OS Unit-5 File System Management (1st Semester)

A file is a collection of related information defined by its creator. Files represent programs and data. Data files may be numeric, alphabetic or alphanumeric. File is a sequence of bits, byte, lines or record whose meaning is defined by its user. A file is named and referred by its name. It has certain other properties such as its type, time of its creation, its length so on.

File Attributes :-

1. Name :- It is a name given to a file by user.
2. Identifier :- This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
3. Type :- It is one of the most important file attributes because this attribute specifies which kind of file is this e.g :- Ms-word file, power point file etc.
4. Location :- Location is the addressable space, which directs the system or user to a file where it get placed.
5. Size :- The current size of the file (in bytes, words or blocks) are included in this attributes.
6. Protection :- A protection attributes about permission that are granted to user with respect to operations performed on that file. e.g :- Read only permission, Read-write permission etc. because of this a file can be protected by unauthorized access of a file.
7. Time, date and user identification :- These set of attributes are used together for protection, security and record keeping purpose because whenever the information is required about a particular file such as when it gets

(11) Created, at what time, when created, modified, accessed, deleted that file etc.

File operation :- Following operations are performed on a file

(1) Creating a file :- A new file is declared, definition is written and obtained that place within the system hierarchy or system structure.

(2) Delete :- It removes a file from the system structure, it's allocated memory and associated relationship such as dependencies will be destroyed.

(3) Open :- When a file gets opened it means that it is available for user i.e for performing different operation on the data contained within the file for a process

(4) Close :- A file is closed means, it is not available for performing any operation. This will be present on the disk only, but cannot be used until an open operation is performed.

(5) Read :- A process inside the system either read whole data inside the file or a part of data.

(6) Write :- A write operation can be performed on file by adding new data elements to a file which increases the size of a file.

(7) Append :- When a record or data is added at the end of file then it is called as append. Actually it is a write operation but with condition write at the end.

(8) Truncate :- With the help of truncate option only file contents get deleted but the structure of file remains there.

(9) Rename :- It will provide a new name to a file. There is a system call which changes old name of file with a new name.

File Types : Four (3) implementing file type is to include the ... as part of file name. The name is split into two parts ie name and extension separated by a period character. In this way user and operating system can tell from the name alone what type of a file is

file type	extension	function
executable	.exe, .com	read to run machine language program
text	~~text~~ .txt, .doc	textual data, documents
object	.obj .o	compiled, machine language
etc.		

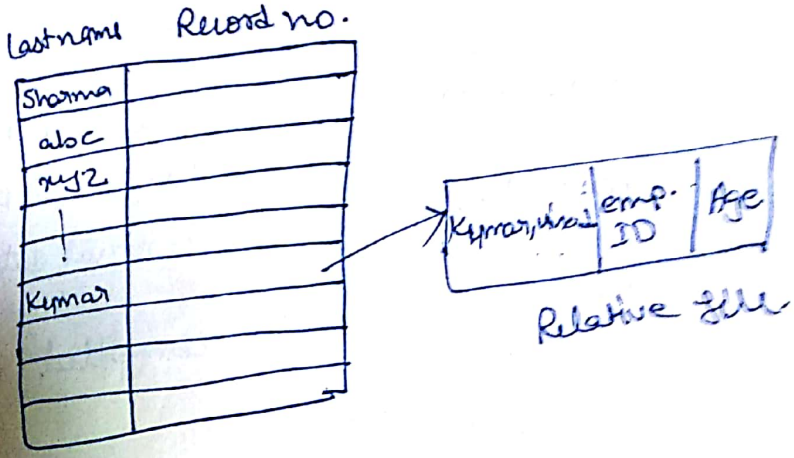
Access Method :-

1. Sequential Access
2. Direct Access
3. Index Access.

1. **Sequential Access**: In this information stored in a file are accessed in an order such that one record is processed after the other. This method is based on the tape model of a file and works well on sequential access devices.
2. **Direct Access**: It is based on the disk model of a file, since disks allow random access to any blocks or record of a file. File operations in this method uses blocks number as a parameter which is provided by the user to the OS.
3. **Indexed Access**: In this method a index is created which contains a key field and pointers to the various blocks. To find an entry in the file for a key value, we first search

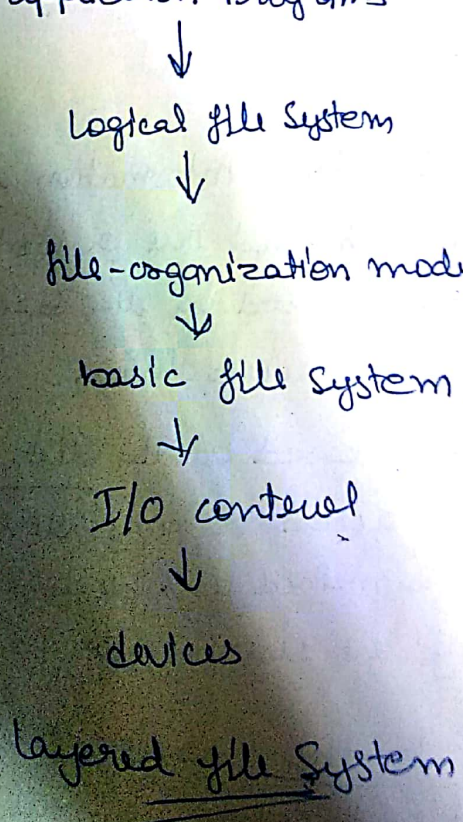
the index and then use the pointer to directly access a file to find the desired entry.

With large files, the index file itself may become too large to be kept in memory. One solution is to create an index for the index file. The primary index file would contain pointers to secondary files, which would point to the actual items.



Relative file

File System Structure: The file system composed of many different levels.



Layered file System

(5)

The lowest level, the I/O control, consist of device drivers and interrupt handlers to transfer information b/w the main memory and the disk system. A device drivers can be thought of as a translator. Its input consist of high level commands and it output consists of low level, H/w-specific instructions that are used by the hardware controller, which interfaces the I/O device to the system.

⇒ The basic file System needs only to issue commands to the device drivers to read and write physical blocks on the disk. Each physical block is identified by its numeric address.

⇒ The file-organization module knows about files and their logical blocks as well as physical blocks. By knowing the type of file allocation used and location of the file, the file-organization module can translate logical block addresses to physical block addresses. The file-organization module also includes the free-space manager, which tracks unallocated blocks and provides these blocks to the file-organization module when requested.

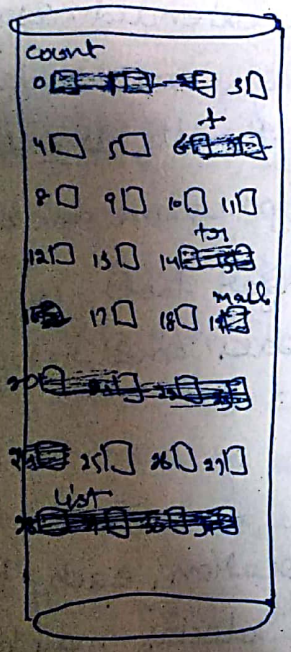
⇒ Logical file System manages metadata information. Metadata includes all of the file-system structure, excluding the actual data. The logical file system manages the directory structure to provide the file-organization module with the information the latter needs, given a symbolic file name. It maintains file structure via file control blocks. A file control block contains information about the file, including ownership, permission, and location of the file contents. The logical file system is also responsible for protection and security.

Three major methods of allocating disk space are in use:

1. Contiguous
2. Linked
3. Indexed.

1. Contiguous Allocation :-

This method requires each file to occupy a set of contiguous blocks on the disk. The directory entry for each file indicates the address of the starting blocks and length of the area allocated for this file.



Directory

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

Contiguous Allocation of disk space

Contiguous allocation of a file is defined by the disk address and length of the first block. If the file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2, ---b+n-1.

Accessing a file that has been allocated contiguously is easy. For sequential access, the file system remembers the disk address of the last block referenced and, when necessary, reads the next block. For direct access to block i of a file that starts at block b, we can access block b+i

Both sequential and direct access can be supported by contiguous allocation.

* One difficulty is finding space for a new file.
  - => First fit and
  - => Best fit

These algorithms suffer from external fragmentation and need solved by compaction.

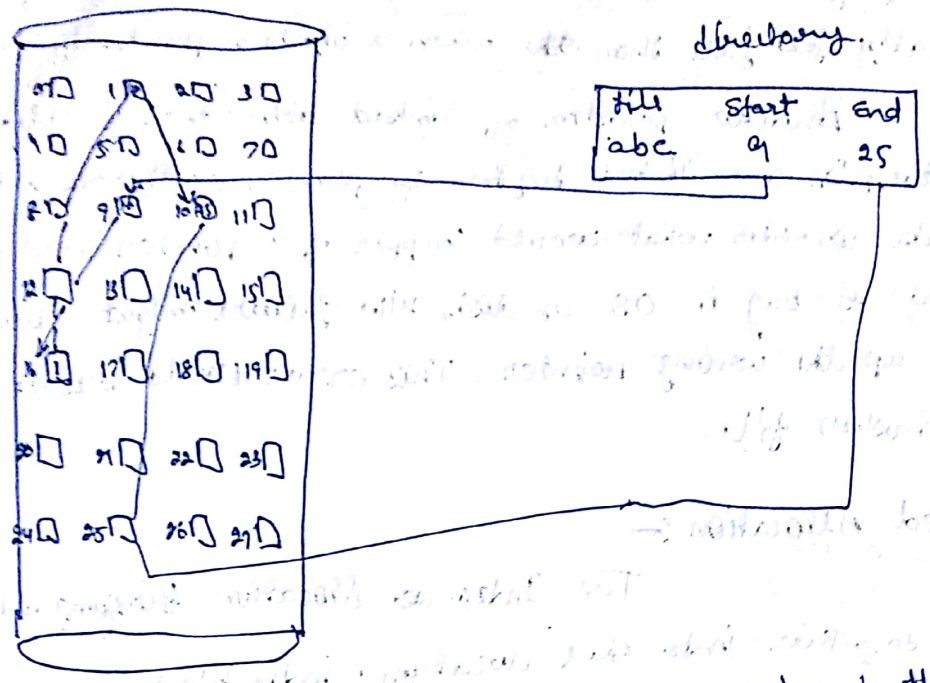
Another problem with contiguous allocation is determining how much space is needed for a file. When a file is created the total amount of space it will need must be found and allocated.

If we allocate too little space to a file we may find that the file cannot be extended. Especially with a best-fit allocation, the space on both sides of the file may be in use. We cannot make the file larger in place. Two possibilities exist: First the user program can be terminated The other possibility is to find a larger hole, copy the contents of the file to the new space and release the previous space.

Linked Allocation:-

With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

e.g:- a file of five blocks might start at block 9, continue at block 16, then block 10 and finally block 25. Each block contains a pointer to the next block.



To create a new file, we create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first block of the file. This pointer is initialized to nil to signify an empty file. The size field is also set to 0. A write to the file causes a free block to be found via the free-space-management system and this new block is written to and linked to end of the file. There is no external fragmentation with linked allocation and any free blocks on the free space list can be used to satisfy a request. The size of a file does not need to be declared when it is created. A file can continue to grow as long as free blocks are available.

The major problem is that it can be used only for sequential access files. To find the i^{th} block of the file, we must start at the beginning of that file and follow the pointers until we get the i^{th} block. It is inefficient to support a direct-access capability for linked allocation files.

Another disadvantage to linked allocation is space required for pointer. If pointer requires 4 bytes out of 512 byte block then 78% of the disk is being used for pointers rather than information.

The solution to this problem is to collect blocks into multiples called clusters and allocate clusters

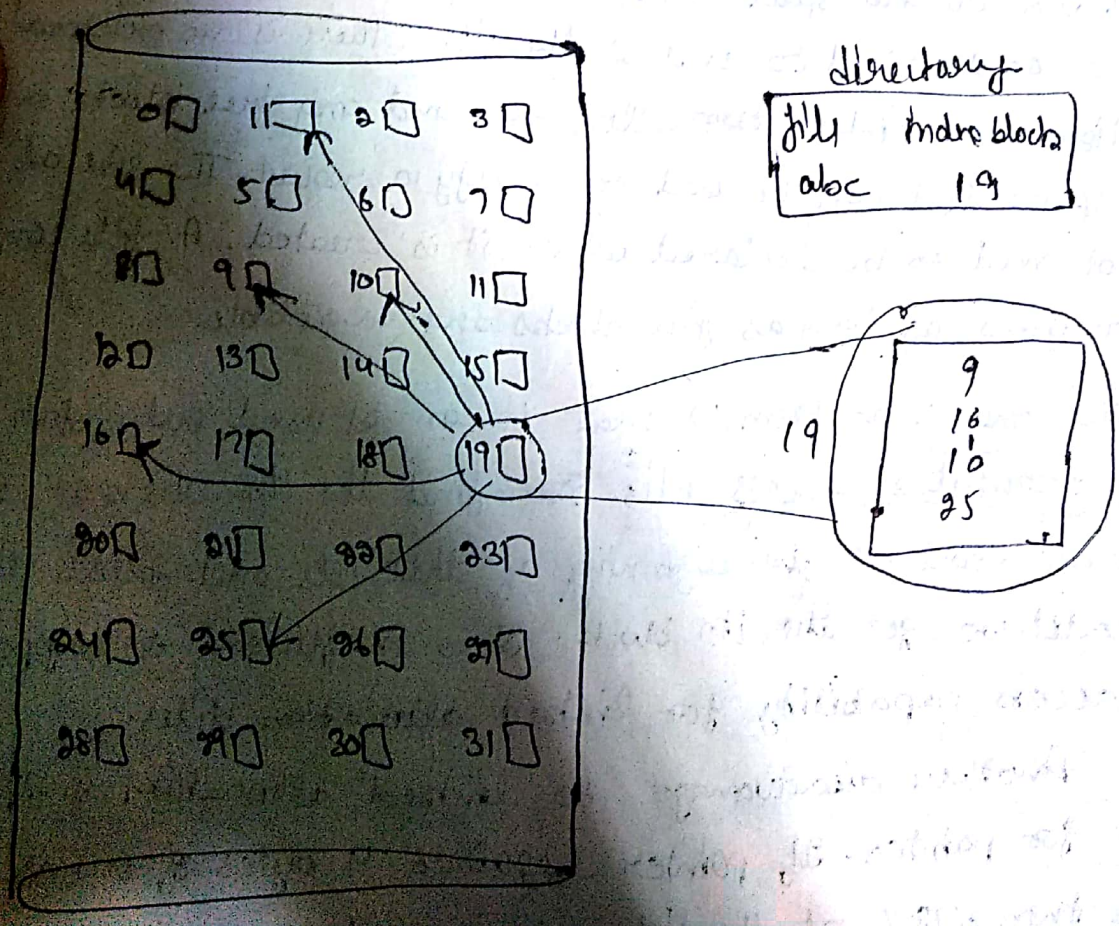
rather than blocks. The cost of this approach is an increase in internal fragmentation, because more space is wasted if a cluster is partially full than when a block is partially full.

Another problem of linked allocation is reliability. Since the files are linked together by pointer scattered all over the disk, consider what would happen if a pointer were lost or damaged. A bug in OS or disk H/w failure might result in picking up the wrong pointer. This error could result in linking into another file.

Indexed Allocation :-

The Index allocation bringing all the pointers together into one location : index block.

Each file has its own index block, which is an array of disk block address. The i^{th} entry in the index block points to the i^{th} block of the file. The directory contains the address of the index table.



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

Index allocation supports direct access, without suffering external fragmentation. The pointer overhead of the index blocks is greater than the pointer overhead of linked allocation.

Free Space Management:

To keep track of free disk space, the system maintains a free-space list. The free space list records all the free disk blocks - those not allocated to some file or directory. To create a file, we search the free-space list for required amount of space and allocate that space to the new file. This space is removed from the free space list. When a file is deleted, its disk space is added to the free-space list.

① Bit Vector:

The free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit. If the block is free, the bit is 1; If the block is allocated, the bit is 0.

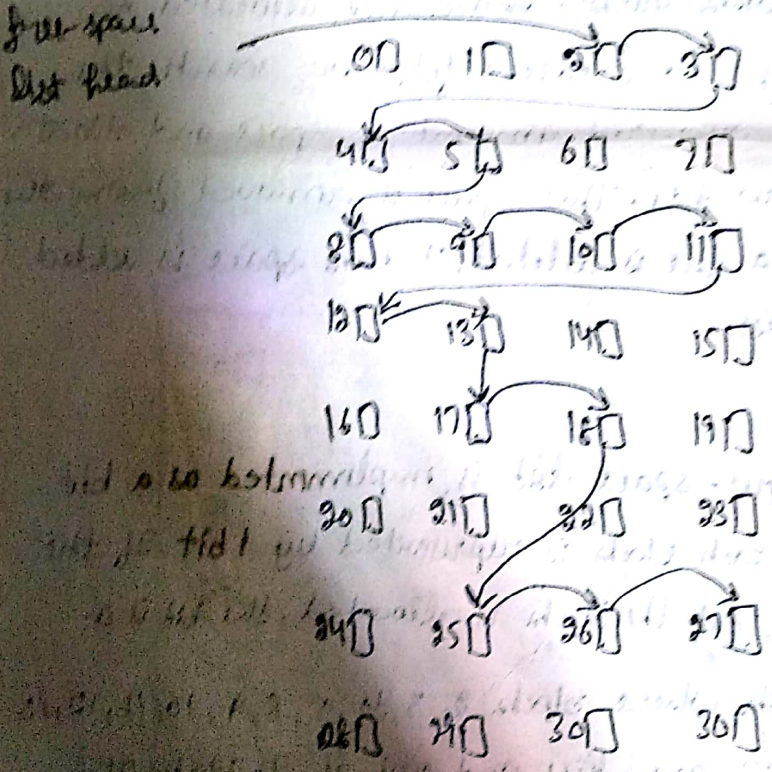
e.g.:- Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13 17, 18, 25, 26 and 27 are free and rest of the blocks are allocated. The free space bit map would be

0011110011111100011000001110000...

Linked List :

Another approach, is to link together all the free disk blocks, keeping a pointer to the first block in a special location on the disk and caching it in memory. The first block contains a pointer to the next free disk block and so on.

In our example, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5 which would point to block 8, and so on. This scheme is not efficient; to traverse the list we must read each block, which requires I/O time.



Linked free space list on disk

Grouping : A modification of the free-list approach is to store the addresses of n free blocks in the first free block. The importance of this implementation is that the addresses of a large number of empty free blocks can be found quickly.

Protection :-

The processes in an operating system must be protected from one another's activities. Protection refers to a mechanism for controlling the access of programs, processes or users to the resources defined by a computer system.

Goals of Protection :- Access restriction by a user.

Domain of Protection :-

A computer system is a collection of Processes and objects. By objects, we mean both hardware objects (CPU, memory segments, printers, disks and tape drives) and software objects (such as files, programs). Each object has a unique name that differentiates it from all other objects in the system and each can be accessed only through operations.

The operations that are possible may depend on the object. e.g:- CPU can only be executed on. Memory can be read & written, CD-ROM, DVD-ROM can only be read. Data files can be created, opened, read, written, closed and deleted; program files can be read, written, executed and deleted. The process should be allowed to access only those resources for which it has authorization.

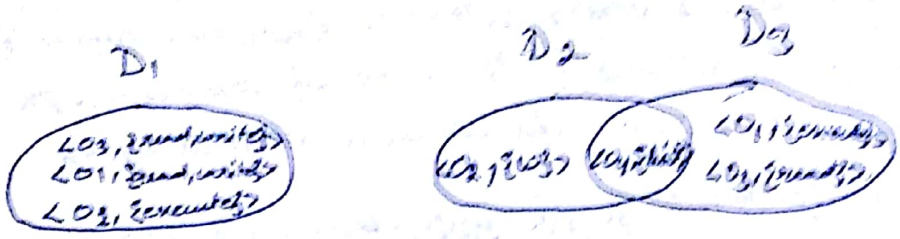
Domain structure :-

A process operates within a protection domain, which specifies the resources that the process may access. Each domain defns set of objects and types of operations that may be invoked on each object.

* The ability to execute an operation on object is an access right.
* A domain is a collection of access rights, each of which is an ordered pair <object-name, rights-set>.

e.g:- If domain D has the access right <file F, {read, write}> then a process executing in domain D can both read and write file F; it cannot perform any other operation on that object.

* Domains may share access rights.



e.g:- We have three domains: D_1, D_2 and D_3. The access right $\langle O_4, \{print\} \rangle$ is shared by both D_2 and D_3, implying that a process executing in either of these two domains can print object O_4. A process must be executing in domain D_1 to read and write object O_1. On the other hand, only processes in domain D_3 may execute object O_1.

Access Matrix :-

* The protection can be viewed as a matrix, called an access matrix.
* The rows of the access matrix represent domains and the columns represent objects.
* There are four domains and four objects, three files (F_1, F_2, F_3) and one printer. When a process executes in Domain D_1 it can read files F_1 and F_3. A process executing in Domain D_4 has the same privileges as it does in domain D_1 but in addition, it can also write onto files F_1 and F_3. The printer can be accessed only by a process executing in domain D_2.

Domain \ object	F_1	F_2	F_3	Printer
D_1	read		read	
D_2				Print
D_3		read	execute	
D_4	read write		read write	

Access Matrix

process should be able to switch from one domain to another. Domain switching from domain D_i to domain D_j is allowed to occur if and only if the access right switch $\in$ access(i, j)

In the following fig. a process executing in Domain D_2 can switch to domain D_3 or D_4. A process in domain D_4 can switch to D_1 and a process in domain D_1 can switch to D_2.

object / domain	F1	F2	F3	Printer	D1	D2	D3	D4
D1	read		read			switch		
D2				Print			switch	switch
D3		read	execute					
D4	read write		read write		switch			

* The contents of the access matrix entries requires three additional operations: copy, owner and control.

* The ability to copy an access right from one domain of the access matrix to another is denoted by asterisk(*) appended to the access right. The copy right allows the copying of the access right only within the column for which the right is defined.
e.g.:

object / domain	F1	F2	F3
D1	Ex		write*
D2	Ex	read*	Ex
D3	Ex		

(a)

object / Domain	F1	F2	F3
D1	Ex		write*
D2	Ex	read*	Ex.
D3	Ex	read	

(b)

Access Matrix with copy right

This Scheme has two variants :

1. A right is copied from access(i,j) to access(k,j); it is removed from the access(i,j); this action is a transfer of right, rather than a copy.
2. Propagation of the copy right may be limited. When the right R^* is copied from access(i,j) to access(k,j), only the right R is created. A process executing in domain D_K cannot further copy the right R.

* We also need a mechanism to allow addition of new rights and removal of some rights.
* The owner right controls these operations. If access(i,j) includes the owner right, then a process executing in domain D_i can add and remove any right in any entry in column j.

e.g. :-

Domain \ Obj	F1	F2	F3
D1	owner execute		write
D2		read* owner	read* owner write*
D3	execute		

(a)

Domain \ Obj	F1	F2	F3
D1	owner execute		
D2		owner read* write*	read* owner write*
		write	write

(b)

Access Matrix with owner right.

* Domain D_1 is the owner of F1 and can add and delete any valid right in column F1. Ily domain D2 is the owner of F2 and F3 and can add and remove any valid right within these two columns.

copy and owner right allow a process to change entries in a column.

* A mechanism is also needed to change the entries in a row. The control right is applicable only to domain objects. If access(i, j) includes the control right then a process executing in domain Di can remove any access right from row j.

e.g.:- We include the control right in access(D2, D4). The a process executing in D2 could modify domain D4

Domain \ object	F1	F2	F3	Printer	D1	D2	D3	D4
D1	read		read			switch		
D2				Print			switch	switch control
D3		read	execute					
D4	write		write		switch			