WEEK 6 : LECTURE 5 - FILE MANAGEMENT

OBJECTIVES/TOPICS

- File System Implementation.
- Access Methods.
- Directory Structure.
- Protection.
- File System Structure.
- Allocation Methods.
- Free Space Management.

FILE SYSTEM IMPLEMENTATION

- Computers can store information on various storage media, such as NVM devices, HDDs, 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.

FILE SYSTEM IMPLEMENTATION

- 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 nonvolatile, so the contents are persistent between system reboots

FILE SYSTEM IMPLEMENTATION

- A file is a named collection of related information that is recorded on secondary storage.

- 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 and data.
  - Data files may be numeric, alphabetic, alphanumeric, or binary.

FILE SYSTEM IMPLEMENTATION

- The information in a file is defined by its creator. 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.

FILE SYSTEM IMPLEMENTATION

- A text file is a sequence of characters organized into lines (and possibly pages).

- 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 SYSTEM IMPLEMENTATION: File Attributes

- A file is named, for the convenience of its human users, and is referred to by its name.
  - A name is usually a string of characters

- Some systems differentiate between uppercase and lowercase characters in names, whereas other systems do not.

- When a file is named, it becomes independent of the process, the user, and even the system that created it.

FILE SYSTEM IMPLEMENTATION: File Attributes

- A file's attributes vary from one operating system to another but typically consist of these:
  - 1. **Name:** The symbolic file name is the only information kept in human-readable form.
  - 2. **Identifier:** This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
  - 3. **Type:** This information is needed for systems that support different types of files.
  - 4. **Location:** This information is a pointer to a device and to the location of the file on that device.

FILE SYSTEM IMPLEMENTATION: File Attributes

- 5. **Size:** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- 6. **Protection:** Access-control information determines who can do reading, writing, executing, and so on.
- 7. **Timestamps 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.

FILE SYSTEM IMPLEMENTATION: File Attributes

- Some newer file systems also support extended file attributes, including character encoding of the file and security features such as a file checksum

- The information about all files is kept in the directory structure, which resides on the same device as the files themselves.

- Typically, a directory entry consists of the file's name and its unique identifier.

FILE SYSTEM IMPLEMENTATION: File Attributes

- The identifier in turn locates the other file attributes.
- It may take more than a kilobyte to record this information for each file.
- In a system with many files, the size of the directory itself may be megabytes or gigabytes.
- Because directories must match the volatility of the files, like files, they must be stored on the device and are usually brought into memory piecemeal, as needed

FILE SYSTEM IMPLEMENTATION: File Operations

- A file is an abstract data type (ADT)

- To define a file properly, we need to consider the operations that can be performed on files.

- The operating system can provide system calls to create, write, read, reposition, delete, and truncate files

FILE SYSTEM IMPLEMENTATION: File Operations

- 1. **Creating a file**
  - Two steps are necessary to create a file.
    - 1st, space in the file system must be found for the file.
    - 2nd, an entry for the new file must be made in a directory.

- 2. **Opening a file**
  - all operations except create and delete require a file open() first.
  - If successful, the open call returns a file handle that is used as an argument in the other calls.

FILE SYSTEM IMPLEMENTATION: File Operations

- **3. Writing a file**
  - To write a file, we make a system call specifying both the open file handle and the information to be written to the file.
  - The system must keep a write pointer to the location in the file where the next write is to take place if it is sequential.
  - The write pointer must be updated whenever a write occurs.

FILE SYSTEM IMPLEMENTATION: File Operations

- 4. **Reading a file**
  - To read from a file, we use a system call that specifies the file handle and where (in memory) the next block of the file should be put.
  - Again, the system needs to keep a read pointer to the location in the file where the next read is to take place, if sequential
  - Once the read has taken place, the read pointer is updated.

FILE SYSTEM IMPLEMENTATION: File Operations

- 5. **Repositioning within a file.**
  - The current-file-position pointer of the open file is repositioned to a given value.
  - Repositioning need not involve any actual I/O.
  - This file operation is also known as a file seek.

- 6. **Deleting a file .**
  - To delete , 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 or mark as free the directory entry.

FILE SYSTEM IMPLEMENTATION: File Operations

- 7. 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.
  - The file can then be reset to length zero, and its file space can be released.

ACCESS METHODS

- 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.
- Some systems provide only one access method for files. Others (such as mainframe operating systems) support many access methods, and choosing the right one for a particular application is a major design problem.

ACCESS METHODS:

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.

ACCESS METHODS:

1. Sequential Access

- Reads and writes make up the bulk of the operations on a file.

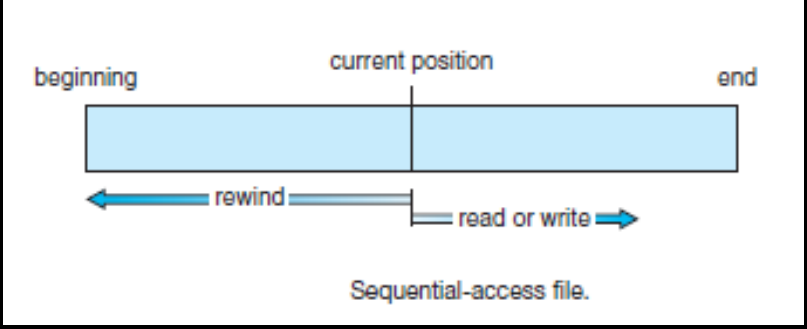
  - A read operation—read_next()—reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.

  - the write operation—write_next()—appends to the end of the file and advances to the end of the newly written material (the new end of file).

ACCESS METHODS:

1. Sequential Access

- Sequential access, which is depicted below, is based on a tape model of a file and works as well on sequential-access devices as it does on random-access ones.



Sequential-access file.

ACCESS METHODS:

2. 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 & 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.

ACCESS METHODS:

2. Direct Access

- 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.

ACCESS METHODS:

2. Direct Access

- 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

DIRECTORY STRUCTURE

- The directory can be viewed as a symbol table that translates file names into their file control blocks.

- If we take such a view, we see that the directory itself can be organized in many ways.

- The organization must allow us to insert entries, to delete entries, to search for a named entry, and to list all the entries in the directory.

DIRECTORY STRUCTURE

- When considering a particular directory structure,we need to keep in mind the operations that are to be performed on a directory:

- 1. **Search for a file**
  - We need to be able to search a directory structure to find the entry for a particular file.
  - Since files have symbolic names, and similar names may indicate a relationship among files, we may want to be able to find all files whose names match a particular pattern.

DIRECTORY STRUCTURE

- 2. **Create a file:**
  - New files need to be created and added to the directory.

- 3. **Delete a file:**
  - When a file is no longer needed, we want to be able to remove it from the directory.
  - Note a delete leaves a hole in the directory structure and the file system may have a method to defragment the directory structure.

DIRECTORY STRUCTURE

- 4. **List a directory:**
  - We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.

- 5. **Rename a file:**
  - Because the name of a file represents its contents to its users, we must be able to change the name when the contents or use of the file changes.
  - Renaming a file may also allow its position within the directory structure to be changed.

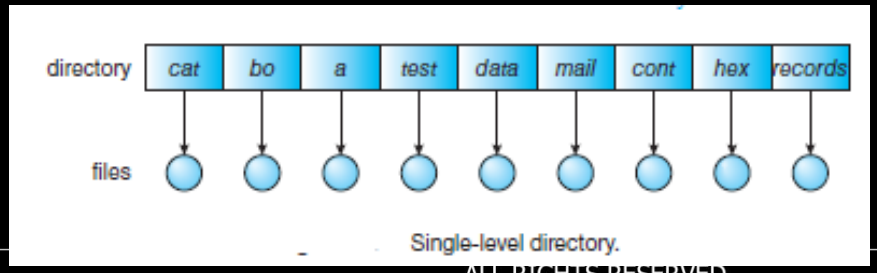
DIRECTORY STRUCTURE

- 6. Traverse the file system:
  - We may wish to access every directory and every file within a directory structure

DIRECTORY STRUCTURE

- In the following sections, we describe the most common schemes for defining the logical structure of a directory:

- **1. 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



Single-level directory.

DIRECTORY STRUCTURE

- **1. Single-Level Directory**
  - 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

DIRECTORY STRUCTURE

- **1. 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.
  - Keeping track of so many files is a daunting task.

DIRECTORY STRUCTURE

- **2. Two-Level Directory**
  - 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.

DIRECTORY STRUCTURE

- **2. Two-Level Directory**
  - The MFD is indexed by user name or account number, and each entry points to the UFD for that user

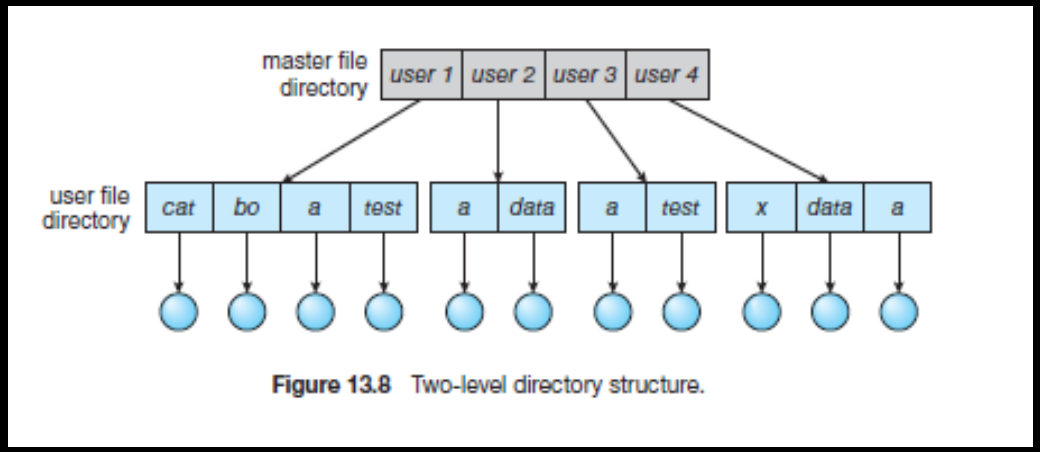


Figure 13.8 Two-level directory structure.

DIRECTORY STRUCTURE

- **2. Two-Level Directory**
  - When a user refers to a particular file, only his own UFD is searched.
  - Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.
  - To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists.

DIRECTORY STRUCTURE

- **2. Two-Level Directory**
  - To delete a file, the operating system confines its search to the local UFD;
  - thus, it cannot accidentally delete another user's file that has the same name.

DIRECTORY STRUCTURE

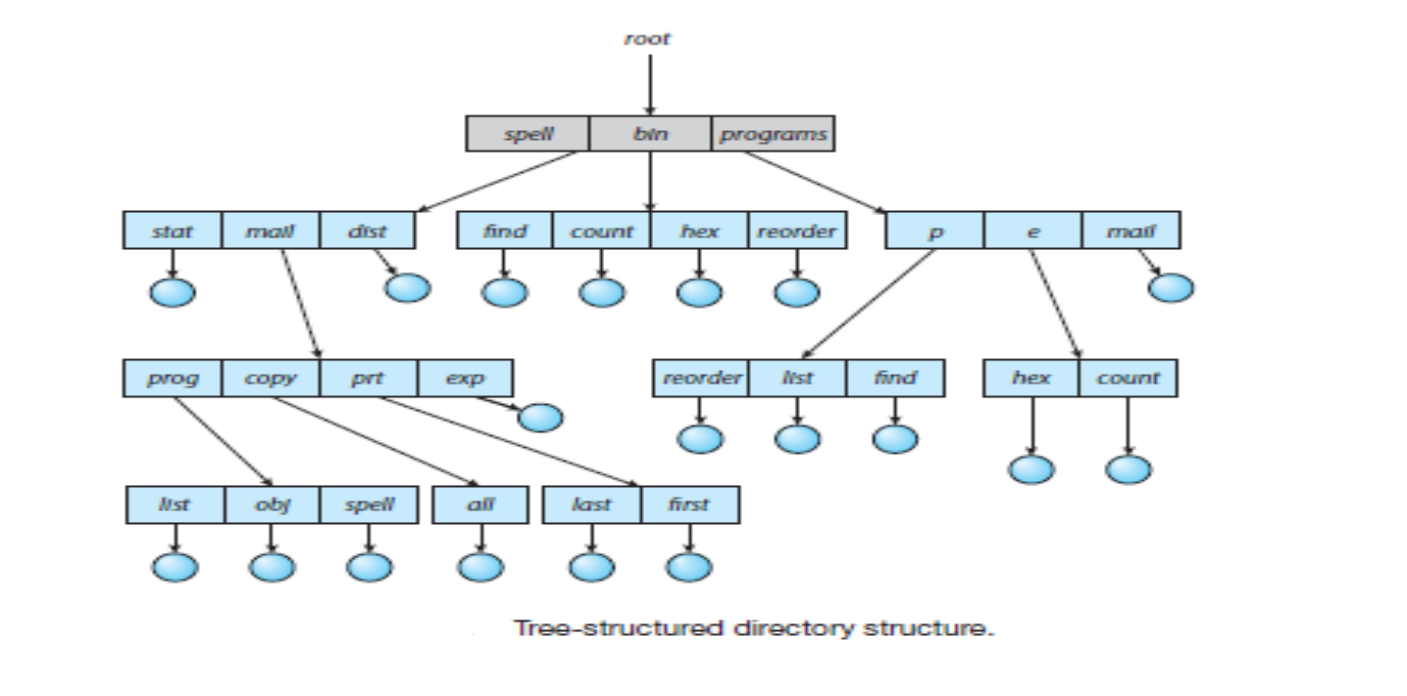
- **2. Two-Level Directory**
  - Although the two-level directory structure solves the name-collision problem, it still has disadvantages.
  - This structure effectively isolates one user from another, & Isolation is an advantage when the users are independent but is a disadvantage when the users want to cooperate on some task & to access 1 another's files
  - Some systems simply do not allow local user files to be accessed by other users.

DIRECTORY STRUCTURE

- **2. 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.

DIRECTORY STRUCTURE

- **2. Tree-Structured Directories**
  - The tree has a root directory, and every file in the system has a unique path name.



Tree-structured directory structure.

DIRECTORY STRUCTURE

- **2. Tree-Structured Directories**
  - A directory (or subdirectory) contains a set of files or subdirectories.
  - In many implementations, 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).

DIRECTORY STRUCTURE

- **2. Tree-Structured Directories**
  - Special system calls are used to create & delete directories.
  - In normal use, each process has a current directory.
  - The current directory should contain most of the files that are of current interest to the process.
  - When reference is made to a file, the current directory is searched.

DIRECTORY STRUCTURE

- **2. Tree-Structured Directories**
  - If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory to be the directory holding that file.

  - To change directories, a system call could be provided that takes a directory name as a parameter and uses it to redefine the current directory.
    - Thus, the user can change her current directory whenever she wants.

DIRECTORY STRUCTURE

- **2. Tree-Structured Directories**
  - Path names can be of two types: absolute and relative.
    - In UNIX and Linux, an absolute path name begins at the root (which is designated by an initial "/") and follows a path down to the specified file, giving the directory names on the path.
    - A relative path name defines a path from the current directory

DIRECTORY STRUCTURE

- **2. 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.

DIRECTORY STRUCTURE

- **2. Acyclic-Graph Directories**
- In this situation, the common subdirectory should be shared.

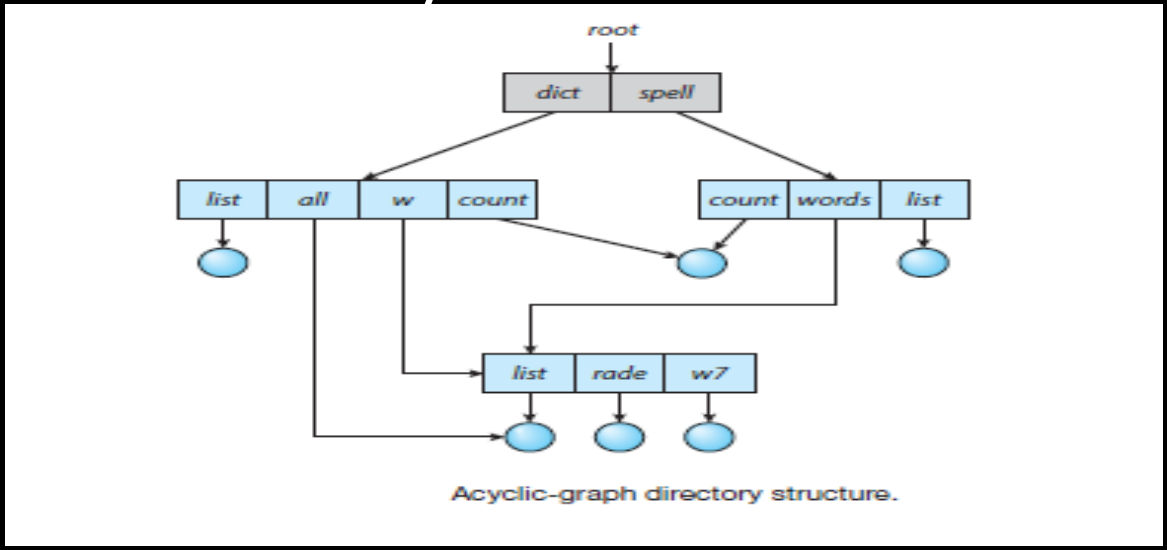
- A shared directory or file exists in the file system in two (or more) places at once.

- A tree structure prohibits the sharing of files or directories

- An acyclic graph—that is, a graph with no cycles—allows directories to share subdirectories and files

DIRECTORY STRUCTURE

- **2. Acyclic-Graph Directories**
- The same file or subdirectory may be in two different directories.
- The acyclic graph is a natural generalization of the tree-structured directory scheme.



Acyclic-graph directory structure.

DIRECTORY STRUCTURE

- **2. Acyclic-Graph Directories**
- It is important to note that a shared file (or directory) is not the same as two copies of the file.

- With two copies, each programmer can view the copy rather than the original, but if one programmer changes the file, the changes will not appear in the other's copy.

- With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the other.

DIRECTORY STRUCTURE

- **2. Acyclic-Graph Directories**
- A common way to implement shared files and directories, exemplified by UNIX systems, is to create a new directory entry called a link.

- A link is effectively a pointer to another file or subdirectory.
  - For example, a link may be implemented as an absolute or a relative path name.

DIRECTORY STRUCTURE

- **2. Acyclic-Graph Directories**
- When a reference to a file is made, we search the directory.
  - If the directory entry is marked as a link, then the name of the real file is included in the link information.

- We resolve the link by using that path name to locate the real file

- READ ON General Graph Directory

PROTECTION

- When information is stored in a computer system, we want to keep it safe from physical damage (the issue of reliability) and improper access (the issue of protection).

- Reliability is generally provided by duplicate copies of files.
  - Many computers have systems programs that automatically (or through computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or month) to maintain a copy should a file system be accidentally destroyed.

PROTECTION

- File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes and vandalism.

- Files may be deleted accidentally.

- Bugs in the file-system software can also cause file contents to be lost.

PROTECTION

- Protection can be provided in many ways.
- For a laptop system running a modern operating system, we might provide protection by :
  - requiring a user name and password authentication to access it,
  - encrypting the secondary storage so even someone opening the laptop and removing the drive would have a difficult time accessing its data, and
  - firewalling network access so that when it is in use it is difficult to break in via its network connection

PROTECTION: 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.

- Alternatively, we could provide free access with no protection.

PROTECTION: Types of Access

- Both approaches are too extreme for general use.
- What is needed is controlled 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.

PROTECTION: Types of Access

- Several different types of operations may be controlled:

  - 1. **Read:** Read from the file.
  - 2. **Write:** Write or rewrite the file.
  - 3. **Execute:** Load the file into memory and execute it.
  - 4. **Append:** Write new information at the end of the file.
  - 5. **Delete:** Delete the file and free its space for possible reuse.
  - 6. **List:** List the name and attributes of the file.
  - 7. **Attribute change:** Changing the attributes of the file.

PROTECTION: 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.

PROTECTION: Access Control

- 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.

PROTECTION: Access Control

- This technique has two undesirable consequences:
  - 1. 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.
  - 2. The directory entry, previously of fixed size, now must be of variable size, resulting in more complicated space management.

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

PROTECTION: Access Control

- The most common recent approach is to combine access-control lists with the more general (and easier to implement) owner, group, and universe access-control scheme just described.

FILE SYSTEM STRUCTURE

- Disks provide most of the secondary storage on which file systems are maintained.
- 2 characteristics make them convenient for this purpose:
  - 1. A disk can be rewritten in place; it is possible to read a block from the disk, modify the block, and write it back into the same block.
  - 2. A disk can access directly any block of information it contains. Thus, it is simple to access any file either sequentially or randomly, and switching from one file to another requires the drive moving the read–write heads and waiting for the media to rotate.

FILE SYSTEM STRUCTURE

- Nonvolatile memory (NVM) devices are increasingly used for file storage and thus as a location for file systems.

- They differ from hard disks in that they cannot be rewritten in place and they have different performance characteristics.

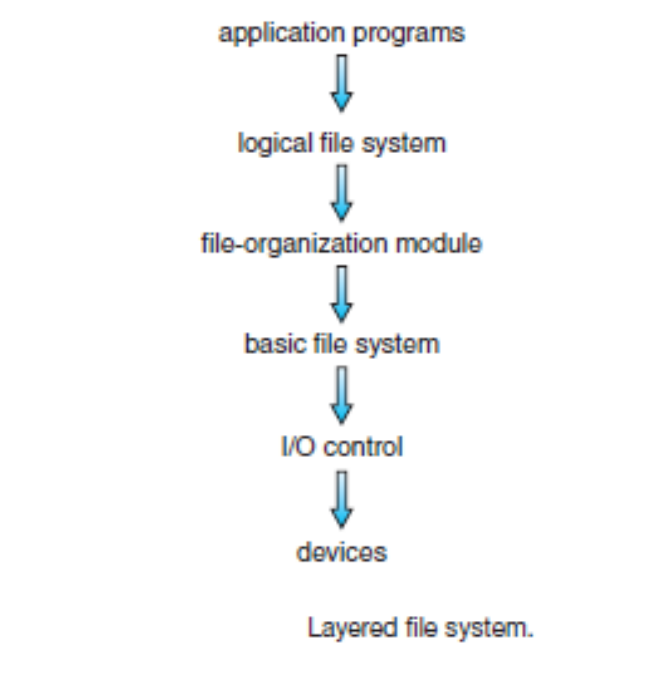
- To improve I/O efficiency, I/O transfers between memory and mass storage are performed in units of blocks.
  - Each block on a hard disk drive has 1 or more sectors.

FILE SYSTEM STRUCTURE

- File systems provide efficient and convenient access to the storage device 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.
  - The second problem is creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices.

FILE SYSTEM STRUCTURE

- The file system itself is generally composed of many different levels, as illustrated below:



Layered file system.

FILE SYSTEM STRUCTURE

- The file system itself is generally composed of many different levels, as illustrated below:
  - 1. The I/O control level consists of device drivers and interrupt handlers to transfer information between the main memory and the disk system
  - 2. The basic file system (called the "block I/O subsystem" in Linux) needs only to issue generic commands to the appropriate device driver to read and write blocks on the storage device.

FILE SYSTEM STRUCTURE

- 3. The file-organization module knows about files and their logical blocks. Each file's logical blocks are numbered from 0 (or 1) through N.
  - The file-organization module also includes the free-space manager, which tracks unallocated blocks and provides these blocks to the file-organization module when requested.

- 4. The logical file system manages metadata information.
  - Metadata includes all of the file-system structure except the actual data (or contents of the files).

ALLOCATION METHODS

- The direct-access nature of secondary storage gives us flexibility in the implementation of files.

- In almost every case, many files are stored on the same device.

- The main problem is how to allocate space to these files so that storage space is utilized effectively and files can be accessed quickly.

ALLOCATION METHODS

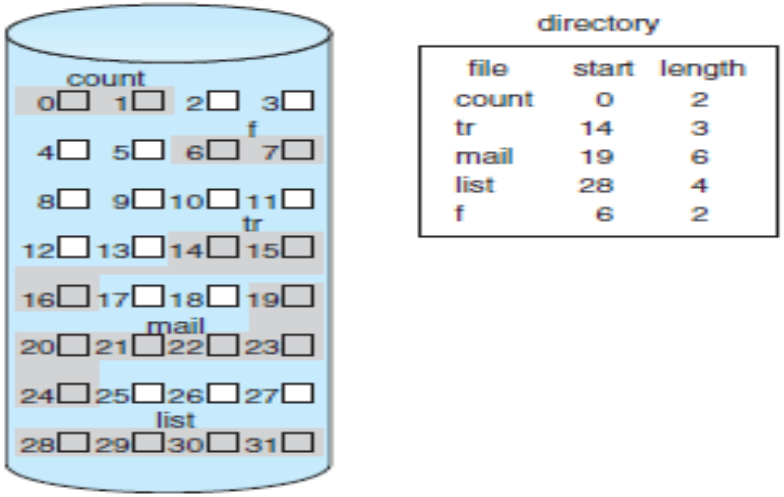
- Three major methods of allocating secondary storage space are in wide use: contiguous, linked, and indexed

- **1. Contiguous Allocation**
  - Contiguous allocation requires that each file occupy a set of contiguous block on the device.
  - Device addresses define a linear ordering on the device.

  - With this ordering, assuming that only one job is accessing the device, accessing block b + 1 after block b normally requires no head movement

ALLOCATION METHODS

- **1. Contiguous Allocation**
  - 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



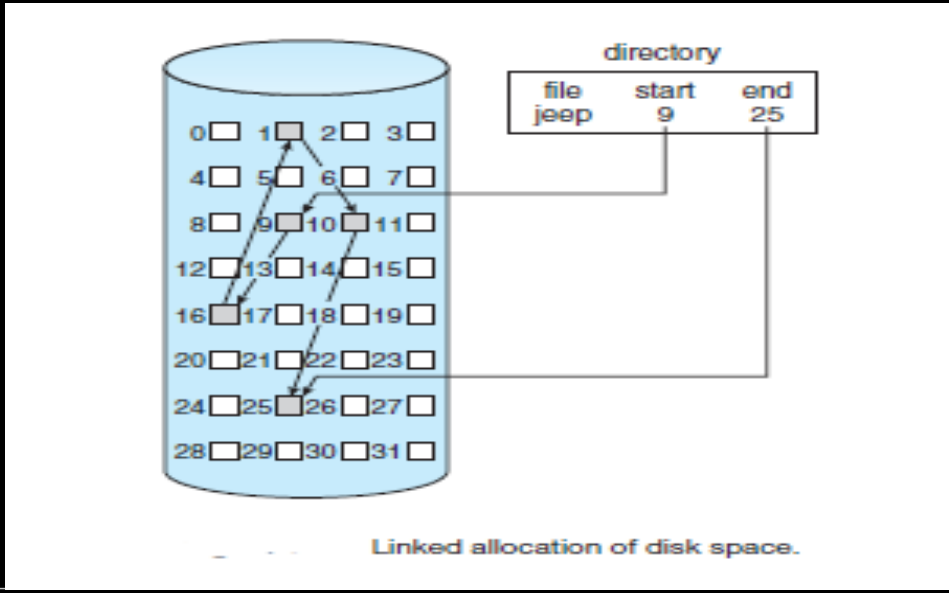
directory

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

Contiguous allocation of disk space.

ALLOCATION METHODS

- **2. Linked Allocation**
  - Here, each file is a linked list of storage blocks; the blocks may be scattered anywhere on the device.
  - The directory contains a pointer to the first and last blocks of the file.



Linked allocation of disk space.

ALLOCATION METHODS

- 3. Indexed Allocation
  - READ ON INDEXED ALLOCATION

FREE SPACE MANAGEMENT

- Since storage 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 device 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

THE END