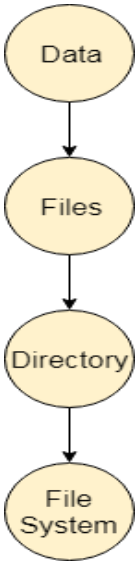
UNIT-V FILE SYSTEM INTERFACE and OPERATIONS

System files and data are kept in the computer system's memory, and when these files are needed by an application, the operating system must have some way to read the memory and access the appropriate files.

FILE

A file can be defined as a data structure which stores the sequence of records. Files are stored in a file system, which may exist on a disk or in the main memory. Files can be simple (plain text) or complex (specially-formatted).



The collection of files is known as **Directory**. The collection of directories at the different levels, is known as **File System**.

ATTRIBUTES OF THE FILE

1. Name

Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.

2. Identifier

Along with the name, Each File has its own extension which identifies the type of the file. For example, a text file has the extension **.txt,** A video file can have the extension **.mp4.**

3. Type

In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.

4. Location

In the File System, there are several locations on which, the files can be stored. Each file carries its location as its attribute.

5. Size

The Size of the File is one of its most important attribute. By size of the file, we mean the number of bytes acquired by the file in the memory.

6. Protection

The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.

7. Time and Date

Every file carries a time stamp which contains the time and date on which the file is last modified.

OPERATIONS ON THE FILE

The various operations which can be implemented on a file such as read, write, open and close etc. are called file operations. These operations are performed by the user by using the commands provided by the operating system.

Some common operations are as follows:

1. Create

This operation is used to create a file in the file system. It is the most widely used operation performed on the file system. To create a new file of a particular type the associated application program calls the file system. This file system allocates space to the file. As the file system knows the format of directory structure, so entry of this new file is made into the appropriate directory.

2. Open

This operation is the common operation performed on the file. Once the file is created, it must be opened before performing the file processing operations. When the user wants to open a file, it provides a file name to open the particular file in the file system. It tells the operating system to invoke the open system call and passes the file name to the file system.

3. Write

This operation is used to write the information into a file. A system call write is issued that specifies the name of the file and the length of the data has to be written to the file. Whenever the file length is increased by specified value and the file pointer is repositioned after the last byte written.

4. Read

This operation reads the contents from a file. A Read pointer is maintained by the OS, pointing to the position up to which the data has been read.

5. Re-position or Seek

The seek system call re-positions the file pointers from the current position to a specific place in the file i.e. forward or backward depending upon the user's requirement. This operation is generally performed with those file management systems that support direct access files.

6. Delete

Deleting the file will not only delete all the data stored inside the file it is also used so that disk space occupied by it is freed. In order to delete the specified file the directory is searched. When the directory entry is located, all the associated file space and the directory entry is released.

7. Truncate

Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

8. Close

When the processing of the file is complete, it should be closed so that all the changes made permanent and all the resources occupied should be released. On closing it deallocates all the internal descriptors that were created when the file was opened.

9. Append

This operation adds data to the end of the file.

10. Rename

This operation is used to rename the existing file.

File Type

File type	Usual extension	Function
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked
Source Code	C, 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
Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video information
Markup	xml, html, tex	It is the textual data and documents
Library	lib, a ,so, dll	It contains libraries of routines for programmers
Print or View	gif, pdf, jpg	It is a format for printing or viewing an ASCII or binary file.

FILE ACCESS METHODS

A file is a collection of bits/bytes or lines which are stored on secondary storage devices like a hard drive (magnetic disks).

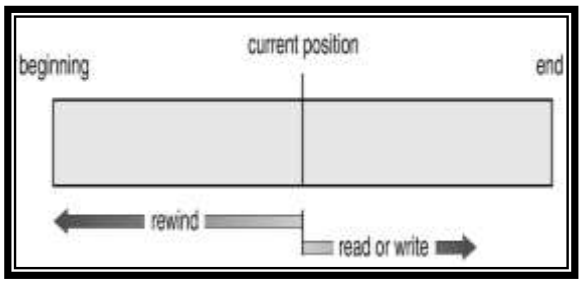
File access methods in OS are nothing but techniques to read data from the system's memory. There are various ways in which we can access the files from the memory like:

- Sequential Access
- Direct/Relative Access, and
- Indexed Sequential Access.

1. Sequential Access

The operating system reads the file word by word in sequential access method of file accessing. A pointer is made, which first links to the file's base address. If the user wishes to read the first word of the file, the pointer gives it to them and raises its value to the next word. This procedure continues till the file is finished. It is the most basic way of file access.

The data in the file is evaluated in the order that it appears in the file and that is why it is easy and simple to access a file's data using sequential access mechanism. For example, editors and compilers frequently use this method to check the validity of the code.



Advantages

- The sequential access mechanism is very easy to implement.
- It uses lexicographic order to enable quick access to the next entry.

Disadvantages

- Sequential access will become slow if the next file record to be retrieved is not present next to the currently pointed record.
- Adding a new record may need relocating a significant number of records of the file.

2. Direct (or Relative) Access

A Direct/Relative file access mechanism is mostly required with the database systems. In the majority of the circumstances, we require filtered/specific data from the database, and in such circumstances, sequential access might be highly inefficient.

sequential access	implementation for direct access
reset	*cp = 0;*
read next	*read cp;* *cp = cp+1;*
write next	*write cp;* *cp = cp+1;*

Assume that each block of storage holds four records and that the record we want to access is stored in the tenth block. In such a situation, sequential access will not be used since it will have to traverse all of the blocks to get to the required record, while direct access will allow us to access the required record instantly.

The direct access mechanism requires the OS to perform some additional tasks but eventually leads to much faster retrieval of records as compared to the sequential access.

Advantages

- The files can be retrieved right away with direct access mechanism, reducing the average access time of a file.
- There is no need to traverse all of the blocks that come before the required block to access the record.

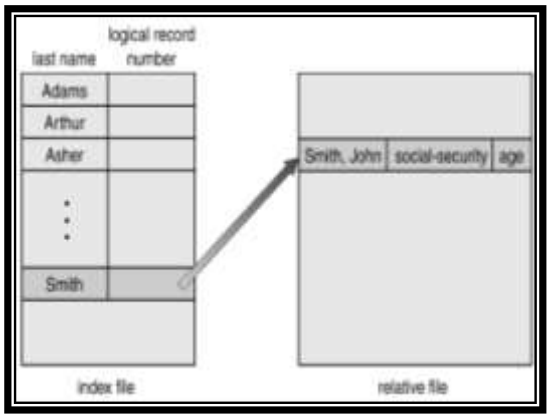
Disadvantages

- The direct access mechanism is typically difficult to implement due to its complexity.
- Organizations can face security issues as a result of direct access as the users may access/modify the sensitive information. As a result, additional security processes must be put in place.

3. Indexed Sequential Access

This method is practically similar to the pointer to pointer concept in which we store an address of a pointer variable containing address of some other variable/record in another pointer variable.

The indexes, similar to a book's index (pointers), contain a link to various blocks present in the memory. To locate a record in the file, we first search the indexes and then use the pointer to pointer concept to navigate to the required file.



Primary index blocks contain the links of the secondary inner blocks which contains links to the data in the memory.

Advantages

- If the index table is appropriately arranged, it accesses the records very quickly.
- Records can be added at any position in the file quickly.

Disadvantages of Indexed Sequential Access

- When compared to other file access methods, it is costly and less efficient.
- It needs additional storage space.

DIRECTORY STRUCTURE

Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes.

To get the benefit of different file systems on the different operating systems, A hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks.

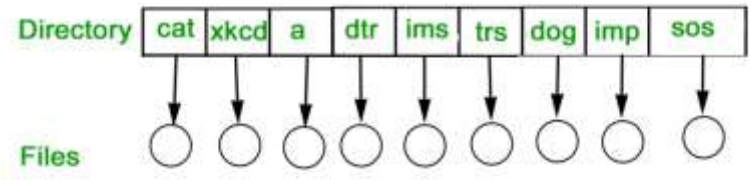
Each partition must have at least one directory in which, all the files of the partition can be listed. A directory entry is maintained for each file in the directory which stores all the information related to that file.

A directory can be viewed as a file which contains the Meta data of the bunch of files.

Every Directory supports a number of common operations on the file:

- File Creation
- Search for the file
- File deletion
- Renaming the file
- Traversing Files
- Listing of files

SINGLE LEVEL DIRECTORY



The simplest method is to have one big list of all the files on the disk. The entire system will contain only one directory which is supposed to mention all the files present in the file system. The directory contains one entry per each file present on the file system.

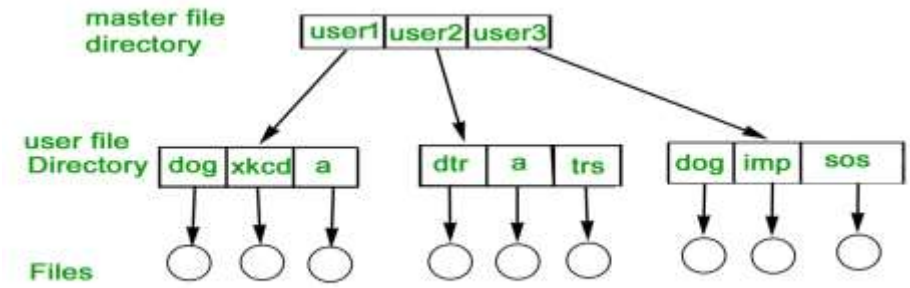
Advantages

1. Implementation is very simple.
2. If the sizes of the files are very small then the searching becomes faster.
3. File creation, searching, deletion is very simple since we have only one directory.

Disadvantages

- **Naming problem:** Users cannot have the same name for two files.
- **Grouping problem:** Users cannot group files according to their needs.

TWO-LEVEL DIRECTORY



In two level directory systems, we can create a separate directory for each user. There is one master directory which contains separate directories dedicated to each user. For each user, there is a different directory present at the second level, containing group of user's file. The system doesn't let a user to enter in the other user's directory without permission.

Path name: Due to two levels there is a path name for every file to locate that file.

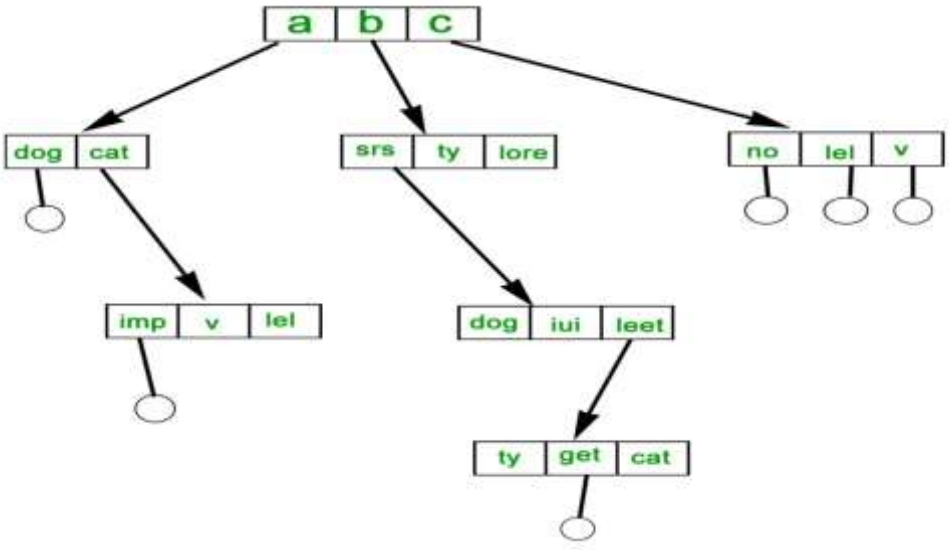
Advantage

- we can have the same file name for different users.
- Searching is efficient in this method.

TREE- STRUCTURED DIRECTORY

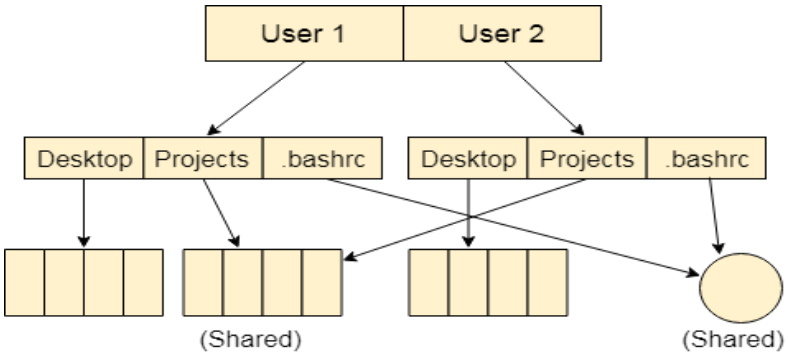
In Tree structured directory system, any directory entry can either be a file or sub directory. Tree structured directory system overcomes the drawbacks of two level directory system. The similar kind of files can now be grouped in one directory.

The directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.



ACYCLIC-GRAPH STRUCTURED DIRECTORY

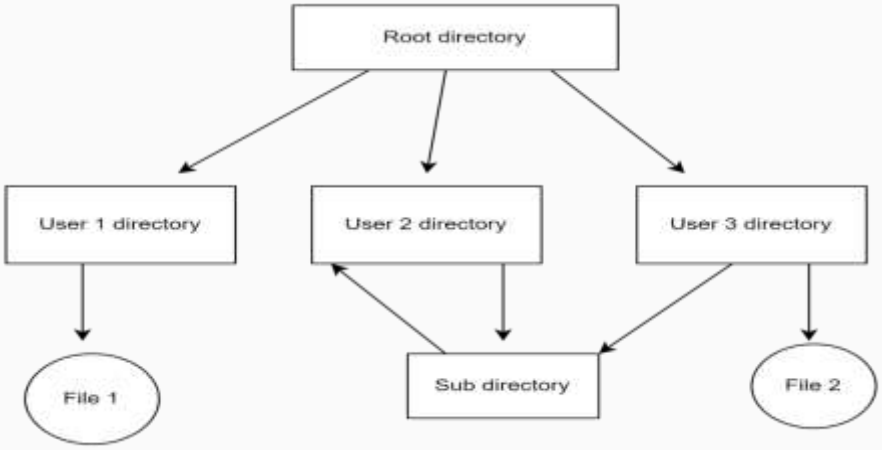
The tree structured directory system doesn't allow the same file to exist in multiple directories therefore sharing is major concern in tree structured directory system. We can provide sharing by making the directory an acyclic graph. In this system, two or more directory entry can point to the same file or sub directory. That file or sub directory is shared between the two directory entries.



Acyclic-Graph Structured Directory System

GENERAL-GRAPH DIRECTORY

This is an extension to the acyclic-graph directory. In the general-graph directory, there can be a cycle inside a directory.



In the above image, we can see that a cycle is formed in the user 2 directory. Although it provides greater flexibility, it is complex to implement this structure.

Advantages

- Compared to the others, the General-Graph directory structure is more flexible.
- Cycles are allowed in the directory for general-graphs.

Disadvantages

- It costs more than alternative solutions.
- Garbage collection is an essential step here.

PROTECTION IN FILE SYSTEM

In computer systems, a lot of user's information is stored, the objective of the operating system is to keep safe the data of the user from the improper access to the system.

Protection can be provided in number of ways. For a single laptop system, we might provide protection by locking the computer in a desk drawer or file cabinet. For multi-user systems, different mechanisms are used for the protection.

Types of Access

The files which have direct access of the any user have the need of protection. The files which are not accessible to other users doesn't require any kind of protection.

The mechanism of the protection provide the facility of the controlled access by just limiting the types of access to the file. Access can be given or not given to any user depends on several factors, one of which is the type of access required.

Several different types of operations can be controlled:

- **Read –** Reading from a file.
- **Write –** Writing or rewriting the file.
- **Execute –** Loading the file and after loading the execution process starts.
- **Append –** Writing the new information to the already existing file, editing must be end at the end of the existing file.
- **Delete –** Deleting the file which is of no use and using its space for the another data.
- **List –** List the name and attributes of the file.

Operations like renaming, editing the existing file, copying; these can also be controlled. There are many protection mechanism. each of them mechanism have different advantages and disadvantages and must be appropriate for the intended application.

Access Control

There are different methods used by different users to access any file. The general way of protection is to associate *identity-dependent access* with all the files and directories a list called access-control list (ACL) which specify the names of the users and the types of access associate with each of the user.

The main problem with the access list is their length. If we want to allow everyone to read a file, we must list all the users with the read access. This technique has two undesirable consequences:

Constructing such a list may be tedious and unrewarding task, especially if we do not know in advance the list of the users in the system.

Previously, the entry of the any directory is of the fixed size but now it changes to the variable size which results in the complicates space management. These

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

- **Owner –** Owner is the user who has created the file.
- **Group –** A group is a set of members who has similar needs and they are sharing the same file.
- **Universe –** In the system, all other users are under the category called universe.

The most common recent approach is to combine access-control lists with the normal general owner, group, and universe access control scheme. For example: Solaris uses the three categories of access by default but allows access-control lists to be added to specific files and directories when more fine-grained access control is desired.

Other Protection Approaches

The access to any system is also controlled by the password. If the use of password is random and it is changed often, this may be result in limit the effective access to a file.

The use of passwords has a few disadvantages:

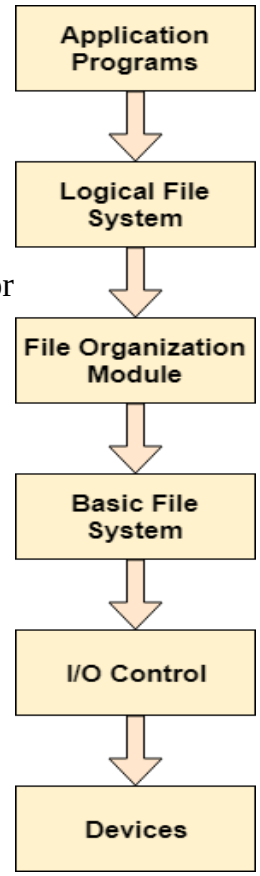
- The number of passwords are very large so it is difficult to remember the large passwords.
- If one password is used for all the files, then once it is discovered, all files are accessible; protection is on all-or-none basis.

FILE SYSTEM STRUCTURE

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown, elaborates how the file system is divided in different layers, and also the functionality of each layer.



- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

ALLOCATION METHODS

There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed.

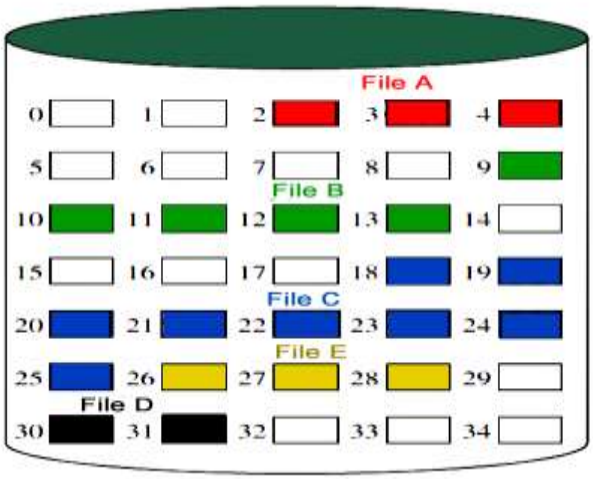
There are following methods which can be used for allocation.

1. Contiguous Allocation.
2. Linked Allocation
3. Indexed Allocation
4. Linked Indexed Allocation
5. Multilevel Indexed Allocation

Contiguous Allocation

A single continuous set of blocks is allocated to a file at the time of file creation. Thus, this is a pre-allocation strategy, using variable size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. This method is best from the point of view of the individual sequential file.

Multiple blocks can be read in at a time to improve I/O performance for sequential processing. It is also easy to retrieve a single block. For example, if a file starts at block b, and the ith block of the file is wanted, its location on secondary storage is simply b+i-1.



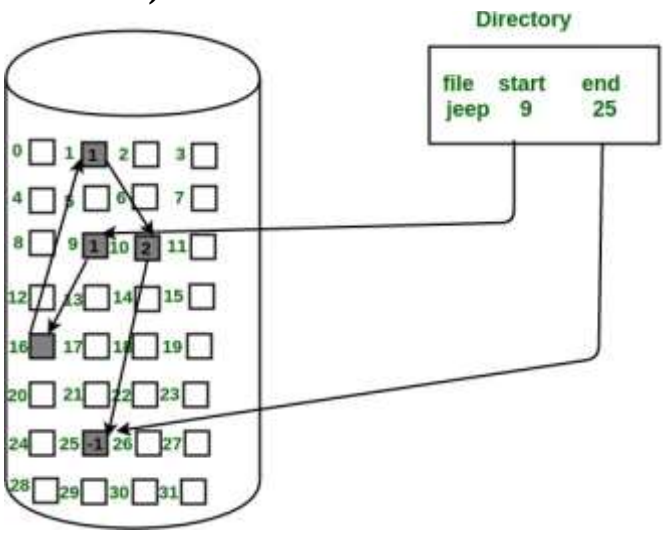
File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Disadvantage

- External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length. A compaction algorithm will be necessary to free up additional space on the disk.
- Also, with pre-allocation, it is necessary to declare the size of the file at the time of creation.

Linked Allocation(Non-contiguous allocation)



Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file, showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. An increase in file size is always possible if a free disk block is available. There is no external fragmentation because only one block at a time is needed but there can be internal fragmentation but it exists only in the last disk block of the file.

Disadvantage

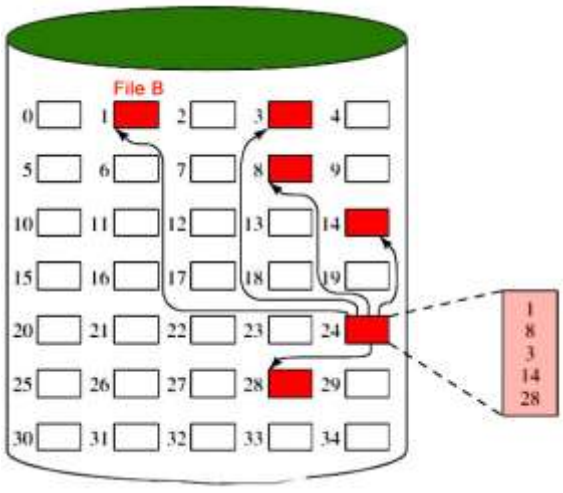
- Internal fragmentation exists in the last disk block of the file.
- There is an overhead of maintaining the pointer in every disk block.
- If the pointer of any disk block is lost, the file will be truncated.
- It supports only the sequential access of files.

Indexed Allocation

It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file: The index has one entry for each block allocated to the file.

The allocation may be on the basis of fixed-size blocks or variable-sized blocks. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size blocks improves locality.

This allocation technique supports both sequential and direct access to the file and thus is the most popular form of file allocation.



File allocation table

File name	Index block
•••	•••
File B	24
•••	•••

FREE SPACE MANAGEMENT

It is not easy work for an operating system to allocate and de-allocate memory blocks (managing free space) simultaneously. The operating system uses various methods for adding free space and freeing up space after deleting a file. There are various methods using which a free space list can be implemented. We are going to explain them below-

1. Bitmap or Bit Vector :

A bit vector is a most frequently used method to implement the free space list. A bit vector is also known as a **Bit map**. It is a series or collection of bits in which each bit represents a disk block.

The values taken by the bits are either **1** or **0**. If the block bit is 1, it means the block is empty and if the block bit is 0, it means the block is not free. It is allocated to some files. Since all the blocks are empty initially so, each bit in the bit vector represents 0.

"Free block number" can be defined as that block which does not contain any value, i.e., they are free blocks.

The formula to find a free block number is :

[Block number = (number of bits per words)*(number of **0**-value word) + Offset of first **1** bit]

2. Linked List :

A **linked list** is another approach for free space management in an operating system. In it, all the free blocks inside a disk are linked together in a **linked list**. These free blocks on the disk are linked together by a pointer. These pointers of the free block contain the address of the next free block and the last pointer of the list points to null which indicates the end of the linked list.

This technique is not enough to traverse the list because we have to read each disk block one by one which requires I/O time.

The operating system can use this linked list to allocate memory blocks to processes as needed.

3. Grouping

The grouping technique is also called the **"modification of a linked list technique"**. In this method, first, the free block of memory contains the addresses of the **n-free** blocks. And the last free block of these **n** free blocks contains the addresses of the next **n** free block of memory and this keeps going on. This technique separates the empty and occupied blocks of space of memory.

4. Counting

In memory space, several files are created and deleted at the same time. For which memory blocks are allocated and de-allocated for the files. Creation of files occupy free blocks and deletion of file frees blocks.

When there is an entry in the free space, it consists of two parameters- **"address of first free disk block (a pointer)"** and **"a number 'n'"**.

SYSTEM CALLS

1. create

The create() function is used to create a new empty file in C. We can specify the permission and the name of the file which we want to create using the create() function. It is defined inside **<unistd.h>** header file and the flags that are passed as arguments are defined inside **<fcntl.h>** header file.

Syntax of create() in C

int **create**(char **filename*, mode_t *mode*);

Parameter

- **filename:** name of the file which you want to create
- **mode:** indicates permissions of the new file.

Return Value

- return first unused file descriptor (generally 3 when first creating use in the process because 0, 1, 2 fd are reserved)
- return -1 when an error

2. open

The open() function in C is used to open the file for reading, writing, or both. It is also capable of creating the file if it does not exist. It is defined inside **<unistd.h>** header file and the flags that are passed as arguments are defined inside **<fcntl.h>** header file.

Syntax of open() in C

int **open** (const char* *Path*, int *flags*);

Parameters

- **Path:** Path to the file which we want to open.
  - Use the **absolute path** beginning with "/" when you are **not working in the same directory** as the C source file.
  - Use **relative path** which is only the file name with extension, when you are **working in the same directory** as the C source file.
- **flags:** It is used to specify how you want to open the file. We can use the following flags.

Flags	Description
O_RDONLY	Opens the file in read-only mode.
O_WRONLY	Opens the file in write-only mode.
O_RDWR	Opens the file in read and write mode.

Flags	Description
O_CREAT	Create a file if it doesn't exist.
O_EXCL	Prevent creation if it already exists.
O_ APPEND	Opens the file and places the cursor at the end of the contents.
O_ASYNC	Enable input and output control by signal.
O_CLOEXEC	Enable close-on-exec mode on the open file.
O_NONBLOCK	Disables blocking of the file opened.
O_TMPFILE	Create an unnamed temporary file at the specified path.

3. close

The close() function in C tells the operating system that you are done with a file descriptor and closes the file pointed by the file descriptor. It is defined inside **<unistd.h>** header file.

Syntax of close() in C

int close(int fd);

Parameter

- **fd: F**ile descriptor of the file that you want to close.

Return Value

- **0** on success.
- **-1** on error.

4. read

From the file indicated by the file descriptor fd, the read() function reads the specified amount of bytes **cnt** of input into the memory area indicated by **buf**. The read() function is also defined inside the <unistd.h> header file.

Syntax of read() in C

size_t **read** (int *fd*, void* *buf*, size_t *cnt*);

Parameters

- **fd:** file descriptor of the file from which data is to be read.
- **buf:** buffer to read data from
- **cnt:** length of the buffer

Return Value

- return Number of bytes read on success
- return 0 on reaching the end of file
- return -1 on error
- return -1 on signal interrupt

5. write

Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT_MAX (defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any other action.

The write() is also defined inside **<unistd.h>** header file.

Syntax of write() in C

size_t **write** (int *fd*, void* *buf*, size_t *cnt*);

Parameters

- **fd:** file descriptor
- **buf:** buffer to write data to.
- **cnt:** length of the buffer.

Return Value

- returns the number of bytes written on success.
- return 0 on reaching the End of File.
- return -1 on error.
- return -1 on signal interrupts.

6. ioctl

- ioctl() is referred to as Input and Output Control.
- ioctl is a system call for device-specific input/output operations and other operations which cannot be expressed by regular system calls.

7. fork

- A new process is created by the fork() system call.
- A new process may be created with fork() without a new program being run-the new sub-process simply continues to execute exactly the same program that the first (parent) process was running.
- It is one of the most widely used system calls under process management.

8. exit

- The exit() system call is used by a program to terminate its execution.
- The operating system reclaims resources that were used by the process after the exit() system call.

9. exec

- A new program will start executing after a call to exec()
- Running a new program does not require that a new process be created first: any process may call exec() at any time. The currently running program is immediately terminated, and the new program starts executing in the context of the existing process.

10. wait

The **wait**() system call suspends execution of the current process until one of its children terminates. The call *wait(&status)* is equivalent to:

waitpid(-1, &status, 0);

11. waitpid

The **waitpid**() system call suspends execution of the current process until a child specified by *pid* argument has changed state. By default, **waitpid**() waits only for terminated children, but this behaviour is modifiable via the *options* argument, as described below.

The value of *pid* can be:

Tag	Description
< -1	meaning wait for any child process whose process group ID is equal to the absolute value of *pid*.
-1	meaning wait for any child process.
0	meaning wait for any child process whose process group ID is equal to that of the calling process.
> 0	meaning wait for the child whose process ID is equal to the value of *pid*.

DISK SCHEDULING ALGORITHMS

A process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.

However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.

The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Seek Time

Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

Rotational Latency

It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

Transfer Time

It is the time taken to transfer the data.

Disk Access Time

Disk access time is given as,

Disk Access Time = Rotational Latency + Seek Time + Transfer Time

Disk Response Time

It is the average of time spent by each request waiting for the IO operation.

Purpose of Disk Scheduling

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

Goal of Disk Scheduling Algorithm

- Fairness
- High throughout
- Minimal traveling head time

Disk Scheduling Algorithms

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.

- FCFS scheduling algorithm
- SSTF (shortest seek time first) algorithm
- SCAN scheduling
- C-SCAN scheduling
- LOOK Scheduling
- C-LOOK scheduling

FCFS Scheduling Algorithm

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

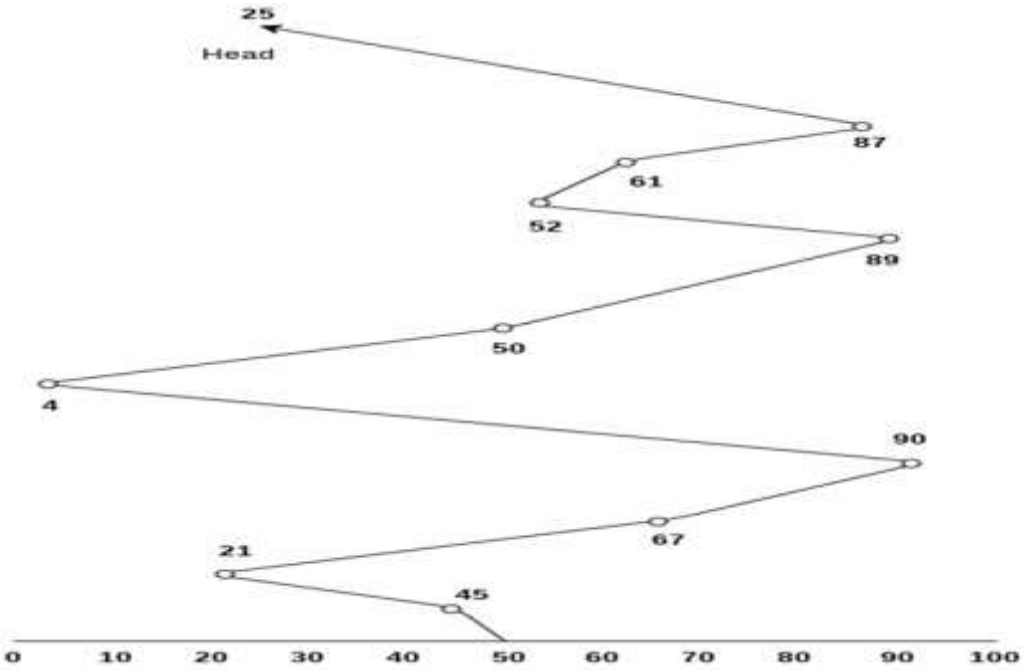
Disadvantages

- The scheme does not optimize the seek time.
- The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

Example

Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25. Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

Solution



Number of cylinders moved by the head

= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376

SSTF Scheduling Algorithm

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction.

It reduces the total seek time as compared to FCFS.

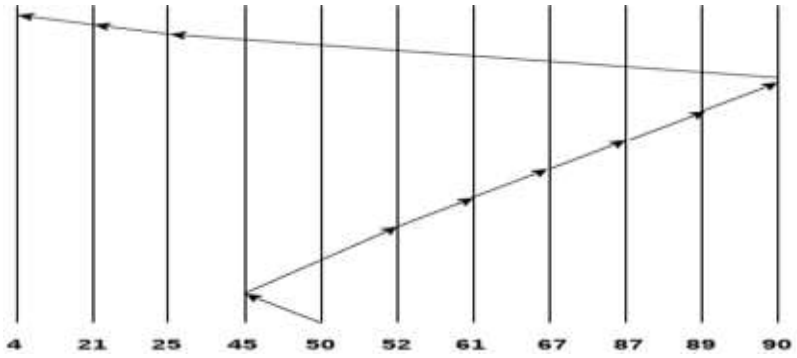
It allows the head to move to the closest track in the service queue.

Disadvantages

- It may cause starvation for some requests.
- Switching direction on the frequent basis slows the working of algorithm.
- It is not the most optimal algorithm.

Example

Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 89, 52, 61, 87, 25. Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.



Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

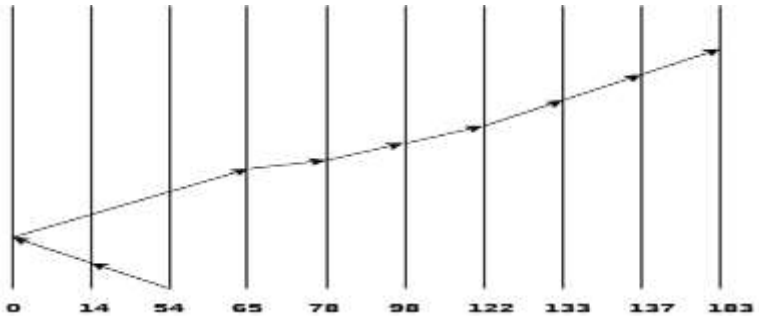
SCAN Algorithm

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path and then it turns back and moves in the reverse direction satisfying requests coming in its path.

It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

Example

Consider the following disk request sequence for a disk with 100 tracks 98, 137, 122, 183, 14, 133, 65, 78. Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using SCAN scheduling.



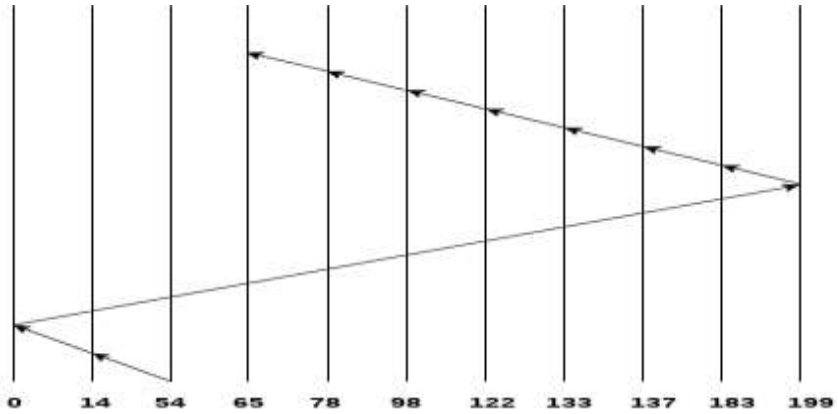
Number of Cylinders = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237

C-SCAN algorithm

In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

Example

Consider the following disk request sequence for a disk with 100 tracks 98, 137, 122, 183, 14, 133, 65, 78. Head pointer starting at 54 and moving in left direction.



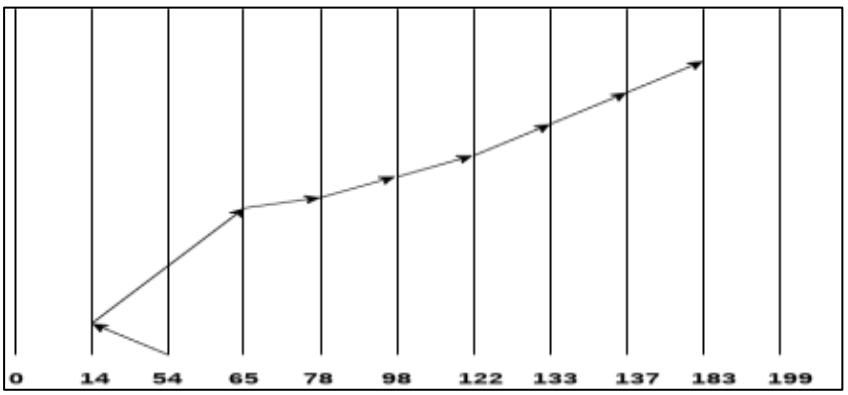
No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

LOOK Scheduling

It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

Example

Consider the following disk request sequence for a disk with 100 tracks 98, 137, 122, 183, 14, 133, 65, 78. Head pointer starting at 54 and moving in left direction.



Number of cylinders crossed = 40 + 51 + 13 + +20 + 24 + 11 + 4 + 46 = 209

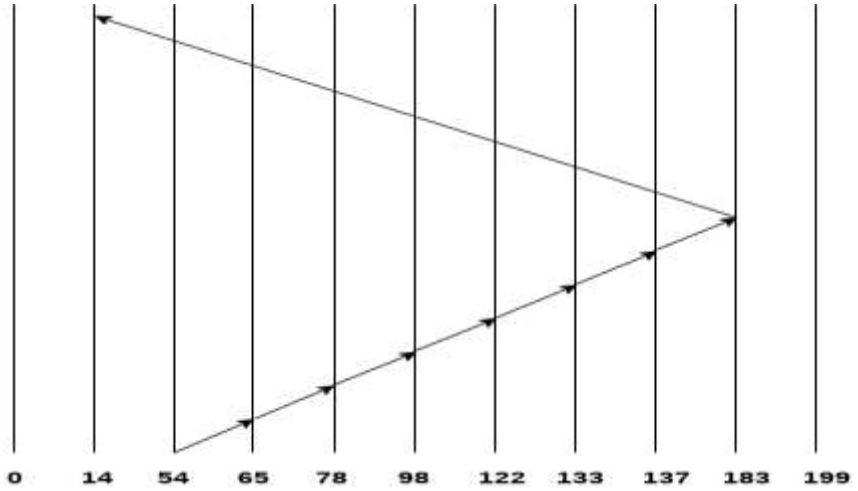
C LOOK Scheduling

C Look Algorithm is similar to C-SCAN algorithm to some extent. In this algorithm, the arm of the disk moves outwards servicing requests until it reaches the highest request cylinder, then it jumps to the lowest request cylinder without servicing any request then it again start moving outwards servicing the remaining requests.

It is different from C SCAN algorithm in the sense that, C SCAN force the disk arm to move till the last cylinder regardless of knowing whether any request is to be serviced on that cylinder or not.

Example

Consider the following disk request sequence for a disk with 100 tracks 98, 137, 122, 183, 14, 133, 65, 78. Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C LOOK scheduling.



Number of cylinders crossed = 11 + 13 + 20 + 24 + 11 + 4 + 46 + 169 = 298