FILE SYSTEM

File system provides a mechanism for on-line storage of and access to both data and programs of OS and all users of the computer system. The file system consists of two distinct parts:

- Collection of files, each storing related data
- Directory structure which organizes and provides information about all files in the system.

What is a file? What are the different types of file?

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

Commonly, file represents program and data. Data in file may be in numeric, alphabetic or binary form. Four types of files are:

1) **Text file**: sequence of characters organized into lines.
2) **Source file**: sequence of subroutines & functions.
3) **Object file**: sequence of bytes organized into blocks.
4) **Executable file**: series of code sections.

Mention any 5 file attributes.

1. **Name:** The only information kept in human-readable form.
2. **Identifier:** It is a unique number which identifies the file within file-system and it is in non-human-readable form.
3. **Type:** It is used to identify different types of files.
4. **Location:** It is a pointer to device and location of file.
5. **Size:** Current-size of file in terms of bytes, words, or blocks. It also includes maximum allowed size.
6. **Protection:** Access-control information determines who can do reading, writing and executing.
7. **Time, date, & user identification**: These information can be kept for creation, last Modification and last use. These data can be useful for protection, security and Usage, monitoring.

Mention any 5 file operations.

1. **Creating a file**: The two steps in creation of files are: i.) Find the space in the file-system for the file. ii) An entry for the new file is made in the directory.

2.Writing a file: Make a system call specifying both file name and information to be written to the file. The system searches the directory to find the file's location.

The write-pointer must be updated whenever a write-operation occurs.

3.Reading a file: Make a system-call specifying both file-name and location of the next block of the file in the memory. The system searches the directory to find the file's location. The read-pointer must be updated whenever a read-operation occurs. Same pointer (rp & wp) is used for both read- & write-operations. This results in → saving space and reducing system-complexity.

4. Repositioning within a file: Two steps are: Search the directory for the appropriate entry.

Set the current-file-position to a given value. This file-operation is also known as file seek.

5.Deleting a file: Two steps are: Search the directory for the named-file. Release all file-space and erase the directory-entry.

6.Truncating a file: The contents of a file are erased but its attributes remain unchanged.

Only file-length attribute is set to zero.

- The OS keeps a small table which contains info. about all open files (called **open-file table**).
- If a file-operation is requested, then
  - → file is specified via an index into open-file table
  - → so no searching is required.
- If the file is no longer actively used, then
  - → process closes the file and
  - → OS removes its entry in the open-file table.
- Two levels of internal tables:

  1) Per-process Table
  - Tracks all files that a process had opened.
  - Includes access-rights to
    - → file and
    - → accounting info.
  - Each entry in the table in turn points to a system-wide table

  2) System-wide Table
  - Contains process-independent info. such as
    - → file-location on the disk
    - → file-size and
    - → access-dates.

- Information associated with an open file:
  - **1) File-pointer**
    - ➢ Used by the system to keep track of last read-write location.
  - **2) File-open Count**
    - ➢ The counter
      - →tracks the no. of opens & closes and
      - →reaches zero on the last close.
  - **3) Disk Location of the File**
    - ➢ Location-info is kept in memory to avoid having to read it from disk for each operation.
  - **4) Access Rights**
    - ➢ Each process opens a file in an access-mode (read, write or execute).
- **File locks** allow one process to
  - →lock a file and
  - →prevent other processes from gaining access to locked-file.

Differentiate between i.) shared lock and exclusive lock ii) Mandatory and advisory lock

Shared Lock	Exclusive Lock
Similar to a reader lock.	Behaves like a writer lock.
Several processes can acquire the lock concurrently.	Only one process at a time can acquire the lock.

Mandatory	Advisory
OS will prevent any other process from accessing the locked-file.	OS will not prevent other process from accessing the locked-file.
OS ensures locking integrity.	It is up to software-developers to ensure that locks are appropriately acquired and released.
Used by windows OS.	Used by UNIX systems.

Explain different common file types

File Types

Common technique for implementing file-types: Include the type as part of the file-name.
Two parts of file-name (Figure 4.14):

1) Name and 2) Extension

The system uses the extension to indicate

→type of file and

→type of operations (read or write).

Example:

→Only a file with a .com, .exe, or .bat extension can be executed.

→.com and .exe are two forms of binary executable files.

→.bat file is a batch file containing, in ASCII format, commands to the OS.

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

Common file types

File Structure

File types can be used to indicate the internal structure of the file.
Disadvantage of supporting multiple file structures: Large size.
All OSs must support at least one structure: an executable file
In Mac OS, file contains 2 parts:

1) **Resource fork**: contains info. of interest to the user.
2) **Data fork**: contains program-code or data.

Too few structures make programming inconvenient.
Too many structures make programmer confusion.

Internal File Structure

Locating an offset within a file can be complicated for the OS.
Disk-systems typically have a well-defined block-size.
All disk I/0 is performed in units of one block (physical record), and all blocks are the same size.
Problem: It is unlikely that physical-record size will exactly match length of desired logical-record.
Solution: **Packing** a number of logical-records into physical-blocks.
Following things determine how many logical-records are in each physical-block:

→ logical-record size
→ physical-block size and
→ packing technique.

The packing can be done either by

→ user's application program or
→ OS.

Disadvantage of packing:

→ All file-systems suffer from internal fragmentation (the larger the block size, the greater the internal fragmentation).

Explain different file accessing methods.

1. Sequential access 2. Direct access 3. Indexed access.

Access Methods

Sequential Access

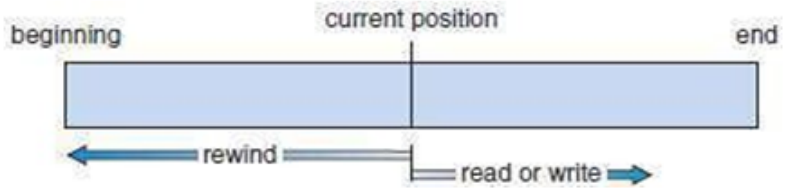
This is based on a tape model of a file.
This works both on
→ sequential-access devices and
→ random-access devices.
Info. in the file is processed in order
For ex: editors and compilers
Reading and writing are the 2 main operations on the file.
File-operations:

1) read next

- This is used to
  → read the next portion of the file and
  → advance a file-pointer, which tracks the I/O location.

2) write next

- This is used to
  → append to the end of the file and
  → advance to the new end of file.



Sequential-access file

Direct Access (Relative Access)

This is based on a disk model of a file (since disks allow random access to any file-block).
A file is made up of fixed length logical records.
Programs can read and write records rapidly in no particular order.
Disadvantages:

1) Useful for immediate access to large amounts of info.
2) Databases are often of this type.

File-operations include a relative block-number as parameter.
The **relative block-number** is an index relative to the beginning of the file.
File-operations

1) **read n**
2) **write n**

where n is the block-number

Use of relative block-numbers:

→ allows OS to decide where the file should be placed and
→ helps to prevent user from accessing portions of file-system that may not be part of his file.

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp; cp = cp + 1;
write_next	write cp; cp = cp + 1;

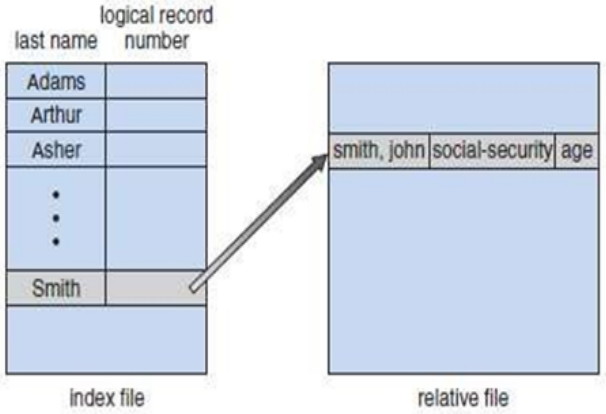
Simulation of sequential access on a direct-access file

Other Access Methods

These methods generally involve constructing a **file-index**.
The index contains pointers to the various blocks (like an index in the back of a book).
To find a record in the file

1) First, search the index and
2) Then, use the pointer to
   → access the file directly and
   → find the desired record.

Problem: With large files, the index-file itself may become too large to be kept in memory.
Solution: Create an index for the index-file. (The primary index-file may contain pointers to secondary index-files, which would point to the actual data-items).



Example of index and relative files

DIRECTORY AND DISK STRUCTURE

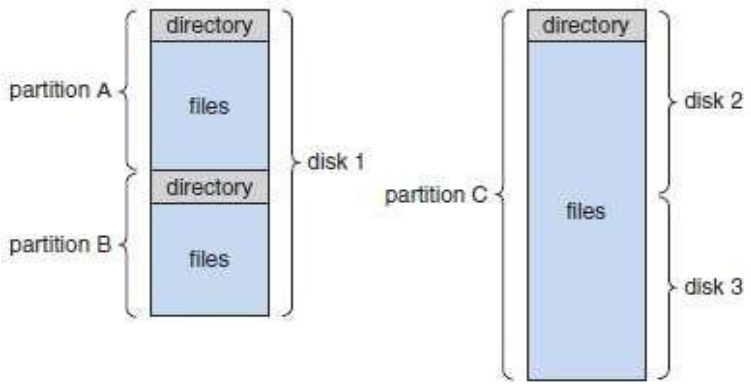
Storage Structure

A storage-device can be used in its entirety for a file-system.

The storage-device can be split into 1 or more partitions (known as slices or minidisk). Any entity containing a file-system is known as a volume. The volume may be a subset of a device or a whole device. Each volume 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).

Device directory (or directory) records following information for all files on that volume as shown in figure below:

- name
- location
- size and
- type.



Directory Overview

The various operations 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.

2) Create a File: We need to be able to create and add new files 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.

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

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.

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

For reliability, it is a good idea to save the contents and structure of the entire file-system at regular intervals.

Describe the methods used for implementing directories.

OR

******Explain various directory structures.**

The various directory structures are:

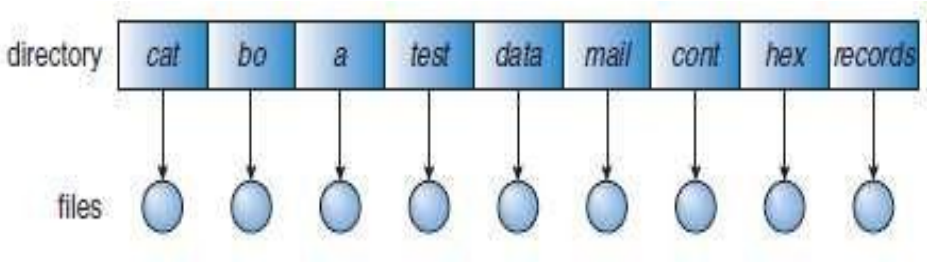
1) Single level directory
2) Two level directory
3) Tree structured directories
4) Acyclic-graph directories
5) General graph directory

1. Single Level Directory

All files are contained in the same directory as shown in below figure.

Disadvantages: i) Naming problem: All files must have unique names.

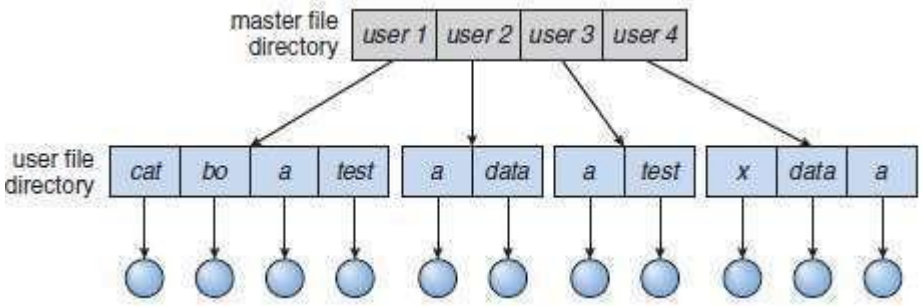
ii) Grouping problem: Difficult to remember names of all files, as number of files increases.



Single-level directory

2. Two Level Directory

A separate directory for each user is maintained. Each user has his own UFD (user file directory). The UFDs have similar structures. Each UFD lists only the files of a single user. When a user job starts, the system's MFD is searched (MFD = master file directory). The MFD is indexed by user-name. Each entry in MFD points to the UFD for that user as shown in below figure.



Two-level directory-structure

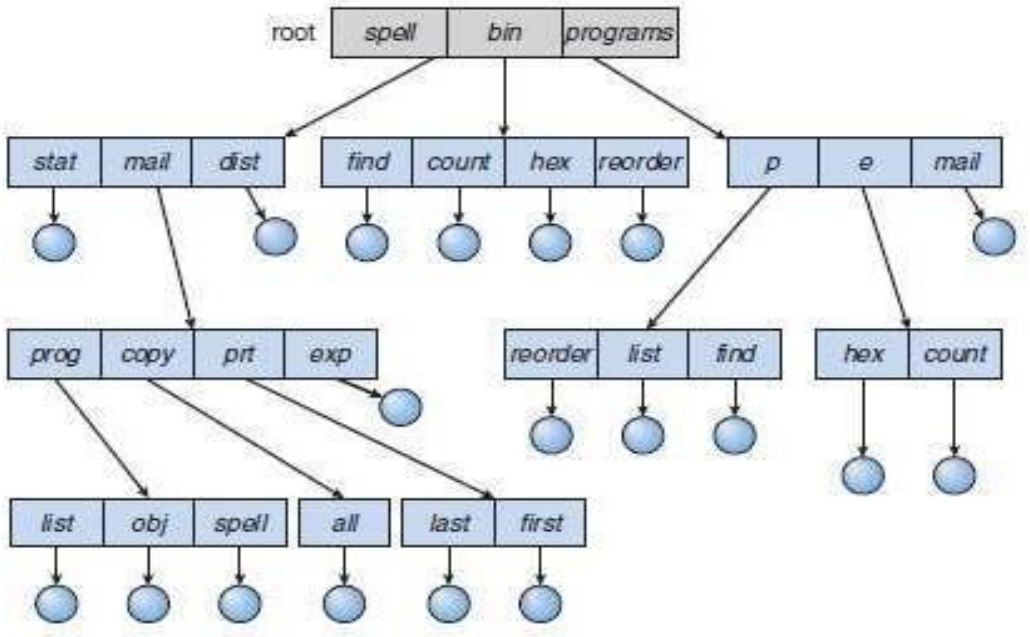
To create a file for a user, the OS searches only that user's UFD to determine whether another file of that name exists. To delete a file, the OS limits its search to the local UFD

Advantages: No filename-collision among different users and efficient searching.

Disadvantage: Users are isolated from one another and can't cooperate on the same task.

3. Tree Structured Directories

Users can create their own subdirectories and organize files as shown in below figure. A tree is the most common directory-structure. The tree has a root directory. Every file in the system has a unique path-name.



Tree-structured directory-structure

A directory contains a set of files (or subdirectories). A directory is simply another file, but it is treated in a special way. In each directory-entry, one bit defines as file (0) or subdirectory (1).

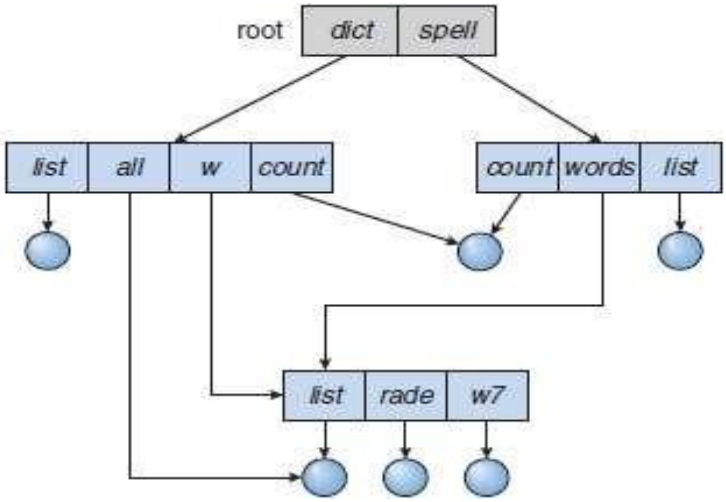
Path-names can be of 2 types: Absolute path-name begins at the root. Relative path-name defines a path from the current directory.

Advantage: Users can be allowed to access the files of other users.

Disadvantages: A path to a file can be longer than a path in a two-level directory. It prohibits the sharing of files (or directories).

4. Acyclic Graph Directories

The directories can share subdirectories and files as shown in below figure. (An acyclic graph means a graph with no cycles). The same file (or subdirectory) may be in 2 different directories. Only one shared-file exists, so any changes made by one person are immediately visible to the other.



Acyclic-graph directory-structure

Two methods to implement shared-files (or subdirectories): 1.Create a new directory-entry called a link. A link is a pointer to another file (or subdirectory). 2. Duplicate all information's about shared-files in both sharing directories.

Two problems: A file may have multiple absolute path-names. Deletion may leave dangling-pointers to the non-existent file. Solution to deletion problem: Use back pointers: Preserve the file until all references to it are deleted.

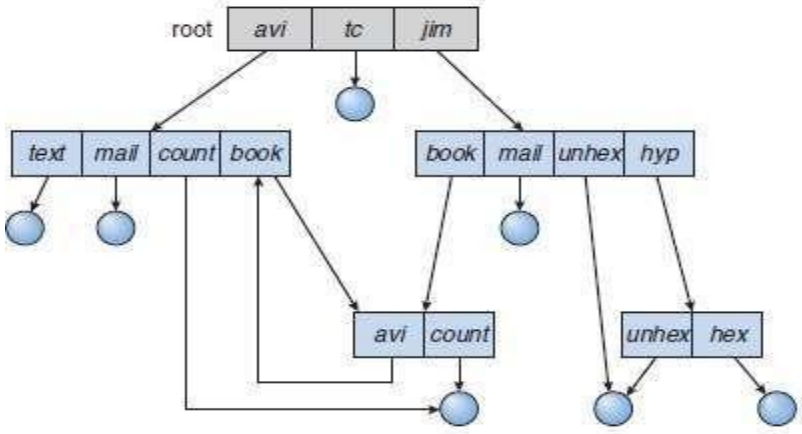
With symbolic links, remove only the link, not the file. If the file itself is deleted, the link can be removed.

5. General Graph Directory

If there are cycles in Acyclic directory then we want to avoid searching components twice as shown in below figure. The solution is to limit the number of directories accessed in a search.

Also the problem with cycles, the reference-count may be non-zero even when it is no longer possible to refer to a directory (or file). (A value of 0 in the reference count means that there are no more references to the file or directory, and the file can be deleted). The solution to this problem is Garbage-collection scheme can be used to determine when the last reference has been deleted. Garbage collection involves: First pass: traverses the entire file-system and marks everything that can be accessed.

A second pass collects everything that is not marked onto a list of free-space.

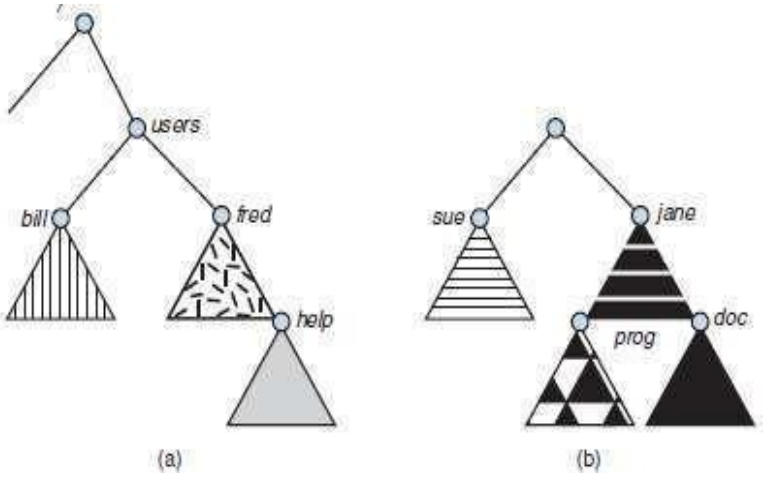


General graph directory

FILE MOUNTING

Briefly explain file mounting concept in operating system

A file-system must be mounted before it can be available to processes on the system. This is as shown in below figure. Mount-point is the location in the file-structure where the file-system is to be attached.



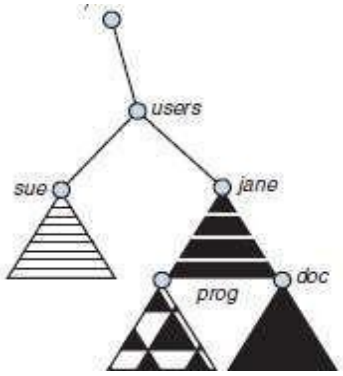
File system. (a) Existing system. (b) Unmounted volume

The file mounting procedure is as follows:

1) OS is given name of the device and mount-point as shown in below figure.

2) OS verifies that the device contains a valid file-system.

3) OS notes in its directory-structure that a file-system is mounted at specified mount-point.



Mount point

FILE SHARING

Sharing of files on multi-user systems is desirable. Sharing may be done through a protection scheme. On distributed systems, files may be shared across a network. Network File-system (NFS) is a common distributed file-sharing method.

Multiple Users:

File-sharing can be done in 2 ways: The system can allow a user to access the files of other users by default or The system may require that a user specifically grant access.

To implement file-sharing, the system must maintain more file- & directory-attributes than on a single-user system. Most systems use concepts of file owner and group.

Owner: The user who may change attributes & grant access and has the most control over the file (or directory). Most systems implement owner attributes by managing a list of user-names and user IDs

Group: The group attribute defines a subset of users who can share access to the file. Group functionality can be implemented as a system-wide list of group-names and group IDs. Exactly which operations can be executed by group-members and other users is definable by the file's owner. The owner and group IDs of a file are stored with the other file-attributes and can be used to allow/deny requested operations.

Remote File Systems:

It allows a computer to mount 1 or more file-systems from 1 or more remote-machines.

Three methods:

1. Manually via: programs like FTP.
2. Automatically DFS (Distributed file-system): remote directories are visible from a local machine.

3. Semi-automatically via www (World Wide Web): A browser is needed to gain access to the remote files, and separate operations (a wrapper for ftp) are used to transfer files.FTP is used for both anonymous and authenticated access. Anonymous access allows a user to transfer files without having an account on the remote system.

Client Server Model

It allows clients to mount remote file-systems from servers. The machine containing the files is called the server. The machine seeking access to the files is called the client. A server can serve multiple clients, and A client can use multiple servers. The server specifies which resources (files) are available to which clients. A client can be specified by a network-name such as an IP address. Disadvantage: 1) Client identification is more difficult. In UNIX and its NFS (network file-system), authentication takes place via the client networking information by default. Once the remote file-system is mounted, file-operation requests are sent to the server via the DFS protocol.

Distributed Information Systems

It provides unified access to the information needed for remote computing. The DNS (domain name system) provides hostname-to-network address translations. Other distributed information systems provide username/password space for a distributed facility.

Failure Modes

Local file-systems can fail for a variety of reasons such as failure of disk (containing the file-system) corruption of directory-structure & cable failure.

Remote file-systems have more failure modes because of the complexity of network-systems.

The network can be interrupted between 2 hosts. Such interruptions can result from hardware failure poor hardware configuration or networking implementation issues.

DFS protocols allow delaying of file-system operations to remote-hosts, with the hope that the remote-host will become available again. To implement failure-recovery, some kind of state information may be maintained on both the client and the server.

Consistency Semantics

These represent an important criterion of evaluating file-systems that supports file-sharing. These specify how multiple users of a system are to access a shared-file simultaneously. In particular, they specify when modifications of data by one user will be observed by other users. These semantics are typically implemented as code with the file-system.

These are directly related to the process-synchronization algorithms. A successful implementation of complex sharing semantics can be found in the Andrew file-system **(AFS).**

FILE PROTECTION

When information is stored in a computer system, we want to keep it safe from physical damage (reliability) and improper access (protection). Reliability is generally provided by duplicate copies of files.

Explain the various file protection methods

1. For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer. File owner/creator should be able to control: what can be done by whom.
2. Protection mechanisms provide controlled access by limiting the types of file access that can be made Read, write, execute, append, delete, list (name and attribute) Other operations like renaming, copying and editing the file.
3. Associate password with each file. This scheme may be effective in limiting access to file. Some systems associate password with subdirectory.

IMPLEMENTING FILE SYSTEM

Explain file implementation methods in operating system

On-disk & in-memory structures are used to implement a file-system in operating system.

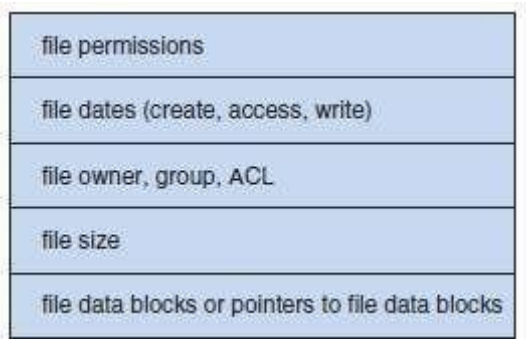
On-disk structures shown in below figure include:

i. Boot Control Block: Contains information needed to boot an OS from the partition. It is typically the first block of a volume. In UFS, it is called the boot block. In NTFS, it is the partition boot sector.

ii. Partition Control Block: Contains partition-details like number of blocks, size of blocks and free-block count. In UFS, this is called a superblock. In NTFS, it is stored in the master file table.

iii. Directory-structure: Used to organize the files. In UFS, this includes file-names and associated inode-numbers. In NTFS, it is stored in the master file table.

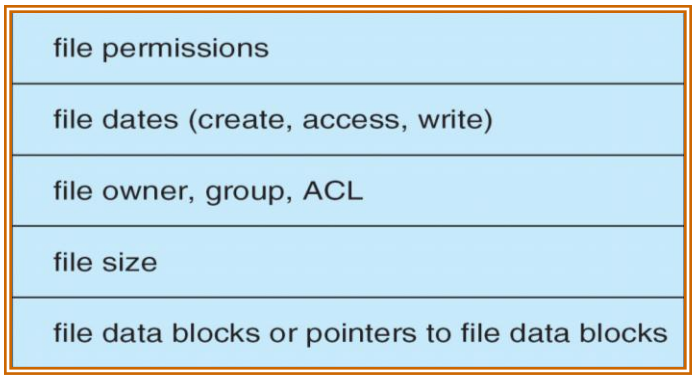
iv. FCB (file control block): Contains file-details including file-permissions, ownership, file-size and location of data-blocks.



A typical file-control block

Explain the structure File control block (FCB)

Directory structure is used to organize the files. Per-file FCB contains many details about the file.



- To create a new file, application program calls the logical file system
- It allocates FCB
- The system then reads the appropriate directory structure to memory and updates it with new file name and FCB, and writes it back to the disk
- When the process closes the file, per-process table entry is removed, and system-wide entry's open count is decremented.

In-Memory File System Structures may include:

i. In-memory Mount Table: Contains information about each mounted partition.

ii. An in-memory Directory-structure: Holds directory information of recently accessed directories.

iii. System-wide Open-file Table: Contains a copy of the FCB of each open file

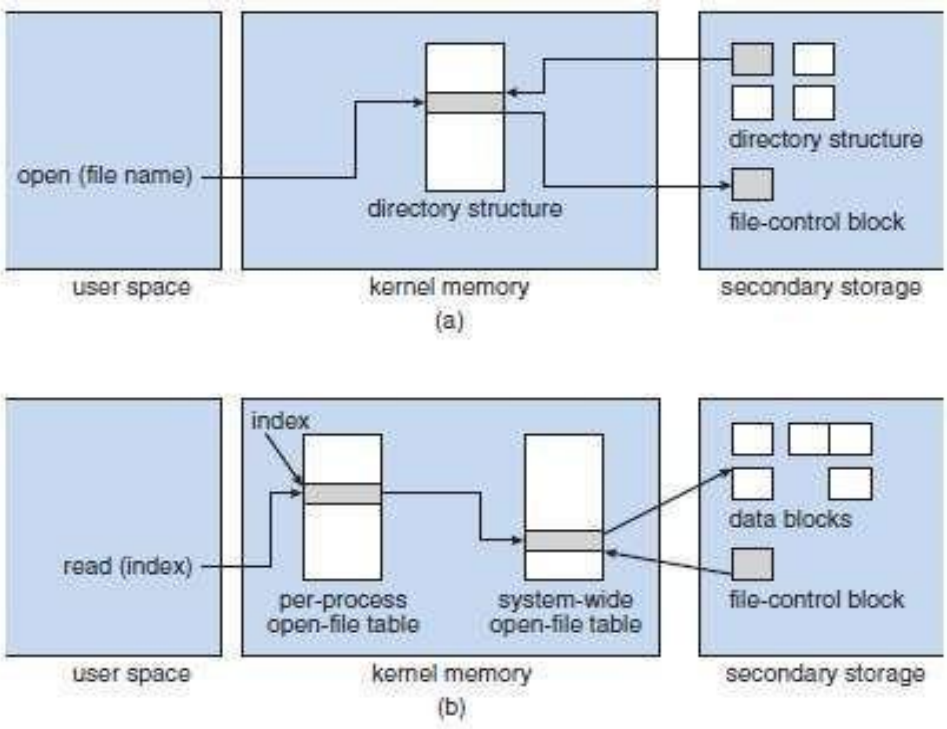
iv. Per-process Open-file Table: Contains a pointer to the appropriate entry in the system-wide open-file table.

Buffers hold file-system blocks when they are being read from disk or written to disk. To create a new file, a program calls the LFS (logical file-system). The „LFS' knows the format of the directory-structures. To create a new file, the LFS

1) Allocates a new FCB.
2) Reads the appropriate directory into memory.
3) Updates LFS with the new file-name and FCB.
4) Writes LFS back to the disk

After a file has been created, it can be used for I/O.

1) First the file must be opened.
2) FCB is copied to a system-wide open-file table in memory.
3) An entry is made in the per-process open-file table, with a pointer to the entry in the system-wide open-file table.
4) The open call returns a pointer to the appropriate entry in the per-process file-system table.
5) All file operations are then performed via this pointer.
6) When a process closes the file: The per-process table entry is removed.The system-wide entry‟s open count is decremented.



In-memory file-system structures. (a) File open. (b) File read

PARTITIONS & MOUNTING

Disk layouts can be: A disk can be divided into multiple partitions or A partition can span multiple disks (RAID). Each partition can either be: Raw i.e. containing no file-system or

Cooked i.e: containing a file-system.

Boot information is a sequential series of blocks, loaded as an image into memory. Execution of the image starts at a predefined location, such as the first byte. The boot information has its own format, because at boot time the system does not have device-drivers loaded and the system cannot interpret the file-system format.

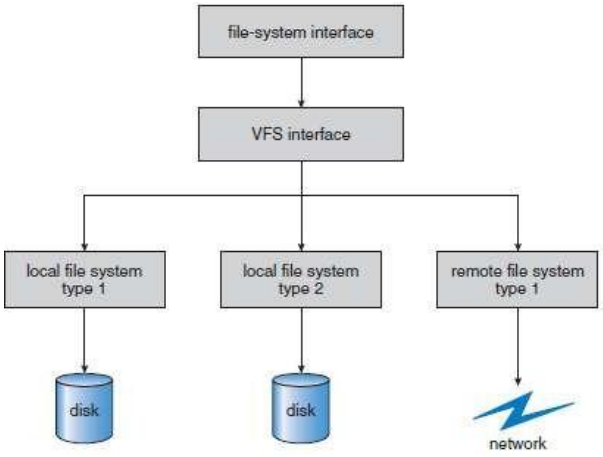
Steps for mounting: The root partition containing the kernel is mounted at boot time. Then, the OS verifies that the device contains a valid file-system. Finally, the OS notes in its in-memory mount table structure that A file-system is mounted and Type of the file-system.

VIRTUAL FILE SYSTEMS

The OS allows multiple types of file-systems to be integrated into a directory-structure.

Three layersare:

1) File-system Interface: This is based on the open(), read(), writeO and closeO calls on file descriptors.
2) File-system (VFS) Interface: This serves 2 functions: Separates file-system basic operations from their implementation by defining a clean VFS interface.The VFS is based on a file-representation structure called a vnode. vnode contains a numerical designator for a network-wide unique file.
3) Local File-system:Local files are distinguished according to their file-system types.



Schematic view of a virtual file system

DIRECTORY IMPLEMENTATION

1. Linear-list **2.** Hash-table

Linear List

A linear-list of file-names has pointers to the data-blocks.

To create a new file: First search the directory to be sure that no existing file has the same name. Then, add a new entry at the end of the directory.

To delete a file: Search the directory for the named-file and Then release the space allocated to the file.

To reuse the directory-entry, there are 3 solutions: Mark the entry as unused (by assigning it a special name). Attach the entry to a list of free directory entries. Copy the last entry in the directory into the freed location & to decrease length of directory.

Problem: Finding a file requires a linear-search which is slow to execute.

Solutions: A cache can be used to store the most recently used directory information. A sorted list allows a binary search and decreases search time.

Advantage: 1) Simple to program.

Disadvantage: 1) Time-consuming to execute.

Hash Table

A linear-list stores the directory-entries. In addition, a hash data-structure is also used.

The hash-table takes a value computed from the file name and returns a pointer to the file name in the linear-list.

Advantages: Decrease the directory search-time. Insertion & deletion are easy.

Disadvantages: Some provision must be made for collisions i.e. a situation in which 2 file-names hash to the same location. Fixed size of hash-table and the dependence of the hash function on

FILE ALLOCATION METHODS

The direct-access nature of disks allows us flexibility in the implementation of files. In almost every case, many files are stored on the same disk.

Main problem: How to allocate space to the files so that disk-space is utilized effectively and files can be accessed quickly. Three methods of allocating disk-space:

1. Contiguous
2. Linked
3. Indexed

********Explain the various storage mechanisms available to store files, with neat diagram.**

OR

*********Explain different file allocation methods**

There are three file allocation methods for allocating disk-space:

1. Contiguous Allocation

Each file occupies a set of contiguous-blocks on the disk as shown in below figure. Disk addresses define a linear ordering on the disk. The number of disk seeks required for accessing contiguously allocated files is minimal. Both sequential and direct access can be supported.

Problems: Finding space for a new file. External fragmentation can occur. Determining how much space is needed for a file. If you allocate too little space, it can't be extended.

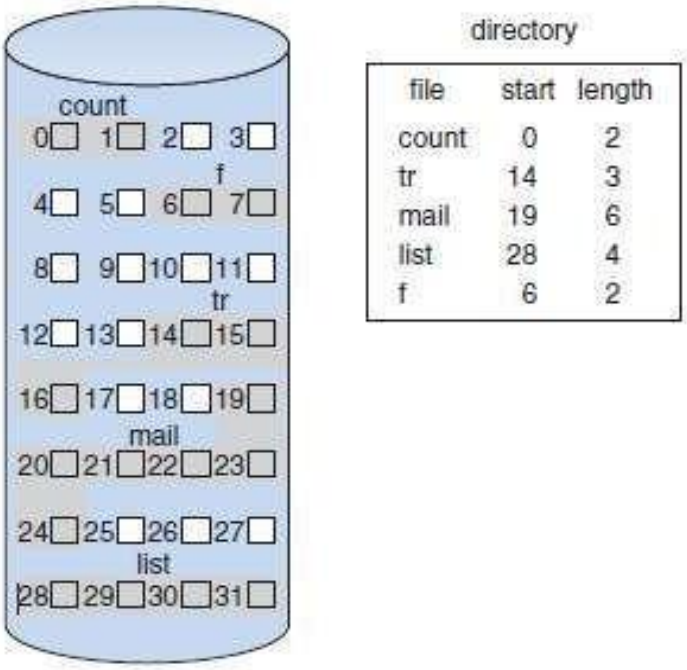
Two solutions:

The user-program can be terminated with an appropriate error-message. The user must then allocate more space and run the program again.

Find a larger hole, copy the contents of the file to the new space and release the previous space.

To minimize these drawbacks: A contiguous chunk of space can be allocated initially and

Then when that amount is not large enough, another chunk of contiguous space is added.



Contiguous allocation of disk-space

2. Linked Allocation

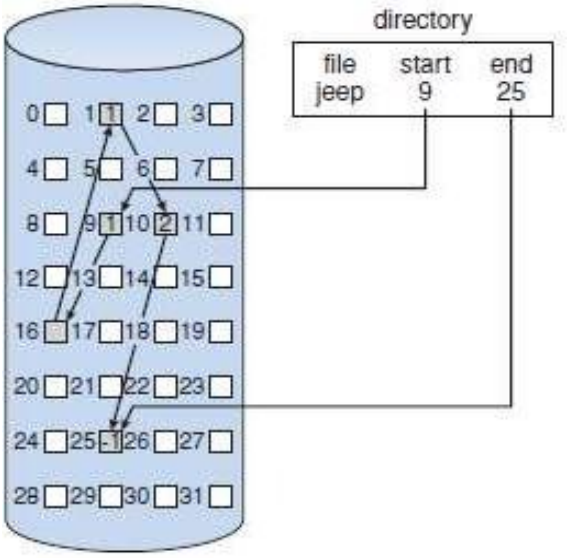
Each file is a linked-list of disk-blocks. The disk-blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file as shown in below figure. To create a new file, just create a new entry in the directory (each directory-entry has a pointer to the disk-block of the file). A write to the file causes a free block to be found. This new block is then written to and linked to the end of file (eof). A read to the file causes moving the pointers from block to block.

Advantages: No external fragmentation, and any free block on the free-space list can be used to satisfy a request. The size of the file doesn't need to be declared on creation. Not necessary to compact disk-space.

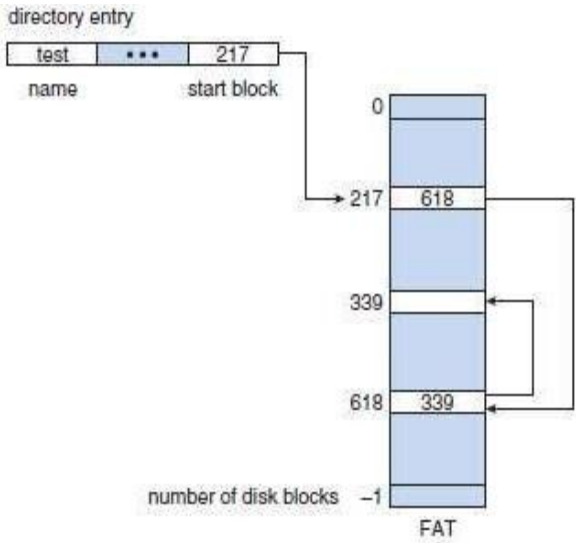
Disadvantages: Can be used effectively only for sequential-access files. Space required for the pointers.

Solution: Collect blocks into multiples (called „clusters") & allocate clusters rather than blocks.

Reliability: Problem occurs if a pointer is lost (or damaged). Partial solutions: i) Use doubly linked-lists. ii) Store file name and relative block-number in each block.



Linked allocation of disk-space



File-allocation table

FAT is a variation on linked allocation (FAT=File Allocation Table). A section of disk at the beginning of each partition is set aside to contain the table as shown in above figure. The table has one entry for each disk-block and is indexed by block-number.

The directory-entry contains the block-number of the first block in the file. The table entry indexed by that block-number then contains the block-number of the next block in the file.

This chain continues until the last block, which has a special end of file has the table entry.

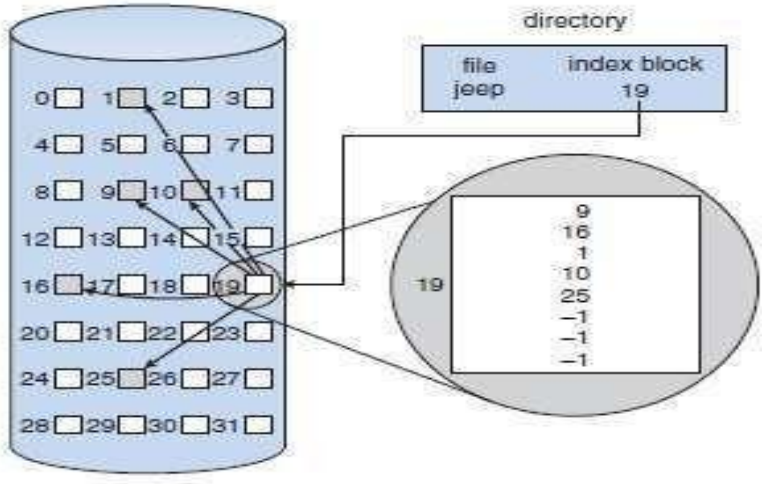
Advantages:

i. Cache can be used to reduce the no. of disk head seeks.

ii. Improved access time, since the disk head can find the location of any block by reading the information in the FAT.

3. Indexed Allocation

It solves the problems of linked allocation (without a FAT) by bringing all the pointers together into an index block. Each file has its own index block, which is an array of disk-block addresses. The i^{th} entry in the index block points to the i^{th} file block. The directory contains the address of the index block.



Indexed allocation of disk space

When the file is created, all pointers in the index-block are set to nil. When writing the ith block, a block is obtained from the free-space manager, and its address put in the ith index-block entry,

Problem: If the index block is too small, it will not be able to hold enough pointers for a large file,

Solution: Three mechanisms to deal with this problem:

Linked Scheme: To allow for large files, link several index blocks,

Multilevel Index: A first-level index block points to second-level ones, which in turn point to the file blocks,

Combined Scheme: The first few pointers point to direct blocks (i.e. they contain addresses of blocks that contain data of the file). The next few pointers point to indirect blocks.

Advantage:

1) Supports direct access, without external fragmentation,

Disadvantages: Suffer from wasted space, The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation. Suffer from performance problems.

Performance of file allocation methods

Contiguous Allocation

It requires only one access to get a disk-block. We can calculate immediately the disk address of the next block and read it directly. It is good for direct access

Linked Allocation

It is good for sequential access. But it is not to be used for an application requiring direct access

Indexed Allocation

If the index block is already in memory, then the access can be made directly but keeping the index block in memory requires considerable space.

FREE SPACE MANAGEMENT

************What do you mean by free space list? With suitable example, explain any two methods of implementation of free space list. OR**

Explain how free space is managed?

A **free-space list** keeps track of all free disk-space. 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. Allocate that space to the new file. This space is then removed from the free-space list. To delete a file, its disk-space is added to the free-space list.

The various methods used in implementation of free space list:

1. Bit-vector
2. Linked list
3. Grouping
4. Counting

1. Bit Vector

The free-space list is implemented as a bit map /bit vector. Each block is represented by a 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 and 7 are free and the rest of the blocks are allocated. The free-space bit map will be 00111101

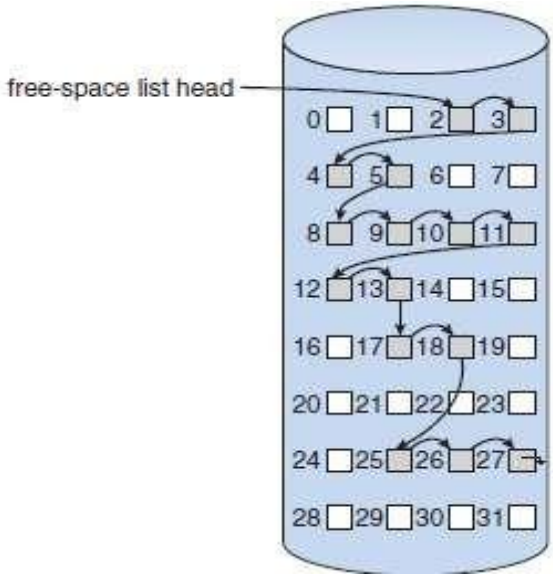
Advantage: 1) Relative simplicity & efficiency in finding the first free block, or „n" consecutive free blocks.

Disadvantages: Inefficient unless the entire vector is kept in main memory. The entire vector is written to disc occasionally for recovery**.**

2. Linked List

The basic idea is Link together all the free disk-blocks as shown in below figure. Keep a pointer to the first free block in a special location on the disk. Cache the block in memory. The first block contains a pointer to the next free one, etc.

Disadvantage: 1) Not efficient, because to traverse the list, each block is read. Usually the OS simply needs a free block, and uses the first one.



Linked free-space list on disk

3. Grouping

The addresses of n free blocks are stored in the 1st free block. The first n-1 of these blocks are actually free. The last block contains addresses of another n free blocks, etc.

Advantage: Addresses of a large no of free blocks can be found quickly.

4. Counting

Takes advantage of the fact that, generally, several contiguous blocks may be allocated /freed simultaneously. To Keep the address of the first free block and the number n of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count.

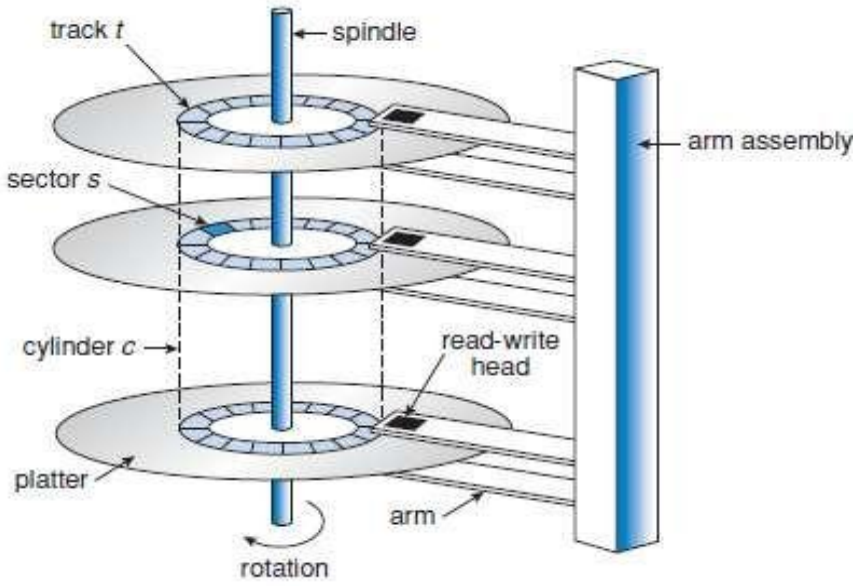
MASS-STORAGE STRUCTURE

Hard-Disks

With neat diagram briefly explain the structure of moving disk.

Hard-disks provide the bulk of secondary-storage for modern computer-systems as shown in figure.

Each disk-platter has a flat circular-shape, like a CD. The 2 surfaces of a platter are covered with a magnetic material. Information is stored on the platters by recording magnetically.



Moving-head disk mechanism

A read/write head flies just above the surface of the platter. The heads are attached to a disk-arm that moves all the heads as a unit. The surface of a platter is logically divided into circular tracks, which are subdivided into sectors. The set of tracks that are at one arm position makes up a cylinder. There may be thousands of concentric-cylinders in a disk-drive, and each track may contain hundreds of sectors.

Disk-speed has 2 parts:

- The transfer-rate is the rate at which data flow between the drive and the computer.
- The positioning-time (or random-access time) consists of 2 parts:
    a) Seek-time refers to the time necessary to move the disk-arm to the desired cylinder.

b) Rotational-latency refers to the time necessary for the desired sector to rotate to the disk-head.

A disk can be removable which allows different disks to be mounted as needed. A disk-drive is attached to a computer by an I/O bus.

Different kinds of buses: Advanced technology attachment (ATA), Serial ATA (SATA), eSATA, Universal serial bus (USB) and Fiber channel (FC).

SOLID-STATE DISKS

An SSD is non-volatile memory that is used like a hard-drive.

For example: DRAM with a battery to maintain its state in a power-failure through flash-memory **technologies.**

Advantages compared to Hard-disks: More reliable: SSDs have no moving parts and are faster because they have no seek-time or latency and Less power consumption.

Disadvantages: More expensive and Less capacity and so shorter life spans, so their uses are somewhat limited.

Applications: One use for SSDs is in storage-arrays, where they hold file-system metadata that require high performance. SSDs are also used in laptops to make them smaller, faster, and more energy-efficient.

MAGNETIC TAPES

Magnetic tape was used as an early secondary-storage medium.

Advantages: It is relatively permanent and can hold large quantities of data.

Disadvantages: Its access time is slow compared with that of main memory and Hard-disk.

In addition, random access to magnetic tape is about a thousand times slower than random access to Hard-disk, so tapes are not very useful for secondary-storage.

Applications: Tapes are used mainly for backup, for storage of infrequently used information.

Tapes are used as a medium for transferring information from one system to another.

DISK STRUCTURE

Modern Hard-disk-drives are addressed as large one-dimensional arrays of logical blocks. The logical block is the smallest unit of transfer.

How one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially?

Sector 0 is the first sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

In practice, it is difficult to perform this mapping, for two reasons.

- Most disks have some defective sectors, but the mapping hides this by substituting spare sectors from elsewhere on the disk.
- The number of sectors per track is not a constant on some drives.

DISK ATTACHMENT

Computers access disk storage in two ways.

- via I/O ports (or host-attached storage); this is common on small systems.
- via a remote host in a distributed file system; this is referred to as network-attached storage.

Host-Attached Storage

Host-attached storage is storage accessed through local I/O ports. These ports use several technologies.

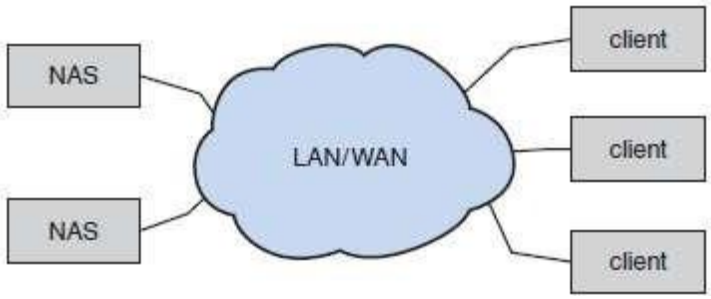
- ✓ The desktop PC uses an I/O bus architecture called IDE or ATA.This architecture supports a maximum of 2 drives per I/O bus.
- ✓ High-end workstations(and servers) use fibre channel (FC), a high-speed serial architecture that can operate over optical fiber.

It has two variants:

- ✓ One is a large switched fabric having a 24-bit address space. This variant is the basis of storage-area networks (SANs).
- ✓ The other FC variant is an arbitrated loop (FC-AL) that can address 126 devices.

A wide variety of storage devices are suitable for use as host-attached storage. For ex: Hard-disk-drives, RAID arrays, and CD, DVD, and tape drives.

Network-Attached Storage



Network-attached storage

A network-attached storage (NAS) device is a special-purpose storage system that is accessed remotely over a data network as shown in figure. Clients access NAS via a remote-procedure-call interface such as NFS for UNIX systems and CIFS for Windows machines.

The remote procedure calls (RPCs) are carried via TCP or UDP over a local area network (LAN). Usually, the same local area network (LAN) carries all data traffic to the clients. The NAS device is usually implemented as a RAID array with software that implements the RPC interface.

Advantage: All computers on a LAN can share a pool of storage with the same ease of naming and access local host-attached storage.

Disadvantages: NAS is less efficient and have lower performance than some direct-attached storage options. The storage I/O operations consume bandwidth on the data network, thereby increasing the latency of network communication.

iSCSI is the latest network-attached storage protocol. iSCSI uses the IP network protocol to carry the SCSI protocol. Thus, networks rather than SCSI cables can be used as the interconnects between hosts and their storage.

Storage-Area Network

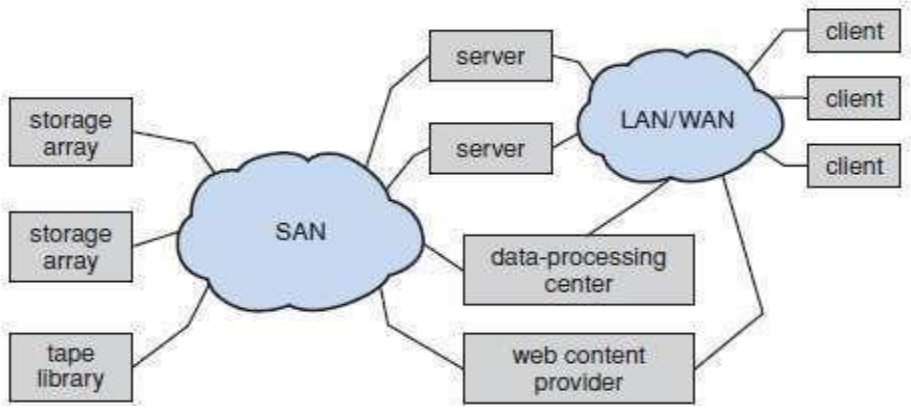
A storage-area network (SAN) is a private network connecting servers and storage units as shown in figure. The power of a SAN lies in its flexibility.

1. Multiple hosts and multiple storage-arrays can attach to the same SAN.
2. Storage can be dynamically allocated to hosts.
3. A SAN switch allows or prohibits access between the hosts and the storage.

4. SANs make it possible for clusters of servers to share the same storage and for storage arrays to include multiple direct host connections.
5. SANs typically have more ports than storage-arrays.

FC is the most common SAN interconnect.

Another SAN interconnect is InfiniBand- a special-purpose bus architecture that provides hardware and software support for high-speed interconnection networks for servers and storage units.



Storage-area network

DISK SCHEDULING

Explain in brief the selection of disk scheduling algorithm.

The selection of disk scheduling algorithm depends on

1. **Access time** = Seek-time + Rotational-latency

Seek-time: The seek-time is the time for the disk-arm to move the heads to the cylinder containing the desired sector.

Rotational-latency: The Rotational-latency is the additional time for the disk to rotate the desired sector to the disk-head.

2. The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

We can improve both the access time and the band width by managing the order in which the disk I/O requests are serviced by selecting appropriate disk scheduling algorithms.

Whenever a process needs I/O to or from the disk, it issues a system call to the operating system. The request specifies several pieces of information:

- ✓ Whether this operation is input or output
- ✓ What the disk address for the transfer is
- ✓ What the memory address for the transfer is
- ✓ What the number of sectors to be transferred is

If the desired disk-drive and controller are available, the request can be serviced immediately.

If the drive or controller is busy, any new requests for service will be placed in the queue of pending requests for that drive.

For a multiprogramming system with many processes, the disk queue may often have several pending requests. Thus, when one request is completed, the operating system chooses which pending request to service next. Any one of several disk-scheduling algorithms can be used.

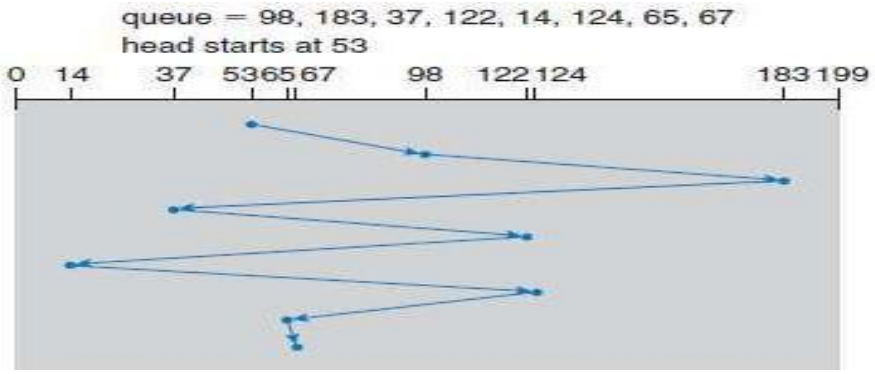
DISK SCHEDULING ALGORIHMS

*********What is disk scheduling? Explain different disk scheduling algorithms. (Any two)**

In a multiprogramming system with many processes, the disk queue may often have several pending disk requests. Thus, when one request is completed, the operating system chooses which pending request to service next. This mechanism is called as disk scheduling.

The various disk scheduling methods are: FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK

FCFS DISK SCHEDULING ALGORITHM: stands for First Come First Serve.The requests are serviced in the same order, as they are received. **For example:**



FCFS Disk Scheduling

Starting with cylinder 53, the disk-head will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67 as shown in above figure.

Head movement from 53 to 98 = 45

Head movement from 98 to 183 = 85

Head movement from 183 to 37 = 146

Head movement from 37 to 122 =85

Head movement from 122 to 14 =108

Head movement from 14 to 124 =110

Head movement from 124 to 65 =59

Head movement from 65 to 67 = 2 ***Total head movement = 640***

Advantage: This algorithm is simple & fair and no starvation of process requests.

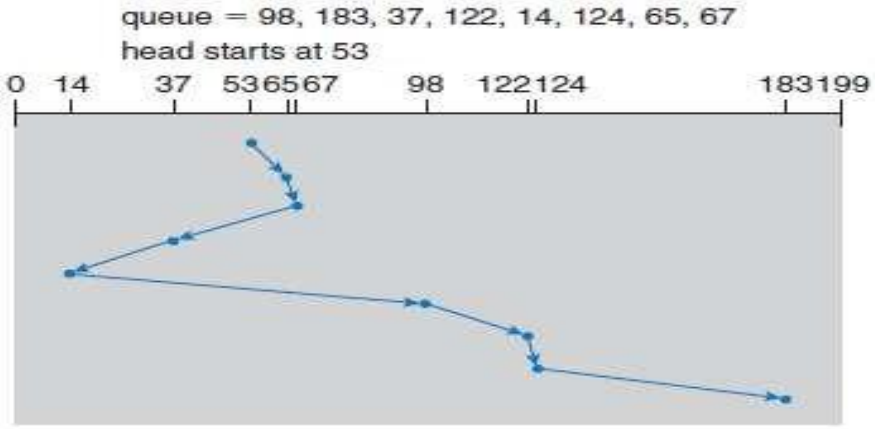
Disadvantage: Generally, this algorithm does not provide the fastest service.

SSTF SCHEDULING

SSTF stands for Shortest Seek-time First. This selects the request with minimum seek-time from the current head-position. Since seek-time increases with the number of cylinders traversed by head, SSTF chooses the pending request closest to the current head-position.

Problem: Seek-time increases with the number of cylinders traversed by head.

Solution: To overcome this problem, SSTF chooses the pending request closest to the current head-position. **For example:**



SSTF disk scheduling

The closest request to the initial head position 53 is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67.From there, the request at cylinder 37 is closer than 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98, 122, 124, and finally 183. It is shown in above Figure.

Head movement from 53 to 65 = 12

Head movement from 65 to 67 = 2

Head movement from 67 to 37 = 30

Head movement from 37 to 14 =23

Head movement from 14 to 98 =84

Head movement from 98 to 122 =24

Head movement from 122 to 124 =2

Head movement from 124 to 183 = 59 ***Total head movement = 236***

Advantage: SSTF is a substantial improvement over FCFS, it is not optimal.

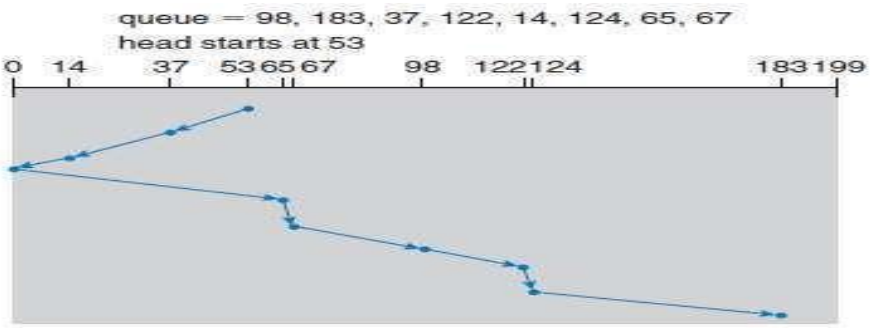
Disadvantage: Essentially, SSTF is a form of SJF and it may cause starvation of some requests.

SCAN SCHEDULING

The SCAN algorithm is sometimes called the elevator algorithm, since the disk-arm behaves just like an elevator in a building.

Here is how it works:

The disk-arm starts at one end of the disk. Then, the disk-arm moves towards the other end, servicing the request as it reaches each cylinder. At the other end, the direction of the head movement is reversed and servicing continues. The head continuously scans back and forth across the disk. ***For example:***



Before applying SCAN algorithm, we need to know the current direction of head movement. Assume that disk-arm is moving toward 0, the head will service 37 and then 14. At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65,67,98, 122, 124, and 183. It is shown in above figure.

Head movement from 53 to 37 = 16

Head movement from 37 to 14 = 23

Head movement from 14 to 0 = 14

Head movement from 0 to 65 =65

Head movement from 65 to 67 =2

Head movement from 67 to 98 =31

Head movement from 98 to 122 =24

Head movement from 122 to 124 = 2

Head movement from 124 to 183 = 59

Total head movement = 236

Disadvantage: If a request arrives just in from of head, it will be serviced immediately.

On the other hand, if a request arrives just behind the head, it will have to wait until the arms reach other end and reverses direction.

C-SCAN SCHEDULING

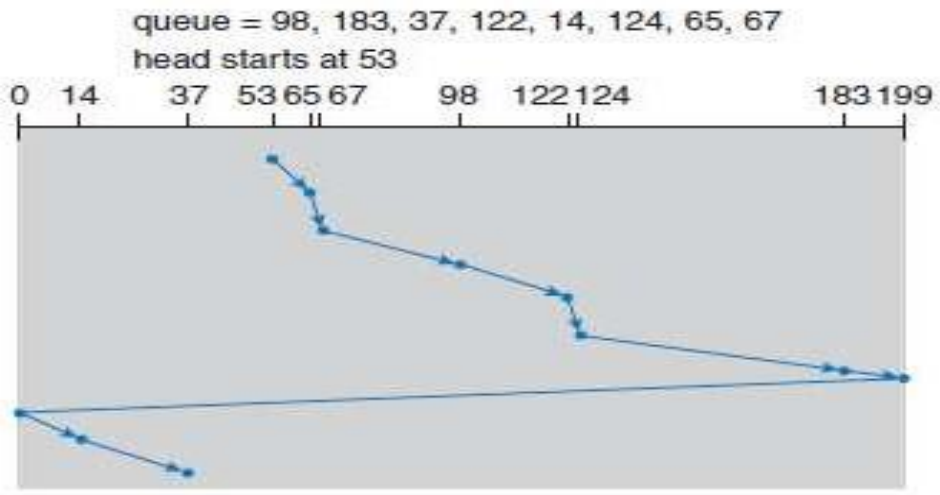
Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip as shown in below figure.

The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

Before applying C - SCAN algorithm, we need to know the current direction of head movement.

Assume that disk-arm is moving toward cylinder number 199, the head will service 65, 67, 98, 122, 124, 183. Then it will move to 199 and the arm will reverse and move towards 0.

While moving towards 0, it will not serve. But, after reaching 0, it will reverse again and then serve 14 and 37. It is as shown in below figure.



C-SCAN disk scheduling

Head movement from 53 to 65 = 12

Head movement from 65 to 67 = 2

Head movement from 67 to 98 = 31

Head movement from 98 to 122 =24

Head movement from 122 to 124 =2

Head movement from 124 to 183 =59

Head movement from 183 to 199 =16

Head movement from 199 to 0 = 199

Head movement from 0 to 14 = 14

Head movement from 14 to 37 = 23

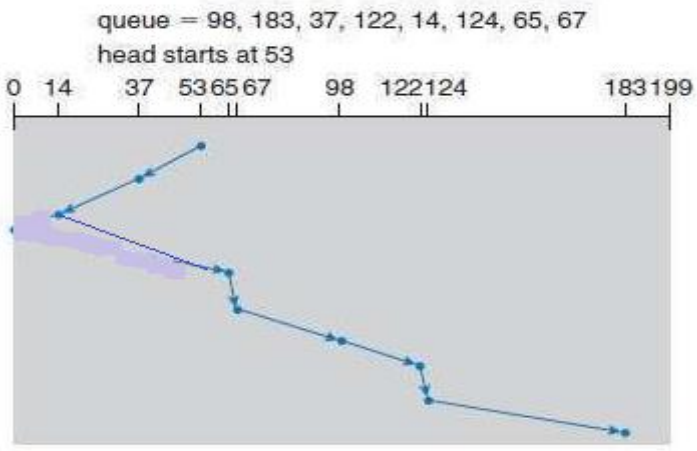
Total head movement = 382

LOOK SCHEDULING

SCAN algorithm, move the disk-arm across the full width of the disk. In practice, the SCAN algorithm is not implemented in this way.

The arm goes only as far as the final request in each direction. Then, the arm reverses, without going all the way to the end of the disk. ***This version of SCAN is called Look scheduling*** because they look for a request before continuing to move in a given direction.

Example:



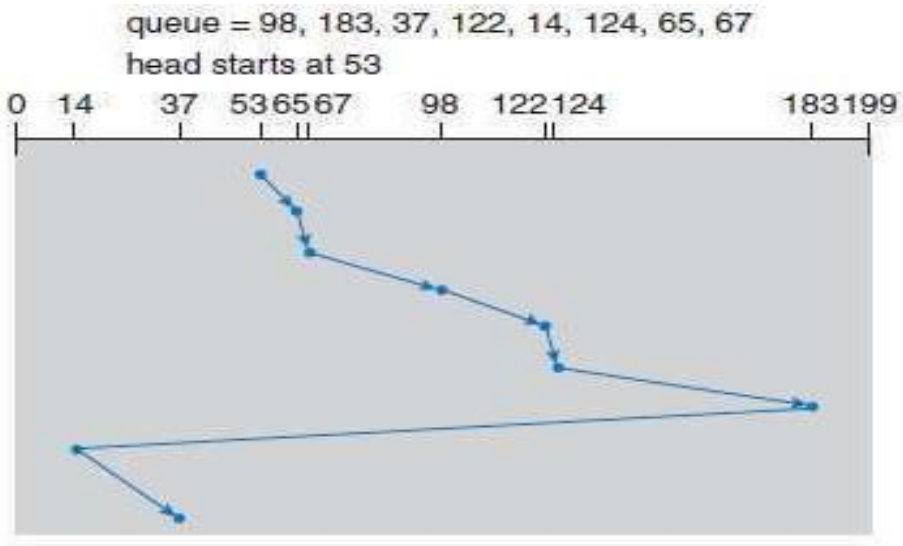
Look scheduling

Total head movement **= (53 – 14) + (183 – 14) = 39 + 169 = 208**

C-LOOK SCHEDULING

Circular LOOK (C-LOOK) scheduling is a variant of LOOK designed to provide a more uniform wait time. Like LOOK, C-LOOK the head goes only as far as the final request in each direction. Then, the arm reverses, without going all the way to the end of the disk. Now it moves the head from one end of the disk to the other, without servicing any requests on the return trip. At the other end, the direction of the head movement is reversed and servicing continues.

Assume that disk-arm is moving toward 199, the head will service 65, 67, 98, 122, 124, 183. Then the arm will reverse and move towards 14. Then it will serve 37. It is as shown in below Figure



C-LOOK disk scheduling

Head movement from 53 to 65 = 12

Head movement from 65 to 67 = 2

Head movement from 67 to 98 = 31

Head movement from 98 to 122 =24

Head movement from 122 to 124 =2

Head movement from 124 to 183 =59

Head movement from 183 to 14 = 169

Head movement from 14 to 37 = 23

Total head movement = **322**

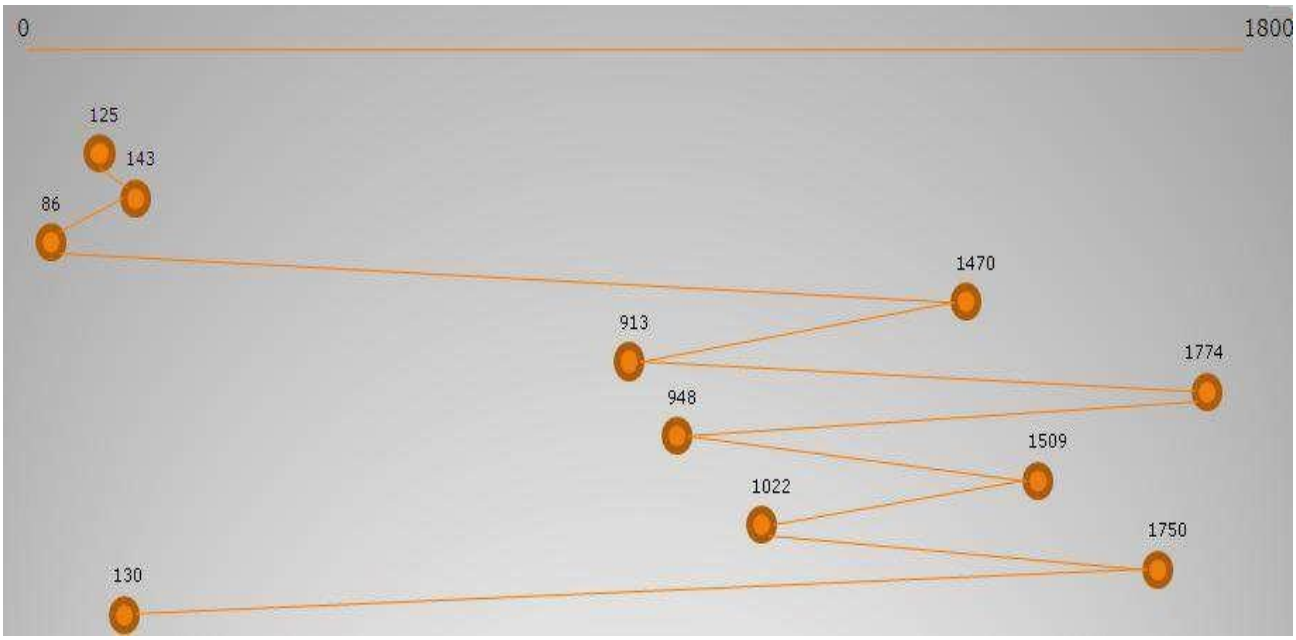
EXERCISE PROBLEMS

Suppose that the disk-drive has 5000 cylinders numbered from 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests in FIFO order is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current (location) head position, what is the total distance (in cylinders) that the disk-arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms?

i. **FCFS**
ii. **SSTF**
iii. **SCAN**
iv. **LOOK**
v. **C-SCAN**
vi. **C-LOOK**

Solution:

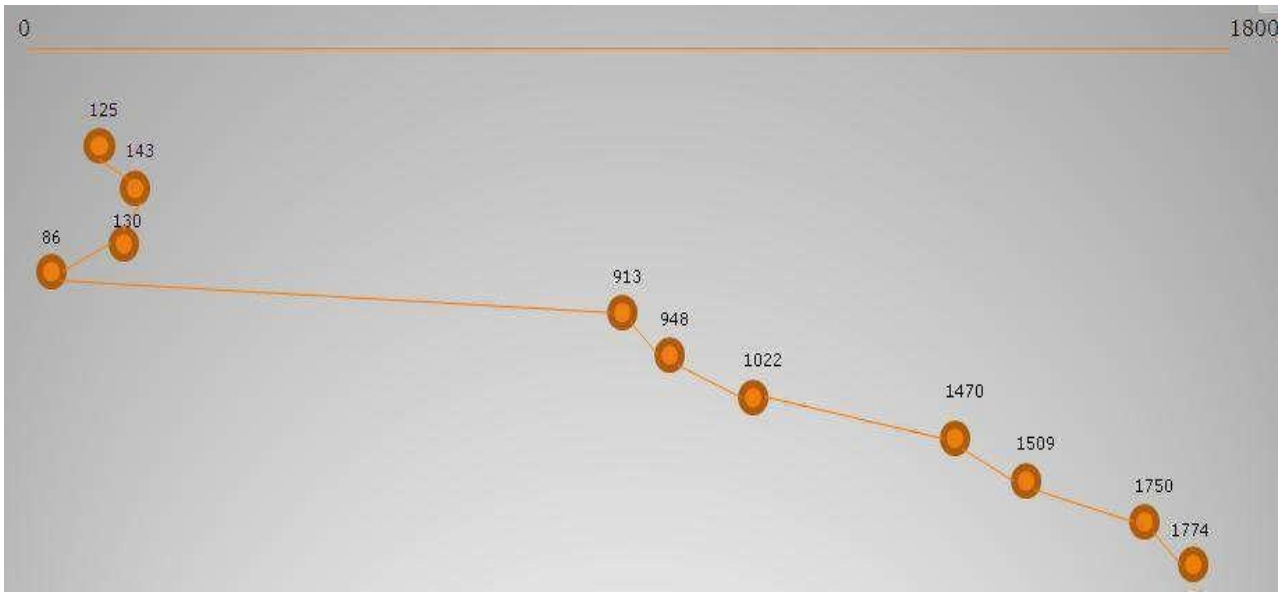
i. **FCFS**



From cylinder	To cylinder	Seek Time
143	86	57
86	1470	1384
1470	913	557
913	1774	861
1774	948	826
948	1509	561
1509	1022	487
1022	1750	728
1750	130	1620
Total Seek Time		7081

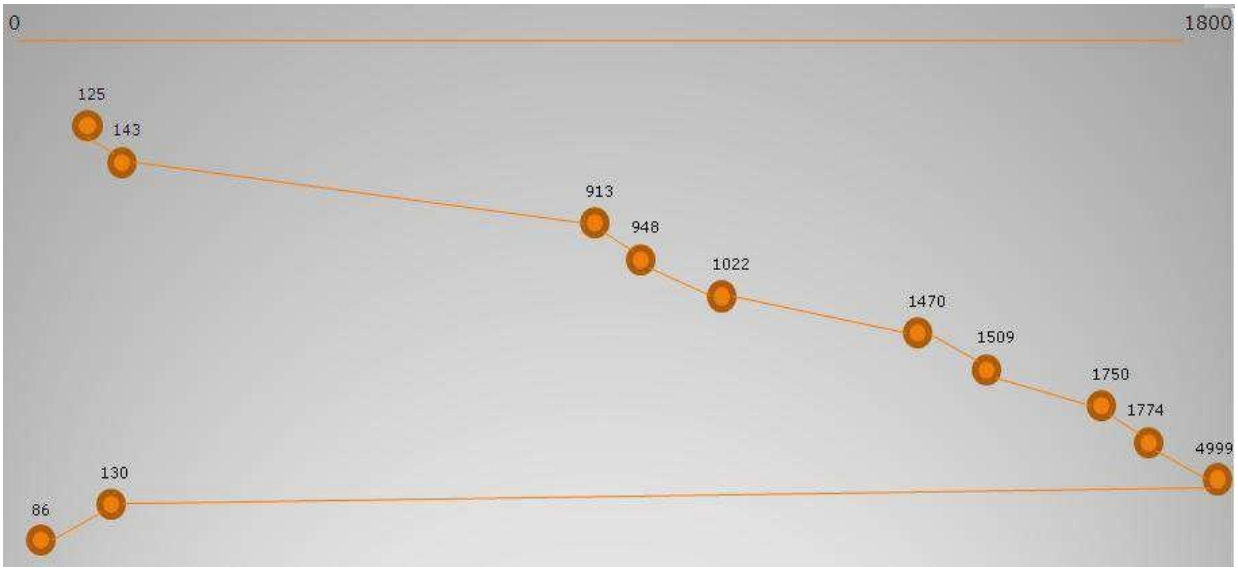
For FCFS schedule, the total seek distance is 7081.

ii. SSTF



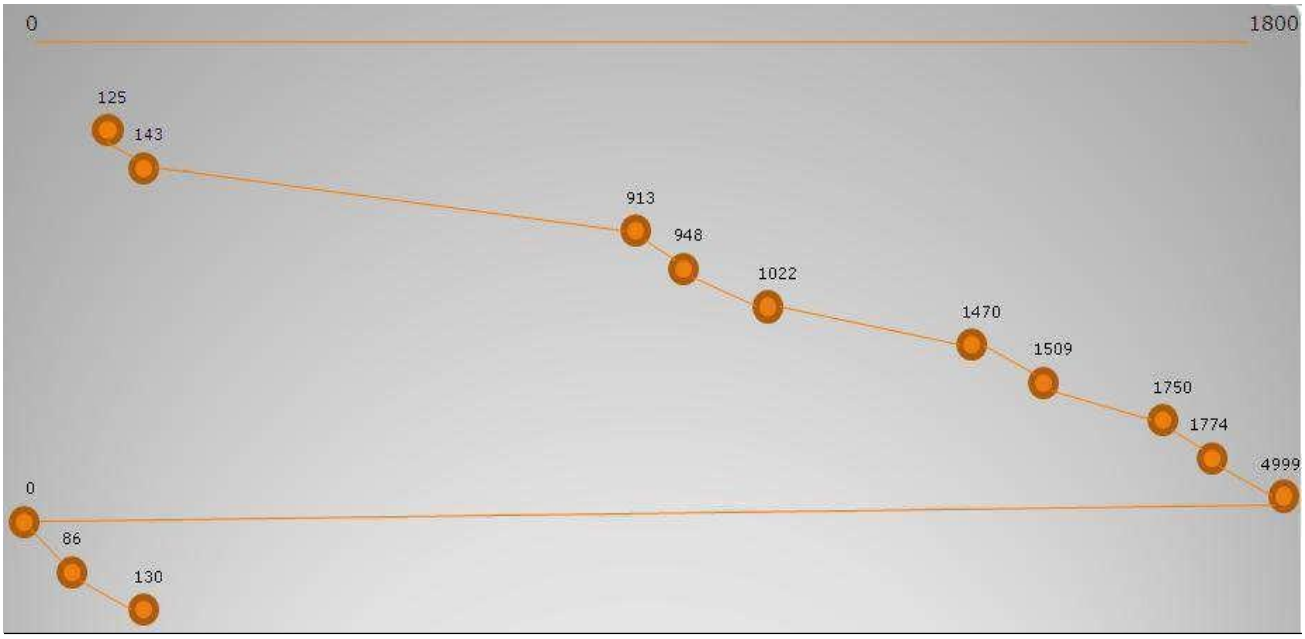
For SSTF schedule, the total seek distance is = (143-130) + (130-86) + (1774-86) = 1745.

iii, SCAN



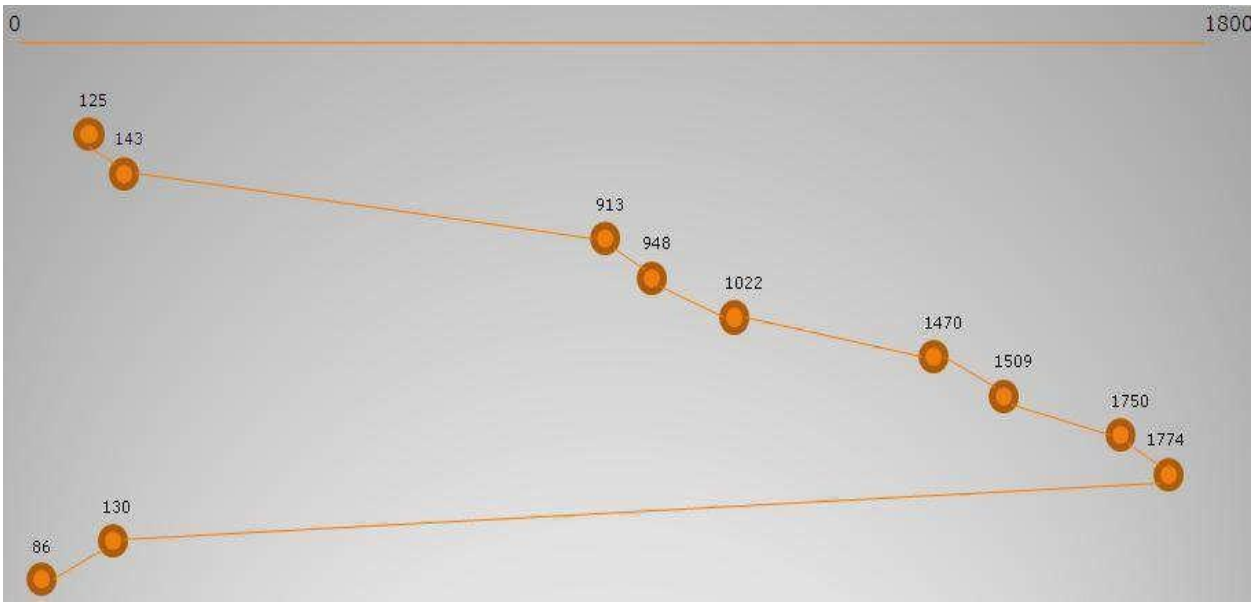
For SCAN schedule, the total seek distance is = (4999- 143) + (4999 -86) = 9769

iv. C-SCAN



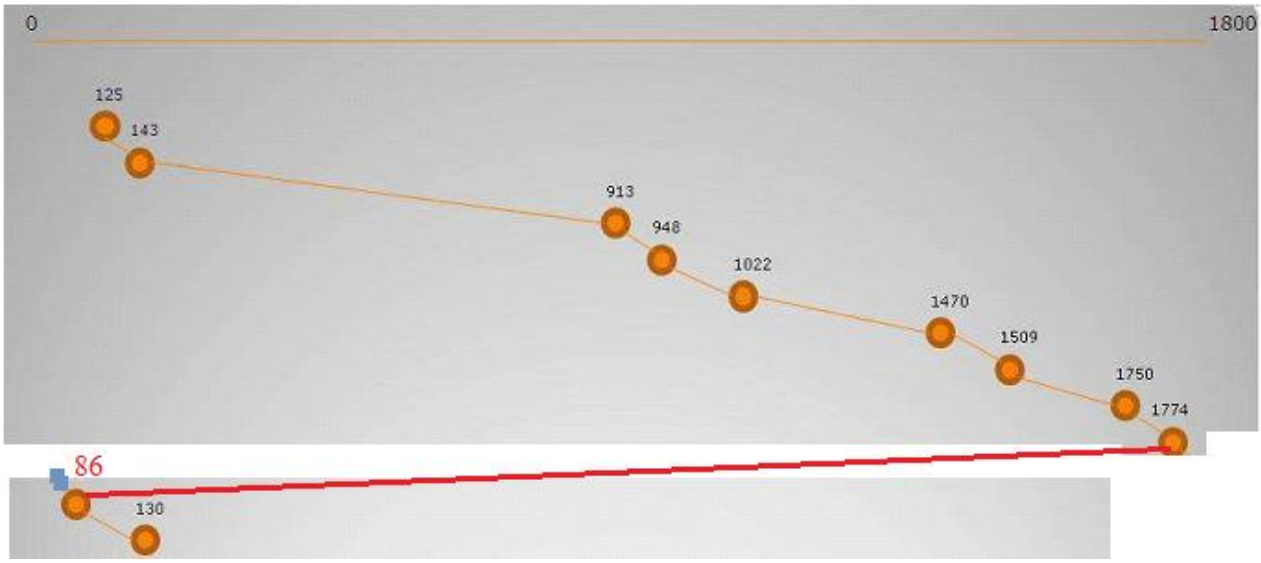
For C-SCAN schedule, the total seek distance is = (4999 – 143) + (4999 – 0)+(130-0) = 9985

v. LOOK



For LOOK schedule, the total seek distance is = (1774 – 143) + (1774 – 86) = **3319.**

vi. C-LOOK



For C- LOOK schedule, the total seek distance is = (1774 – 143) +(1774 -86) + (130-86) = 3363

1) Suppose that a disk has 50 cylinder named 0 to 49. The R/W head is currently serving at cylinder

Suppose that a disk has 50 cylinders named 0 to 49. The R/W head is currently serving at cylinder 15. The queue of pending request are in order: 4, 40, 11, 35, 7, 14 starting from the current head position, what is the total distance traveled (in cylinders) by the disk-arm to satisfy the request using algorithms

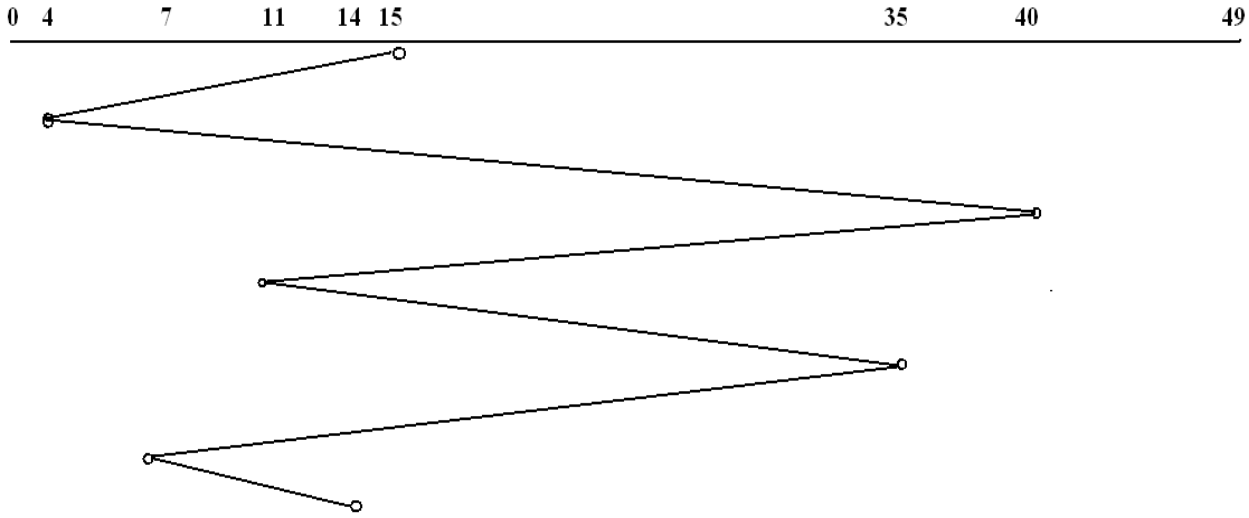
i. **FCFS**
ii. **SSTF and**
iii. **LOOK.**

Illustrate with figure in each case.

FCFS

Queue: 4, 40, 11, 35, 7, 14

Head starts at 15

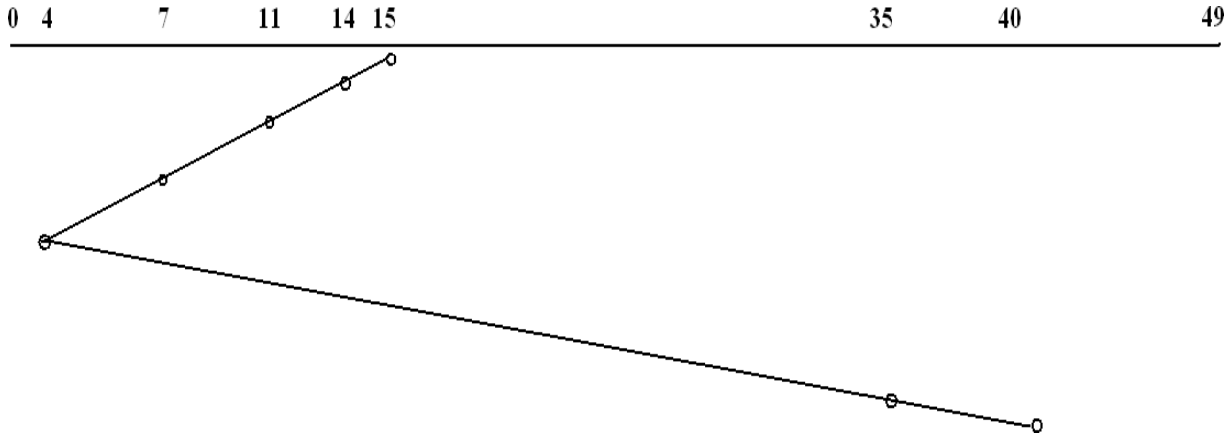


For FCFS schedule, the total seek distance is **135**

SSTF

Queue: 4, 40, 11, 35, 7, 14

Head starts at 15

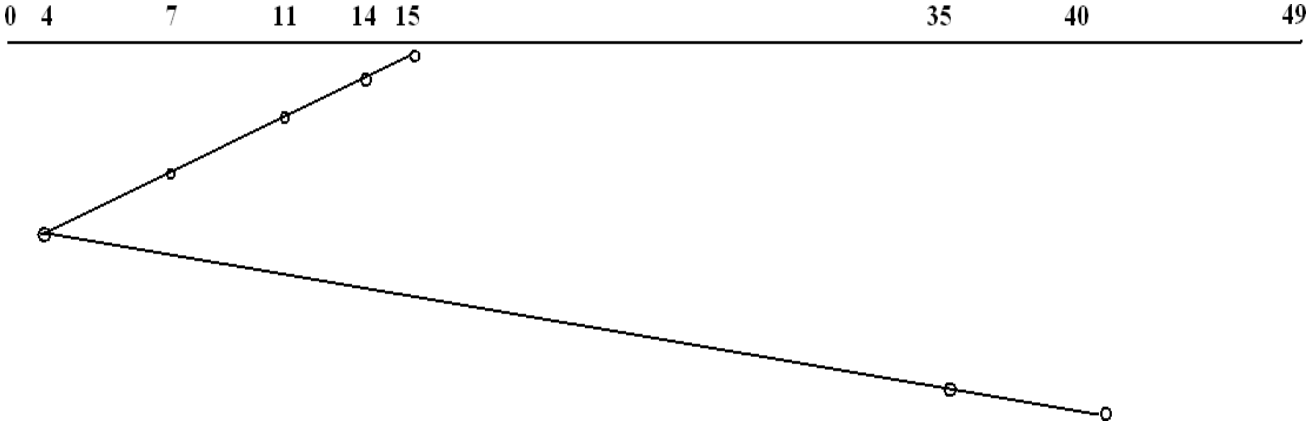


For SSTF schedule, the total seek distance is **47**

LOOK

Queue: 4, 40, 11, 35, 7, 14

Head starts at 15

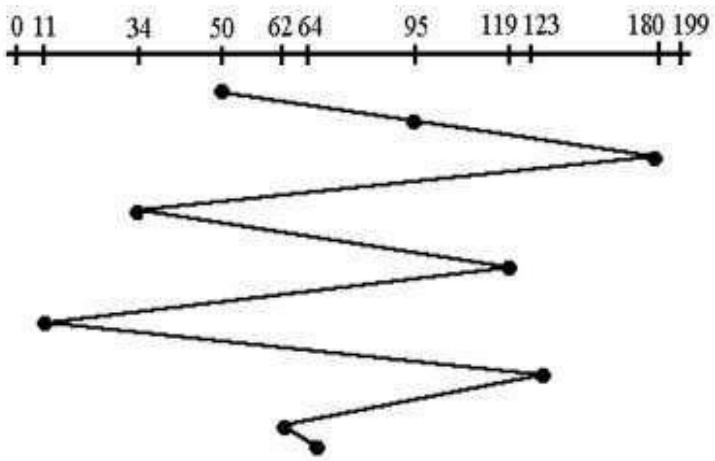


For LOOK schedule, the total seek distance is **47**

Given the following queue 95, 180, 34, 119, 11, 123, 62, 64 with head initially at track 50 and ending at track 199. Calculate the number moves using:

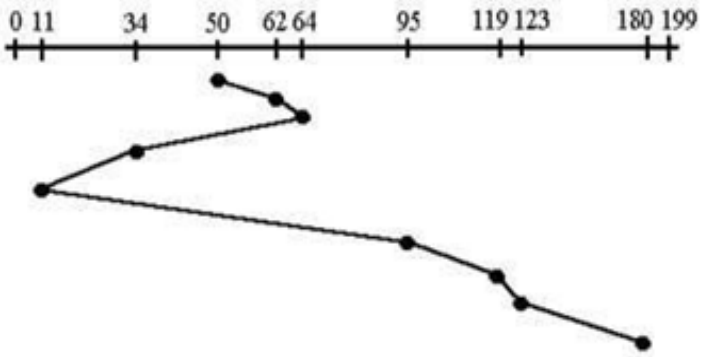
i. **FCFS**
ii. **SSTF**
iii. **Elevator and**
iv. **C-look.**

FCFS



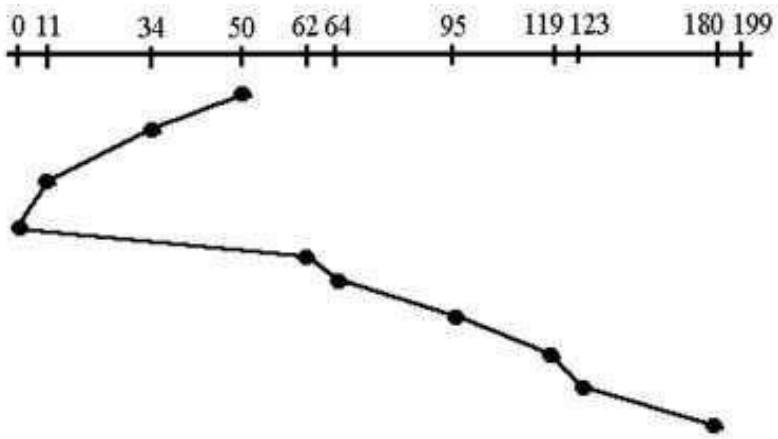
For FCFS schedule, the total seek distance is **640**

SSTF



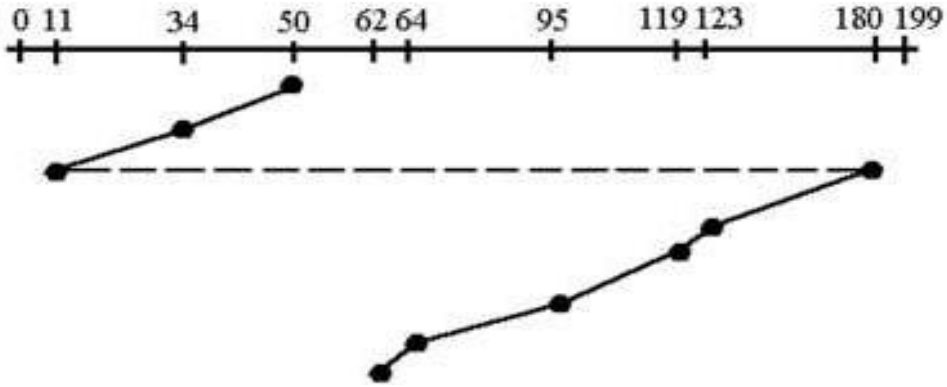
For SSTF schedule, the total seek distance is 236

Elevator (SCAN)



For Elevator (Scan) schedule, the total seek distance is 230

C-LOOK



For C-LOOK schedule, the total seek distance is = (50 – 11) + (180 – 11) + (180 -62) = **326**

DISK MANAGEMENT

The operating system is responsible for several other aspects of disk management. For example: Disk initialization, Booting from disk and Bad-block recovery.

Disk Formatting: Usually, a new Hard-disk is in a blank slate: it is just a platter of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. ***This process is called low-level formatting***, or ***physical formatting.*** Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area (usually 512 bytes in size), and a trailer. The header and trailer contain information used by the disk controller, such as sector number and error-correcting code (ECC). Before a disk can store data, the operating system still needs to record its own data structures on the disk. It does so in two steps.

- ✓ Partition the disk into one or more groups of cylinders.
- ✓ The operating system can treat each partition as a separate disk.

For example: one partition can hold a copy of the operating system's executable code, another partition can hold user files.

Logical formatting or creation of a file system: The operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space and an initial empty directory. To increase efficiency, most file systems group blocks together into larger chunks, frequently called clusters. Disk I/O is done via blocks, File system I/O is done via clusters.

BOOT BLOCK

What are boot blocks? Explain.

For a computer to start running, it must have a bootstrap program to run. The Bootstrap program initializes CPU registers, device controllers and the contents of main memory and then starts the operating system. For most computers, the bootstrap is stored in read-only memory (ROM). To change the bootstrap code, the ROM hardware chips has to be changed. To solve this problem, most systems store a tiny bootstrap loader program in the boot-ROM. This loader program in ROM will bring bootstrap program from disk. The full bootstrap program is stored in the form of ***boot blocks*** at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk.

In the boot-ROM, the code instructs the disk-controller to read the boot blocks into memory and then starts executing that code.

BAD BLOCKS

What are bad blocks? Explain

Because disks have moving parts and small tolerances, they are prone to failure. Sometimes, the disk needs to be replaced. The disk-contents need to be restored from backup media to the new disk. One or more sectors may become defective. From the manufacturer, most disks have ***bad-blocks.***

How to handle bad-blocks?

On simple disks, bad-blocks are handled manually.

One strategy is to scan the disk to find bad-blocks while the disk is being formatted. Any bad-blocks that are discovered are flagged as unusable. Thus, the file system does not allocate them.

If blocks go bad during normal operation, a special program (such as Linux bad-blocks command) must be run manually to search for the bad-blocks and to lock the bad-blocks. Usually, data that resided on the bad-blocks are lost.

Bad blocks are recovered by using:

1. Sector sparing method 2. Sector slipping

The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as ***sector sparing or forwarding***

As an alternative to sector sparing some controllers can be instructed to replace a bad block by ***sector slipping***

Example: A typical bad-sector transaction might be as follows:

The operating system tries to read logical block 87. The controller calculates the ECC and finds that the sector is bad. It reports this finding to the operating system. The next time the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare. After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.

SWAP SPACE MANAGEMENT

Swap-space management is a low-level task of the operating system. Virtual memory uses disk space as an extension of main memory.

What is swap space management in OS? Explain

The main goal of swap space: to provide the best throughput for the virtual memory system.

Here, we discuss about 1) Swap space use 2) Swap space location.

Swap-Space Use: Swap space can be used in 2 ways.

- ✓ Swapping-Systems may use swap space to hold an entire process image, including the code and data segments.
- ✓ Paging-systems may simply store pages that have been pushed out of main memory.

The amount of swap space needed on a system can therefore vary from a few megabytes of disk space to gigabytes, depending on amount of physical memory, amount of virtual memory it is backing, and way in which the virtual memory is used.

Swap-Space Location: A swap space can reside in one of two places:

1, The swap space can be a large file within the file system: Here, normal file-system routines can be used to create it, name it, and allocate its space.

Advantage: This approach easy to implement,

Disadvantage: This approach is inefficient. This is because navigating the directory structure and the disk structures takes time and extra disk accesses. External fragmentation can greatly increase swapping times by forcing multiple seeks during reading or writing of a process image.

2.The swap space can be in a separate raw (disk) partition: No file system or directory structure is placed in the swap space. Rather, a separate swap-space storage manager is used to allocate and de-allocate the blocks from the raw partition. This manager uses algorithms optimized for speed rather than for storage efficiency, because swap space is accessed much more frequently than file system.

PROTECTION

Protection vs. Security Protection

- Protection controls access to the system-resources by Programs, Processes or Users.
- Protection ensures that only processes that have gained proper authorization from the OS can operate on memory-segments, CPU and other resources.
- Protection must provide means for specifying the controls to be imposed, means of enforcing the controls.
- Protection is an internal problem. Security, in contrast, must consider both the computer-system and the environment within which the system is used.

Security

- Security ensures the authentication of system-users to protect integrity of the information stored in the system (both data and code) physical resources of the computer-system.
- The security-system prevents unauthorized access malicious destruction alteration of data or accidental introduction of inconsistency.

Goals of Protection

Explain the goals of protection

Operating system consists of a collection of objects, hardware or software. Each object has a unique name and can be accessed through a well-defined set of operations.

Protection problem: ensure that each object is accessed correctly & only by those processes that are allowed to do so.

Reasons for providing protection: To prevent mischievous violation of an access restriction. To ensure that each program component active in a system uses system resources only in ways consistent with policies.

Mechanisms are distinct from policies:

Mechanisms determine how something will be done. Policies decide what will be done. This principle provides flexibility.

Principles of Protection

Explain the principles of protection

A key principle for protection is the principle of least privilege. Principle of Least Privilege: Programs, users, and even systems are given just enough privileges to perform their tasks. The principle of least privilege can help produce a more secure computing environment. An operating system which follows the principle of least privilege implements its features, programs, system-calls, and data structures. Thus, failure of a component results in minimum damage. An operating system also provides system-calls and services that allow applications to be written with fine-grained access controls.

Access Control provides mechanisms to enable privileges when they are needed, to disable privileges when they are not needed. Audit-trails for all privileged function-access can be created. Audit-trail can be used to trace all protection/security activities on the system.

The audit-trail can be used by Programmer, System administrator or Law-enforcement officer.

Managing users with the principle of least privilege requires creating a separate account for each user, with just the privileges that the user needs. Computers implemented in a computing facility under the principle of least privilege can be limited to running specific services, accessing specific remote hosts via specific services accessing during specific times. Typically, these restrictions are implemented through enabling or disabling each service and through using Access Control Lists.

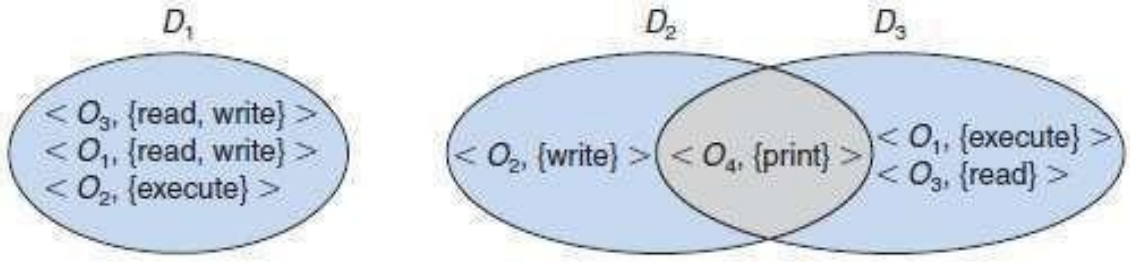
DOMAIN OF PROTECTION

A process operates within a protection domain. Protection domain specifies the resources that the process may access. Each domain defines set of objects and types of operations that may be invoked on each object. The ability to execute an operation on an object is an access-right.

A domain is a collection of access-rights. The access-rights are an ordered pair <object-name, rights-set>.

For example:

If domain D has the access-right <file F, {read, write}>; Then a process executing in domain D can both read and write on file F. As shown in below Figure, domains may share access-rights. The access-right <O4, {print}> is shared by D2 and D3.



System with three protection domains

The association between a process and a domain may be either static or dynamic.

If the association between processes and domains is static, then a mechanism must be available to change the content of a domain. Static means the set of resources available to the process is fixed throughout the process's lifetime.

If the association between processes and domains is dynamic, then a mechanism is available to allow domain switching. Domain switching allows the process to switch from one domain to another. A domain can be realized in a variety of ways:

Each user may be a domain.

Each process may be a domain.

Each procedure may be a domain.

Domain Structure

A protection domain specifies the resources a process may access. A domain is a collection of access rights, each of which is an ordered pair <object-name, rights-set>

Access right = the ability to execute an operation on an object. Access-right = <object-name, rights-set> where rights-set is a subset of all valid operations that can be performed on the object.

Domains also define the types of operations that can be invoked.

The association between a process and a domain may be:

Static (if the process' life-time resources are fixed): Violates the need-to-know principle

Dynamic: A process can switch from one domain to another.

A domain can be realized in several ways:

- Each user may be a domain
- Domain switching occurs when a user logs out.

- Each process may be a domain
- Domain switching occurs when a process sends a message to another process and waits for a response
- Each procedure may be a domain
- Domain switching occurs when a procedure call is made

ACCESS MATRIX

Explain Access Matrix with an example.

Access-matrix provides mechanism for specifying a variety of policies. The access matrix is used to implement policy decisions concerning protection. In the matrix, 1) Rows represent domains.2) Columns represent objects. Each entry consists of a set of access-rights (such as read, write or execute). In general, Access(i, j) is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

Example: Consider the access matrix shown in below Figure.There are Four domains: D_1, D_2, D_3, and D_4 and three objects: F_1, F_2 and F_3

A process executing in domain D_1 can read files F_1 and F_3.

object / domain	F_1	F_2	F_3
D_1	read		read
D_2			
D_3		read	execute
D_4	read write		read write

Access matrix

i. OR

Explain access matrix with domain as an object.

Domain switching allows the process to switch from one domain to another. When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain).We can include domains in the matrix to control domain switching. Consider the access matrix shown in Figure below: A process executing in domain D_2 can switch to domain D_3 or to domain D_4.

object / domain	F_1	F_2	F_3	D_1	D_2	D_3	D_4
D_1	read		read		switch		
D_2						switch	switch
D_3		read	execute				
D_4	read write		read write	switch			

Access matrix with domains as objects

Explain the operation in access matrix with example for each: i. Copy ii. Transfer iii. Limited copy

Allowing controlled change in the contents of the access-matrix entries requires 3 additional operations as shown in below figure:

- ✓ **Copy**(*) denotes ability for one domain to copy the access right to another domain.
- ✓ Owner denotes the process executing in that domain can add/delete rights in that column.
- ✓ Control in access (D_2, D_4) means: A process executing in domain D_2 can modify row D_4.

object / domain	F_1	F_2	F_3	D_1	D_2	D_3	D_4
D_1	read		read*		switch		
D_2						switch	switch control
D_3		read owner	execute				
D_4	write		write	switch			

Access matrix with Copy rights, Owner rights & Control rights

IMPLEMENTATION OF ACCESS MATRIX

************What is an access matrix? Explain the different methods of implementing access matrix.**

Access-matrix provides mechanism for specifying a variety of policies. The access matrix is used to implement policy decisions concerning protection in an operating system. In the matrix, 1) Rows represent domains.2) Columns represent objects. Each entry consists of a set of access-rights (such as read, write or execute).

In general, Access(i, j) is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

The different methods of implementing access matrix are:

1. By using Global table
2. Access lists for objects
3. Capability lists for domains
4. A Lock-Key mechanism

Global Table

A global table consists of a set of ordered triples <domain, object, rights-set>. Whenever an operation M is executed on an object O_j within domain D_i, the global table is searched for a triple $< D_i , O_j , R_k >$, with $M \in R_k$. If this triple is found, Then, we allow the access operation;Otherwise, access is denied, and an exception condition occurs.

Disadvantages: The table is usually large and can't be kept in main memory.

It is difficult to take advantage of groupings, e.g. if all may read an object, there must be an entry in each domain.

Access Lists for Objects

In the access-matrix, each column can be implemented as an access-list for one object. Obviously, the empty entries can be discarded. For each object, the access-list consists of ordered pairs <domain, rights-set>.

Working : Whenever an operation M is executed on an object O_j within domain D_i, the access list is searched for a triple $< D_i , R_k >$, with $M \in R_k$.

If this entry is found, Then, we allow the access operation; Otherwise, we check the default-set. If M is in the default-set, we allow the access operation; Otherwise, access is denied, and an exception condition occurs.

Advantages:

The strength is the control that comes from storing the access privileges along with each object.This allows the object to revoke or expand the access privileges in a localized manner.

Disadvantages:

The weakness is the overhead of checking whether the requesting domain appears on the access list. This check would be expensive and needs to be performed every time the object is accessed. Usually, the table is large & thus cannot be kept in main memory, so additional I/O is needed. It is difficult to take advantage of special groupings of objects or domains.

Capability Lists for Domains

For a domain, a capability list is a list of objects & operations allowed on the objects. Often, an object is represented by its physical name or address, called a capability. To execute operation M on object O_j, the process executes the operation M, specifying the capability (or pointer) for object O_j as a parameter. The capability list is associated with a domain. But capability list is never directly accessible by a process. Rather, the capability list is maintained by the OS & accessed by the user only indirectly. Capabilities are distinguished from other data in two ways:

- ✓ Each object has a tag to denote whether it is a capability or accessible data.
- ✓ Program address space can be split into 2 parts. One part contains normal data, accessible to the program. Another part contains the capability list, accessible only to the OS.

A Lock–Key Mechanism

The lock–key scheme is a compromise between 1) Access-lists and 2) Capability lists.

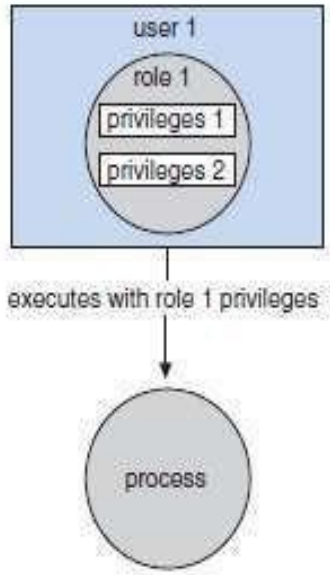
Each object has a list of unique bit patterns, called locks. Similarly, each domain has a list of unique bit patterns, called keys. A process executing in a domain can access an object only if that domain has a key that matches one of the locks of the object.

ACCESS CONTROL

Protection can be applied to non-file resources as shown in below figure:

Solaris 10 provides role-based access control (RBAC) to implement least privilege. Privilege is right to execute system call or use an option within a system call. Privilege can be assigned to processes.

Users assigned roles granting access to privileges and programs



Role-based access control in Solaris 10.

REVOCATION OF ACCESS RIGHTS

In a dynamic protection system, we may sometimes need to revoke access rights to objects shared by different users.

*******Explain the various questioning that arise in revocation of access rights**

Following questions about revocation may arise:

1. Immediate versus Delayed

Does revocation occur immediately, or is it delayed? If revocation is delayed, can we find out when it will take place?

2. Selective versus General

When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a select group of users whose access rights should be revoked?

3. Partial versus Total

Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?

4. Temporary versus Permanent

Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again? Schemes that implement revocation for capabilities include the following:

i. Reacquisition
   - ✓ Periodically, capabilities are deleted from each domain.
   - ✓ The process may then try to reacquire the capability.

ii. Back-Pointers
   - ✓ A list of pointers is maintained with each object, pointing to all capabilities associated with that object.
   - ✓ When revocation is required, we can follow these pointers, changing the capabilities as necessary

iii. Indirection
   - ✓ Each capability points to a unique entry in a global table, which in turn points to the object.
   - ✓ We implement revocation by searching the global table for the desired entry and deleting it.

iv. Keys
   - ✓ A key is associated with each capability and can't be modified / inspected by the process owning the capability
   - ✓ Master key is associated with each object; can be defined or replaced with the set-key operation