File-System

File Concept

- A file is a collection of related information that is recorded on secondary storage
- Contiguous logical address space
- Types:
  - Data files
  - Program files
- File content is defined by its creator
- Many different types of information may be stored in a file - source or executable programs, numeric or text data, imagess, music, video, and so on.

File concept (contd…)

- A file has a certain defined structure, which depends on its type.
  - A **text file** is a sequence of characters organized into lines
  - A **source file** is a sequence of functions, each of which is further organized as declarations followed by executable statements
  - An **executable file** is a series of code sections that the loader can bring into memory and execute.

File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring

File Operations

- **Creating a file**
- **Writing a file**
- **Reading a file**
- **Repositioning within a file - seek**
- **Deleting a file**
- **Truncating a file**
- **Renaming a file**

Open Files

- The OS Keeps a table, called the **open-file table**, containing information about all open files.
- Several pieces of data are associated with an open files:
  - **File pointer**: pointer to last read/write location, per process that has the file open
  - **File-open count**: counter of number of times a file is open
  - **Disk location of the file**: cache of data access information
  - **Access rights**: per-process access mode information

Open File Locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock

- Operating systems may provide either **mandatory** or **advisory** file-locking mechanisms:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

File Types – Name, Extension

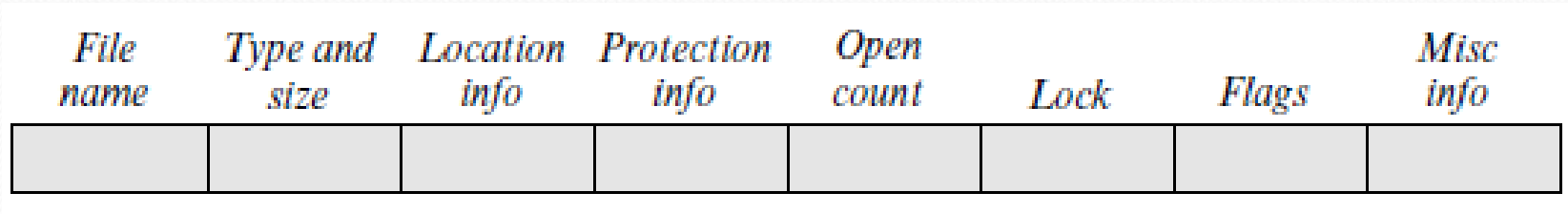
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access Methods

- The information in the file can be accessed in several ways
  - **Sequential Access** - Information in the file is processed in order, one record after the other
  - **Direct Access or Relative Access** - a file is made up of fixed-length **logical records** that allow programs to read and write records rapidly in no particular order
    - The direct-access method is based on a disk model of a file, since disks allow random access to any file block
  - **Indexed Access** – it involves in the construction of an index for the file.
    - The **index** contains pointers to the various blocks.
    - To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.

Directories

- A directory contains information about a group of files
- Each entry in a directory contains the attributes of one file



- File name - Name of the file
- Type and size -The file's type and size
- Location info - Information about the file's location on a disk.
- Protection info - Information about which users are permitted to access this file and in what manner.

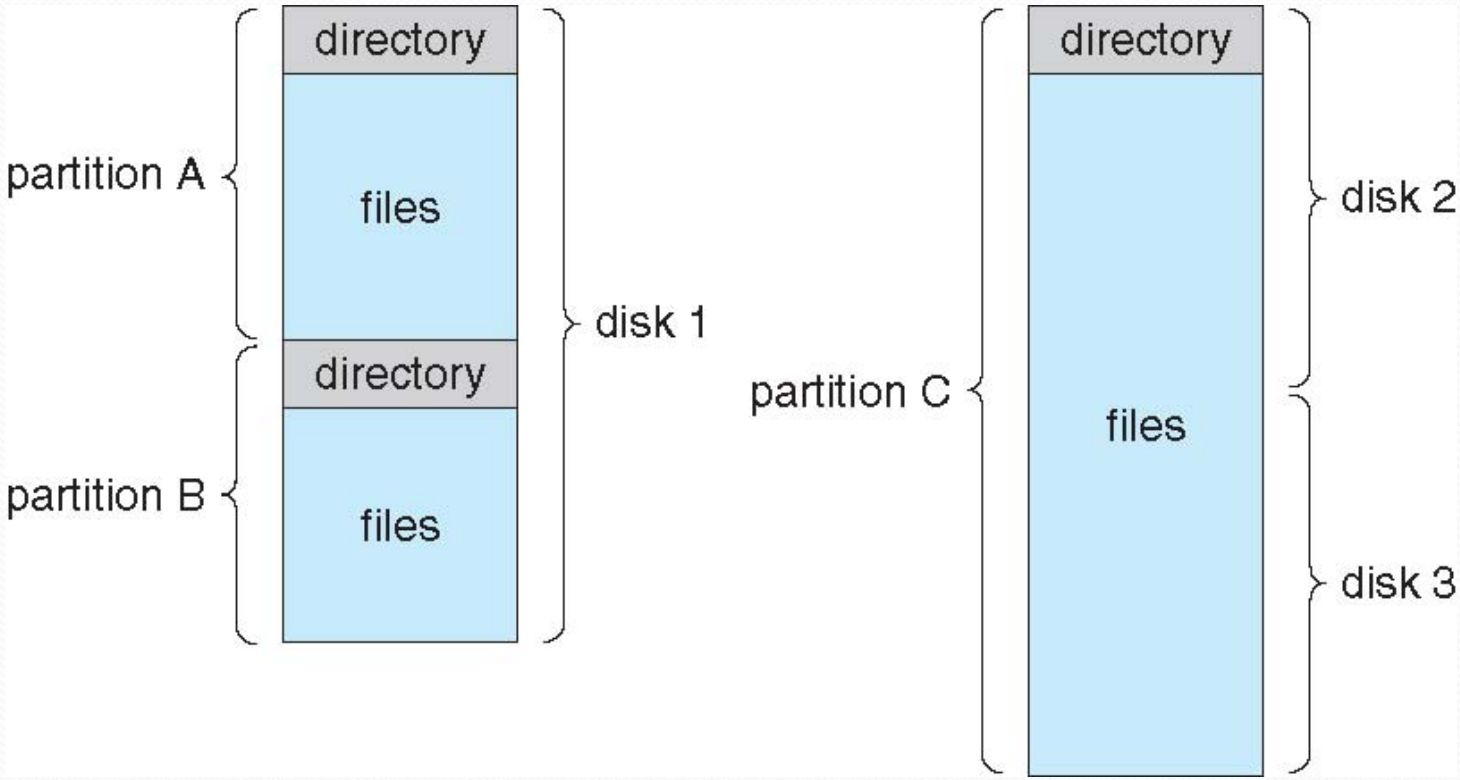
Directories (contd...)

- Open count - Number of processes currently accessing the file.
- Lock - Indicates whether a process is currently accessing the file in an exclusive manner.
- Flags - Information about the nature of the file—whether the file is a directory, a link, or a mounted file system.
- Misc info - Miscellaneous information like id of owner, date and time of creation, last use, and last modification.

Directory and Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a **volume**

A Typical File-system Organization



Directory

The directory can be viewed as a symbol table that translates file names into their directory entries

Operations performed on Directory

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

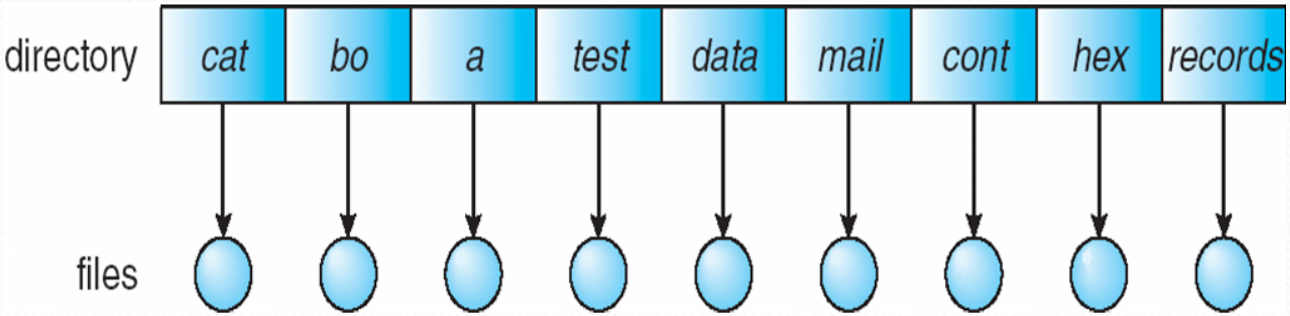
Directory Organization

The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

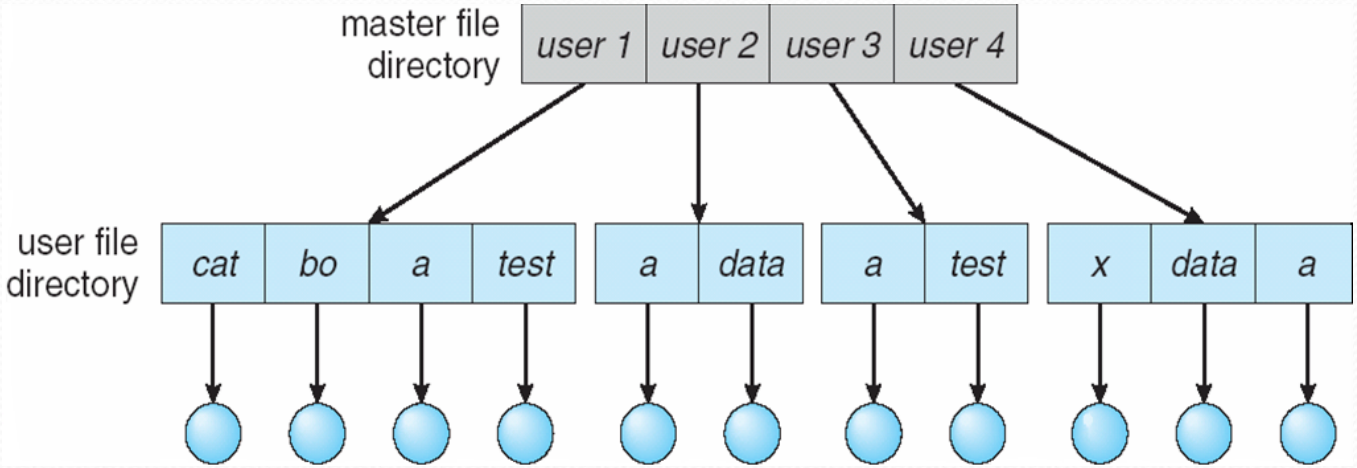
- A single directory for all users



- Naming problem
- Grouping problem

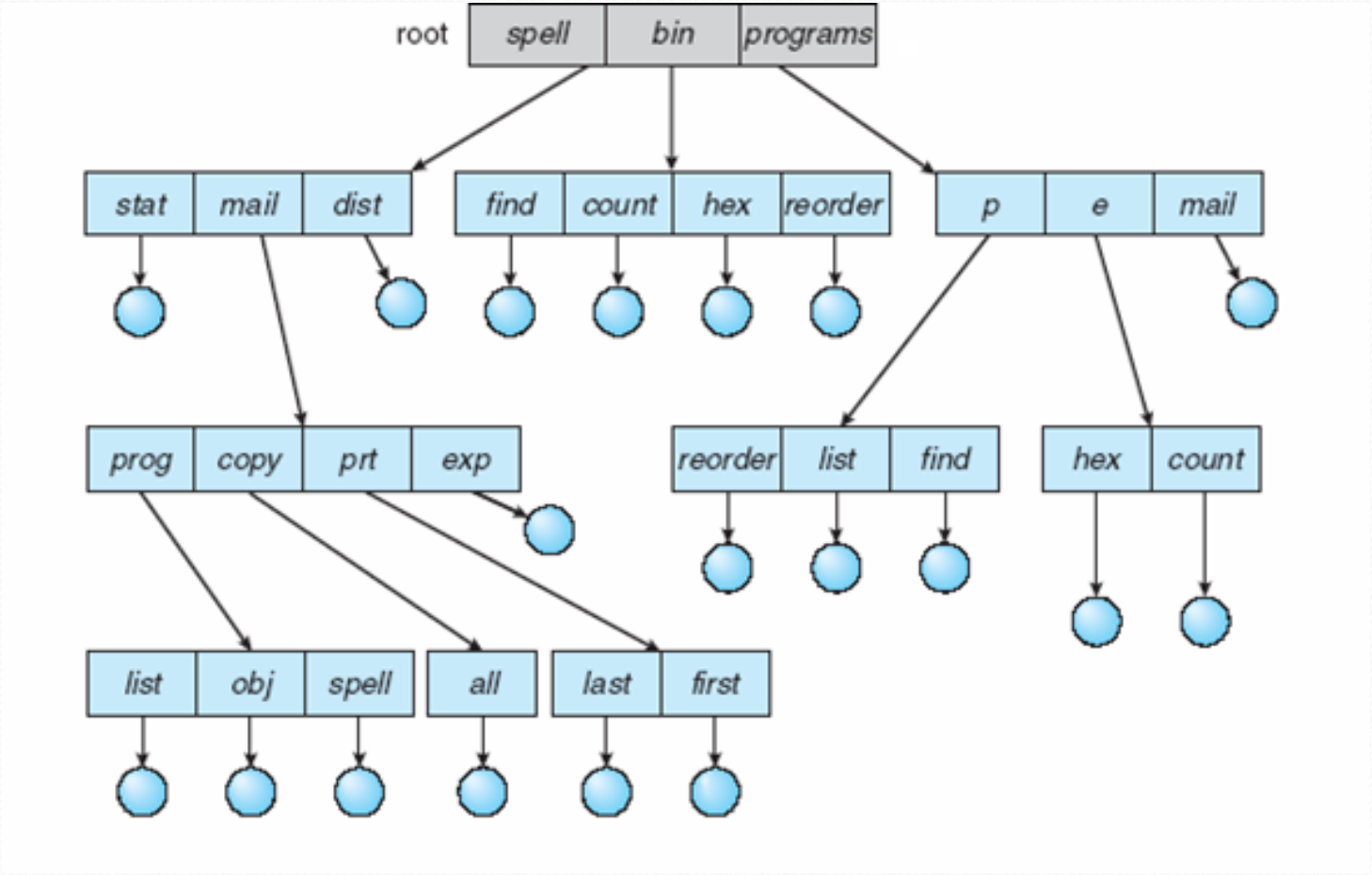
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



Tree-Structured Directories (Cont.)

Advantage:

- Efficient searching

- Grouping Capability

Disadvantage :

- Sharing of file – not possible

Tree-Structured Directories (Cont)

Path names can be of two types:

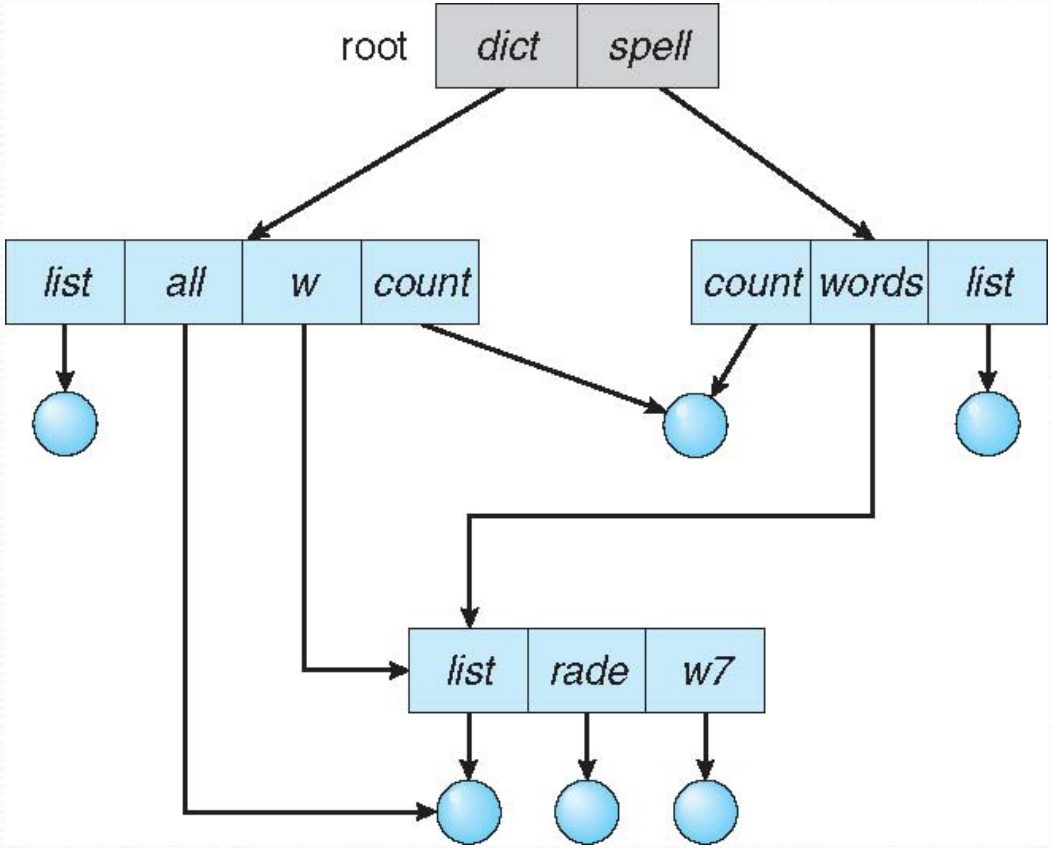
- An **absolute path name** begins at the root and follows a path down to the specified file, givingthe directory names on the path.
- A **relative path name** defines a path from the current directory.

An interesting policy decision in a tree-structured directory concerns how to handle the deletion of a directory.

If a directory is empty, its entry in the directory that contains it can simply be deleted.

Acyclic-Graph Directories

- Graph with no cycles
- Have shared subdirectories and files



Acyclic-Graph Directories

- Shared files and subdirectories can be implemented in several ways.
- A common way is to create a new directory entry called a link.
- A **link** is effectively a pointer to another file or subdirectory
- An acyclic-graph directory structure is more flexible than a simple tree structure, but it is also more complex.
- A file may now have multiple absolute path names.
- Another problem involves deletion

File System Mounting

- Each file system is constituted on a *logical disk*, i.e., on a partition of a disk.
- Files contained in a file system can be accessed only when the file system is *mounted*
- The mount operation is what "connects" the file system to the system's directory structure
- An unmount operation disconnects a file system.
- The operating system is given the name of the device and the **mount point**—the location within the file structure where the file system is to be attached
- Typically, a mount point is an empty directory
- A unmounted file system is mounted at a **mount point**

Mounting of a File System

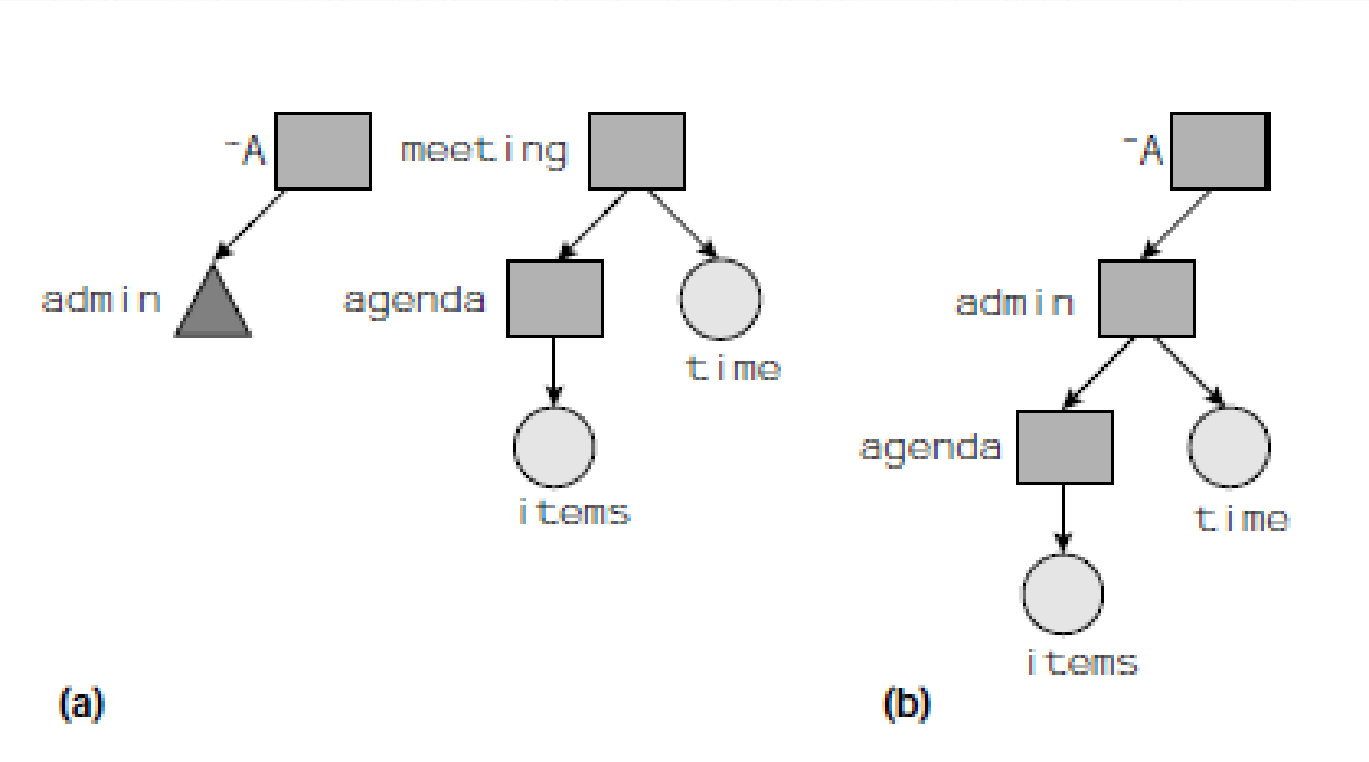


Fig (a)˜A/admin is a mount point in a directory structure, and meeting is the root directory of another file system.

Fig (b) shows the effect of the command mount (meeting,˜A/admin).

Directory Implementation

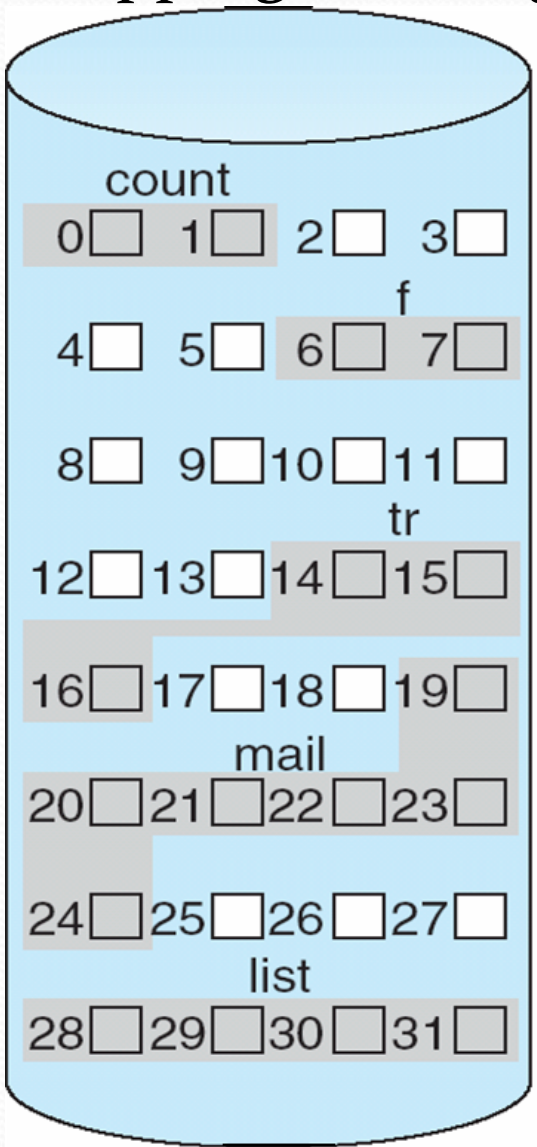
- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Allocation Methods - Contiguous

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Contiguous Allocation

- Mapping from logical to physical



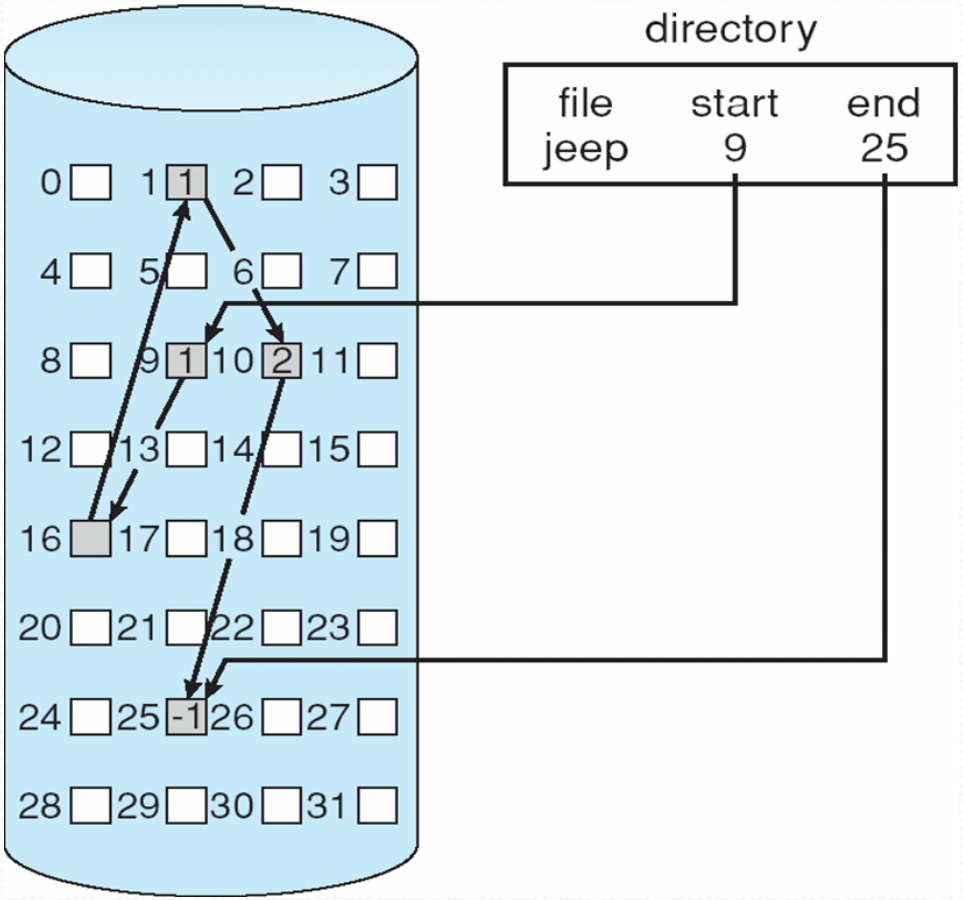
directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocation Methods - Linked

- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation
  - Locating a block can take many I/Os and disk seeks

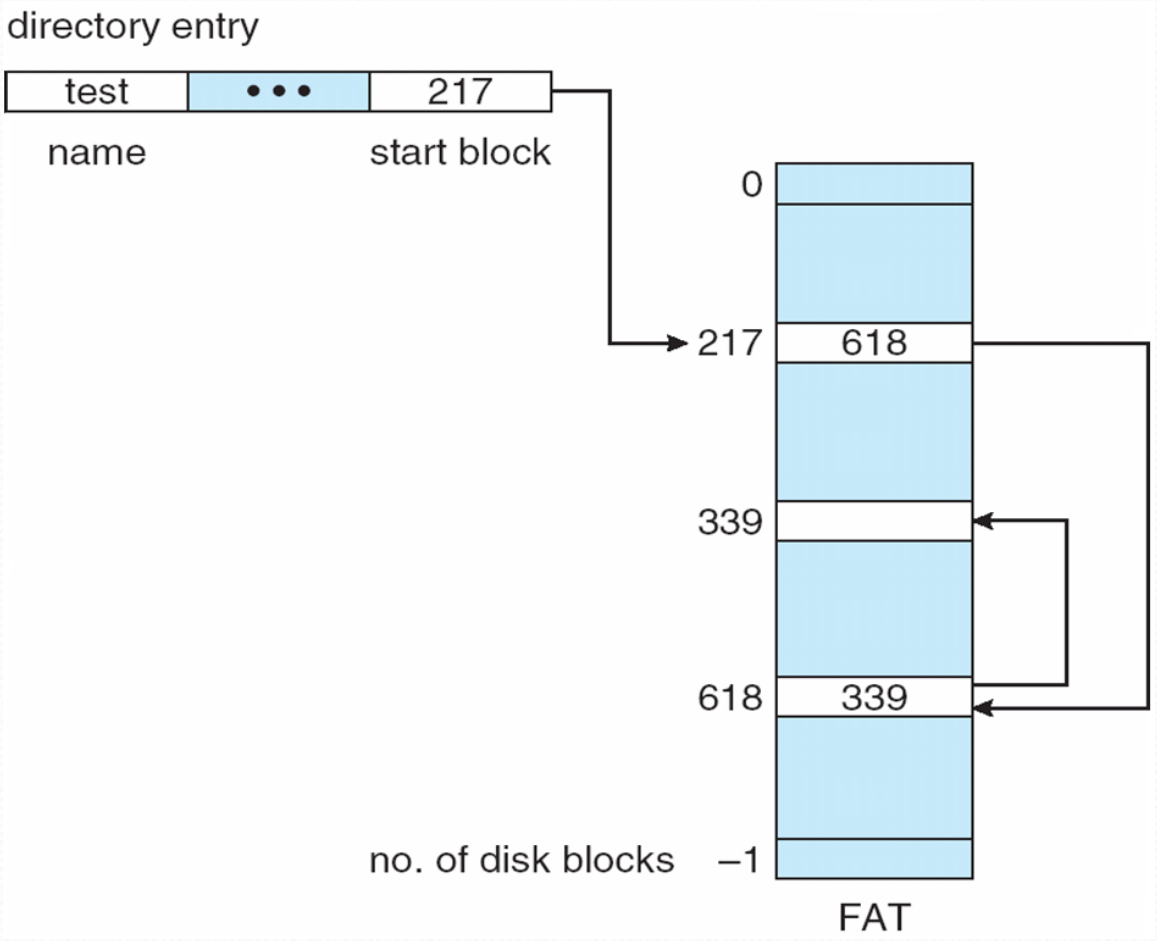
Linked Allocation



Allocation Methods – Linked (Cont.)

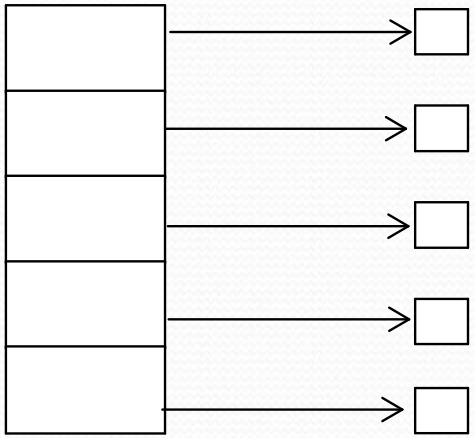
- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

File-Allocation Table

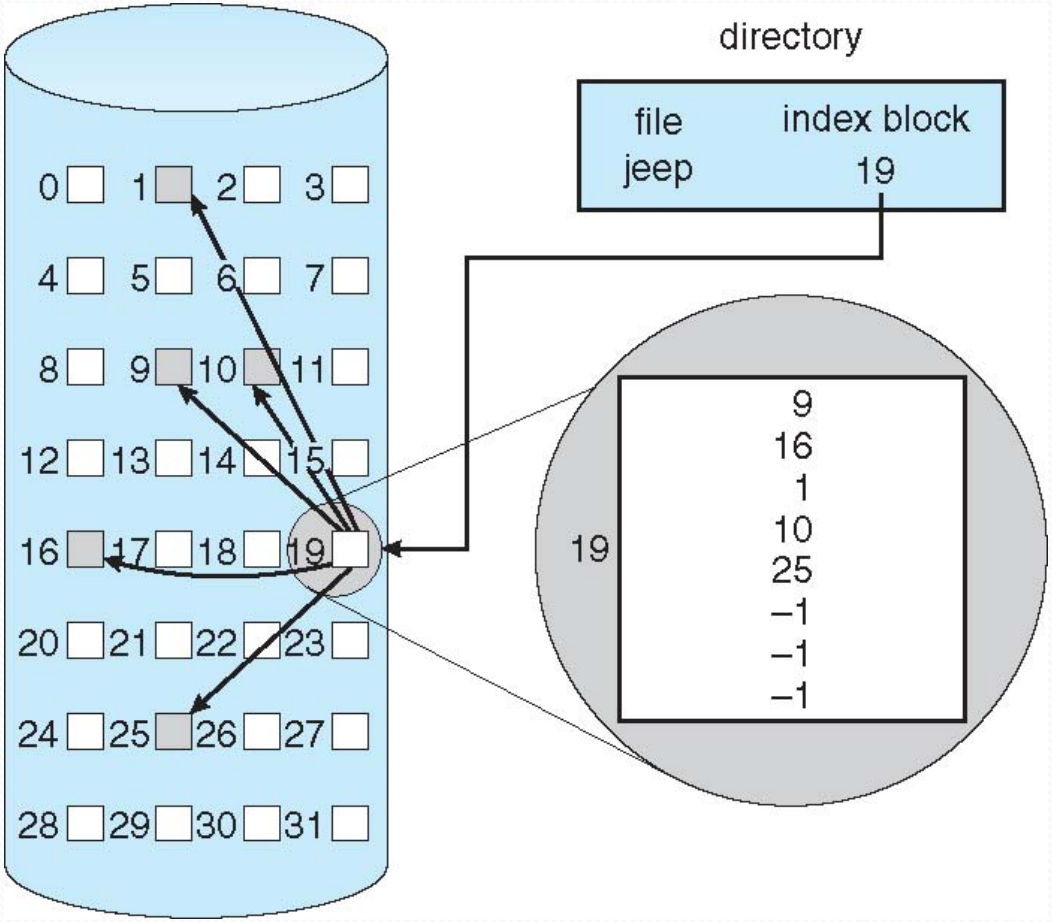


Allocation Methods - Indexed

- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks
- Logical view



Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- We need a block for index table

Distributed File System

Terms in DFS Context

- **Service** - software entity running on one or more machines and providing a particular type of function to clients
- **Server** is the service software running on a single machine
- **Client** is a process that can invoke a service using a set of operations that form its **client interface**
- File system provides file services to clients
- A client interface for a file service is formed by a set of primitive file operations, such as create a file, delete a file, read from a file, and write to a file.

DFS

- A DFS is a file system whose clients, servers, and storage devices are dispersed among the machines of a distributed system
- Instead of a single centralized data repository, the system frequently has multiple and independent storage devices
- The distinctive features of a DFS are the multiplicity and autonomy of clients and servers in the system
- The most important performance measure of a DFS is the amount of time needed to satisfy service requests and its transparency

Naming and Transparency

- **Naming** is a mapping between logical and physical objects
- Usually, a user refers to a file by a textual name
- It is mapped to a lower-level numerical identifier that in turn is mapped to disk blocks
- This multilevel mapping provides users with an abstraction of a file that hides the details of how and where on the disk the file is stored.
- In a transparent DFS, a new dimension is added to the abstraction: that of hiding where in the network the file is located
- In this abstraction, both the existence of multiple copies and their locations are hidden

Naming Structures

We need to differentiate two related notions regarding name mappings in a DFS:

1. Location transparency. The name of a file does not reveal any hint of the file's physical storage location

2. Location independence. The name of a file does not need to be changed when the file's physical storage location changes

Naming Schemes

There are three main approaches to naming schemes in a DFS.

1. In the simplest approach, a file is identified by some combination of its host name and local name, which guarantees a unique system-wide name
    - The Internet URL system also uses this approach. This naming scheme is neither location transparent nor location independent

Naming Schemes (contd…)

2. . The second approach was popularized by Sun's network file system

- NFS provides a means to attach remote directories to local directories, thus giving the appearance of a coherent directory tree
- **Automount** feature allowed mounts to be done on demand based on a table of mount points and file-structure names

Naming Schemes (contd…)

3. We can achieve total integration of the component file systems by using the third approach.

- Here, a single global name structure spans all the files in the system
- In practice, it is difficult to attain

Remote File Access

- Requests for accesses to files are delivered to the server, the server machine performs the accesses, and their results are forwarded back to the user.
- One of the most common ways of Implementing remote service is the RPC paradigm
- Caching –is used to ensure reasonable performance of a remote-service mechanism
- The main goal of caching is to reduce both network traffic and disk I/O

Basic Caching Scheme

- The idea is to retain recently accessed disk blocks in the cache, so that repeated accesses to the same information can be handled locally, without additional network traffic
- Files are still identified with one master copy residing at the server machine, but copies (or parts) of the file are scattered in different caches
- When a cached copy is modified, the changes need to be reflected on the master copy to preserve the relevant consistency semantics

Cache Location

- Where should the cached data be stored—on disk or in main memory?
- Disk caches have one clear advantage over main-memory caches: they are reliable
- Modifications to cached data are lost in a crash if the cache is kept in volatile memory
- Moreover, if the cached data are kept on disk, they are still there during recovery, and there is no need to fetch them again

Cache Location

Main-memory caches have several advantages of their own, however:

- Main-memory caches permit workstations to be diskless

- Data can be accessed more quickly from a cache in main memory than from one on a disk

- The server caches will be in main memory regardless of where user caches are located

Cache-Update Policy

- The policy used to write modified data blocks back to the server's master copy has a critical effect on the system's performance and reliability.

1. **write-through policy** - The simplest policy is to write data through to disk as soon as they are placed in any cache.

- Advantage - reliability: little information is lost when a client system crashes.
- However, this policy requires each write access to wait until the information is sent to the server, so it causes poor write performance.

Cache-Update Policy

2. Delayed-write policy, also known as **write-back caching**, where we delay updates to the master copy.

- Modifications are written to the cache and then are written through to the server at a later time.
- Advantages:
  - Because writes are made to the cache, write accesses complete much more quickly.
  - Only the last update needs to be written
- Disadvantages :
  - reliability problems, since unwritten data are lost whenever a user machine crashes.

Cache-Update Policy

3. write-on-close policy - write data back to the server when the file is closed.

- For files that are open for long periods and are modified frequently, however, the performance advantages of this policy over delayed write with more frequent flushing are apparent.

Consistency

- If the client machine determines that its cached data are out of date, it must cache an up-to-date copy of the data before allowing further accesses.
- There are two approaches to verifying the validity of cached data:

1. **Client-initiated approach**. The client initiates a validity check, in which it contacts the server and checks whether the local data are consistent with the master copy.
   - The frequency of the validity checking is the crux of this approach

Consistency

2. Server-initiated approach. The server records, for each client, the files (or parts of files) that it caches.

- When the server detects a potential inconsistency, it must react.

Any Queries?