University Institute of Engineering
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Unit-2

Chapter : File System and Management

Prepared By: Er. Inderjeet Singh

DISCOVER . LEARN . EMPOWER

Outline

- File Concept
- Attributes of File
- File Operation
- File Types
- File Access Methods
- Directory and Directory Structure
- File protection: ACL
- File Allocation Methods
- Free Space Management

File Concept

- Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks. So that the computer system will be convenient to use, the operating system provides a uniform logical view of stored information. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file.
- Files are mapped by the operating system onto physical devices. These storage devices are usually non-volatile, so the contents are persistent between system reboots.
- A **file** is a named collection of related information that is recorded on secondary storage.

File Concept

- From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free form, such as text files, or may be formatted rigidly. In general, a file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user.
- Many different types of information may be stored in a file — source or executable programs, numeric or text data, photos, music, video, and so on. A file has a certain defined structure, which depends on its type.

Files

- When you use an application to do work - e.g., write a paper, make a spreadsheet, or draw a picture, the work is stored in RAM (memory) first.

- It is in danger of being lost if the power goes off (RAM is **volatile**!)

- When you save it, it is copied to a secondary storage device like the hard drive or a flash drive.

- It is saved as a FILE with a name, extension, time, date, size
  - The extension, if there is one, is at the right end of the name, with a period before it, like file1.abc

Attributes of File

A file's attributes vary from one operating system to another but typically consist of these:

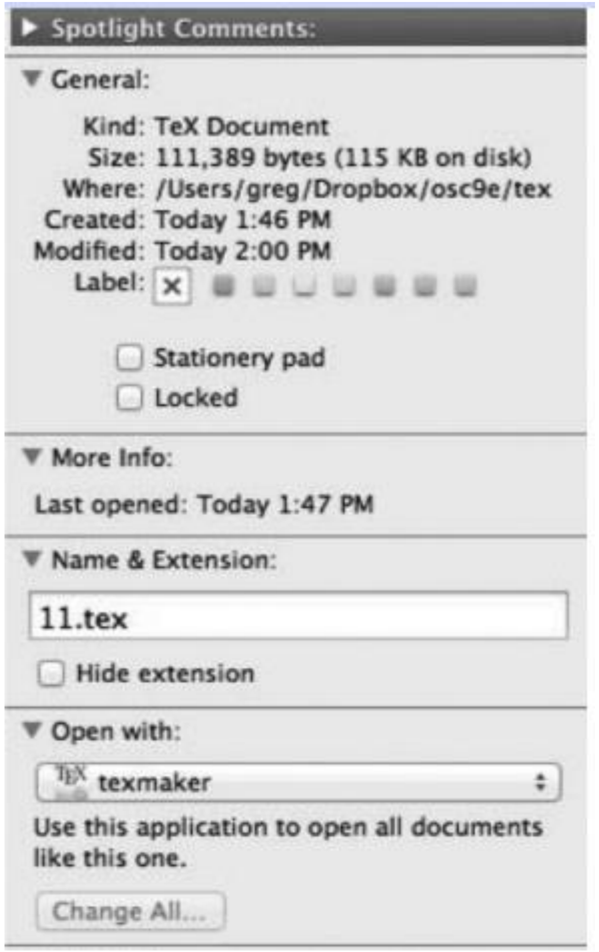
- **Name:** The symbolic file name is the only information kept in human-readable form.
- **Identifier:** This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- **Type:** This information is needed for systems that support different types of files.

Attributes of File

- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing, and so on.
- **Time, date, and user identification.** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

The information about all files is kept in the directory structure, which also resides on secondary storage. Typically, a directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes.

Attributes of File



Kind: TeX Document
Size: 111,389 bytes (115 KB on disk)
Where: /Users/greg/Dropbox/osc9e/tex
Created: Today 1:46 PM
Modified: Today 2:00 PM
Label: ✕

☐ Stationery pad
☐ Locked

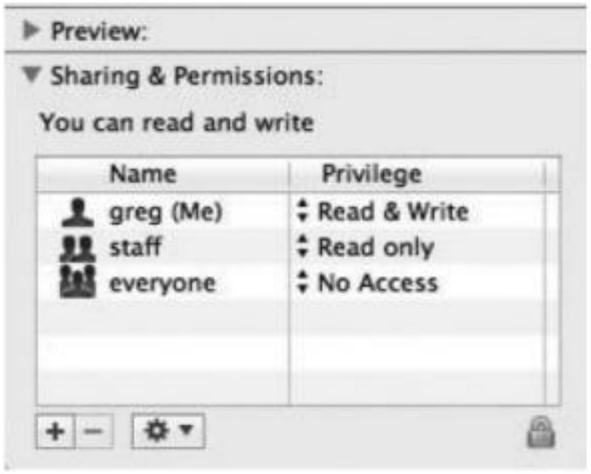
Last opened: Today 1:47 PM

11.tex

☐ Hide extension

texmaker

Use this application to open all documents like this one.



You can read and write

Name	Privilege
greg (Me)	Read & Write
staff	Read only
everyone	No Access

Figure 11.1 A file info window on Mac OS X.

File Operations

- **Creating a file**: Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Writing a file**: To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- **Reading a file**: To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current-file-position pointer. Both the read and write operations use this same pointer, saving space and reducing system complexity.

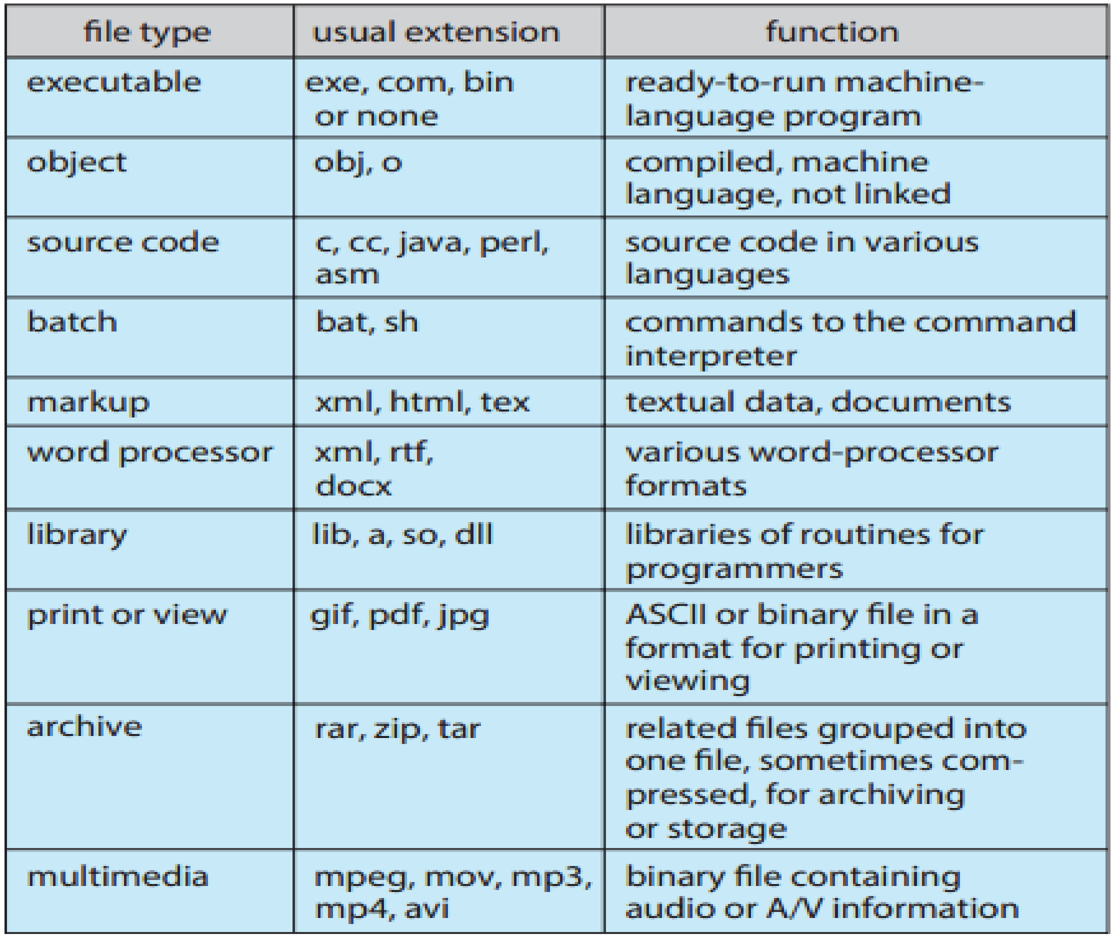
File Operations

- **Repositioning within a file:** The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O .This file operation is also known as a file seek.
- **Deleting a file:** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.
- **Truncating a file**: The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged — except for file length — but lets the file be reset to length zero and its file space released.

File Types

- When we design a file system — indeed, an entire operating system — we always consider whether the operating system should recognize and support file types. If an operating system recognizes the type of a file, it can then operate on the file in reasonable ways.
- A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts — a name and an extension, usually separated by a period.
- In this way, the user and the operating system can tell from the name alone what the type of a file is. Most operating systems allow users to specify a file name as a sequence of characters followed by a period and terminated by an extension made up of additional characters. Examples include resume.docx , server.c and ReaderThread.cpp.
- The system uses the extension to indicate the type of the file and the type of operations that can be done on that file. Only a file with a .com, .exe or .sh extension can be executed, for instance. The .com and .exe files are two forms of binary executable files, whereas the.sh file is a shell script containing, in ASCII format, commands to the operating system.

File Types



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, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Figure 11.3 Common file types.

File Names and Extensions

- You must adhere to file-naming conventions when saving files
  - Case sensitivity – upper and lower case are different
    - True in Linux and Unix variations, not in Windows
    - If “ABC” and “abc” are different names, it IS case sensitive
  - Maximum length (Windows 260 characters)
  - Spaces allowed – be careful if using multiple spaces, can you see the difference between 2 spaces and three spaces?
  - Digits allowed
  - \ / : * ? " < > | not allowed
- File extensions provide clues to the file contents
- OS uses extensions to know which application created the file and the internal format of the file

Access Methods: Sequential Access

Files store information. When it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways.

1. **Sequential Access**: The simplest access method is sequential access. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

Sequential Access

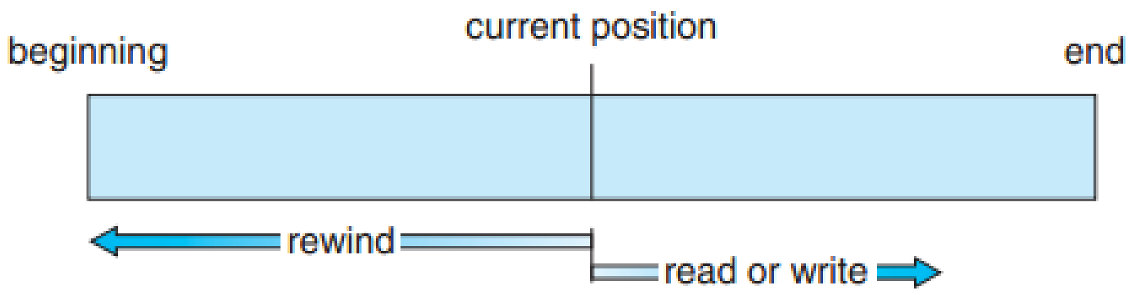


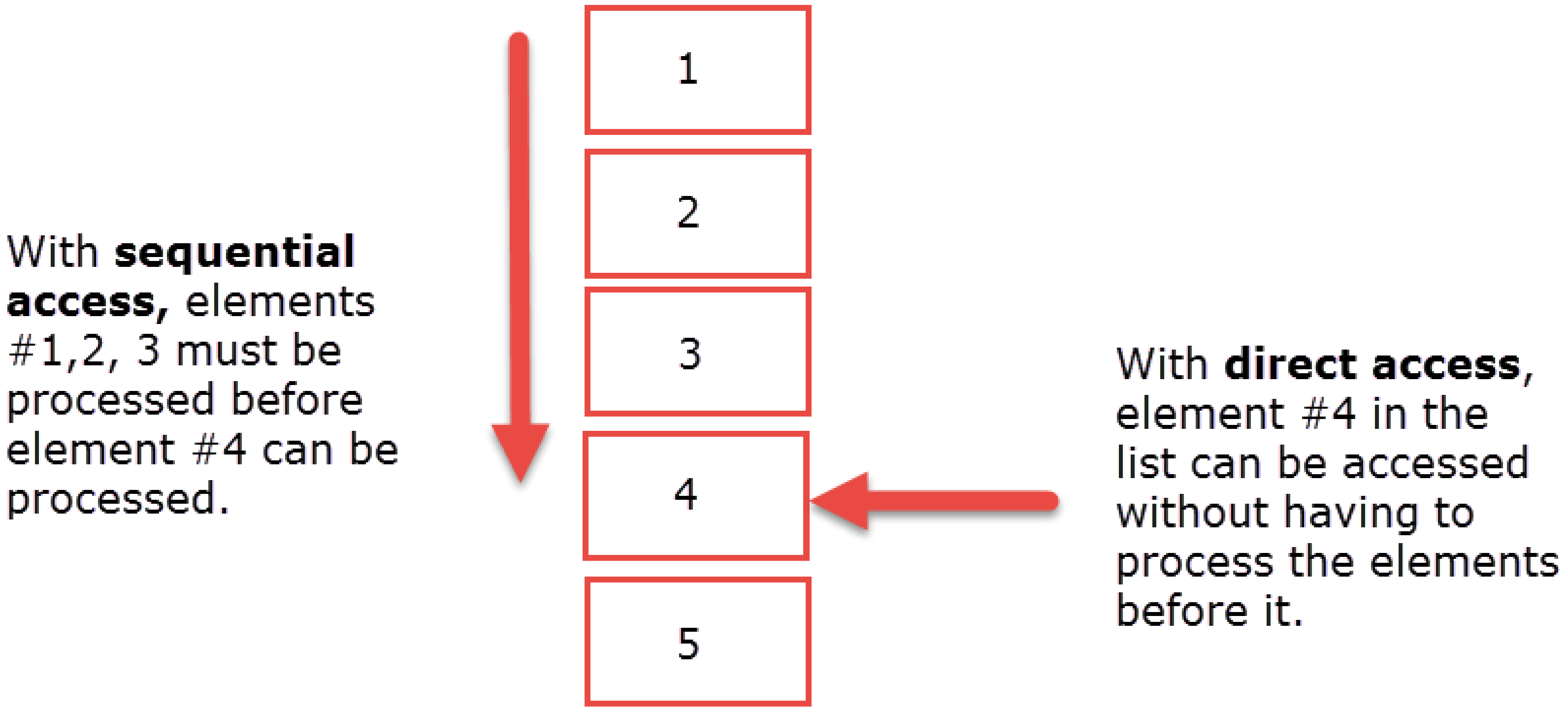
Figure 11.4 Sequential-access file.

- Reads and writes make up the bulk of the operations on a file. A read operation —readnext()— reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, the write operation —writenext()— appends to the end of the file and advances to the end of the newly written material.

Direct Access

- Another method is direct access (or relative access). Here, 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. For direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.
- Direct-access files are of great use for immediate access to large amounts of information. Databases are often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information.

Direct Access



With **sequential access,** elements #1,2, 3 must be processed before element #4 can be processed.

With **direct access**, element #4 in the list can be accessed without having to process the elements before it.

ISAM (Indexed Sequential Access Method)

- IBM's indexed sequential-access method (ISAM) uses a small master index that points to disk blocks of a secondary index. The secondary index blocks point to the actual file blocks.
- The file is kept sorted on a defined key. To find a particular item, we first make a binary search of the master index, which provides the block number of the secondary index. This block is read in, and again a binary search is used to find the block containing the desired record.
- Finally, this block is searched sequentially. In this way, any record can be located from its key by at most two direct-access reads. Figure 11.6 shows a similar situation as implemented by VMS index and relative files.

ISAM (Indexed Sequential Access Method)

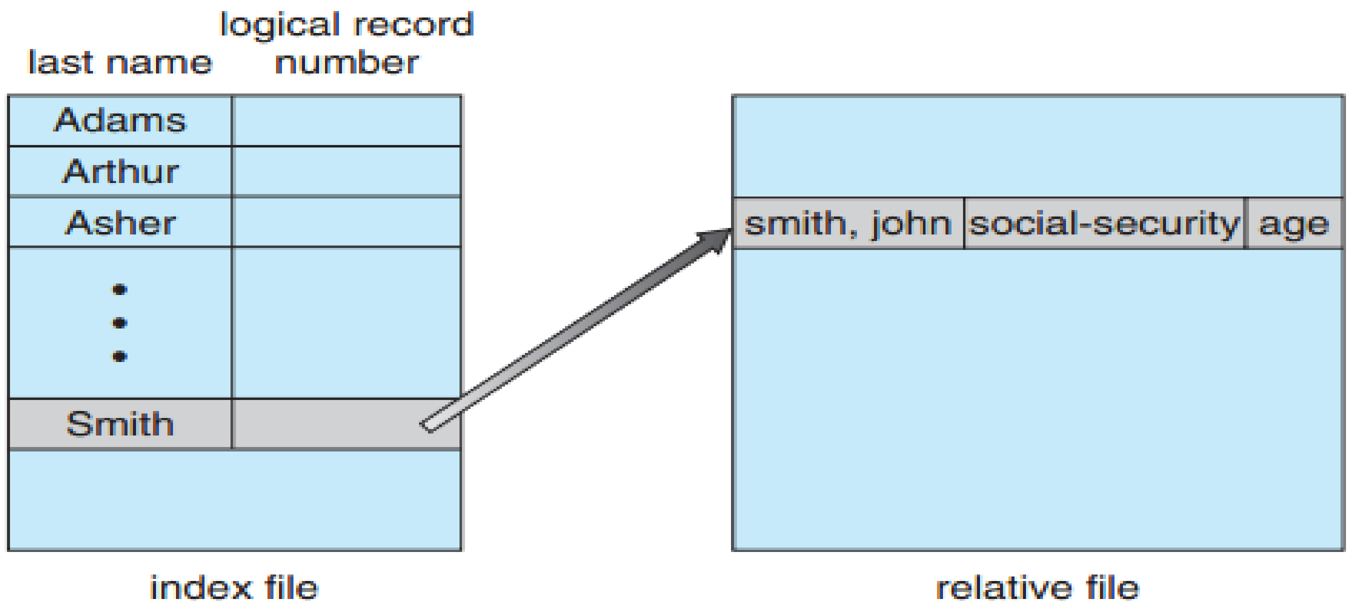


Figure 11.6 Example of index and relative files.

Definitions

- Database
  - Group of related files
  - Interconnected at various levels
    - Give users flexibility of access to stored data
- Program files
  - Contain instructions
- Data files
  - Contain data
- Directories (folders)
  - Listings of filenames and their attributes

Definitions

- Field
  - Group of related bytes
  - Identified by user (name, type, size)
- Record
  - Group of related fields
- File
  - Group of related records
  - Information used by specific application programs
    - Report generation
  - Flat file
    - No connections to other files; no dimensionality

File Management

- The operating system provides an organizational structure to the computer's data and programs
- Hierarchical structure of directories:
  - Drives
    - Folders
      - and more Folders …
        - » Files
- Storage metaphors help you visualize and mentally organize the files on your disks and other storage devices

File Directories and Folders

- Every storage device has a directory containing a list of its files
  - Root directory (like “C:\”)
  - Subdirectory
    - Depicted as folders
- A computer’s file location is defined by a **path**
- Examples: D:\ is the root of the D drive
- Examples: C:\Notes\CS 101\Week 1\notes.txt
- Examples: F:\1999\Music\CDs\Prince\

File Directories and Folders

- A storage device can be used in its entirety for a file system. It can also be subdivided for finer-grained control. For example, a disk can be partitioned into quarters, and each quarter can hold a separate file system. Storage devices can also be collected together into RAID sets that provide protection from the failure of a single disk.
- A file system can be created on each of these parts of the disk. Any entity containing a file system is generally known as a **volume.**
- Volumes can also store multiple operating systems, allowing a system to boot and run more than one operating system.

Directory and Disk Structure

Each volume that contains a file system must also contain information about the files in the system. This information is kept in entries in a device directory or volume table of contents. The device directory (more commonly known simply as the directory) records information — such as name, location, size and type — for all files on that volume. Figure 11.7 shows a typical file-system organization

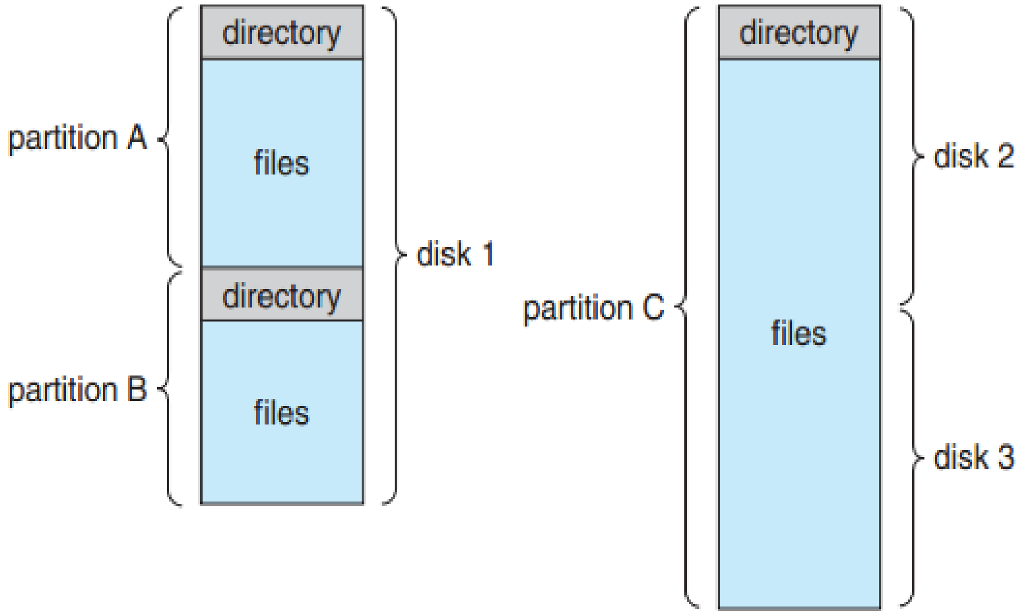


Figure 11.7 A typical file-system organization.

Directories Overview

Single-Level Directory

- The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand (Figure 11.9).
- A single-level directory has significant limitations, however, when the number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names. If two users call their data file test.txt, then the unique-name rule is violated.

Directories Overview: Single Level Directory

- Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases. It is not uncommon for a user to have hundreds of files on one computer system and an equal number of additional files on another system.

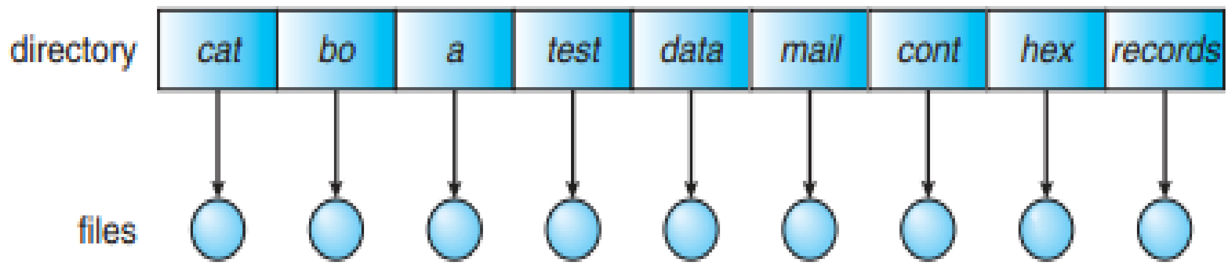


Figure 11.9 Single-level directory.

Directories Overview: Two-Level Directory

- As we have seen, a single-level directory often leads to confusion of file names among different users. The standard solution is to create a separate directory for each user. In the two-level directory structure, each user has his own user file directory (UFD).
- The UFDs have similar structures, but each lists only the files of a single user. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user

Directories Overview: Two-Level Directory

- Although the two-level directory structure solves the name-collision problem, it still has disadvantages. This structure effectively isolates one user from another. Isolation is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another’s files. Some systems simply do not allow local user files to be accessed by other users.

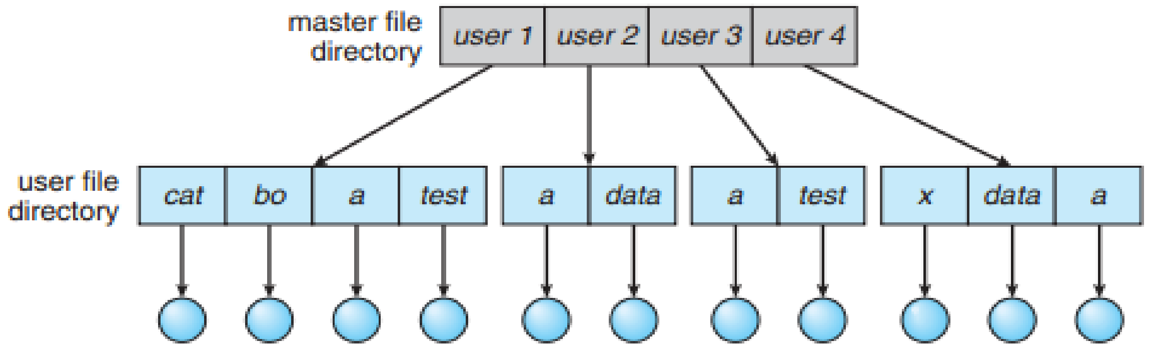


Figure 11.10 Two-level directory structure.

Directories Overview: Tree-Structured Directories

- Once we have seen how to view a two-level directory as a two-level tree, the natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way. All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1). Special system calls are used to create and delete directories

Directories Overview: Tree-Structured Directories

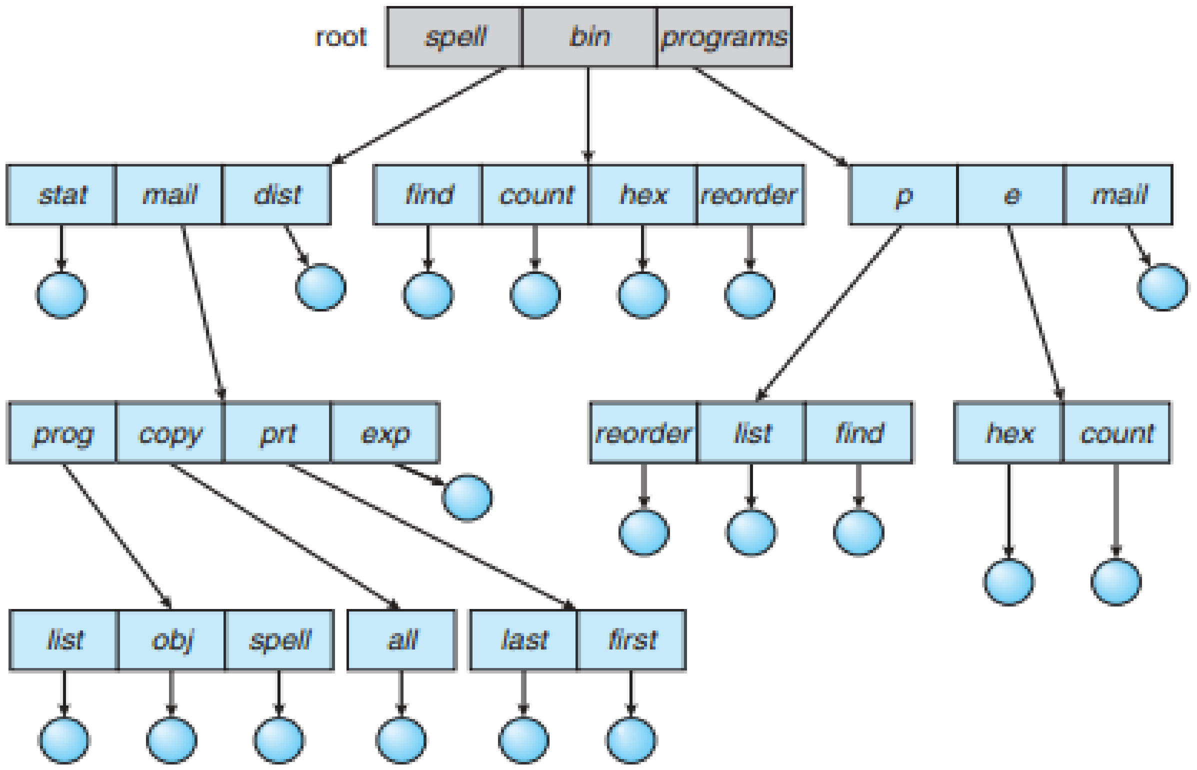
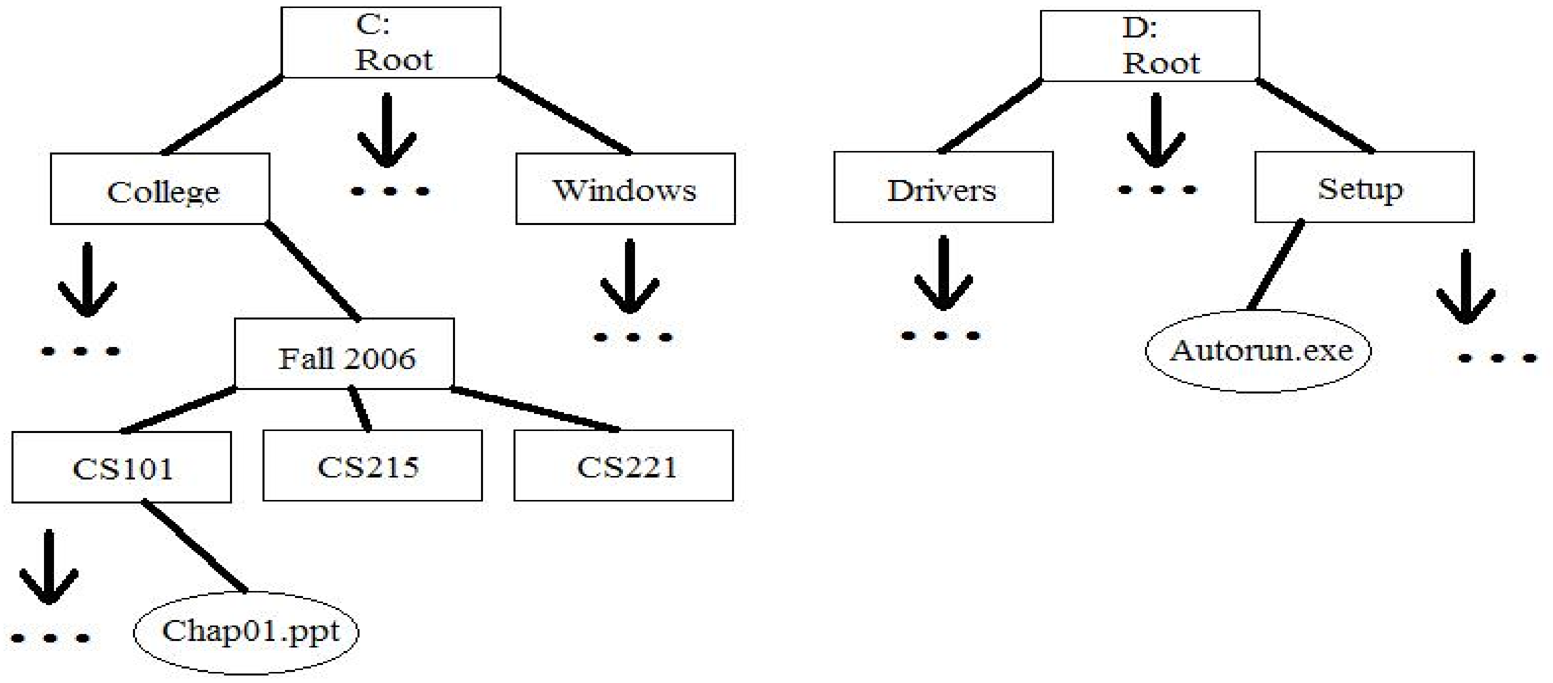
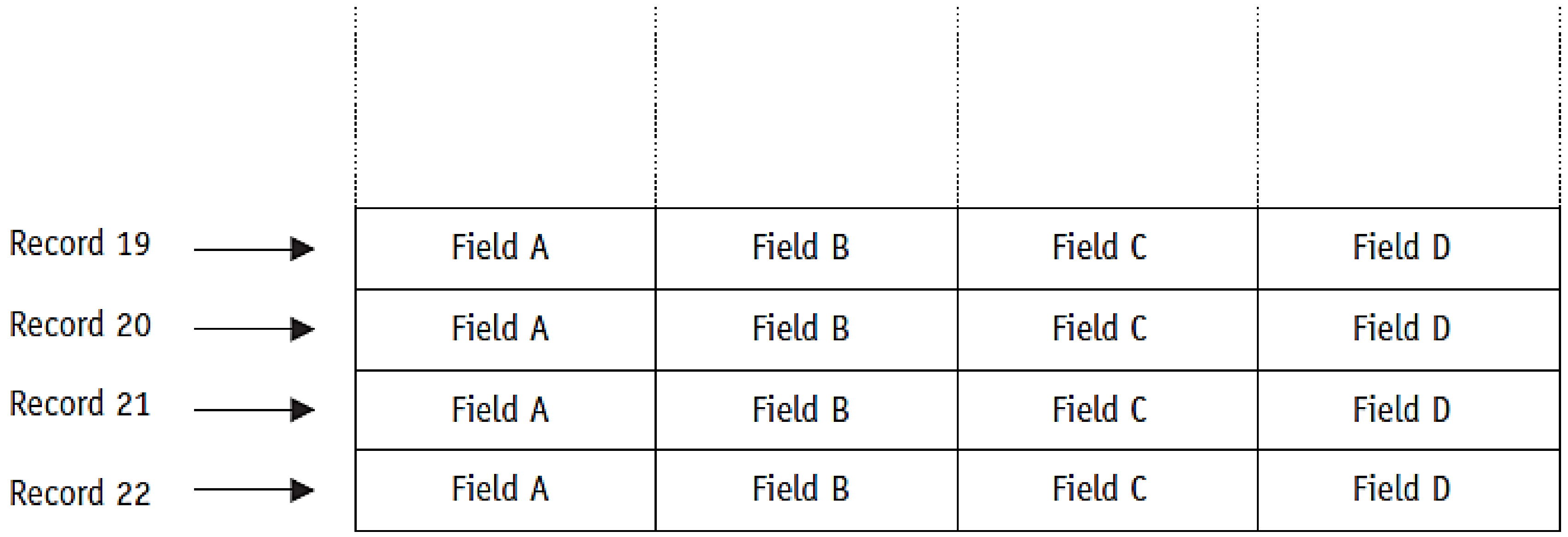


Figure 11.11 Tree-structured directory structure.

A File System Tree (2 devices)



Records and Files



Record 19	Field A	Field B	Field C	Field D
Record 20	Field A	Field B	Field C	Field D
Record 21	Field A	Field B	Field C	Field D
Record 22	Field A	Field B	Field C	Field D

Files are made up of records. Records consist of fields.

Acyclic-Graph Directories

- Consider two programmers who are working on a joint project. The files associated with that project can be stored in a subdirectory, separating them from other projects and files of the two programmers. But since both programmers are equally responsible for the project, both want the subdirectory to be in their own directories. In this situation, the common subdirectory should be shared. A shared directory or file exists in the file system in two (or more) places at once.
- An acyclic graph — that is, a graph with no cycles — allows directories to share subdirectories and files.

Acyclic-Graph Directories

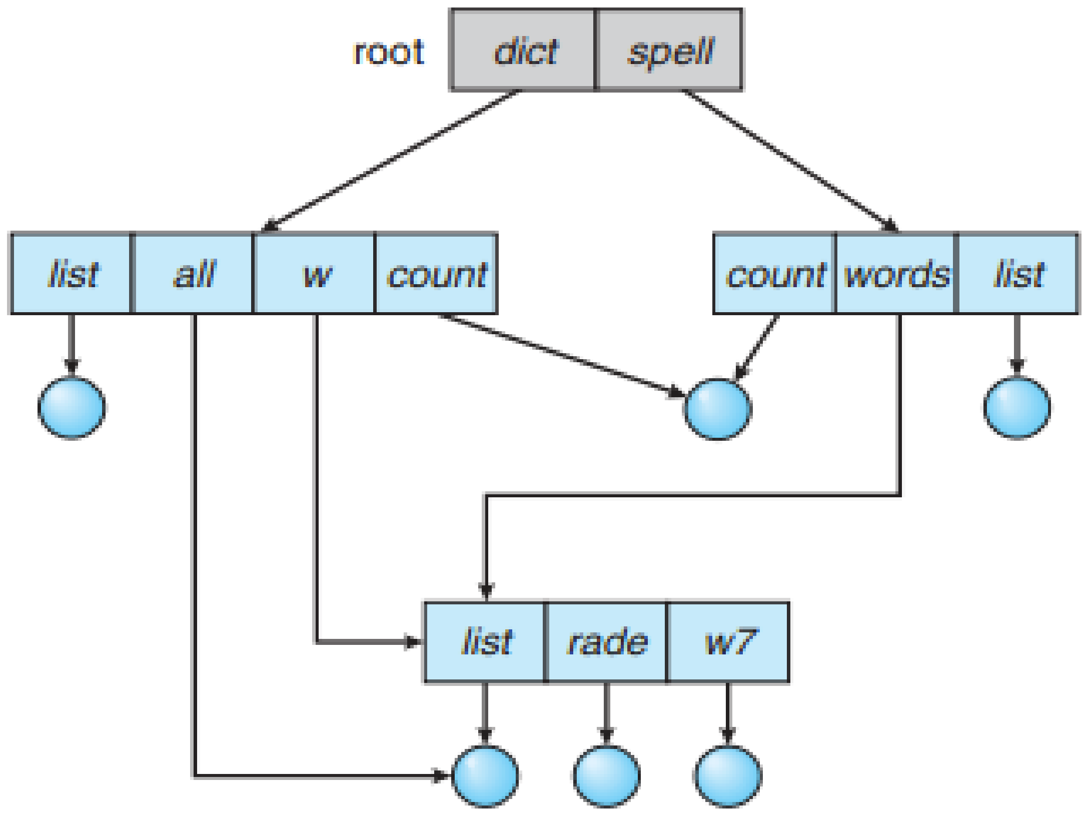


Figure 11.12 Acyclic-graph directory structure.

The File Manager

- File management system
  - Software
- File access responsibilities
  - Creating, deleting, modifying, controlling

Responsibilities of the File Manager

- Four tasks
  - File storage tracking
  - Policy implementation
    - Determine where and how files are stored
    - Efficiently use available storage space
    - Provide efficient file access
  - File allocation if user access cleared
    - Record file use
  - File deallocation
    - Return file to storage
    - Communicate file availability

Responsibilities of the File Manager (cont'd.)

- Policy determines:
  - File storage location
  - System and user access
    - Uses device-independent commands
- Access to material
  - Two factors
- Factor 1: flexibility of access to information
  - Share files
  - Provide distributed access
  - Allow users to browse public directories

Responsibilities of the File Manager (cont'd.)

- Factor 2: subsequent protection
  - Prevent system malfunctions
  - Security checks
    - Account numbers and passwords
- File allocation
  - Activate secondary storage device, load file into memory, and update records
- File deallocation
  - Update file tables, rewrite file (if revised), and notify waiting processes of file availability

Types of Access

- The need to protect files is a direct result of the ability to access files. Systems that do not permit access to the files of other users do not need protection. Thus, we could provide complete protection by prohibiting access.
- Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:
- **Read:** Read from the file.
- **Write.** Write or rewrite the file.
- **Execute**: Load the file into memory and execute it.
- **Append:** Write new information at the end of the file.
- **Delete**: Delete the file and free its space for possible reuse.
- **List:** List the name and attributes of the file.

Access Control

- The most common approach to the protection problem is to make access dependent on the identity of the user. Different users may need different types of access to a file or directory. The most general scheme to implement identity-dependent access is to associate with each file and directory an **access-control list (ACL)** specifying user names and the types of access allowed for each user. When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.

This technique has two undesirable consequences:

- Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.
- The directory entry, previously of fixed size, now must be of variable size, resulting in more complicated space management.

Access Control

These problems can be resolved by use of a condensed version of the access list. To condense the length of the access-control list, many systems recognize three classifications of users in connection with each file:

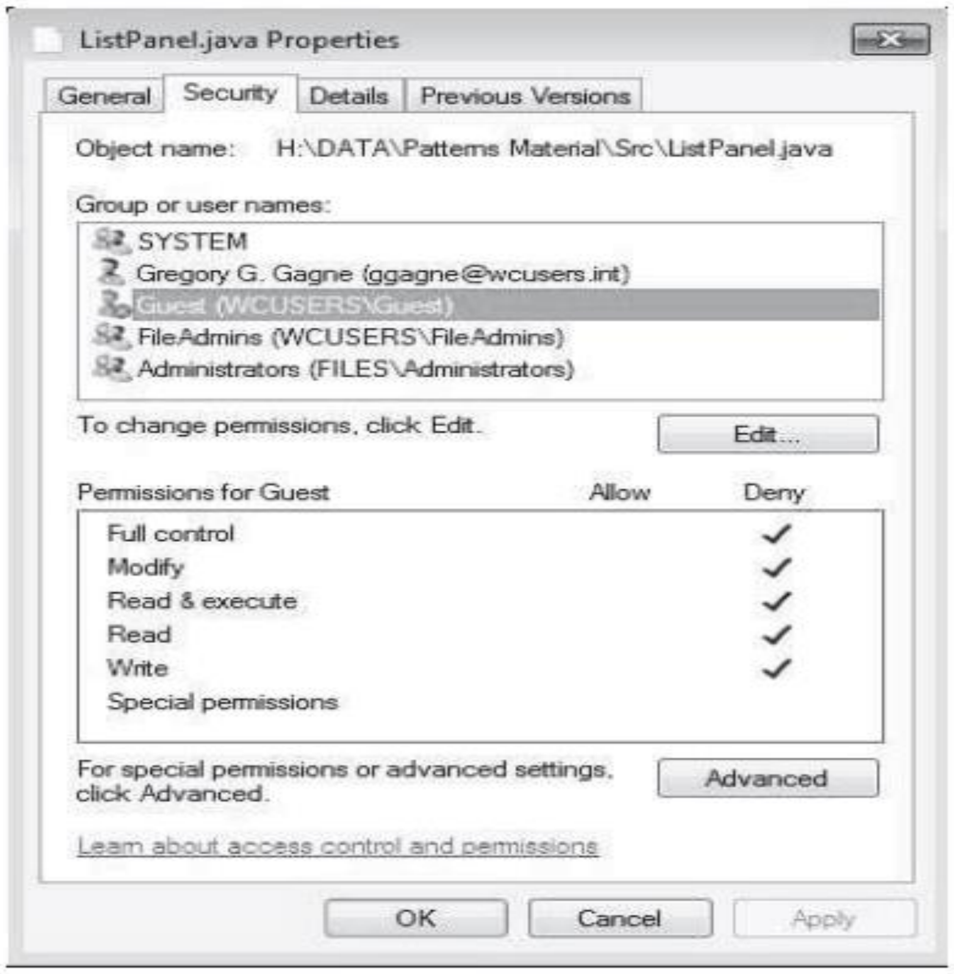
- **Owner**: The user who created the file is the owner.
- **Group**: A set of users who are sharing the file and need similar access is a group, or work group.
- **Universe**: All other users in the system constitute the universe.

To illustrate, consider a person, Sara, who is writing a new book. She has hired three graduate students (Jim, Dawn, and Jill) to help with the project. The text of the book is kept in a file named book.tex. The protection associated with this file is as follows:

- Sara should be able to invoke all operations on the file.
- Jim, Dawn, and Jill should be able only to read and write the file; they should not be allowed to delete the file.
- All other users should be able to read, but not write, the file. (Sara is interested in letting as many people as possible read the text so that she can obtain feedback.)

Access Control

With the more limited protection classification, only three fields are needed to define protection. Often, each field is a collection of bits, and each bit either allows or prevents the access associated with it. For example, the UNIX system defines three fields of 3 bits each — r w x , where r controls read access, w controls write access, and x controls execution



ListPanel.java Properties

General | Security | Details | Previous Versions

Object name: H:\DATA\Patterns Material\Src\ListPanel.java

Group or user names:
- SYSTEM
- Gregory G. Gagne (ggagne@wcusers.int)
- Guest (WCUSERS\Guest)
- FileAdmins (WCUSERS\FileAdmins)
- Administrators (FILES\Administrators)

To change permissions, click Edit. [Edit...]

Permissions for Guest	Allow	Deny
Full control		✓
Modify		✓
Read & execute		✓
Read		✓
Write		✓
Special permissions		

For special permissions or advanced settings, click Advanced. [Advanced]

Learn about access control and permissions

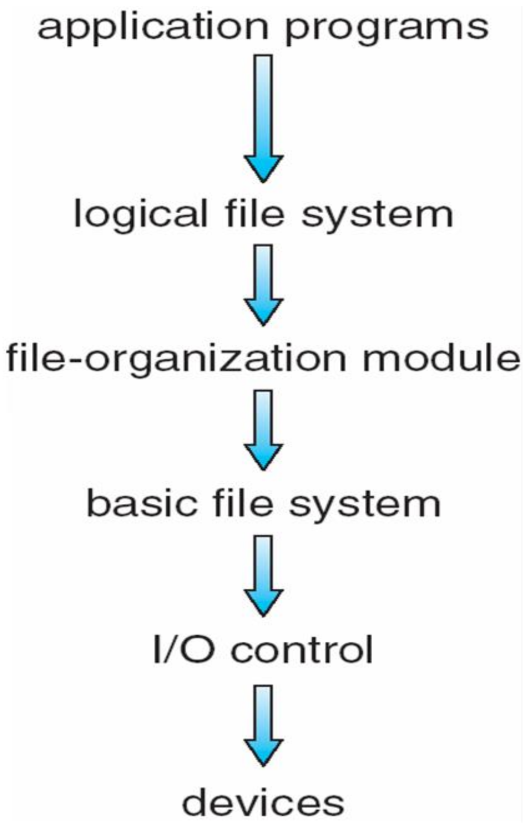
OK | Cancel | Apply

Figure 11.16 Windows 7 access-control list management.

File-System Structure

- File system provides the mechanism for on-line storage and access to file contents, including data and programs. The file system resides permanently on secondary storage, which is designed to hold a large amount of data permanently.
- File systems provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily. A file system poses two quite different design problems. The first problem is defining how the file system should look to the user. This task involves defining a file and its attributes, the operations allowed on a file and the directory structure for organizing files. The second problem is creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices.
- The **file system** itself is generally composed of many different levels. The structure shown in Figure 12.1 is an example of a layered design.

Layered File System



File-System Structure

- The **I/O control** level consists of device drivers and interrupts handlers to transfer information between the main memory and the disk system. A device driver can be thought of as a translator. Its input consists of high-level commands such as “retrieve block 123.” Its output consists of low-level, hardware-specific instructions that are used by the hardware controller, which interfaces the I/O device to the rest of the system.
- The **basic file system** needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (for example, drive 1, cylinder 73, track 2, and sector 10).

File-System Structure

- The **file-organization** module knows about files and their logical blocks, as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file-organization module can translate logical block addresses to physical block addresses for the basic file system to transfer.
- Finally, **the logical file system** manages metadata information. Metadata includes all of the file-system structure except the actual data (or contents of the files). The logical file system manages the directory structure to provide the file-organization module with the information the latter needs, given a symbolic file name. It maintains file structure via file-control blocks. A file-control block (FCB) (an inode in UNIX file systems) contains information about the file, including ownership, permissions, and location of the file contents.

File-System Implementation

- **Boot control block** contains info needed by system to boot OS from that volume
- **Volume control block** contains volume details such as the number of blocks in partition
- Directory structure organizes the files. In UFS,

this includes file names and associated inode numbers.

- Per-file **File Control Block (FCB)** contains many details about the file

Note: A file-control block (FCB) (an inode in UNIX file systems) contains information about the file, including ownership, permissions, and location of the file contents.

A Typical File Control Block

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

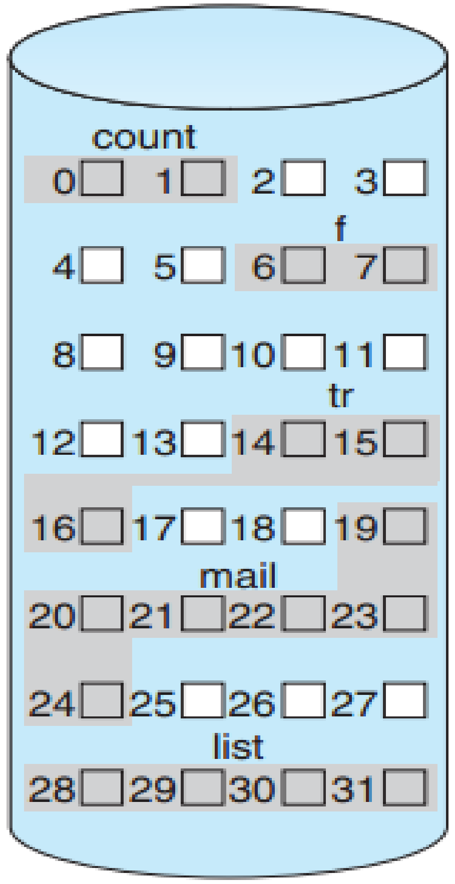
Allocation Methods

- The direct-access nature of disks gives us flexibility in the implementation of files. In almost every case, many files are stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.
- Three major methods of allocating disk space are in wide use: contiguous, linked, and indexed. Each method has advantages and disadvantages

Contiguous Allocation

- Contiguous allocation requires that each file occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessing the disk, accessing block b+1 after block b normally requires no head movement. When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), the head need only move from one track to the next. Thus, the number of disk seeks required for accessing contiguously allocated files is minimal, as is seek time when a seek is finally needed.
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is n blocks long and starts at location b, then it occupies blocks b, b +1, b + 2, ..., b + n − 1. The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file (Figure 12.5).

Contiguous Allocation



directory

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

Figure 12.5 Contiguous allocation of disk space.

Contiguous Allocation

- Contiguous allocation has some problems, however. One difficulty is finding space for a new file. The system chosen to manage free space determines how this task is accomplished;
- The contiguous-allocation problem can be seen as a particular application of the general dynamic storage-allocation problem, which involves how to satisfy a request of size n from a list of free holes. First fit and best fit are the most common strategies used to select a free hole from the set of available holes. Simulations have shown that both first fit and best fit are more efficient than worst fit in terms of both time and storage utilization. Neither first fit nor best fit is clearly best in terms of storage utilization, but first fit is generally faster.
- All these algorithms suffer from the problem of external fragmentation.

Contiguous Allocation

- Another problem with contiguous allocation is determining how much space is needed for a file. When the file is created, the total amount of space it will need must be found and allocated.
- If we allocate too little space to a file, we may find that the file cannot be extended. Especially with a best-fit allocation strategy, the space on both sides of the file may be in use.
- To minimize these drawbacks, some operating systems use a modified contiguous-allocation scheme. Here, a contiguous chunk of space is allocated initially. Then, if that amount proves not to be large enough, another chunk of contiguous space, known as an extent, is added.

Linked Allocation

- Linked allocation solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25.
- **Disadvantage:**Linked allocation does have disadvantages, however. The major problem is that it can be used effectively only for sequential-access files. To find the ith block of a file, we must start at the beginning of that file and follow the pointers until we get to the ith block. Each access to a pointer requires a disk read, and some require a disk seek. Consequently, it is inefficient to support a direct-access capability for linked-allocation files.
- Another disadvantage is the space required for the pointers. If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information.

Linked Allocation

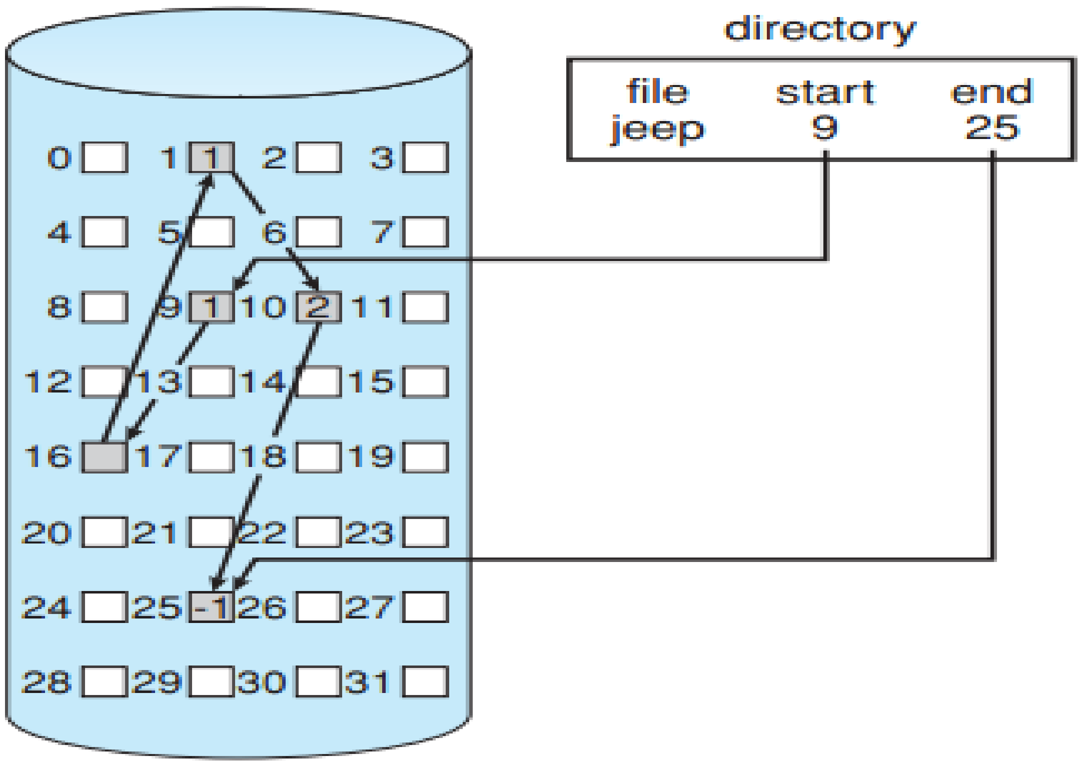


Figure 12.6 Linked allocation of disk space.

Linked Allocation

- An important variation on linked allocation is the use of a **file-allocation table (FAT)**. This simple but efficient method of disk-space allocation was used by the MS-DOS operating system. A section of disk at the beginning of each volume is set aside to contain the table. The table has one entry for each disk block and is indexed by block number. The FAT is used in much the same way as a linked list. The directory entry contains the block number of the first block of the file. The table entry indexed by that block number contains the block number of the next block in the file. This chain continues until it reaches the last block, which has a special end-of-file value as the table entry.

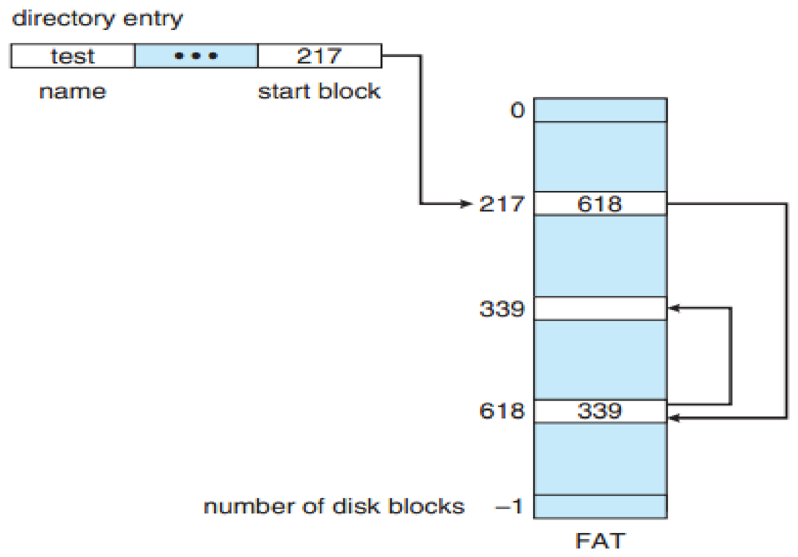


Figure 12.7 File-allocation table.

Indexed Allocation

- In this each file has its own index block, which is an array of disk-block addresses. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block.
- To find and read the ith block, we use the pointer in the ith index-block entry. This scheme is similar to the paging scheme.

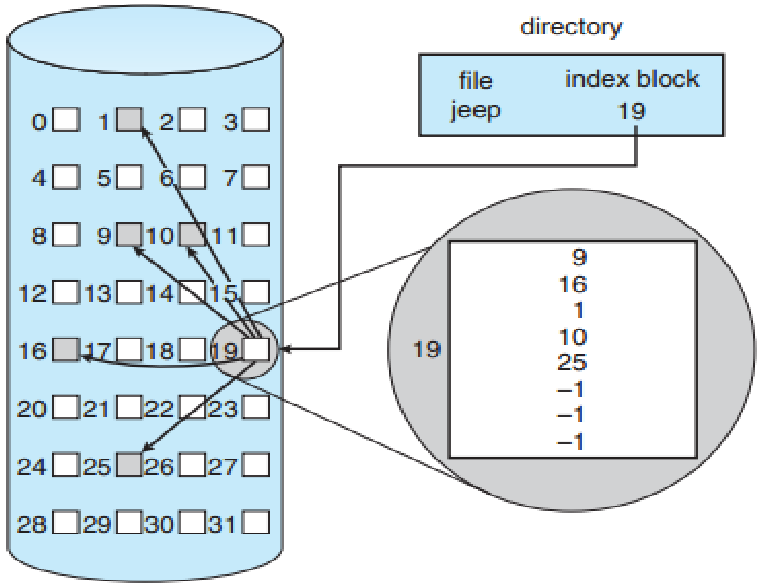


Figure 12.8 Indexed allocation of disk space.

Free Space Management

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. (Write-once optical disks allow only one write to any given sector, and thus reuse is not physically possible.)
- To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks — those not allocated to some file or directory. To create a file, we search the free-space list for the required amount of space and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

Free Space Management: Bit Vector

- Frequently, the free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bit map would be

01111001111110001100000011100000 ...

- The main advantage of this approach is its relative simplicity and its efficiency in finding the first free block or n consecutive free blocks on the disk.
- Unfortunately, bit vectors are inefficient unless the entire vector is kept in main memory (and is written to disk occasionally for recovery needs).

Free Space Management: Linked List

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory. This first block contains a pointer to the next free disk block, and so on.
- Recall, in which blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated. In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.

Free Space Management: Linked List

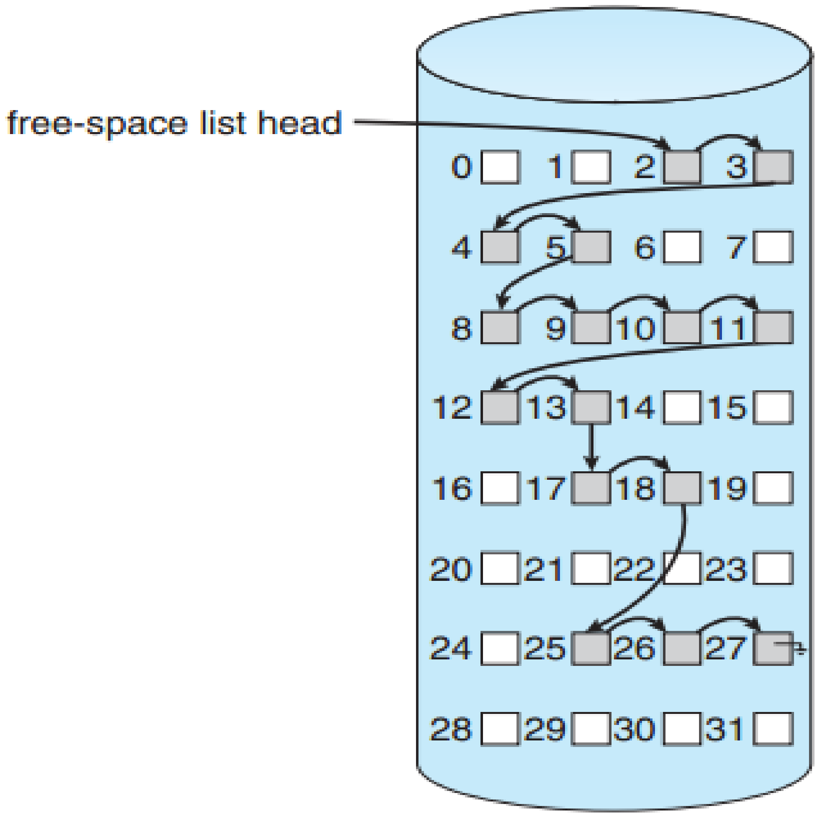


Figure 12.10 Linked free-space list on disk.

For any Query, you can contact

Er. Inderjeet Singh(e8822)
Ph. No: 86-99-100-160
Email id: inderjeet.e8822@cumail.in