SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

23CST202 – Operating Systems

II YEAR - IV SEM

UNIT 5 – FILE SYSTEMS

Syllabus

- UNIT I OVERVIEW AND PROCESS MANAGEMENT 9
- Introduction - Computer System Organization, Architecture, Operation, Process Management – Memory Management – Storage Management – Operating System – Process concept – Process scheduling – Operations on processes – Cooperating processes – Inter process communication. Threads - Multi-threading Models – Threading issues.
- UNIT II PROCESS SCHEDULING AND SYNCHRONIZATION 10
- CPU Scheduling - Scheduling criteria – Scheduling algorithms – Multiple-processor scheduling – Real time scheduling – Algorithm Evaluation. Process Synchronization - The critical-section problem – Synchronization hardware – Semaphores – Classical problems of synchronization. Deadlock - System model – Deadlock characterization – Methods for handling deadlocks – Deadlock prevention – Deadlock avoidance – Deadlock detection – Recovery from deadlock.
- UNIT III MEMORY MANAGEMENT 9
- Memory Management - Background – Swapping – Contiguous memory allocation – Paging – Segmentation – Segmentation with paging. Virtual Memory - Background – Demand paging – Process creation – Page replacement – Allocation of frames – Thrashing.
- UNIT IV FILE SYSTEMS 8
- File concept - Access methods – Directory structure – Files System Mounting – File Sharing – Protection. File System Implementation - Directory implementation – Allocation methods – Free-space management.
- UNIT V I/O SYSTEMS 9
- I/O Systems - I/O Hardware – Application I/O interface – Kernel I/O subsystem – Streams – Performance. Mass-Storage Structure: Disk scheduling – Disk management – Swap-space management – RAID – Disk attachment – Stable storage – Tertiary storage. Case study: Implementation of Distributed File system in Cloud OS / Mobile OS.

- L :45 P:0 T: 45 PERIODS

FILE SYSTEMS

- File concept
- Access methods
- Directory structure
- Files System Mounting
- File Sharing
- Protection

Introduction

- The operating system defines a logical storage unit called a file.
- The files are stored in disk blocks in the disk.
- The mapping of files onto the disk physical devices is done by the operating system.
- The physical devices are nonvolatile.
- Information is stored on different storage media.
- For example, hard disks, pen drives etc. are used to store information.

Introduction

- In a disk, data are stored in small units called as disk blocks.
- That is, the disk is logically divided into disk blocks in which data are stored.
- The user need not be aware that there are disk blocks in the disk where information is stored.
- It is enough for the users to understand information in terms of files.
- In this module we learn the uses of files and file systems, different attributes and types of files, understand file operations.
- We also discuss different file access methods.

File

- A file is a named collection of related information that is recorded on secondary storage.
- The file is the smallest allotment on secondary storage.
- A file may represent programs or data.
- That is, a file may be a program file or a data file.
- The program files can be source programs, objects programs and so on.
- The data files can have numeric data, alphabetic data, alphanumeric data, binary data and so on.

File

- A file has a defined structure depending on the type of the file.
- For example, the text file is a sequence of characters organized into lines.
- A source file has a source program that has a sequence of subroutines and functions, organized as declarations followed by executable statements.
- An object file has a sequence of bytes organized into blocks understandable by the linker.
- An executable file has a series of code sections that the loader can bring into the memory and execute.

File System

- The file system consists of a collection of files.
- When there are a number of files kept in the secondary memory, it is better to keep the files organized.
- For example, similar types of files can be grouped and the group can be given a name.
- This group is called a directory.
- Many directories can be grouped under another directory and so on.
- Thus, the directory structure organizes and provides information about all the files in the system.
- This forms the file system.
- To physically or logically separate large collections of directories, partitions are maintained.
- Each partition can have a different file system.

File Attributes

- Each file has a number of attributes indicating some information of the file.
- The most common attributes are the
- name,
- type,
- location,
- size,
- protection bits,
- time,
- date and
- user identification.

File Attributes

- **Name** is the only information kept in human-readable form. This is the name given to identify the file.
- **Type** is the type of the file and is needed for systems that support different types. For example, the different types of files are text files, image files and so on.
- **Location** points to the location of the file on the disk. That is the disk block in the disk where the contents of the file are kept.
- **Size** refers to the current file size.

File Attributes

Protection bits control who can do reading, writing, executing. Some users can read, some can write, some can execute or some can have a combination of different permissions.

Time, date, and user identification

– Information kept for last creation, last modification and last use

– Data useful for protection, security, and usage monitoring.

Information about the attributes of the files is kept in the directory structure, which is maintained on the disk.

File Operations

- The operating system provides a number of system calls to create, write, read, reposition, delete and truncate files. We now see how each of these operations is carried out.

File Operations

- Create
- This operation creates a new file. First, it is necessary to find if there is space for the file in the disk. Then, an entry is made for the new file in the directory. The directory entry stores information about the file like the name of the file, location of the file in disk and so on.
- Write
- This operation is used to write contents into a file. The system searches the directory and finds the location of the file. Using this location, contents can be written into the file. The system keeps a write pointer to the location in the file where the next write has to take place. The write pointer is updated after a write occurs.

File Operations

- **Read**
- The read operation is used to read the contents of a file. The system call used for reading specifies the name of the file and where the next block must be read from. The system needs to keep a read pointer to the location in the file where the next read is to take place. The read pointer is updated after read has taken place. For a particular file, the file position pointer need not be the same for all processes that access the file. Each process maintains its own file position pointer for a particular file.

File Operations

- **File seek**
- This operation is used to reposition the file pointer within the file. Whenever a read or a write operation is done, the read or write is done on the location which is pointed to by the file pointer. This value of the file pointer or the current-file-position can be modified using the seek operation. The system call used for seek searches the directory for the appropriate entry. The current-file-position maintained in the directory structure is set to the given value. This operation does not access the contents of the file kept in the disk and hence, no I/O is needed. The value of the file pointer alone is changed in the directory structure.

File Operations

Delete

This operation is used to delete a file. The name of the file to be deleted is provided in the system call. The file is searched in the directory. The space allocated for the file in the disk is released. The directory entry created for the file is erased.

Truncate

This operation releases all the contents of the file. The file is not deleted. The file length is reset to zero so that it can be overwritten. The space allocated to the file in the disk is released.

File Operations

Append

This operation is used to add new information to the end of the file. The current file position is moved to the end of the file and the contents to be added are written from that position.

Rename

The name of the file is changed to the new name provided in the system call. For this, the directory entry is modified. The old name is removed and the new name is entered.

File Operations

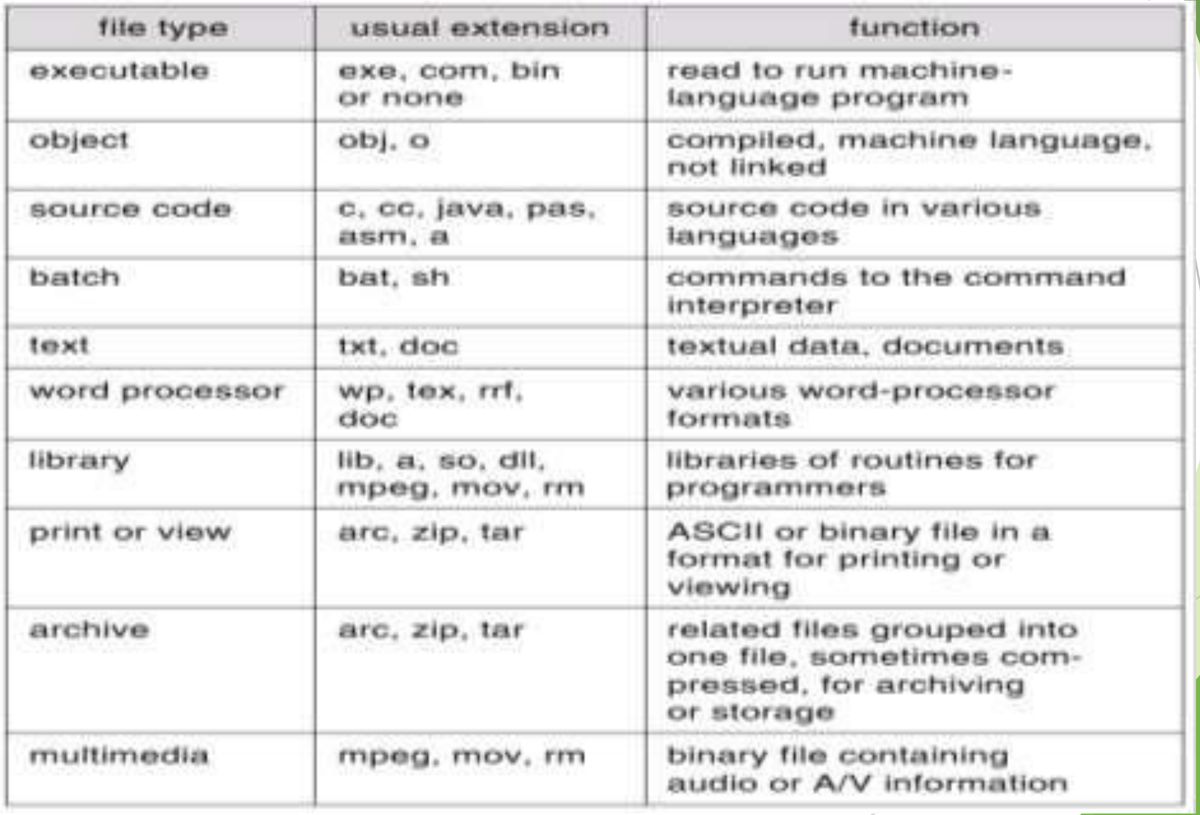
Copy

- This operation is used to make a copy of an existing file. The name of the old file and the name of the new file (copy) to be created are provided through the system call. A new file is created, contents of the old file are read and written to the new file.

File Types – Name, Extension

- Different file types are supported by operating systems.
- Each file type can have different extensions.
- Figure shows examples of different file types and different extensions for each file type.
- For example a source code file can have the extensions .c or .cc or .java and so on.

File Types – Name, Extension



file type	usual extension	function
executable	exe, com, bin or none	read 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, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

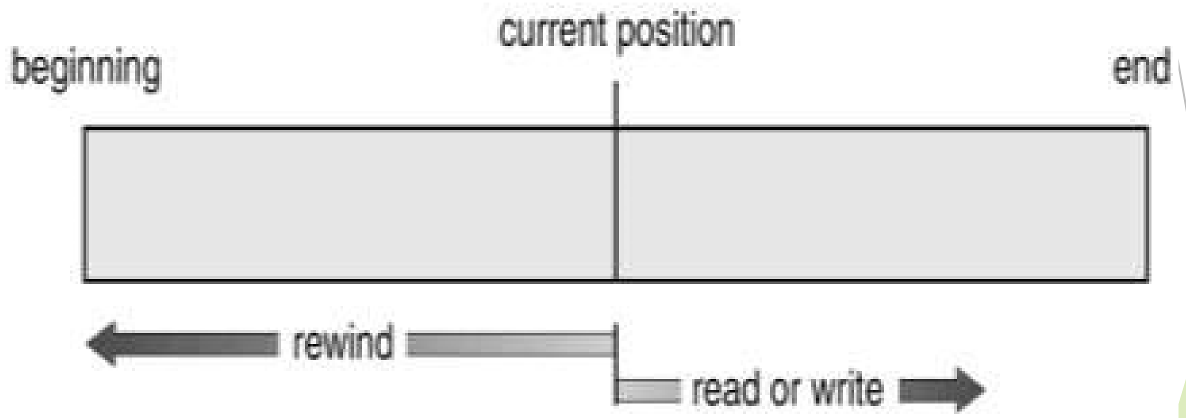
File Access Methods

- Information stored in files must be accessed.
- There are different methods to access files.
- Some systems support only one method.
- Some systems support multiple methods and the right one is chosen based on the application.
- The different methods of file access are sequential access, direct access and indexed sequential access.
- We learn each of these methods in this section one after the other.

Sequential Access

- The sequential access is the simplest and the most common file access method. In this method, the contents of a file are accessed one after the other. That is, information is processed one record after another. The possible operations can be
- read next – read the next record
- write next – write the next record
- reset – reset the file pointer to the initial record

Sequential Access



Sequential Access

- Figure shows the possible operations using sequential access method.
- The current file position is shown.
- From the current file position, it is possible to read the next record or write the next record or rewind back to the beginning of the file.
- Using the sequential access method, it is not possible to access any disk block in random.

Direct Access

- The direct access method is based on the disk model of the file.
- File contents are stored in disk blocks in the disk.
- Disks allow access to any random block.
- Direct access also allows arbitrary blocks to be read and written.

Direct Access

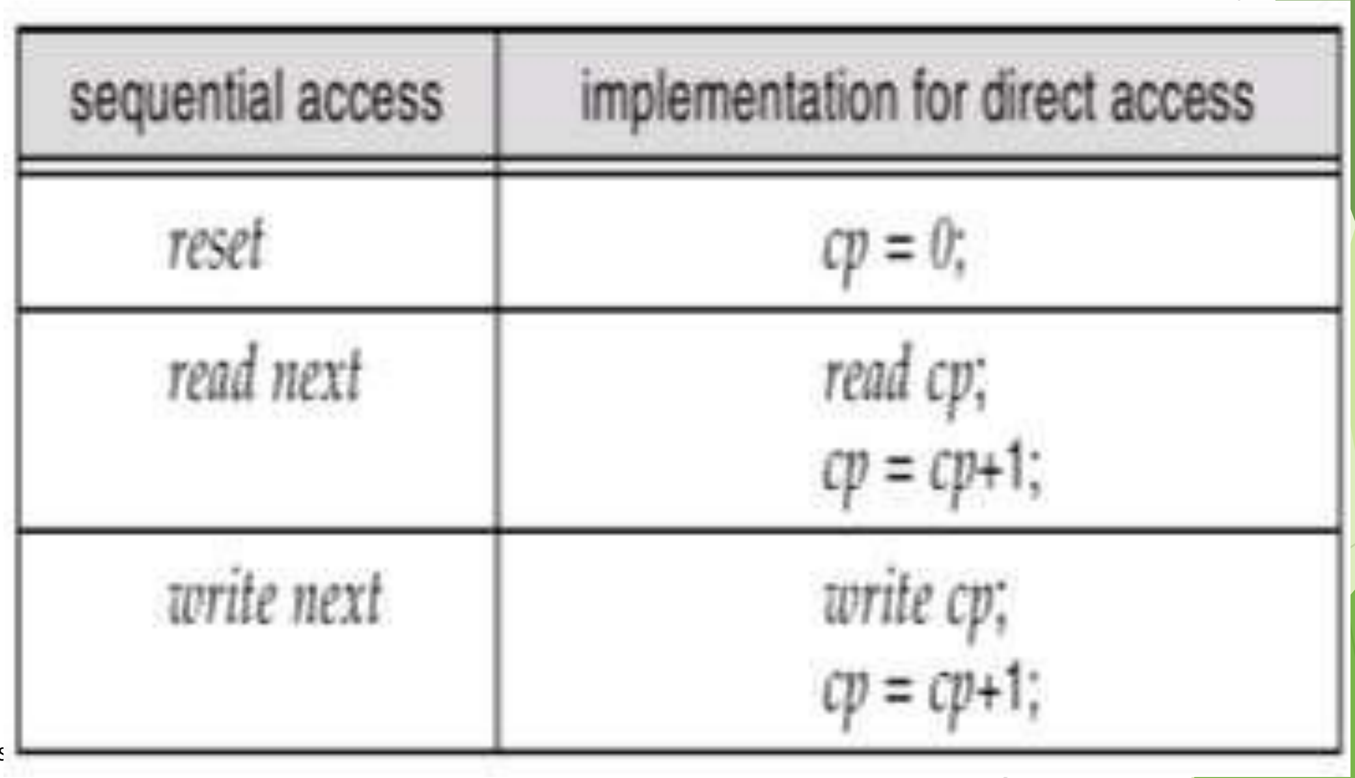
- With direct access, the following operations are possible:
-
- *read n* – read the nth block
- *write n* – write to the nth block
-
- *position to n* – move the pointer to the nth block
- *read next* – read the next block after moving to the nth block
- *write next* – write to the next block after moving to the nth block
- *rewrite n* – rewrite the nth block

Direct Access

- where n is the block number relative to beginning of file

- it is possible to implement sequential access using the operations used for direct access.
- Figure shows the sequential access operations and the corresponding operations in direct access method.
- Thus, it is possible to simulate sequential access on a direct file.
- But, direct access cannot be simulated on a sequential access file.

Direct Access



sequential access	implementation for direct access
reset	*cp = 0;*

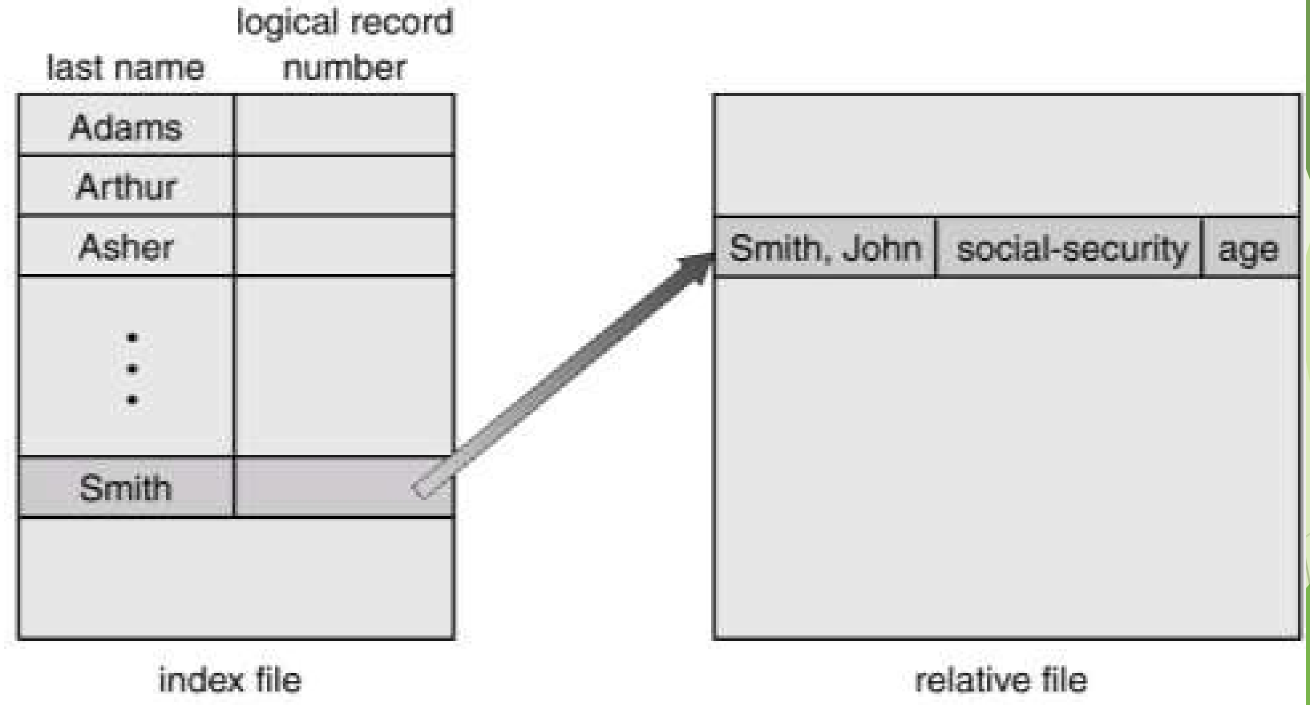
read next	*read cp;*
	cp = cp+1;

write next	*write cp;*
	cp = cp+1;

Other Access Methods

- In another access method, an index is built which contains pointers to various blocks of the file.
- To find a record, the index is searched, and the pointer is used to access the file and the desired record.
- Searching through an index is faster.
- When the file size becomes large, the index file also becomes large.
- Therefore, to make the access faster, another secondary index can be maintained for the primary index file.
- The primary index file will have pointers to secondary index files, and the secondary index files point to blocks of the file.
- Figure shows an example of how an index file is used.

Other Access Methods



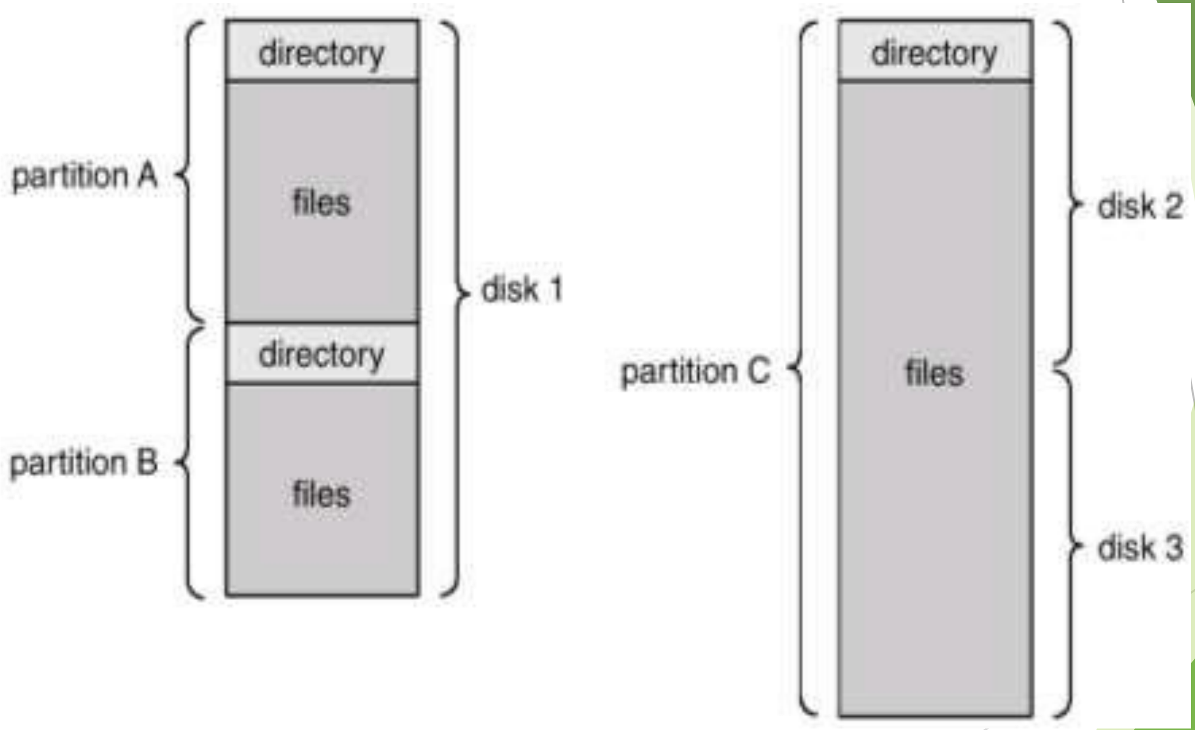
Disk Structure

- The disk which can be in the size of terabytes can be subdivided into partitions.
- A disk or partition can be used raw – without a file system, or can be formatted with a file system.
- The partitions of the disk are also known as minidisks or slices.
- Each disk has at least one partition.
- Partitions can store multiple operating systems.
- That is, each partition can have a different operating system.

Disk Structure

- Each entity containing a file system is known as a volume.
- Each volume containing a file system also needs to track that file system's information.
- In each volume, this information is maintained in a device directory or volume table of contents.
- The device directory records information such as name, location, size, type for all files in that volume.

Disk Structure

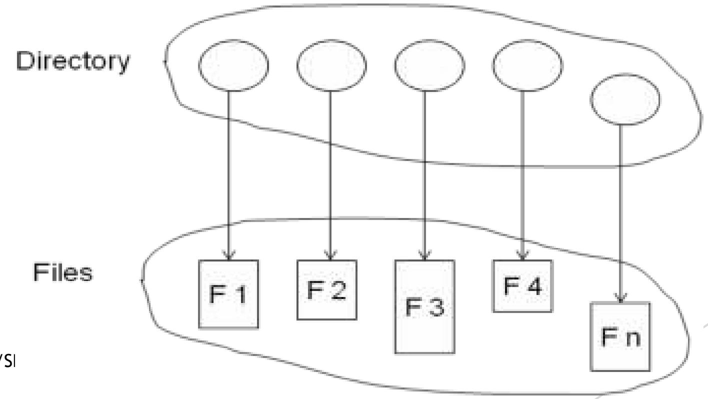


Disk Structure

- Figure shows the organization of a typical file-system.
- Disk1 has two partitions.
- Partition A has a file system and a corresponding device directory.
- Partition B has another file system and a corresponding device directory.
- It is also possible for a partition to cover two disks.
- Partition C has a file system that is kept in disk2 and disk3.

Directory Structure

- A directory structure is a collection of nodes containing information about all files that are kept in the disk.
- Both the directory structure and the files reside on the disk.
- The backups of these two structures are kept on tapes.
- Figure shows a few directories and files under the directories.



Operations Performed on Directory

- Similar to how operations can be performed on files, there are operations that can be performed on directories.
- Some of the operations that can be performed on directories are given below:

- Search for a file – A directory has information about all the files present in the directory. To search for a file, it is necessary to search the directory.

- Create a file – To create a file, an entry is created in the directory. For this, it is necessary to write into the directory.

Operations Performed on Directory

- Delete a file – To delete a file, it is necessary to remove the name of the file and all other details about the file from the directory. This again needs write permissions in the directory.

- List a directory – For listing the contents of a directory, it is necessary to read from the directory.

- Rename a file – To change the name of the file, it is necessary to read, write and search in the directory. Note that renaming a file may change the position of the file name in the directory.

- • Traverse the file system – This also needs reading and searching operations on the directory.

Organizing the Directory

- It is necessary to organize a directory logically so that the following are achieved:

- **Efficiency**

  locating a file quickly.

  The directory should be organized such that files can be located quickly when searching.

Organizing the Directory

- **Naming**
  - convenient to users.
  - The names of the files cannot be arbitrary names and must be easier for the users to remember the names.
  - Two users cannot have the same name for different files.
  - The same file can have several different names.

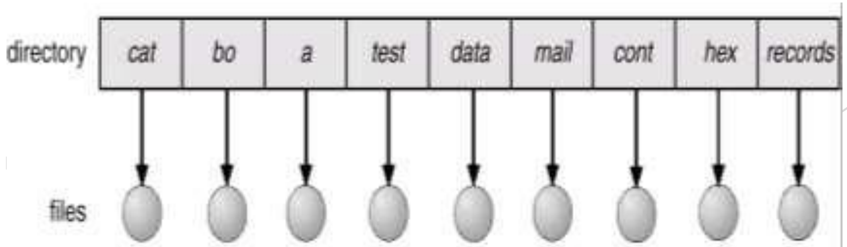
- **Grouping**
  - Based on the properties of files, it is necessary to logically group the files.
  - For example, all Java programs may have to be put under one directory, all games under another directory and so on.

Logical Structure of directory

- Single-Level Directory
- Two-Level Directory
- Tree-Structured Directories
- Acyclic-Graph Directories
- General Graph Directory

Logical Structure of directory

- **Single-Level Directory**
- In this scheme, a single directory is kept for all the users.
- This is the simplest scheme. But there are limitations in this scheme.
- Since there is only one directory, it becomes difficult to manage the files when the number of files increases.
- When the system has more than one user, it becomes difficult for different users to assign unique names to files.
- Even a single-user finds it difficult to remember the names of his/her files because all files are kept in the same directory.
- Figure shows an example of a single-level directory.
- The files cat, bo, a, ..., records are kept in a single directory.



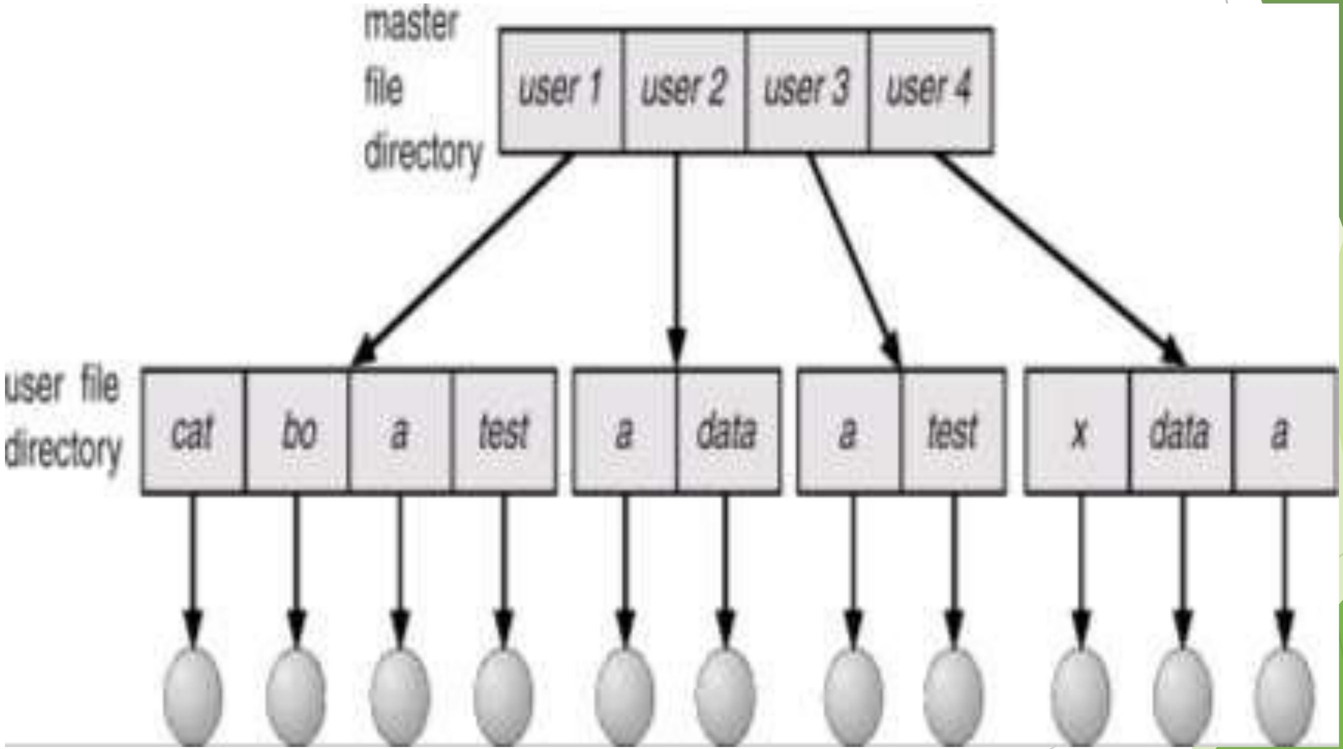
Logical Structure of directory

- **Two-Level Directory**
- In a two-level directory, there is a master file directory (MFD) and under this directory, a separate directory is assigned for each user
- This scheme has more advantages compared to the single-level directory.
- Different users can use the same file name.
- Creation and deletion of files are confined to the user's user file directory (UFD).
- Therefore, naming becomes easier in this scheme.
- But the limitation is that users can't cooperate and access one another's files.
- To access another user's file, the path name of that file should be specified.
- The path name of a file is specified using the name of the MFD (root) followed by the name of UFD and the name of file.

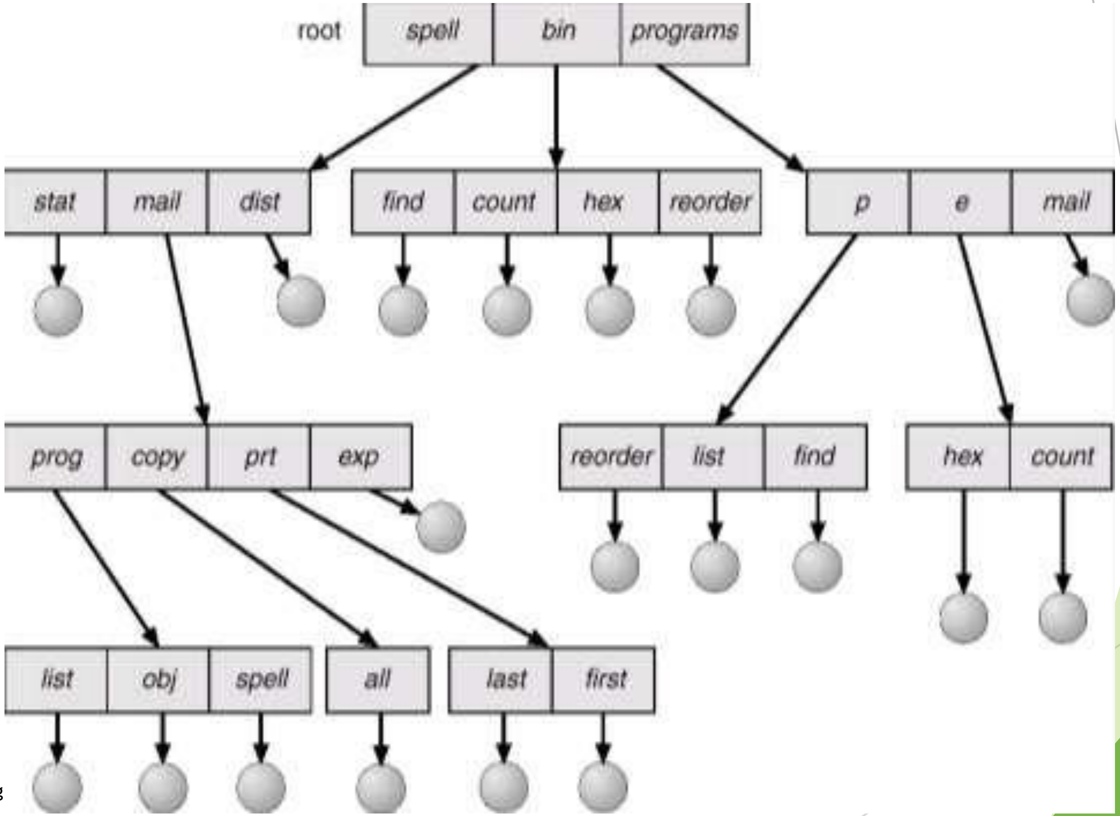
Logical Structure of directory

- Another limitation in this system is the placing of system files like loaders, assemblers and compilers.
- These files are needed by all the users.
- If these files are copied to each UFD, it is waste of space.
- Therefore, a special user directory can be created to contain the system files.
- For accessing files that are not kept in the current directory, a search path can be defined. By default, files are first searched in the current directory.
- If the files are not available in the current directory, the files are searched in the directories specified in the search path.
- The system files can also be searched in this manner.

Logical Structure of directory



Tree-Structured Directories



Tree-Structured Directories

- The two-level directory was extended to have multiple levels and the tree-structured directory was formed.
- The tree is the most common directory structure.
- Figure shows an example of a tree-structured directory.
- The directory is treated as a special kind of file.
- Under a directory, there can be files as well as other subdirectories.
- A bit is used to differentiate between a file and a directory present in the same level.
- The tree has a root directory and beneath the root directory, there are a number of subdirectories.
- System calls are available for creating and deleting directories. Every file has a unique path name.

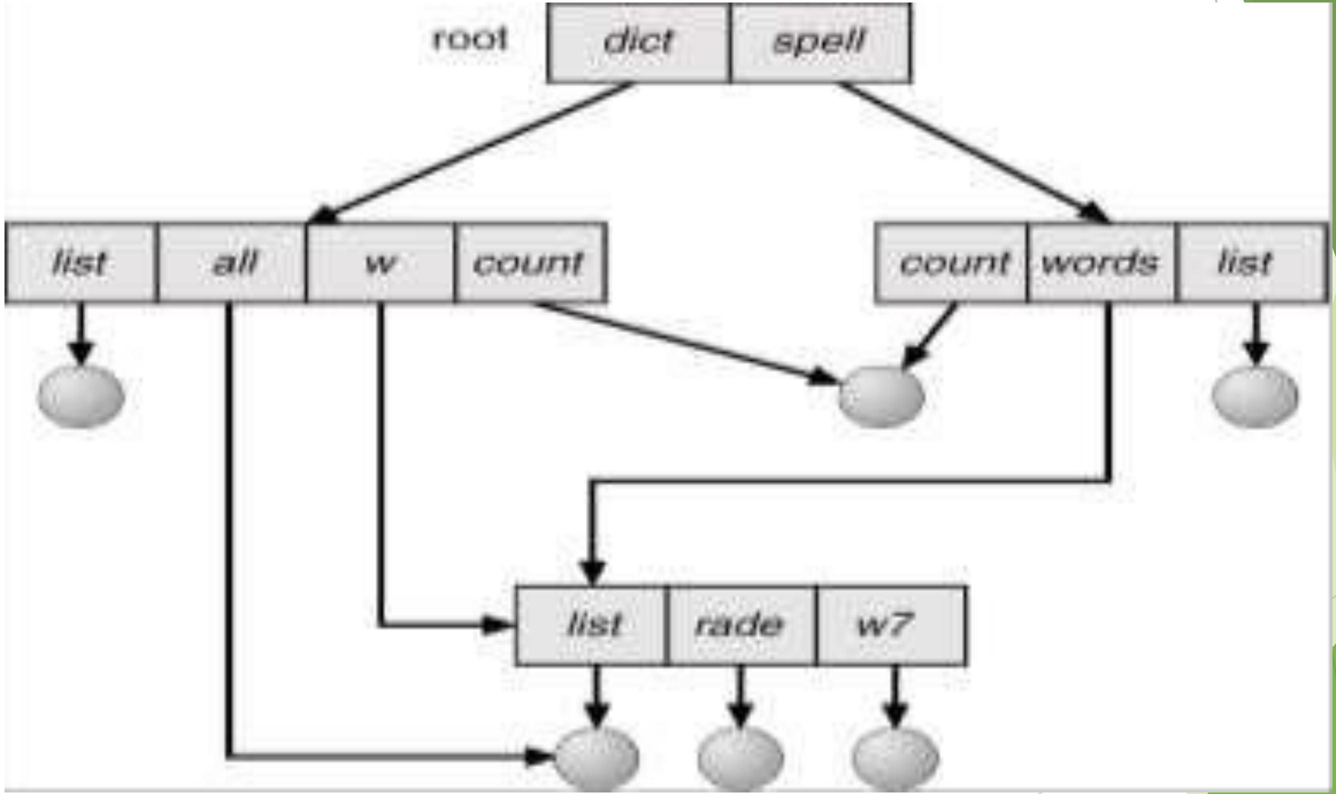
Tree-Structured Directories

- The introduction of tree-structured directories introduced two types of path names called the absolute path name and the relative path name.
- If the path name starts from the root, it is called as absolute path name.
- If the path name starts from the current directory, it is called as relative path name.
- For example, if root/spell/mail is the current directory, prt/first is the relative path name and absolute path name is root/spell/mail/prt/first.

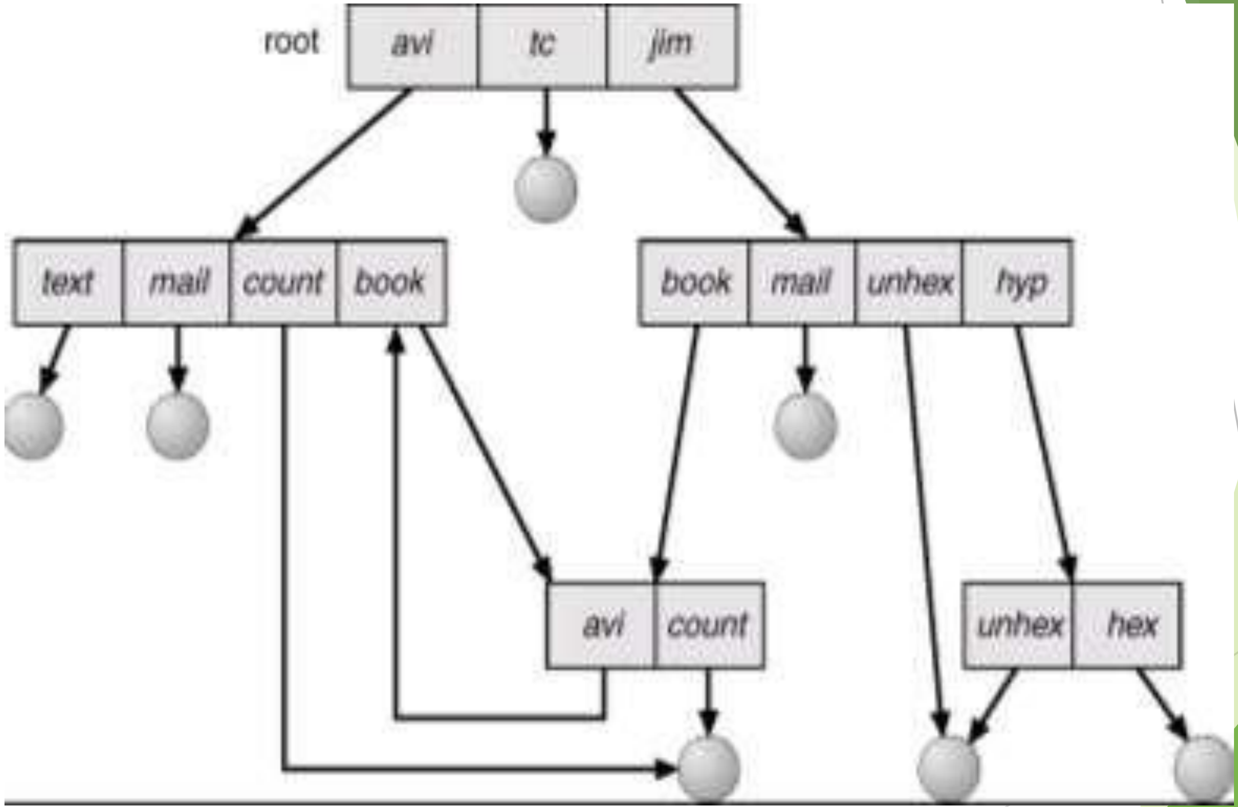
Acyclic-Graph Directories

- To overcome the disadvantage of tree-structured directories, acyclic-graph directories were formed which have shared subdirectories and files.
- Figure shows that file *count* is shared by the directory *spell* and the directory *dict*.
- Therefore, the file *count* has two different path names, /dict/count and /spell/count.
- This sharing of files is helpful when there is more than person working on a project and have to access a common file.
- This is not the same as having two copies of the same file.

Acyclic-Graph Directories



General Graph Directory



General Graph Directory

- The general graph directory is similar to the tree-structured directory except that cycles are allowed in the structure.
- A cycle is path of edges and vertices wherein a vertex is reachable from itself.
- Here, vertices denote directories or files and there is an edge between a file/subdirectory and a directory if the directory is the parent directory of the file/directory.

File-System Mounting

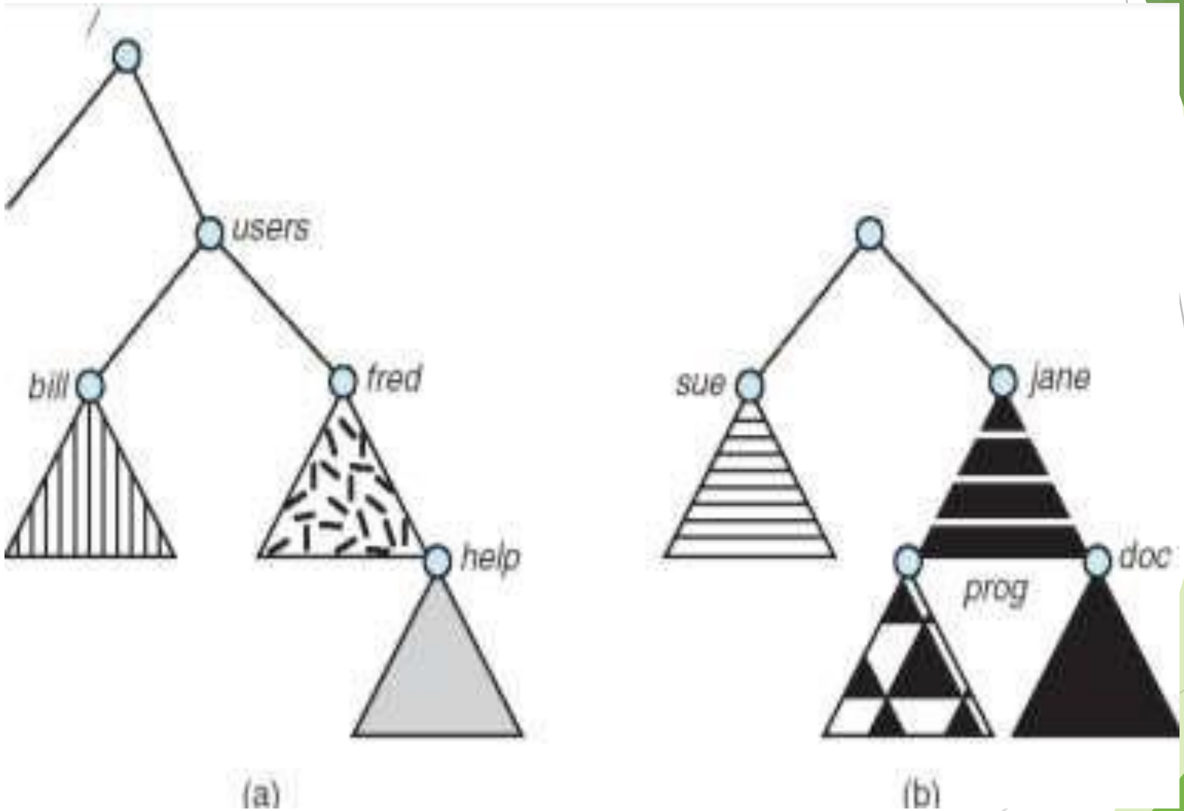
- The file system is the most visible part of an operating system.
- File systems are kept in secondary storage devices.
- File systems can be kept in different partitions, different disks, pen drives and so on.
- In this module we will learn how a file system can be mounted on another file system, how sharing of files is supported in file systems and how files can be protected.

File-System Mounting

- Similar to how a file must be opened before use, a file system must be mounted before it can be accessed.
- A single file system can be built out of multiple partitions or there can be a separate file-system in each partition.
- To logically attach different file systems together and to view the files of different file systems, mounting is done.

- The mount procedure is as follows:
- There is a *mount* command/system call which is used for mounting.
- The system is provided with the name of device containing the file system to be mounted and the mount point as arguments to the mount call.
- The mount point is the directory at which the mounted file system will be attached.

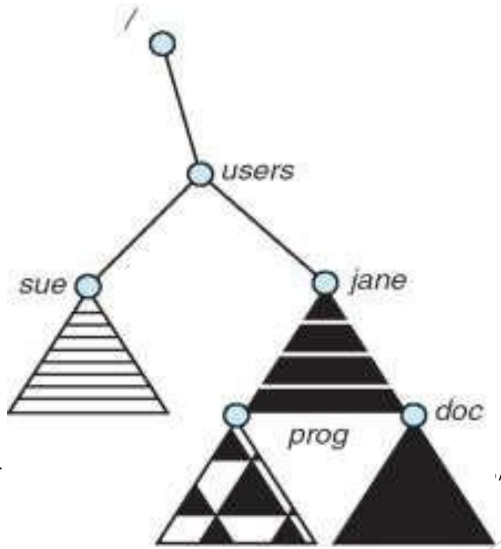
File-System Mounting



(a) Existing file system. (b) Unmounted file system

File Sharing

- Sharing of files is desirable when users want to collaborate and want to achieve a computing goal.
- Therefore, operating systems must provide support to share files, in spite of the difficulties.
- In this section, we will discuss different issues that may arise when files are shared.



File Sharing – Multiple Users

- In a single user system, the need for the sharing of files may not arise.
- But, in a multiuser system, more protection and access control are needed for file sharing.
- Therefore, the system must maintain more file and directory attributes than are needed in a single-user operating system.
- Most systems use the concepts of owner (user) and group.
- The owner is responsible for changing attributes and granting access and has the most control over the file.
- A group defines the subset of users who can share access to the file.

File Sharing – Multiple Users

- For example, in UNIX, the owner of a file can issue all operations on a file like reading, writing and executing.
- A group can have permissions to issue a subset of operations and all others (other users) can have different access permissions.
- The owner and group are assigned IDs and the owner and group IDs are stored along with the other attributes of the file.
- Even with multiple local file systems, ID checking and permission matching are straightforward, once the file systems are mounted.
- But, what happens when the file systems are not local, but placed in different locations connected through a network?

File Sharing – Remote File Systems

- With the advent of computer networks, communication among remote computers became possible.
- Networking allows the sharing of resources spread across the world.
- Data in the form of files is one such resource.

- Files can be shared using the following methods:
- In the first method, the files are transferred manually via programs like FTP.
- In the second method, a distributed file system is used, in which remote directories are visible from a local machine In the third method, the World Wide Web is used.
- A browser is used to gain access to the remote files.
- The World Wide Web uses anonymous file exchange.

File Sharing – Remote File Systems

- The client-server model allows clients to mount remote file systems from servers.
- A server can serve multiple clients and a client can use multiple servers.
- The NFS is a standard UNIX client-server file sharing protocol.
- The Common Internet File System (CIFS) is standard Windows protocol.

File Protection

- It is necessary to keep files safe from physical damage (reliability) and from improper access (protection).
- For keeping the files reliable, it is necessary to have duplicate copies of files.
- We can also periodically copy disk files to tape at regular intervals.

- We now see how files can be protected from improper access.
- The owner/creator of the file should be able to control what can be done on a file and by whom.
- The types of access that can be controlled are read, write, execute, append, delete, list, renaming, copying etc.

- One way in which access can be controlled is to have access control lists and groups.

Access Control Lists and Groups

- With each file and directory an access-control list (ACL) is attached.
- The ACL has the names of users and the types of access allowed for each user.
- When a user requests access to a particular file, the access list is checked.
- If the user is listed for that particular access, access is allowed. Else, user is denied access.

Access Control Lists and Groups

- With each file and directory an access-control list (ACL) is attached.
- The ACL has the names of users and the types of access allowed for each user.
- When a user requests access to a particular file, the access list is checked.
- If the user is listed for that particular access, access is allowed. Else, user is denied access.

Implementation of File Systems

- Introduction

- A file is a collection of related information.
- Files are organized into a structure called the file-system.
- The file systems reside on secondary storage (disks).
- The file systems provide efficient and convenient access to the disk by allowing data to be stored, located and retrieved easily.
- In this module, we understand the different layers in a layered file system, the on-disk structures and the in-memory structures used for the implementation of file systems and issues related to the implementation of directories.

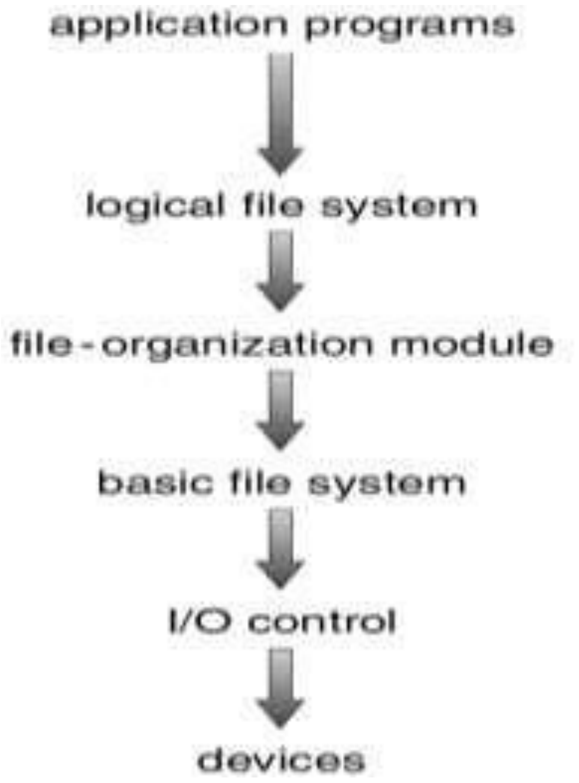
Implementation of File Systems

File-System structure

- A file system is organized into many layers.
- The different layers of a file system are shown in Figure.
- The application programs written by users are shown in the top most layer.
- The users' programs use the logical file system.
- The logical file system invokes different data structures and the file-organization module.
- The file-organization module uses the basic file system, which, in turn uses I/O control, which, in turn accesses the I/O devices.
- We now see the functionalities of each of these layers in the subsequent subsections.

Implementation of File Systems

File-System structure



Implementation of File Systems

File-System structure

- **I/O control level**
- This layer is placed just above the I/O devices.
- This level comprises of the device drivers and the interrupt handlers.
- The device drivers act as an interface between the devices and the operating system.
- They help to transfer information between the main memory and the disk.
- An example of input given to a device driver is – 'retrieve block 123'.
- In response to this, the device driver has to send low-level hardware specific instructions to the disk controller.
- The disk controller assists in reading block 123 from the disk.

Basic file system

- This layer issues generic commands to the device driver to read and write physical blocks on the disk.
- The input received from the I/O control layer is sent from this layer.
- Memory buffers and caches are maintained by the operating system in the main memory.
- The basic file system layer is responsible for managing the memory buffers and caches.
- Each memory buffer can hold contents equal to the size of a block. Each buffer holds the contents of a disk block.

Basic file system

- The contents that are read from the disk are copied to the buffers and can even be used later.
- The cache holds frequently used file-system metadata.
- This can be the contents of the file or attributes of the file such as the owner of the file, size of the file and so on.
- If the file is used frequently, the metadata can be used from the cache itself. It is not necessary to read from the disk each time.

File-organization module

- This layer uses the functionalities of the basic file system layer.
- This layer knows about files and their logical and physical blocks.
- The logical blocks are with respect to a particular file.
- The logical blocks are numbered from 0 to N for a particular file.
- Physical blocks do not match the logical numbers.
- Logical block i need not be kept in block number i in the physical memory.
- It can be kept in any physical disk block.

File-organization module

- Therefore, it is necessary to know the location of the location of the file in the disk (That is the locations of the disk blocks where the contents of the file are kept in the disk).
- Therefore, appropriate data structures are maintained by the file-organization module to know the mapping between the logical block number and the physical bock number.
- The file-organization module also has a free-space manager, which tracks unallocated disk blocks.

Logical file system

- This layer lies above the file-organization module.
- This layer manages the metadata information of a file.
- The metadata information includes all details about a file except the actual contents of the file, for example, the name of the file, the size of the file and so on.
- This layer also manages the directory structure.
- It maintains the file-structure via file-control blocks.
- A File-control block (FCB) (inode in UNIX) has information about a file – owner, size, permissions, time of access, location of file contents and so on.

Advantages of layered file system

- Duplication of code is minimized.
- The code for I/O control and basic file-system layers can be used by multiple file systems.
- The layers above these two layers can be modified for different file systems.
- That is, each file system can then have its own logical file system and file-organization modules, while the I/O control and basic file-system layers being the same.

Disadvantages

- Having a layered file system can introduce more operating-system overhead, resulting in decreased performance.
- The decision about how many layers to use, what each layer should do is a challenge.
-
- Many file-systems are in use today – UNIX file system, FAT, FAT32, NTFS, ext3, ext4, Google.
- The FAT, FAT32 and NTFS are used in Windows operating systems.
- Ext3 and ext4 are used in Linux operating systems. Google has its own distributed file system called the Google File System (GFS).

File-System Implementation

- In this section, we learn different data structures that used to assist in the implementation of file systems.
- There are several on-disk and in-memory structures used for the implementation of file systems.
- The on-disk structures are kept in the disks.
- The on-disk structures contain information about how to boot an OS stored in the disk, total number of disk blocks, number and location of free disk blocks, directory structure and individual files.
- The in-memory structures are kept in the main memory.
- The in-memory structures are helpful for file-system management, caching and so on.

File-System Implementation

On-Disk Structures

- In this section, we understand the functionalities of different on-disk data structures.

File-System Implementation

Boot control block

- Operating system is kept in the boot control block.
- If the disk has no operating system, this block is empty.
- In UNIX, the boot control block is called as a boot block.
- In NTFS, the boot control block is called as a partition boot sector.
- The boot control block is usually the first block of the volume where the file system is kept.
- If a partition of a disk has an operating system, information about how to boot.

File-System Implementation

Volume control block

- This data structure contains details about a volume (partition).
- The information maintained in the volume control block are number of blocks in the partition, size of each block, number of free blocks in the partition, free-block pointers (addresses of free disk blocks) and so on.
- In UNIX the volume control block is called a superblock and is the block next to the boot block.
- In NTFS, these details are stored in the master file table.

File-System Implementation

Directory structure (for each file system)

- The directory structure is used to organize files.
- In the directory structure, the names of files and associated information are kept.
- In UNIX, the directory structure includes file names and associated inode numbers.
- In NTFS, these details are stored in the master file table.

File-System Implementation

Per-file file control block (FCB)

- For each and every file, information about that file is maintained in a file control block.
- The FCB has a unique identifier number to allow association with a directory entry.
- In UNIX, the per-file file control block is nothing but the inode.
- The inode has an inode number, which is the unique identifier.
- In NTFS, the details about the file is stored in the master file table.

File-System Implementation

In-Memory Structures

These are data structures that are maintained inside the main memory.

- Mount table

– Information about each mounted volume is maintained in the mount table.

- Directory-structure cache

– Holds directory information of recently accessed directories.

– If the same directory has to be accessed again, it is not necessary to read from the disk.

– The details can be taken from the directory-structure cache.

File-System Implementation

- System-wide open-file table

– This data structure is common to all the processes present in the system. This contains a copy of the FCB of each open file.

- Per-process open-file table

– This is a table that is available for each and every process. This per-process open-file table points to the appropriate entry in the system-wide open-file table.

- Buffers

– These are buffers that are kept in the memory to hold the contents of disk blocks.

File-System Implementation

- Each buffer can hold the contents of one disk block.
- When contents of a disk block are read from the disk, they are stored in these buffers kept in memory.
- If the contents of the same disk block are needed again, the contents are taken from the buffer and need not be read from the disk.
- Similarly, the contents present in the buffer may be modified and need not be written to the disk for each and every modification.
- It is enough to write the contents of the buffer to the disk when the buffer is to be used for some other disk block contents.

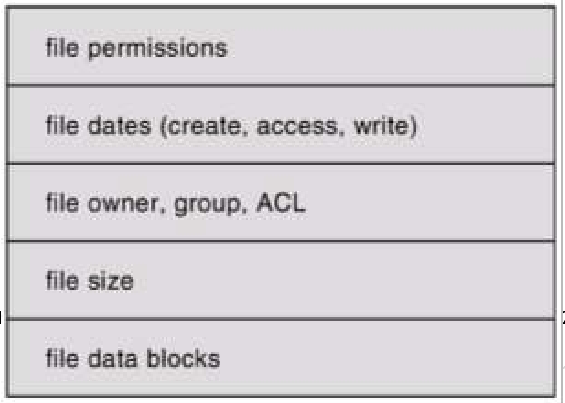
File Operations

Create a new file

- The application program calls the logical file system and gives the name of the file to be created to the logical file system.
- The logical file system knows the name of the directory structures.
- It finds the name of the directory in which the file is to be created from the file name given by the application program.
- The logical file system allocates a new FCB.

File Operations

- In the case of UNIX, a new inode is allocated.
- The system reads the appropriate parent directory into memory.
- It updates the directory with the new file name and FCB and writes the directory back to disk.
- Figure shows a typical file control block.
- The FCB has information about the file like the owner of the file, file size, file permissions and so on.



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

File Operations

- **Open an Existing File**

- The open() call called by the application program passes a file name to the logical file system.
- The call first searches the system-wide open-file table to see if the file is already in use by another process.
- If it is, a per-process open-file table entry is created pointing to the existing system-wide open-file table entry.
- If file is not already open, the directory structure is searched for the given file name.
- There is a possibility that parts of the directory structure are cached in memory.

File Operations

- If the directory structure is present in the cache, it is taken from the cache.
- Else, the directory structure is read from the disk.

- Once the file is found, the FCB is copied into an entry in the system-wide open-file table in memory.
- The FCB entry also keeps track of the number of processes that have opened the file.
- An entry is made in the per-process open-file table.
- This entry points to the system-wide open-file table.

File Operations

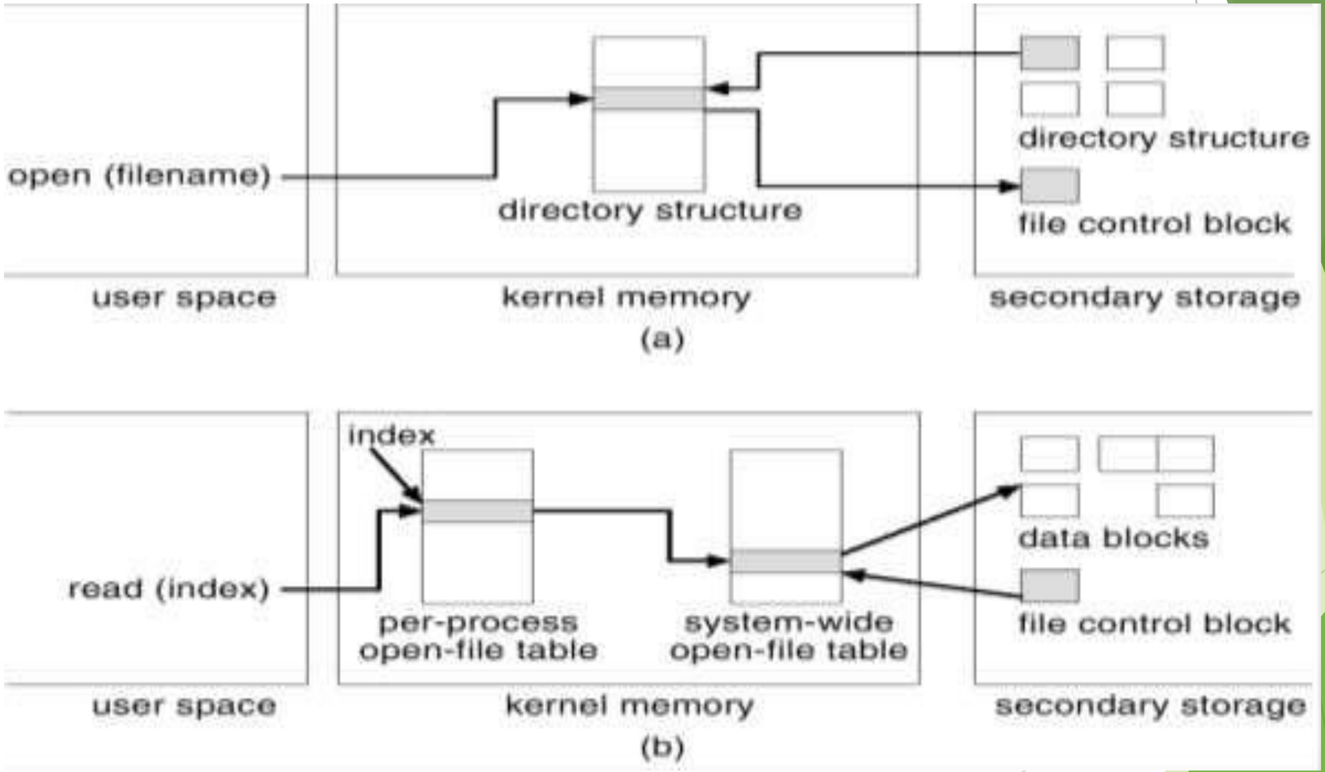
- The FCB entry also has information about where the next read/write should be done on the file (file offset), access mode in which the file is open.
- open() returns a pointer to the entry in the per-process file-system table. All file operations after the open() system call use this pointer.
- In UNIX this pointer is called the file descriptor (file handle in Windows).

File Operations

- **Close a File**

- When a process closes a file, the per-process open-file table's entry is removed.
- The count (count of the number of processes using the file) in the system-wide open-file table entry is decremented.
- When the count becomes zero, the updated metadata is copied to the directory structure in the disk and the system-wide open-file table entry is removed.

File Operations



File Operations

- Figure shows that the open call accesses the directory structure in memory.
- If the directory structure is not cached in memory, it is read from the disk.
- The file control block is accessed using the directory structure.
- If a copy of the FCB is not present in the memory, it is read from the disk.

File Operations

- Figure shows how the read system call uses the in-memory data structures.
- The read uses the index returned by the open call to access the entry in the per-process open-file table.
- The entry in the system-wide open-file table is obtained using the pointer from the per-process open-file table.
- The file control block is accessed from the system-wide open-file table and the data blocks are accessed using the entries in the FCB.

Directory Implementation

- This section explains two different ways in which a directory can be implemented.
- In the first method, a linear list of file names is maintained with pointers to the data blocks.
- This method is simple to program but time-consuming to execute.
- To create a new file, the directory is searched to find if another file with a similar name exists.
- If no such file exists, a new entry is added to the end of the directory.

Directory Implementation

- To delete a file, the directory is searched for that file and the space allocated to it is released.
- To reuse this deleted entry, the entry is marked as unused by assigning it a special name or it is attached to a list of free directory entries or the last entry in the directory is copied to the freed location and the length of the directory is decreased.
- The entries can also be maintained as a linked list to reduce the time required for deletion.
- The disadvantage of a linear list is that finding a file requires linear search and this makes access slow.

Directory Implementation

To reduce this search time

- Operating systems implement a software cache to store the most recently used directory information.
- Maintaining a sorted list also allows a binary search and reduces the search time.

Directory Implementation

- But maintaining a sorted list is difficult.
- When new entries are added, the new entries should be added to the appropriate position.

- Using a hash data structure
- –name and returns a pointer to the file name in the linear list.
- This reduces the
- – In addition to having a linear list for storing the directory entries, a hash data structure can also be used.
- The hash table takes a value computed from the file search time.
- But provision must be made for collisions, that is, it is to be ensured that two file names do not hash to the same location.

File Allocation Methods

- File system is the most visible part of the operating system.
- Files are stored in disks.
- Whenever files are created or appended, space for the files has to be allocated in disks.
- For this, the operating system should keep track of which disk blocks are free and which disk blocks are allocated.
- In this module, we learn how disk blocks are allocated to files (file allocation methods) and how the operating system keeps track of free and allocated disk blocks (free space management).

File Allocation Methods

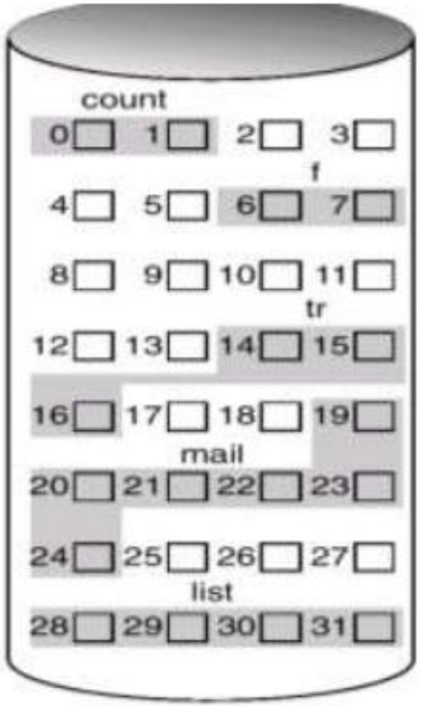
- **Allocation Methods**

- An allocation method refers to how disk blocks are allocated for a file.
- The size of disk bocks is fixed.
- Whenever a new file is created or when the size of a file increases because of writing to the file, disk blocks are allocated for the file.
- Disk space should be allocated to files such that the disk space is utilized effectively and files can be accessed quickly.
- There are three different methods of allocation:
- Contiguous allocation, Linked allocation and Indexed allocation.

File Allocation Methods
Contiguous Allocation

- In contiguous allocation, each file occupies a set of contiguous blocks on the disk.
- If only one job is accessing the disk, accessing block *b*+1 after block *b* does not require any head movement.
- It is enough to move the next sector under the current head position. Therefore, disk seeks are minimal.
- This method is simple to implement.
- It is enough to remember only starting location (starting disk block number) and the length of the file (number of blocks in the file).

File Allocation Methods
Contiguous Allocation



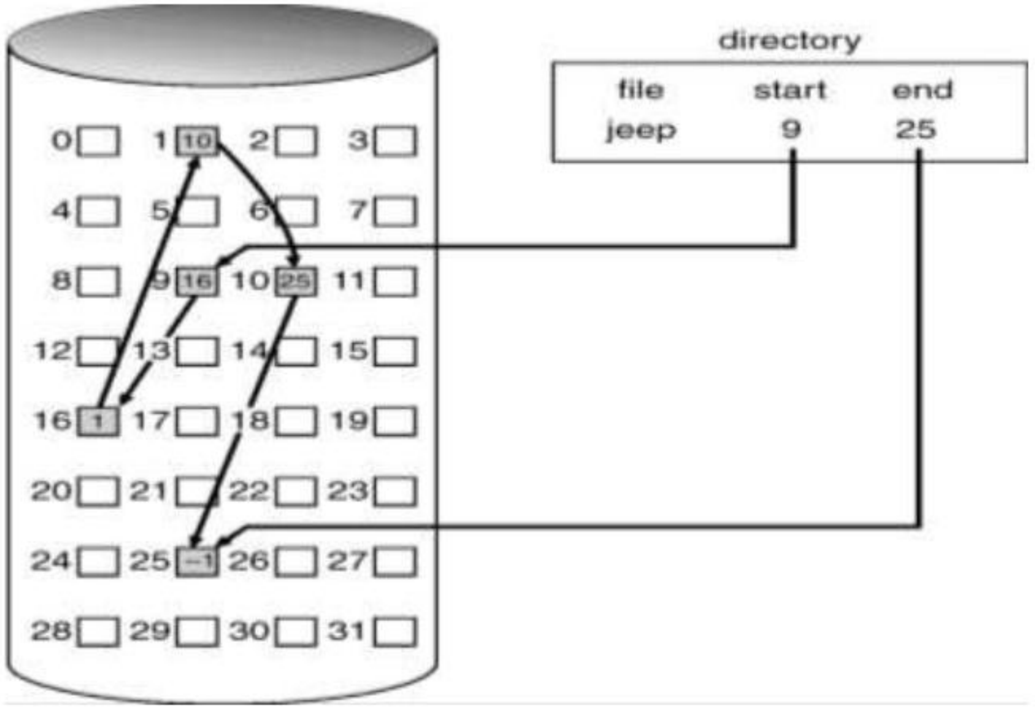
directory

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

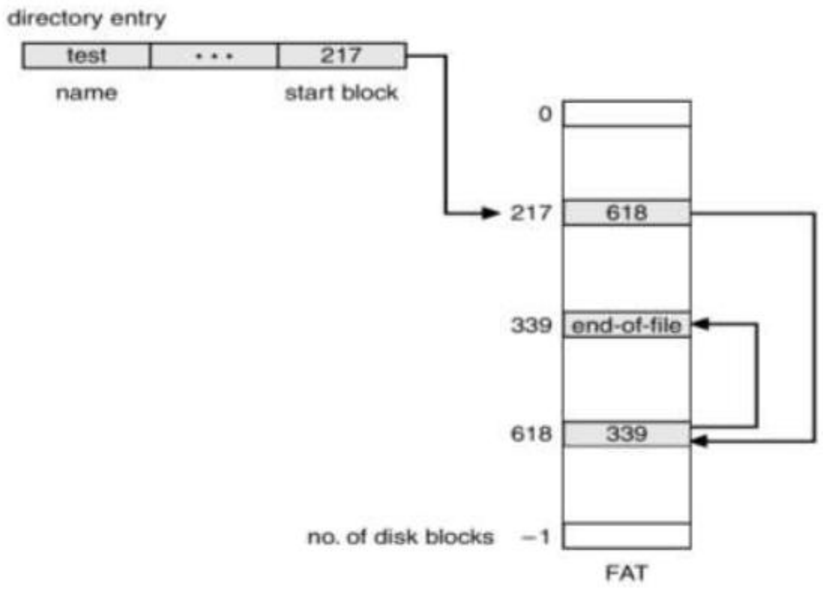
File Allocation Methods
Linked Allocation

- The linked allocation solves all problems of contiguous allocation.
- In linked allocation, each file is a linked list of disk blocks.
- The allocated disk blocks may be scattered anywhere on the disk.
- The directory entry has a pointer to the first and the last blocks of the file.
- Figure shows an example of linked allocation.
- There is a file named *jeep*.
- The starting disk block number 9 is given in the directory entry.
- Disk block 9 has the address of the next disk block, 16.
- Disk block 16 points to disk block 1 and so on.
- The last disk block of the file, disk block 25 has a pointer value of -1 indicating that it is the last disk block of the file.

File Allocation Methods
Linked Allocation



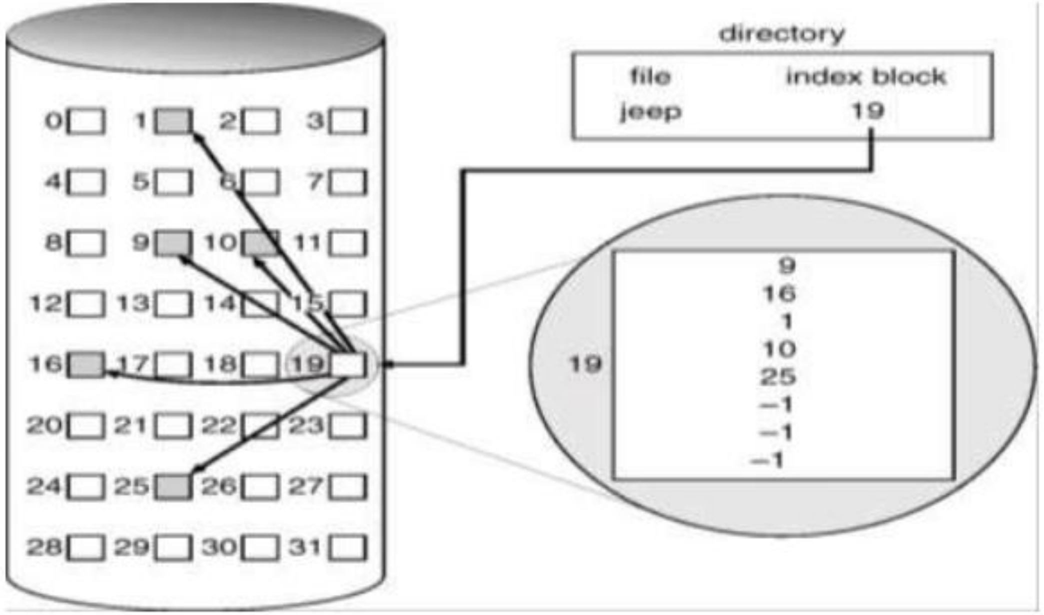
File Allocation Methods
Linked Allocation



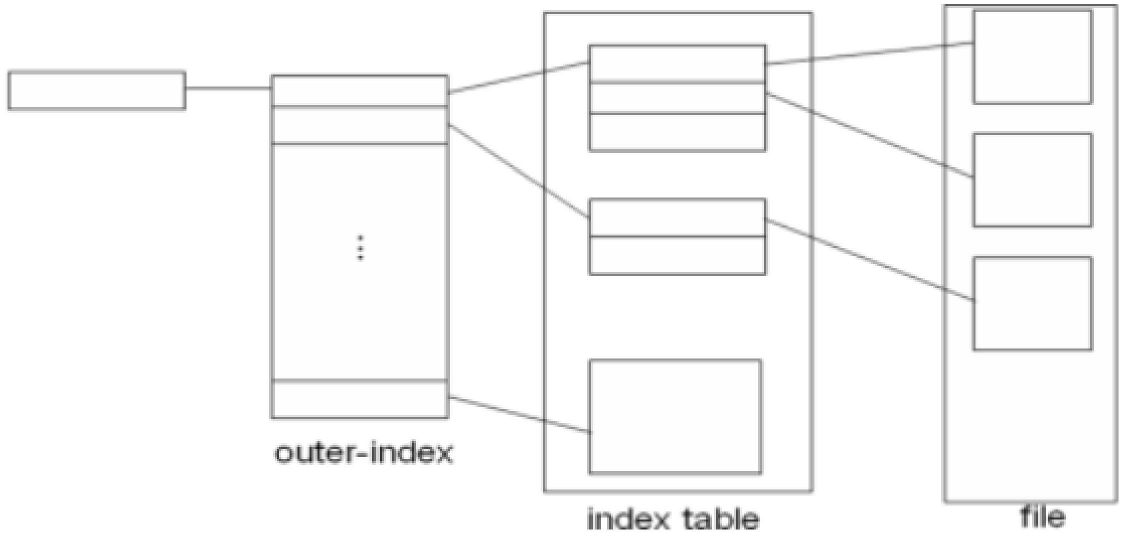
File Allocation Methods
Indexed Allocation

- The indexed allocation solves the external fragmentation and size-declaration problem of contiguous allocation.
- It also solves the direct access problem in linked allocation.
- In indexed allocation all pointers are brought together into one block called the *index block*.
- Each file has an index block, which is an array of disk-block addresses.
- The *i*th entry in the index block points to the *i*th block of the file.

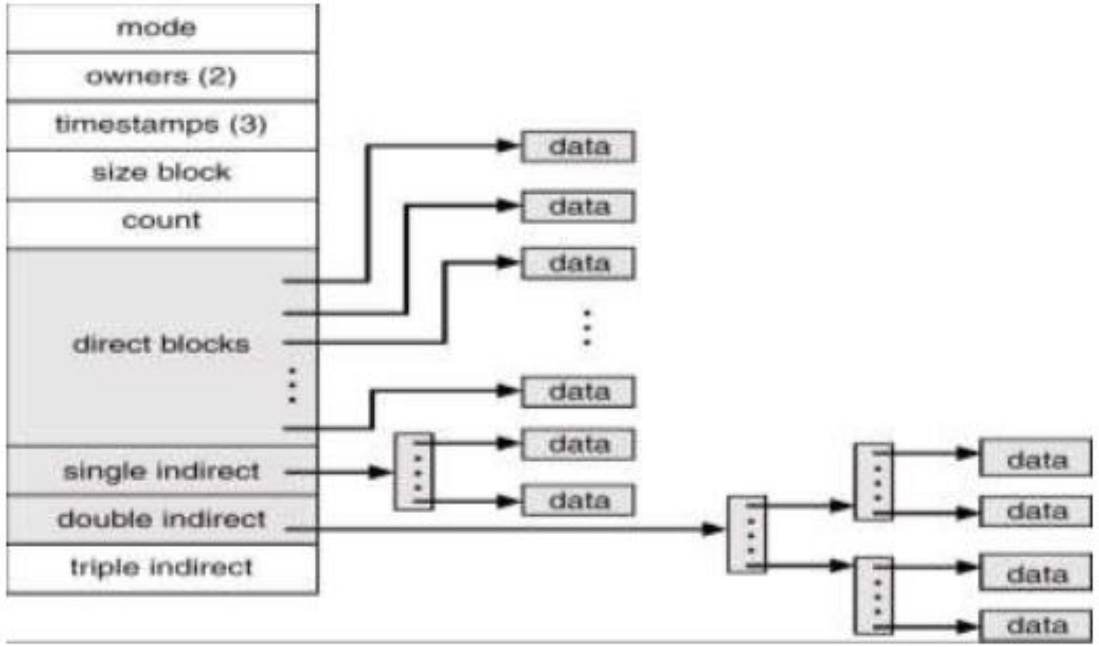
File Allocation Methods
Indexed Allocation



File Allocation Methods
Indexed Allocation



File Allocation Methods
Indexed Allocation


mode
owners (2)
timestamps (3)
size block
count
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data
data