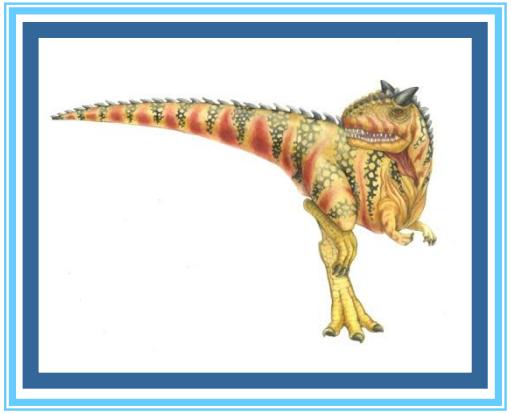
Xi'an Jiaotong-Liverpool University
西交利物浦大學

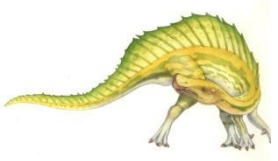
CPT104 - Operating Systems Concepts

STORAGE MANAGEMENT
File System



File-System

- File Concept
- Access Methods
- Directory and Disk Structure
- File-System Mounting
- File Sharing
- Protection
- File System Structure
- File-System Implementation
- Allocation Methods
- Free-Space Management



What Are File Systems?

- Everything is stored as files in a computer system. The files can be *data files* or *application files*.
- A file is a named, *linear region of bytes that can grow and shrink.*
- The operating system performs this management with the help of a program called **File System**.
- Different operating systems use different file systems.
  - **Windows**: NTFS (New Technology File System), exFAT (Extended File Allocation Table)
  - **Linux**: ext2, ext3, ext4, JFS, ReiserFS, XFS, and Btrfs.
  - **Mac**: Mac OS Extended File System or HFS+ (Hierarchical File System).
  - **Apple**: APFS (Apple File System) - Mac OS X, iPhones, iPads and other iOS devices

File System

FILE SYSTEM INTERFACE

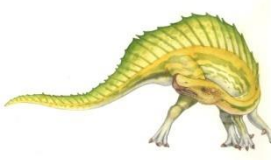
The user level (more visible) of the file system.

- Access methods
- Directory Structure
- Protection
- File-System Mounting
- File Sharing

FILE SYSTEM IMPLEMENTATION

The OS level (less visible) of the file system.

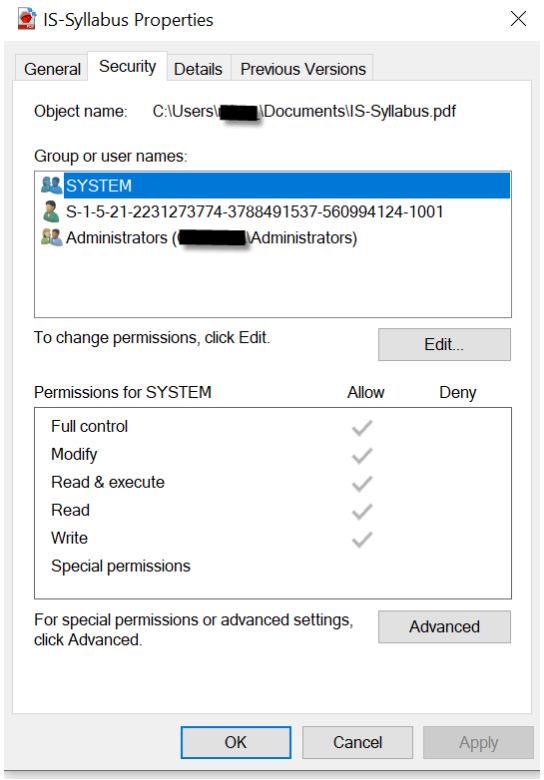
- Allocation and Free Space Management
- Directory Implementation

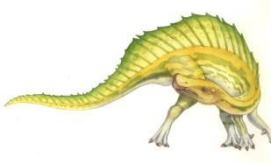


FILE SYSTEM INTERFACE

The user level of the file system.

- Access methods
- Directory Structure
- Protection
- File-System Mounting
- File Sharing





Attributes of a File

A file is a named, *linear region of bytes that can grow and shrink*.

- **Name** The user's visible name.
- **Type** The file is a directory, a program image, a user file, a link, etc.
- **Location** Device and location on the device where the file header is located.
- **Size** Number of bytes/words/blocks in the file.
- **Position** Current next-read/next-write pointers.
- **Protection** Access control on read/write/ execute/delete.
- **Usage** Open count
- **Usage** time of creation/access, etc.

In Memory Only!

There is a need to **PROTECT** files and directories.
Actions that might be protected include: **read, write, execute, append, delete, list**

Basic File Operations

Unix

- creat(name)
- open(name, how)
- read(fd, buf, len)
- write(fd, buf, len)
- sync(fd)
- seek(fd, pos)
- close(fd)
- unlink(name)

Windows NT

- CreateFile(name, CREATE)
- CreateFile(name, OPEN)
- ReadFile(handle, …)
- WriteFile(handle,…)
- FlushFileBuffers(handle,…)
- SetFilePointer(handle,…)
- CloseHandle(handle,…)
- DeleteFile(name)

File systems break files down into two logical categories:

- **Shareable** vs. **Unsharable** files
- **Variable** vs. **Static** files

- *Shareable files* - can be accessed locally and by remote hosts;
- *Unsharable files* - only available locally.
- *Variable files* - can be changed at any time;
- *Static files* - cannot be changed without an action from the system administrator (such as binaries).

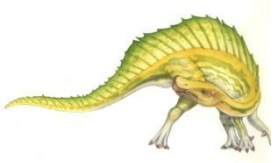
Open File Table - Since the open operation fetches the attributes of the file to be opened, the OS uses a data structure known as open file table (OFT), to keep the information of an opened file.

Access Methods

When it is used, the information must be accessed and read into computer memory.

The information in the file can be accessed in several ways:

- Sequential Access
- Direct Access
- Indexed Access

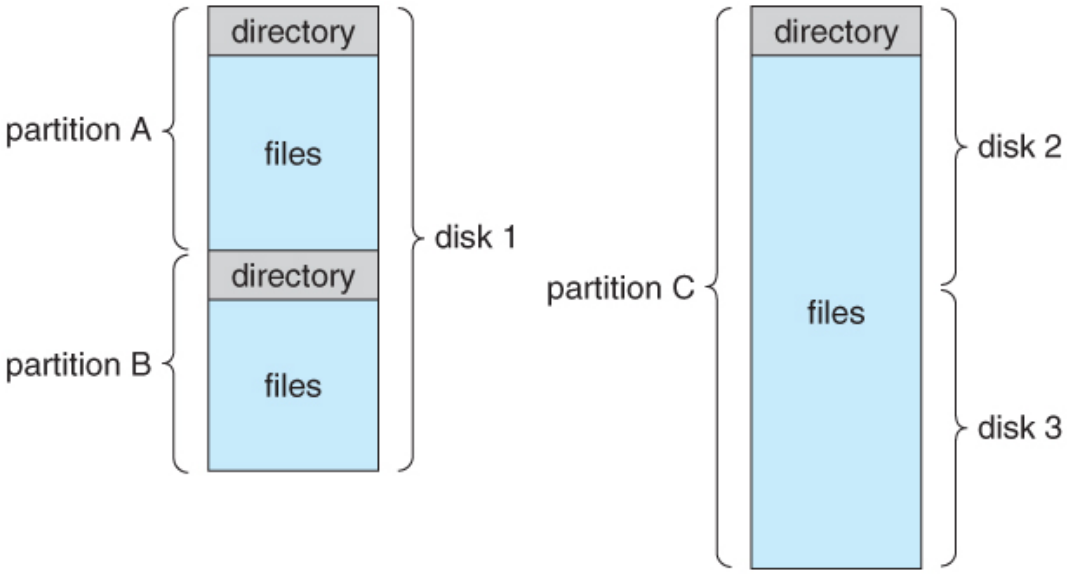


Access Methods

- **Sequential access**
  - Data is accessed one record right after the last.
  - Reads cause a pointer to be moved ahead by one.
  - Writes allocate space for the record and move the pointer to the new *End Of File.*
  - Such a method is reasonable for tape.
- **Direct access**
  - Method useful for disks.
  - The file is viewed as a numbered sequence of blocks or records.
  - There are no restrictions on which blocks are read/written in any order.
- **Indexed access**
  - Uses multiple indexes
  - An index block says what's in each remaining block or contains pointers to blocks containing particular items.
  - Suppose a file contains many blocks of data arranged by name alphabetically.

Disk Structure - Storage Structure

- A disk can be used in its **entirety** for a file system.
- A disk can be broken up into **multiple partitions**, **slices**, or **mini-disks**, each can have its own filesystem.
- Disk/partition is partitioned into **Blocks** or **Sectors**
  - Modern disks have 512-byte or more sectors
  - File Systems usually work in block sizes of 4 KB



Directory

The directories are used to maintain the structure of a file system.

Directories serve two purposes:

- For **User** – they provide a structured way to organize files;
- For the **File System** – they provide an *interface* that allows the implementation to separate logical file organization from physical file placement on the disk.

Operations Performed on Directory

- **Search** for a file
- **Create** a file
- **Delete** a file
- **List** a directory
- **Rename** a file
- **Traverse** the file system - to access every directory and every file within a directory structure.

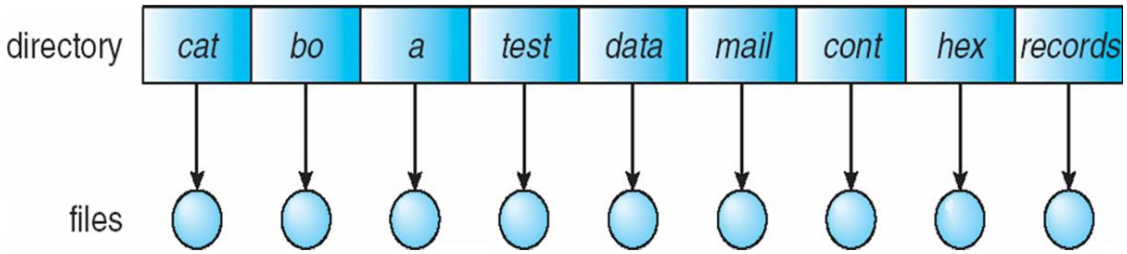
/	ufs
/devices	devfs
/dev	dev
/system/contract	ctfs
/proc	proc
/etc/mnttab	mntfs
/etc/svc/volatile	tmpfs
/system/object	objfs
/lib/libc.so.1	lofs
/dev/fd	fd
/var	ufs
/tmp	tmpfs
/var/run	tmpfs
/opt	ufs
/zpbge	zfs
/zpbge/backup	zfs
/export/home	zfs
/var/mail	zfs
/var/spool/mqueue	zfs
/zpbg	zfs
/zpbg/zones	zfs

Solaris file systems.

Schemes of logical structure of a directory

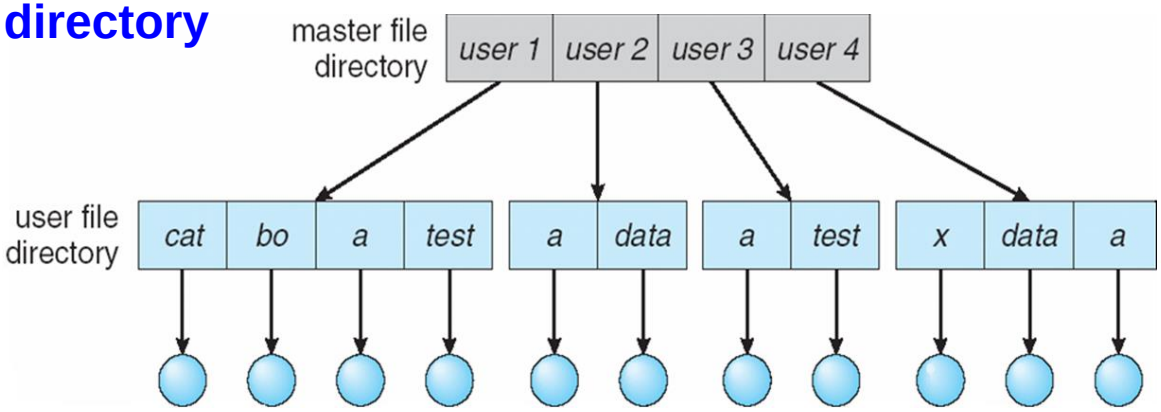
Single-Level Directory

- root director, and all the files are stored only under it

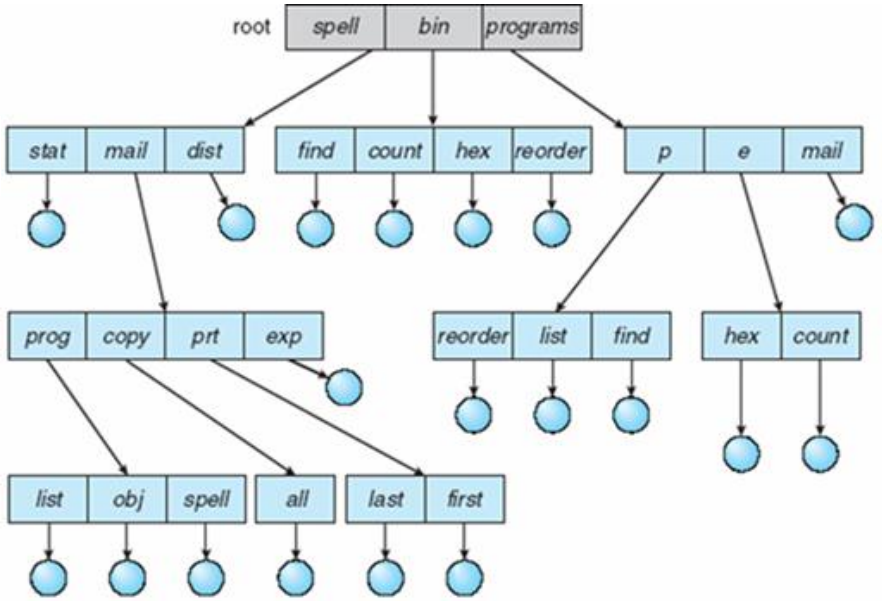


Two-Level Directory

- separate directories for each user
- there are two levels: **master directory** and **user directory**



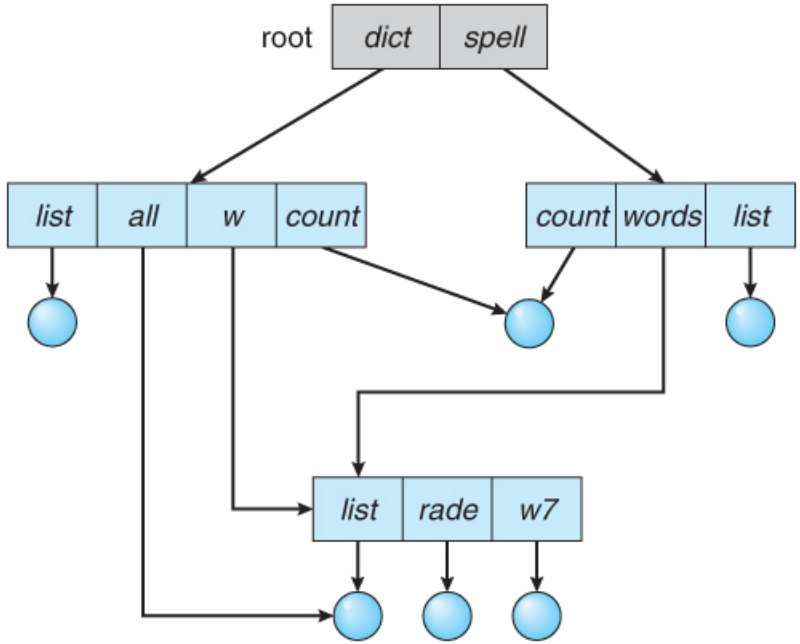
Hierarchical / Tree-Structured Directories



Two types of paths:

- **absolute path** - the root and follows a path down to the specified file
- **relative path** - defines a path from the current directory.

Acyclic-Graph Directories

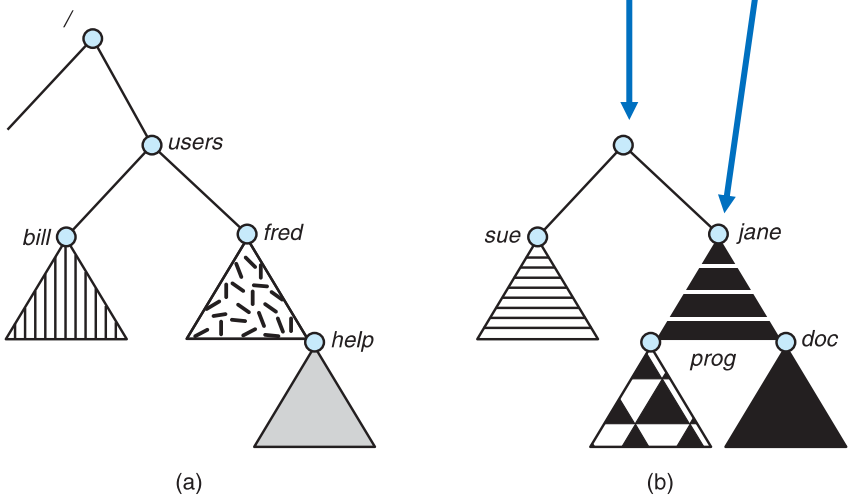


- the directory structure must allow sharing of files or sub-directories.

File System - Mounting

Mounting = attaching portions of the file system into a directory structure.

- The directory where the device is attached, is known as a **mount point**.
- Similarly, **unmounting** is done to remove the device from the mount point.



In **Windows**, the devices and/or partition can be accessed by opening *My Computer* on the desktop.

File Sharing

Sharing must be done through a **protection** scheme

May use **networking** to allow file system access between systems

- **Manually** via programs like **FTP** or **SSH**
- **Automatically**, seamlessly using **distributed file systems**
- **Semi automatically** via the **world wide web**

Client-server model allows clients to mount *remote file systems* from servers

- Server can serve multiple clients
- Client and user-on-client identification is insecure or complicated
- **Network File System NFS** is standard *UNIX client-server file sharing* protocol
- **Common Internet File System CIFS** is standard *Windows* protocol
- Standard operating system file calls are translated into *remote calls*

Distributed Information Systems implement unified access to information needed for remote computing (LDAP, DNS, NIS, Active Directory).

Protection

- Protection mechanisms provide **controlled access by limiting the types of file access that can be made.**
- File owner/creator should be able to control:
  - **what can be done**
  - **by whom**

Types of access

Read

Write

Execute

Append

Delete

List

```
-rw-rw-r--   1 pbg   staff     31200  Sep 3 08:30   intro.ps
drwx------   5 pbg   staff       512  Jul 8 09.33   private/
drwxrwxr-x   2 pbg   staff       512  Jul 8 09:35   doc/
drwxrwx---   2 jwg   student     512  Aug 3 14:13   student-proj/
-rw-r--r--   1 pbg   staff      9423  Feb 24 2012   program.c
-rwxr-xr-x   1 pbg   staff     20471  Feb 24 2012   program
drwx--x--x   4 tag   faculty     512  Jul 31 10:31  lib/
drwx------   3 pbg   staff      1024  Aug 29 06:52  mail/
drwxrwxrwx   3 pbg   staff       512  Jul 8 09:35   test/
```

Access-Control List and Groups

- general scheme to implement **identity dependent access** is to associate with each file and directory an **access-control list (ACL)** specifying usernames and the types of access allowed for each user.

Mode of access: **read, write, execute** (R, W, X)

The classifications:

a) **owner access** - the user who created the file is the owner.

b) **group access** - a set of users who are sharing the file and need similar access is a group, or work group.

c) **public access / universe** - all other users in the system constitute the universe.

Example 1

The following is an access verification technique, listing several files and the access allowed for a single user. Identify the control technique used here and for each, explain the type of access allowed.

a. File_1 RX

b. File_12 RWX

c. File_13 RW

d. File_14 X

Solution

This is an access control list.

a. File_1 This user can Read and Execute File 1.

b. File_12 This user can Read, Write, and Execute File 12 .

c. File_13 This user can Read and Write File 13.

d. File_14 This user can only Execute File 14 but cannot Read, Write the file.

Example 2

A leader of a group on a project wants Alice, Bob and Eve to be able to **read** and **write**, but **not delete data** on the project directory.
Peter and John may be allowed **only to read the files** under the project directory.
Besides this, the project leader should have **all access rights**.
The mapping of every type of user in the project with their access rights is as depicted as follows:

Solution:

TYPE OF USER	PERMISSIONS
Alice, Bob and Eve	Read and Write
Peter and John	Read Only
Project leader	All

Example 3

Alice can read and write to the file **X**, can read the file **Y**, and can execute the file **Z**.

Bob can read X, can read and write to **Y**, and cannot access **Z**.

Write a set of *Access Control Lists* for this situation. Which list is associated with which file?

Solution:

ACL for:

X = (Alice, read/write), (Bob, read)

Y = (Alice, read), (Bob, read/write)

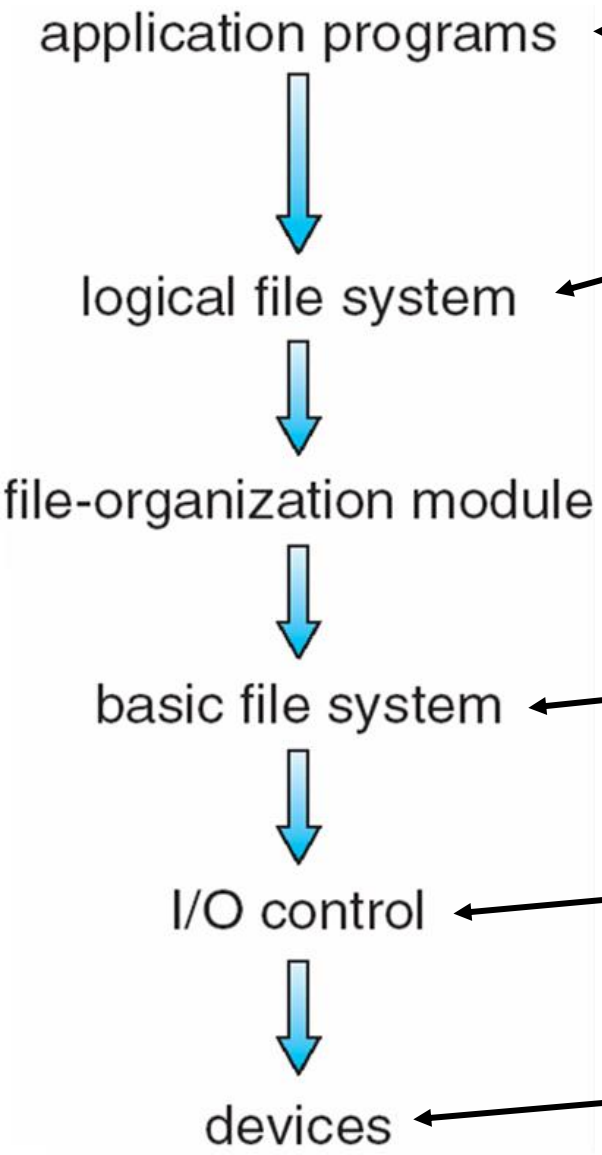
Z = (Alice, execute), (Bob, -----)

FILE SYSTEM IMPLEMENTATION

The OS level of the file system.

- Allocation and Free Space Management
- Directory Implementation

Layered File System



handles the CONTENT of the file.
knows the file's internal structure.

manages all the information about file attributes.
manages the directory structure where a file is located.

knows about the logical blocks of the file and translates them to physical blocks.
File allocation methods

determines where on the disk blocks are located to read and write physical blocks.

communicates with the device, starts operation on the device and processes the I/O request.

File system resides on secondary storage (disks).

File System Implementation

File system needs to maintain *on-disk* or *in-memory* structures

- **On-disk** for data storage.

  On disk, the file system may contain information about how to boot an operating system stored there, the total number of blocks, the number and location of free blocks, the directory structure, and individual files.

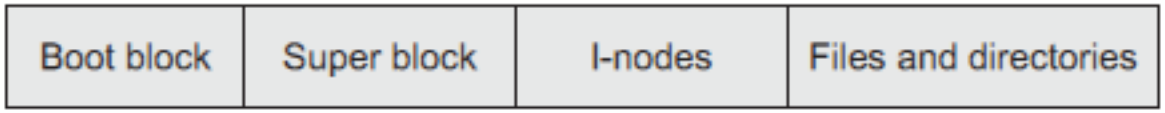
- **In-memory** for data access.

  The in-memory information is used for both file-system management and performance improvement via caching.

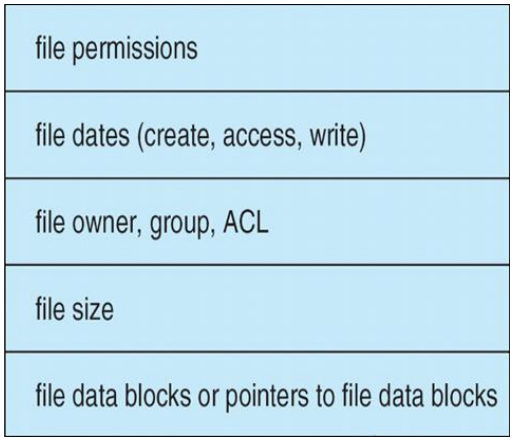
File System Implementation

On-disk Structure

- **Boot control block** is the first block of volume, and it contains information needed to boot an operating system.
- **File Control Block** (FCB *per file*) contains details about file, and it has a unique identifier number to allow association with directory entry. It is also known as ***Inode*** (*Index node*).
- **Volume control block / Superblock** contains volume/partition details: no. of blocks in the partition, block size, free block count, block pointers, etc.
- **Files and Directory Structure** stores file names and associated file names.



Disk layout implementing the file system



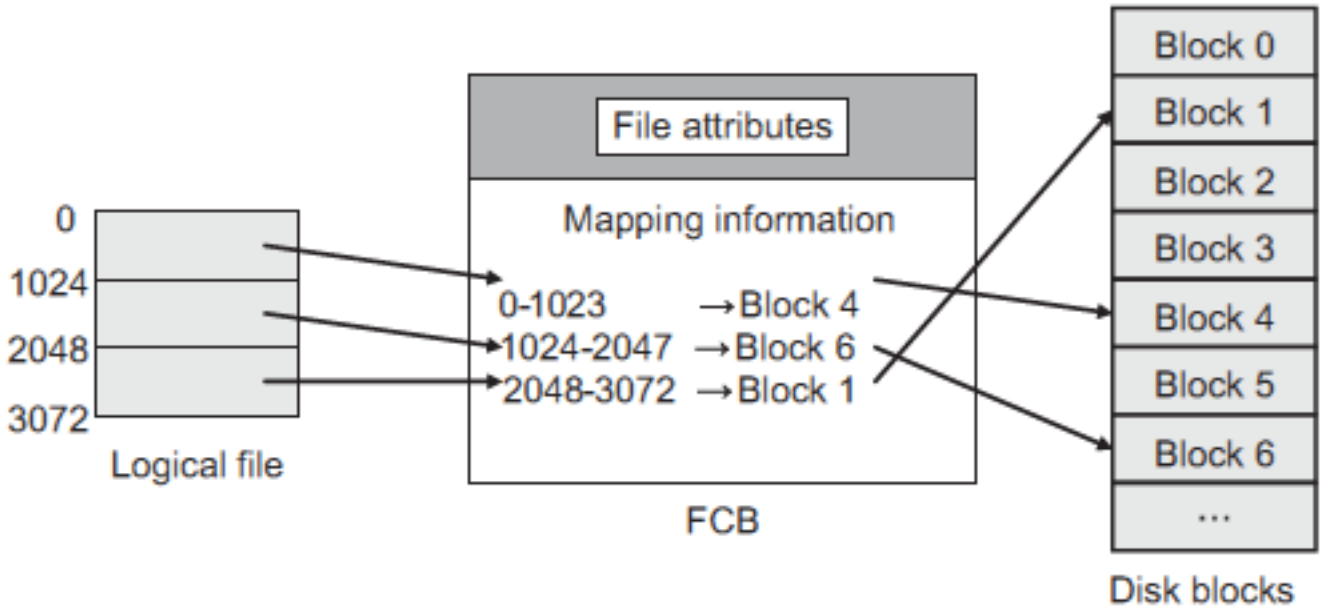
File Control Block

File mapping through FCB

The file system uses the **logical position** in a file **stored by the FCB** to **map** it **to a physical location on the disk**.

The FCB contains a list of blocks of a file and their corresponding disk block addresses.

To retrieve the data at some position in a file, the file system first translates the logical position to a physical location in the disk.



File System Implementation

In-memory Structure

- **Mount table** stores file system mounts, mount points, file system types.
- **Directory** (structure cache) holds information of recently accessed directories.
- **System-wide open-file table SOFT** maintains the information about the open files in the system.
- **Per-process open-file table OFT** maintains the detail of every file opened by a process and an entry in the OFT points to a SOFT.
- **Buffer area** is a temporary storage area in the memory for assisting in the reading/writing of information from/to disk.

In-memory data structures implementing the file system

Structure of the SOFT

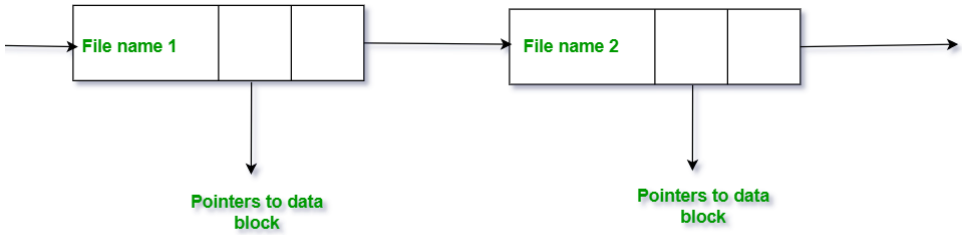
File name	FCB	Open_count
C:/admin/dbase/fileA.doc	File permissions File access dates File owner File size Address of file on disk	

Directory Implementation in O.S.

A file system uses directory to provide a way to name and organize multiple files. Directory implementation in the operating system can be done as:

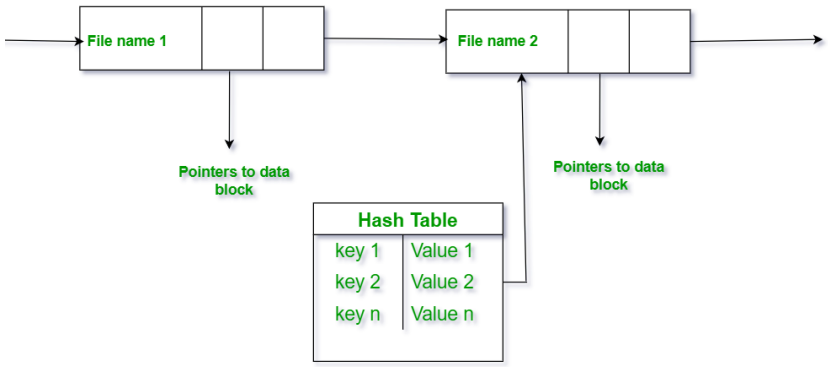
Linear list - all the files in a directory are maintained as single lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

- simple to program



Hash Table – the hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.

- it can greatly decrease the directory search time.



Allocation methods

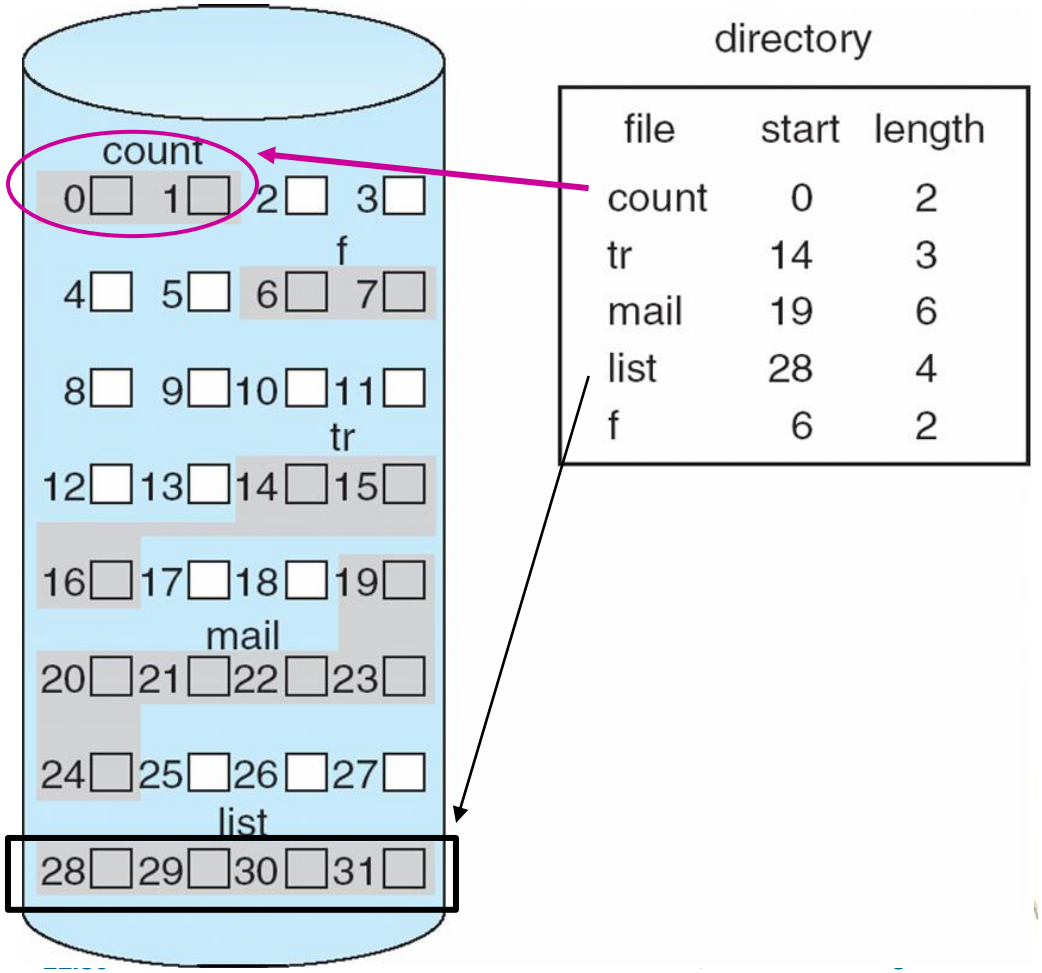
How are disk blocks allocated to files?

There are three major allocation strategies of storing files on disks:

- **contiguous,**
- **linked,**
- **indexed.**

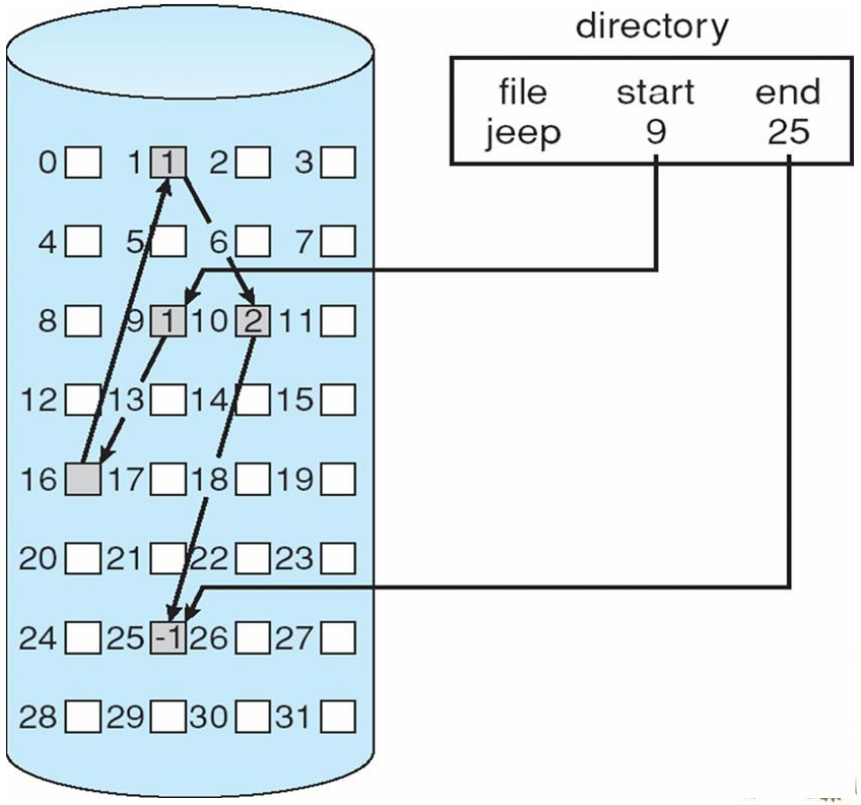
1. Contiguous Allocation

- Each file occupies a set of **contiguous** blocks on the disk.
- **Simple** – only starting location (block #) and length (number of blocks) are required
- **Random access**
- **Wasteful of space** (dynamic storage-allocation problem)
- Files cannot grow
- **external fragmentation**, need for compaction off-line (downtime) or on-line



2. Linked Allocation

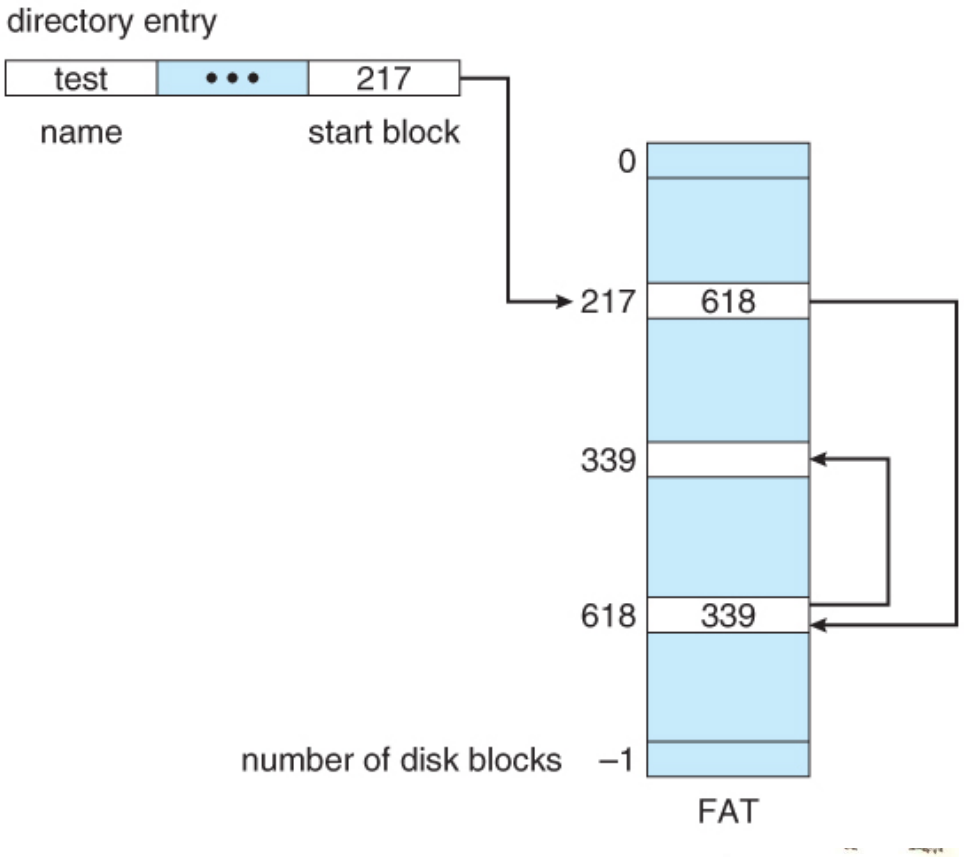
- each file is a **linked list of disk blocks**
- only efficient for **sequential access files**,
- **random access** requires starting at the beginning of the list for each new location access.



Example

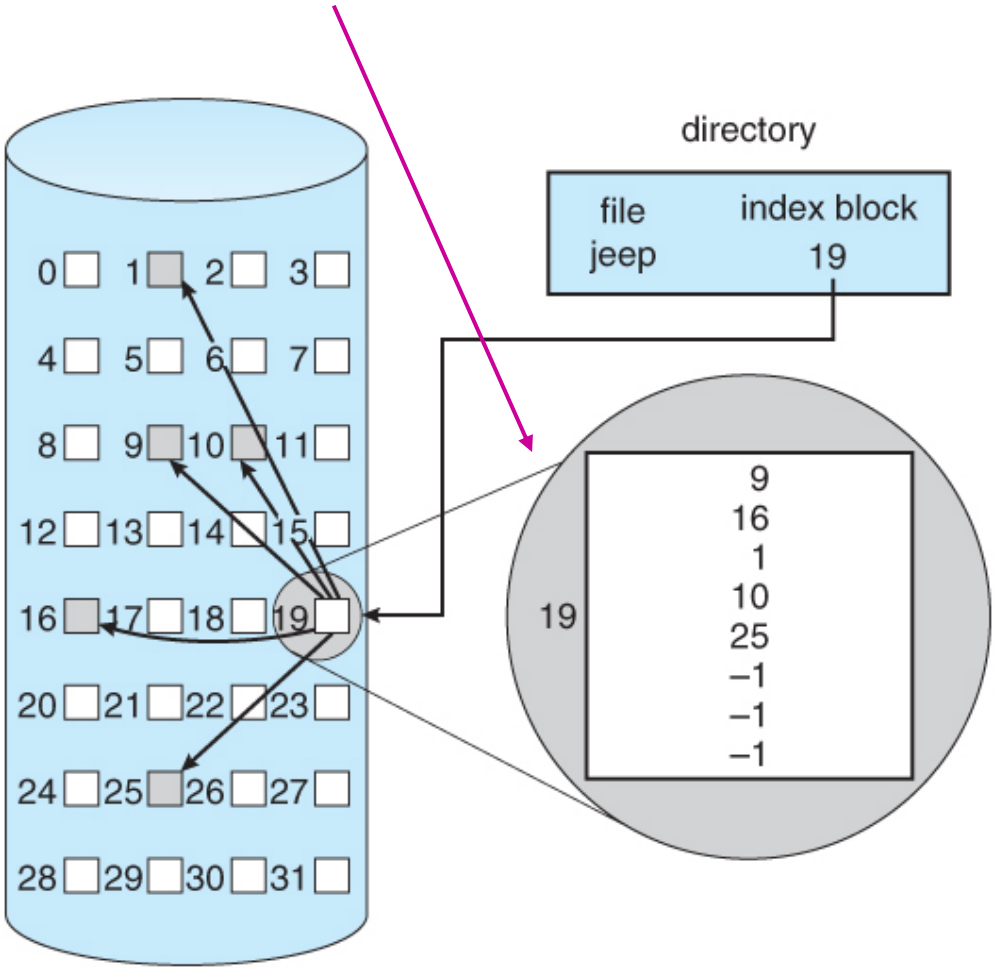
The **File Allocation Table**, **FAT**

- used by MS-DOS operating system
- all the links are stored in a separate table at the beginning of the disk.



3. Indexed Allocation

- Each file has its own index block(s) of pointers to its data blocks
- Brings all pointers together into the **index block**.
- Need **index table**
- **Random access**
- **Dynamic access**



Unix Inode

- An **inode** is a data structure in the Unix File System (UFS) to list the addresses of a file's data blocks.
- Each file or directory is associated with an inode, which contains metadata such as the *file's permissions, ownership, size*, and *location* on the disk.
- When a file is created, the operating system allocates an inode and assigns it to the file.
- The inode keeps track of the physical location of the file's data on the disk. This allows the operating system to efficiently locate and manage files and directories.
- Inodes are crucial for the file system to organize and manage files and directories. They play a key role in file system operations such as file *creation*, *deletion*, and *access control*.

How large index block should be implemented?

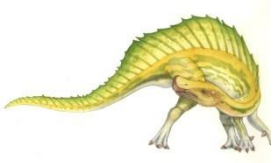
There are several approaches:

- **Linked Scheme** - an index block is one disk block, which can be read and written in a single disk operation.

To allow for large files, several index blocks will be link together.

- **Multi-Level Index** - uses a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks. To access a block, the operating system uses the first-level index to find a second-level index block and then uses that block to find the desired data block.

- **Combined Scheme** – see next!



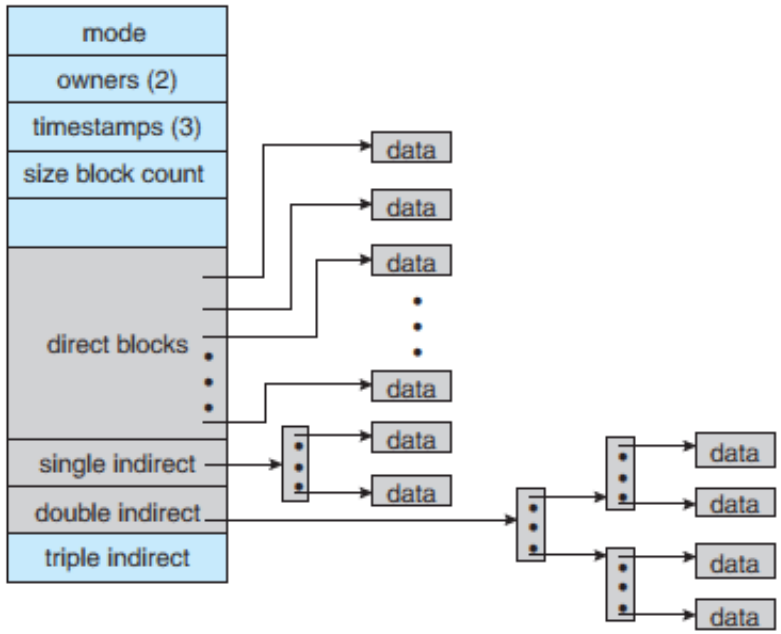
Unix Inode (Combined scheme)

Unix uses an indexed allocation structure.

- An **inode** (indexed node) stores both the attributes (infos) and the pointers to disk blocks.

The **index block** in the file's inode contains **15 pointers** as follows:

- 12 **direct blocks** - that point directly to blocks
- 1 **indirect block** - points to indirect blocks; each indirect block contains pointers that point directly to blocks
- 1 **double indirect block** - that points to doubly indirect blocks, which are blocks that have pointers that point to additional indirect blocks
- 1 **triple indirect block** – etc.



Free-Space Management

Disk management maintains free-space list to track available blocks / free space.

- **Bit Vector** - each bit represents a **disk block**, set to **1 if free** or **0 if allocated**.
- **Linked List** - **link together all the free disk blocks**, keeping a pointer to the first free block **.**
- **Grouping** - stores the **addresses of *n* free blocks in the first free block.**
- **Counting** - **the number of contiguous free blocks**.
- **Space Maps** - free-space list is implemented as a **bit map**, bit maps must be modified both when blocks are allocated and when they are freed.

End of Lecture

- **Summary**
  - File System Interface
  - File System Implementation
- **Reading**
  - Textbook 9th edition, **Ch.11 + Ch.12 of the module textbook**