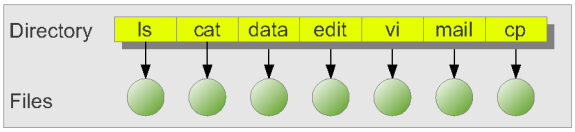


Operations on a directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Single-Level Directory

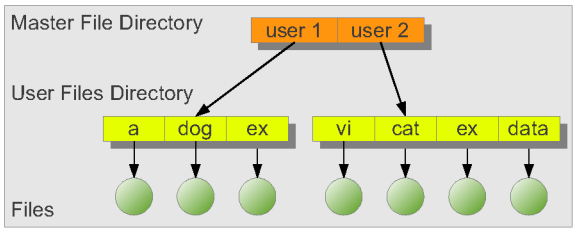
- All files are contained in the same directory.
- Limitation when the number of files increases, or has more than one user
  - All files must have unique names
  - An operating system allows only fixed number of characters file names.
    - DOS – 11 characters
    - UNIX – 255 characters



Two-Level Directory

- The system has a *master file directory* (*MFD*) is indexed by user name (or ID), each points to the *user file directory* (*UFD*).
- Allow same file name for different user
- Users access one another's file ?
  - Introduce a *path name* = user name + file name
  - Path name must be unique, file name may not.
- Path name may contain partition identifier, e.g.,
  - DOS: `C:\user\file`
  - VMS: `u:[user][file]`

(cont'd.)

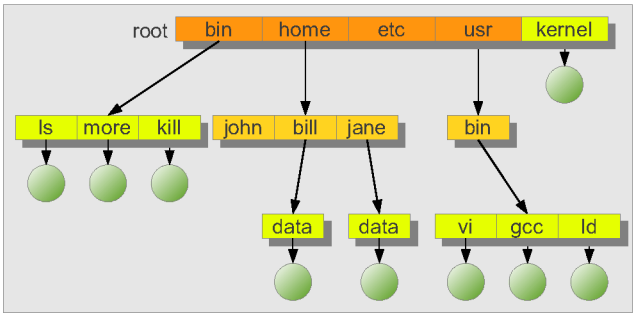


- Some operating systems allow to define the sequence of directories to be searched for a specified file – *search path*.

Tree-Structured Directory

- Most common directory structure
- Extend the directory structure to a tree of arbitrary height
- Allow user to create their own subdirectories and organize their files accordingly
- The topmost directory – *root*
- All path names must be unique, still.
- A directory or subdirectory contains a set of files or subdirectories.
- The *current directory* specifies the current working position in the tree structure.
  - A search path by default

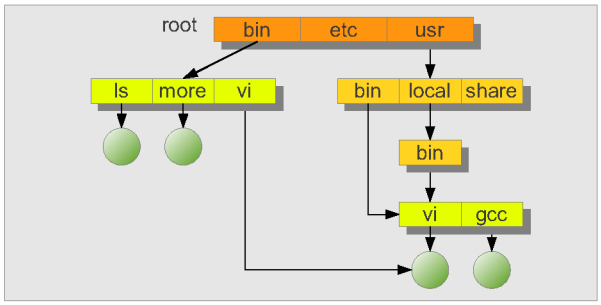
(cont'd.)



- Two types of path name: *absolute*, and *relative*

Acyclic-Graph Directory Structure

- Extend the tree structure to share files or directories

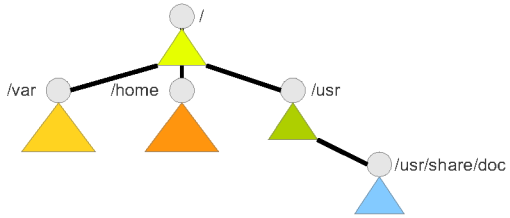


(cont'd.)

- A common way to share files and subdirectories is to create a *link*.
  - A pointer to another file or subdirectory
  - UNIX: symbolic link, hard link
  - Windows: shortcut
- If the file/subdirectory being pointed is deleted ?
- General Graph Directory allows cycles.
  - Loop, self-referenced, must be careful when search, delete, etc ...
  - Garbage collection to traverse through the file system and make sure that they are all fine.
    - This is time consuming, and not that easy to do so.

File-System Mounting

- A file system must be *mounted* before it can be available to process on the system.
- A directory structure can be built out of multiple partitions, each must be mounted to be available within the file system name space.

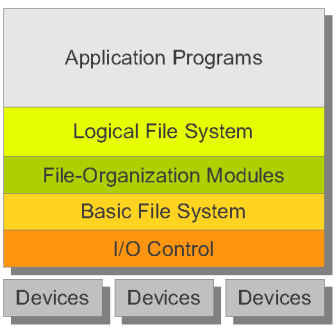


Protection

- Two aspects:
  - Reliability – protect from physical damages.
  - Protection – protect from improper accesses.
- Reliability
  - Redundant
  - File system software is also important.
- Protection
  - Access control, mostly, based on
    - Permissions – what can be done, e.g., read, write, execute, ...
    - User ID – by whom

File-System Implementation

- Generally, a file system is composed of many layers:



(cont'd.)

- **I/O control** consists of device drivers and interrupt handlers to transfer data between memory and storage devices.
- **Basic file system** issues device-specific commands to the devices.
- **File-organization modules** map logical blocks to physical blocks, manage disk space allocation.
- **Logical file system** manages logical structures, e.g., directory structure, file structure (via a *file control block*).
  - Also responsible for protection and security

On-Disk Structure

- A *boot control block* contains information needed by the system to boot up an operating system from that partition
  - Usually, it is the first block of a partition
    - UNIX: boot block
    - NTFS: (partition) boot sector
- A *partition control block* contains partition information, e.g., number of blocks, allocated blocks, free blocks, bootable flag, FCB pointers, ...
  - UNIX: superblock
  - NTFS: master file table

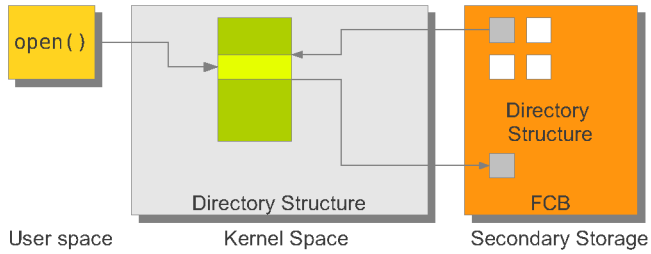
(cont'd.)

- A directory structure
- An FCB contains file information, e.g., permission, ownership, size, location of the data block.
  - UNIX: inode
  - NTFS: reside in the master file table

In-Memory File-System Structure

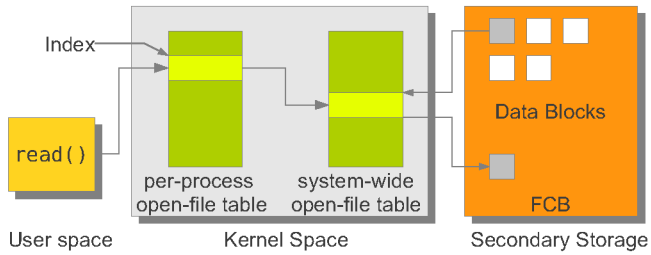
- To manage file system properly and efficiently
- The structure may includes:
  - An in-memory partition table contains all mounted partition.
    - Some system has to reboot after modifying partitions.
  - An in-memory directory structure contains recently accessed directories.
  - A system-wide open-file table contains a copy of the FCB of each opened file and other information.
  - A per-process open-file table contains a pointer to the appropriate entry in the system-wide open-file table and other information.

(cont'd.)



- The `open()` take a file name as an argument.
- An operating system searches for an opening file in the directory structure, which may be cached in memory to speed up the operation.

(cont'd.)



- Once it is found, the FCB is copied to the system-wide open-file table in memory.
- An entry in per-process open-file table is created. The `open()` returns a pointer to this entry.

(cont'd.)

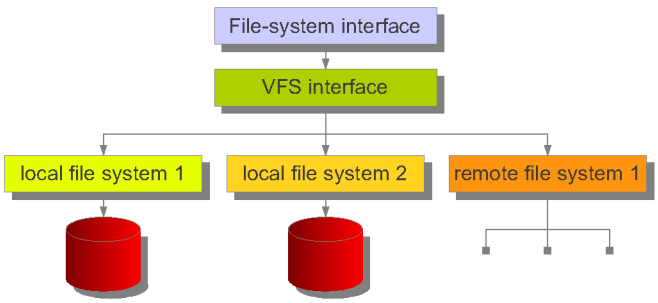
- An entry in the per-process open-file table is sometimes called a *file descriptor* (UNIX), or a *file handle* (MS Windows).
- The `read()` or `write()`, take the pointer as an argument.
- The operating system access the per-process open-file table to search an entry, then get the pointer to the system-wide open-file table to actually access the data in the secondary storage.

(cont'd.)

- When a process closes file, the entry is removed from the per-process open-file table.
- When all processes that have opened the file close it, the information, e.g., statistics, is copied back to the disk structure, and the entry in the system-wide open-file table is removed.
  - That's why, for some file systems, the file system is not consistent when the system has crashed.
  - For `write()` the operating system may immediately update the disk structure.

Virtual File Systems

- A layer to build a uniform interface for all different types of file systems.
  - Local file systems, remote file systems, ..



Directory Implementation

- Linear List
  - The simplest one, can be implemented by a linked list.
  - Linear search, easy to program, but it is slow and it is the real disadvantage of this kind of implementation.
  - The list may be sorted to improve the average search time.
  - B-Tree may also help.
- Hash Table
  - Directly compute the index of the entry from the file name.
  - Collision, chained

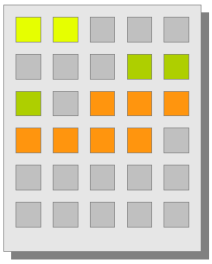
Allocation Methods

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk.
- Need only starting block and file length
- The number of seeks is minimal.
- Both sequential and direct access can be supported.
- Problems:
  - Finding space for a new file (dynamic storage allocation problem)
  - Allocate more space for existing file
  - External fragmentation
    - Compaction may help, but what about down time ?

(cont'd.)



Directory		
file	**start**	**length**
ls	0	2
more	8	3
vi	12	7

(cont'd.)

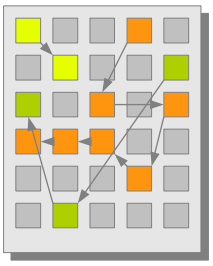
- Extent-based file allocation
  - A modified contiguous file allocation
  - An extent is a set of contiguous blocks.
  - Allocate a set of extents instead of blocks
  - Veritas File System

Linked Allocation

- Each file is a linked list of disk blocks.
- Directory contains pointers to the first and the last block.
- No external fragmentation
- Effectively only for sequential-access files
  - To access i^{th} block, you need to begin at the starting block :(
- Every block are required to store the pointer to the next block.
  - The last block points to nil.
  - Cluster (of blocks) for better space utilization

(cont'd.)

- Reliability
  - What if system failed during update the pointer ?
  - Double linked list may help, but what about overhead ?

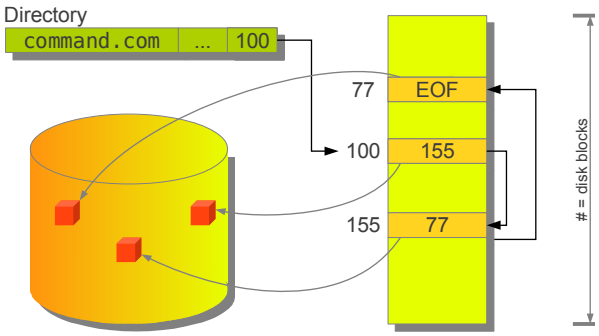


Directory

file	start	end
ls	0	6
more	9	10
vi	3	15

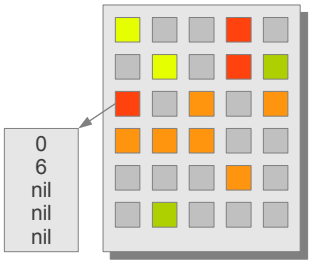
(cont'd.)

- File Allocation Table (FAT)



Indexed Allocation

- Link allocation cannot support direct access efficiently because a pointer is stored in each block.
- Index block keeps all the pointers.



Directory

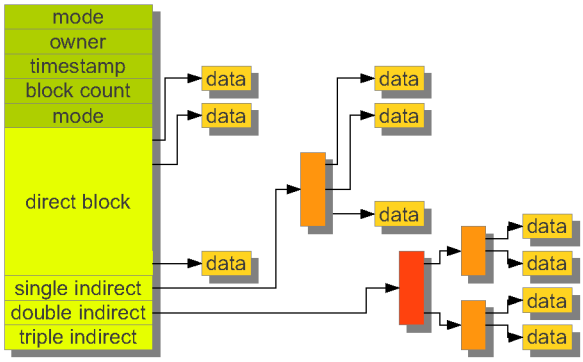
file	index block
ls	10
more	8
vi	3

(cont'd.)

- Overhead
  - Caching to the memory helps

In the Real World ..

- UFS's inode



Journaling File System

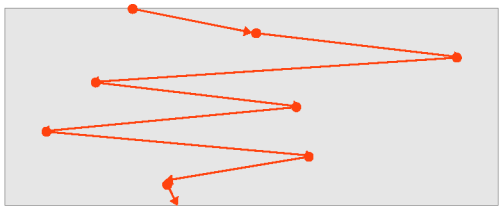
- a.k.a., log-structured file system
- Log changes to a journal before actually write to the file system.
- Reliable
  - If system has crashed, the operating system can replay those changes before the crash from the journal.
- Ext3, Ext4, JFS, UFS, LFS, NTFS, Veritas, Reiser4, ...

Disk Scheduling

- Disk access time = seek time + rotational time
  - Remember ?
- The operating system can do nothing to reduce rotational time
  - Well, actually, *interleave* may help
    - For those systems that their I/O are much slower than disk rotation.
- What about seek time ?
  - The operating system may schedule arm moving across cylinders to reduce seek time.

First Come First Serve (FCFS)

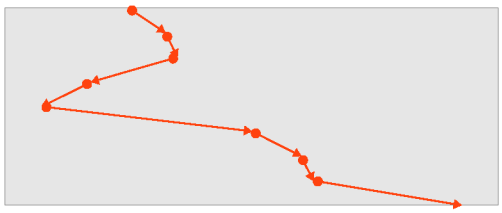
- Current Head Position = Cylinder #53
- Queue: 98, 183, 37, 122, 14, 124, 65, 67



- Total = 640 cylinders

Shortest Seek Time First (SSTF)

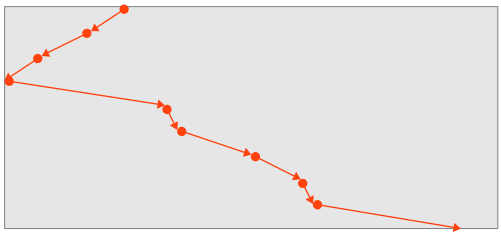
- Queue: 98, 183, 37, 122, 14, 124, 65, 67
- Schedule: 65, 67, 37, 14, 98, 122, 124, 183



- Total = 236 cylinders

SCAN

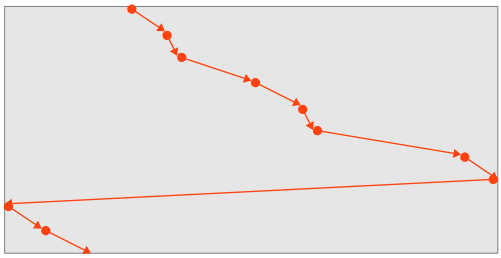
- Move arm from one end to the other
- Queue: 98, 183, 37, 122, 14, 124, 65, 67
- Schedule: 37, 14, 0, 65, 67, 98, 122, 124, 183



- Total = 236 cylinders

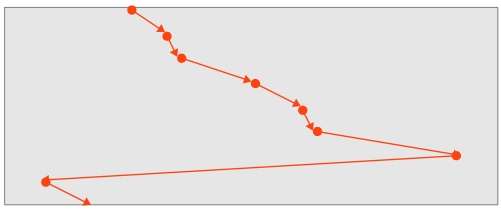
Circular SCAN (C-SCAN)

- Cylinder-as-circular-queue
- Queue: 98, 183, 37, 122, 14, 124, 65, 67
- Schedule: 65, 67, 98, 122, 124, 183, 199, 0, 14, 37



C-LOOK

- C-SCAN, but to the farthest cylinder in the queue
- Queue: 98, 183, 37, 122, 14, 124, 65, 67
- Schedule: 65, 67, 98, 122, 124, 183, 14, 37



Which algorithm ?

- SSTF is common and has a natural appeal.
  - But it may cause starvation.
- SCAN and C-SCAN perform better on heavy-disk-load systems.
- SSTF or C-LOOK is a reasonable choice for the default algorithm.
- Algorithms should be implemented as modules so that appropriate algorithm can be selected for each different situation.