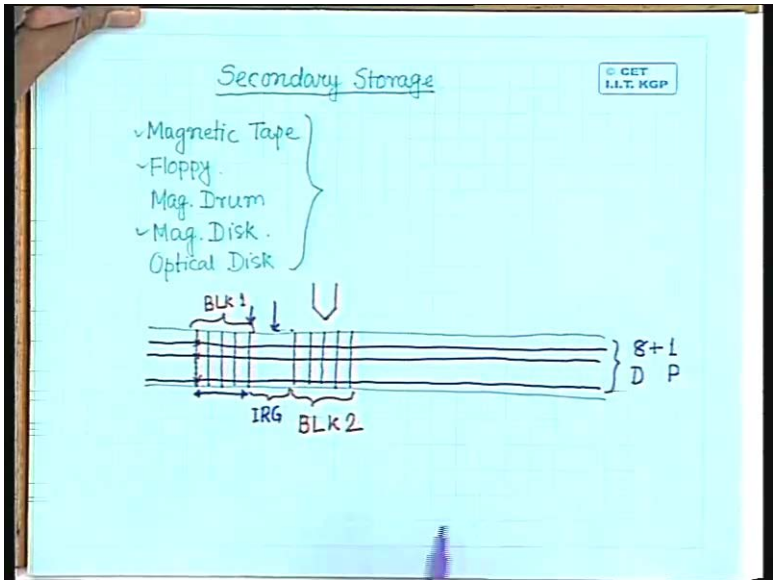
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic & Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture No. # 24
Secondary Storage Organization – I

Till the last class we have discussed about the buffer cache management and in that case we have seen that whenever a process puts a request for a block or the data contained in a particular block then it will be the responsibility of the operating system to check in the buffer pool that whether that particular block is present in the buffer pool or not. If it is present in the buffer pool then it can be directly copied into a user space, user area from the buffer pool but in case it is not available in the buffer pool, in that case the block containing the requested data has to be read from the secondary storage and put into one of the buffer pools. Then from that buffer it can be subsequently transferred to user space, the space which is allocated to the particular application program which has requested for the data. Now before you go further, let us see what is meant by a block.

(Refer Slide Time: 00:02:02 min)



For that we discuss about the secondary storage devices and you know that we can have different kinds of secondary devices for example we can have a storage device like magnetic tape, we can have a storage drive of say floppy. Of course these are all magnetic materials, we can have magnetic drum, we can have magnetic disk, the hard disk that we use is nothing but a magnetic disk and we also have what is optical disk or compact disk. So you can have this various kinds of secondary storage devices and whenever we want to access any data from any of the secondary storage devices, we have to access the data in the form of a block. Either we have to write an entire block on to the secondary storage or we have read a complete block on the secondary storage. From the secondary storage we cannot access any particular byte or any particular character.

Now the typical characteristics we will mainly discuss about the magnetic tape, floppy and magnetic disk. Now as you know that this magnetic tape is nothing but a tape something like a video tape or an audio tape which is coated with some magnetic material. So I can simply put it in the form of a linear tape like this and this tape will consist of a number of tracks, parallel tracks which are parallel to the length of the tape, so it is something like this. So these are a number of parallel tracks and these are the tracks which contains the data. Typically we have 8 tracks for containing the data bits and one track which contains the parity bit, parity to take the correctness of the data. So 8 bits use 8 tracks containing the data bits and this additional track contains the parity bit. Then to access any data which is stored on this magnetic tape or to write a data on this magnetic tape what we use is a read write head and when you say that read write head for the magnetic tape, it is also a linear array because for every track I have to have a read write head. So for a magnetic tape which has got such 9 magnetic tracks, 9 tracks, the read write head will be an array of 9 read write heads. So for reading or writing anything on this magnetic tape, the driving mechanism will pause this magnetic tape over this read write head.

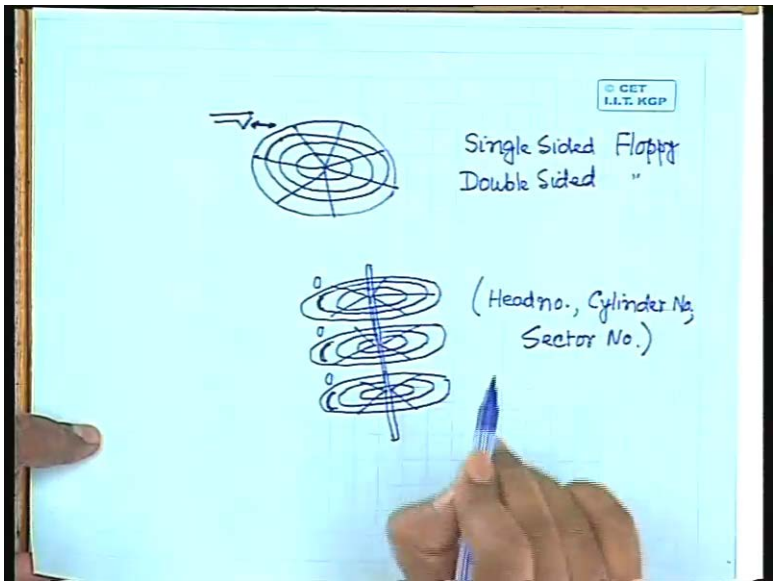
So when this magnetic tape moves over this read write head, either the data can be written on to this different tracks or the data can be read from this tracks. Now usually the way the data is stored on this track is in the form of blocks. A block consists of a number of records. So it is stored like this say every byte will be stored in perpendicular direction, this location will contain one bit of a particular byte, this location will contain another bit of a particular and so on. So when you put a number of records on this magnetic tape, a number of records form a particular block. Suppose this is block number one. Similarly we have another set of records that form say block number 2 in between two blocks, we have to have an empty space which is known as inter record gap or IRG.

The purpose of maintaining this IRG is like this that a data can be written onto this magnetic tape or data can be accessed from the magnetic tape only when the magnetic tape moves at a constant velocity under this read write head and this being an electromechanical device, it will have certain inertia. So if we assume that suppose initially the magnetic tape is halted that means tape is not moving. Now I want to read some data, read a particular block from this magnetic tape may be the first data that I want to read is block number one and the next block that I will be interested to read is block number 2. So what happens is once the read request comes, the driving mechanisms start moving this tape and because of inertia it will take a finite amount of time to attain that particular speed, the specified speed and only when the specified speed is attained then the read write head can start reading from the magnetic tape or it can start writing onto the magnetic tape and the constant velocity must be maintained until and unless the last byte of the block is read. So during this time taken to pass this part of the magnetic tape under read write head, the tape speed must be remaining constant.

Now once reading this block number one is complete, the tape will come to halt and again because of inertia it will have certain deceleration and may be when the tape comes to halt in that case the read write head will be positioned somewhere here. The next request comes is for reading block number two. Again it will take some time to attain the specified speed before which the read write head can read from the tape. So we have to allow certain space so that during the time when this space is passed over the read write head, by that time the tape will attain the constant speed.

So this read write head will be ready to read the next block from the tape and this is the purpose of maintaining this inter record gap and obviously the length of the inter record gap will be depending upon what is the acceleration and what is the deceleration that is provided by the magnetic tape drive. So if the acceleration is more or the deceleration is more, may be this inter record gap, length of the inter record gap will be less whereas if the acceleration or deceleration is less in that case the length of the inter record gap has to be more depending upon what is the specific velocity at which the tape has to move under the read write head.

(Refer Slide Time: 00: 09:37 min)



Now similarly coming to the floppy or the magnetic disk, as you know that floppy is nothing but a circular plate again coated with some magnetic material and on this tape, the data tracks are nothing but a set of concentric circles, so I have a set of concentric circles which stores the data. Now these concentric circles are again divided radially into a number of sectors, so like this. Again to read the data or write data on to any of the tracks, I have to have a read write head. Now in case of magnetic tape the read write head was stationary, it was the tape which was moving under the read write head.

Now in case of a magnetic disk like a floppy disk, this head has to access different tracks. So to access different tracks which are placed in a concentric manner, the read write head has to move radially over this disk and when the read write head moves over the disk, there is only a gap of few microns between the track and the read write head. So given a particular track number from which the data has to be read, the first operation that is to be done is you have to place the read write head over the track and once this placed onto the track, you have to identify the sector which is to be read. Now it is this sector which is called a block in case of a disk. So this sector will contain a number of data, every sector will have an unique address and we refer this sector by a block. So every block on a magnetic disk like this will have a two component address. The first component will identify that what is the track number, track number is same for every circle and the second component is what is the sector number. So the track number and the sector number taken together identifies a particular block of the disk.

Now here we can have these tracks, data tracks either on single surface, one surface of the plate of the magnetic disk or it can be on both the surfaces of the magnetic disk. So accordingly we can have either a single sided floppy drive or we can have a double sided floppy drive. Now in case of a magnetic disk, the disk is nothing but a stack of such magnetic plates. So what will happen is if I place such magnetic plates, circular plates one above the other like this and fix them by a spindle along the axis, the tracks as before will be concentric circles on these plates may be on same side, may be on single side or both sides, like this. Now these tracks are so placed that track number 0 on one disk is at the same special location as track number 0 of the other disk. So all these tracks, concentric tracks having the same number on different disks taken together is called a cylinder.

Now when I specify a cylinder number, it specifies this track, it specifies this track, it also specifies this track. So as you have done in this case that the tracks are divided radially into a number of sectors, here also every track will be divided radially into a number of sectors. The tracks can be on both sides of the plate, they can be on a single side of the plate. Accordingly the number of read write heads will be different because here if I have tracks on both sides of the plate, then I have to have two read write heads one for accessing the top surface, the other one for accessing the bottom surface.

Similarly here if I have three such disks which are stacked together, three such plates stacked together to form a magnetic disk and I have tracks, magnetic tracks, data tracks on one side of the disk, one side of every disk then I need three read write heads. Whereas if the tracks are present on both sides of the plates in that case I need 6 read write heads, every head will have an unique head number. So just in this case we have said that if we have a single sided disk then for identification of a particular block or particular sector, we need the track number and the sector number. Had it been a double sided disk, we need track number, sector number along with the side and the side is referred as head number that which particular head has to be active for reading that particular sector. So I will have three component address if it is a double sided disk, one for the head number which has to be active, the other one is for the track number and the other one is for the sector number.

Similarly in this case, the address will be three component address, head number then as I said that all the circles, all these circles, concentric tracks having the same number is called a cylinder. So the next component that I need is what is the cylinder number and then the sector number. So these three components taken together uniquely identifies a particular sector or a particular block on the disk. So as we said that whenever a page fault occurs and the process wants to read some block from the disk that means it has to supply what is the address of that block and the address has to come in the form of this three component address.

Now generally the process does not know what is the address of the block which is needed. So you might be remembering that in case of when we discussed about the page fault interrupt, we have said cylinder number is say all these stacks at the same special location, at the same distance from the center, on different disks they form a particular cylinder. When I say cylinder number 0, this is the track number 0 on this disk, this is track number 0 on this disk, this is track 0 on this disk, so all these three tracks taken together is called a cylinder.

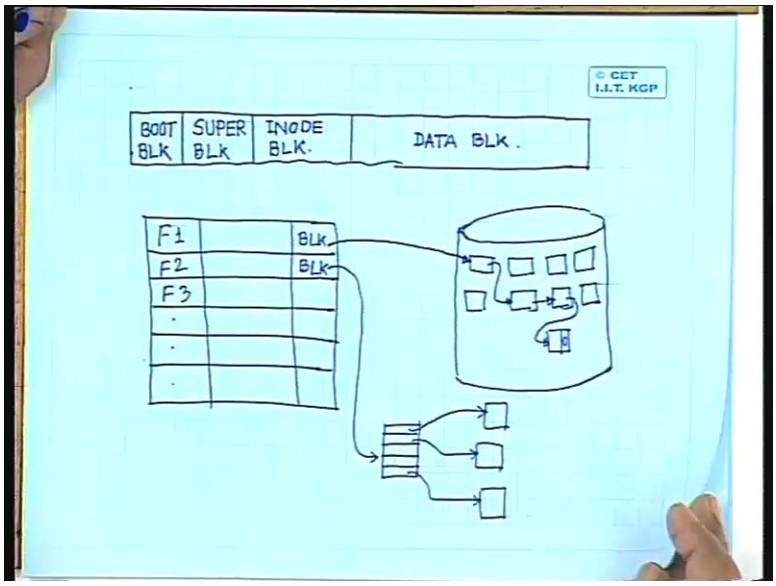
Now you see that how these three component address uniquely identifies a particular sector or a particular block. First what I have to do is I have to specify the head number, maybe I specify head number to be three. Now a read write head which accesses the top surface of the top plate that will be having an head number zero, a head which accesses the bottom surface of the top plate that will be having an head number 1, similarly here head number 2, head number 3. So head number 3 is the head which accesses the bottom surface of the second plate. So I identify the surface on which I have to find that particular block. So once I identify the surface then I have to identify the track on which the block is present. So in this case the track number is identical with the cylinder number because for all the tracks at the same radial distance from the center, the track number is same and all of them taken together is called a circle is called a cylinder, so track number and cylinder number they are identical.

So first I have identified the surface then I am interested in say zeroth track on that surface and that is same as the cylinder number. So once I identify the track, so now I come to a particular track, on that particular track I am interested in a particular sector, so these three taken together uniquely identifies a particular block on the disk because as I said that this sector is equivalent to a block. Now when a process puts a request for a particular data for getting the data, the block address or the block number containing the data must be known but the process application does not know it. So when we discussed about the page fault interrupt, we have said that whenever a page fault interrupt occurs, the operating system or the kernel refers a particular table which we have said as file map table, FMT so it is the FMT which will indicate that what is the address of the block which is needed.

So on getting that block address, now it will invoke the device driver to read the block from the secondary storage and put it into the buffer, if the buffer already does not contain the data. So this FMT is an interface between the application program and the actual hard disk, the physical hard disk or any physical secondary storage because all this information has to come from FMT. In some systems it is called file map table and the particular one that we will discuss in details is called an index node or inode which is used in UNIX operating system.

Now this FMT can be maintained, file map table can be maintained in various ways so before I go into that let us see what is a disk layout? So as we have said that the disk consists of a number of blocks, every block is identified by a three component address that is head number, cylinder number and sector number. So if I put all those blocks in a sequential manner say head number 0, cylinder number 0, sector number 0 is the block number 0, head number 0 cylinder number 1 say cylinder number 0, block number 1, sector number 1 is the block number 1.

(Refer Slide Time: 00:21:17 min)



So if put these blocks in a linear form, in an linear arrangement then what I will have is on a hard disk, I will have a series of blocks which logically can be considered as a linear array of blocks. Few of these blocks are special blocks for example the zeroth block or say first few blocks starting from the zeroth block they are known as boot blocks. These boot blocks contain the boot program that means whenever you switch on the power or you reset the machine, a part of the operating system which is stored in the boot block is read from the boot block and put into the OS part, OS partition of the main memory and that becomes resident. So this boot block is used to store the boot volume or the booting program or initialization program of the computer system. Next few blocks one or more blocks are called super blocks.

Now this is a layout which I am taking with reference to unique file system layout. So the purpose of this super block is to maintain the information of the file system itself that is how big the file system is, how many files it can store, where you will find the free space on the disk, so all those information's will be maintained in this super block. I will come to details on this super block later.

The next few blocks are called inode blocks. as I said that a part of the inode contains what we refer to as FMT or file map table. These are called inode blocks and the remaining blocks are actually the data block, which contain the file data. Now when I talk about the FMT or file map table, the FMT can be maintained in various ways. See one of the way can be that on this file system, I maintain a table common for the entire file system. What the table will do? The table will have a list of the files or list of the directories which are present on this file system. So it can be something like this. I have a table and in this table I store a list of files, so I can have files say F_1 F_2 F_3 and so on. With every file I can store a block address or a block number which contains the initial data or the first block containing the data of that file, so here I can put the block address. So for every file I will have a block address and this is the address of the block which contains the starting data of the file. Now when you come to the disk as I said the disk is nothing but consist of a number blocks.

So it will consist of a number of blocks and these are the blocks which contain the file data. So one way to keep track of what are the blocks which contain the data of say file F_1, I can go for different kinds of block allocation. One kind of block allocation can be contiguous block allocation that means if F_1 needs 5 blocks to contain its data then this block number, it gives the starting block number and 5 consecutive blocks starting from that block number contains the data of file F_1. So that can be one kind of allocation and this is an allocation which is possible if I don't have any dynamic situation that is once I create a file that will remain as it is but in case of a dynamic situation, it may so happen that suppose I had a file of say 3 kilo bytes earlier, the 3 kilo bytes may be taking 3 blocks assuming that every block is 1 kilo byte size then later on, it may so happen that I wanted to expand the file, modify the file so that the size will be changing from say 3 kilo bytes to 4 bytes.

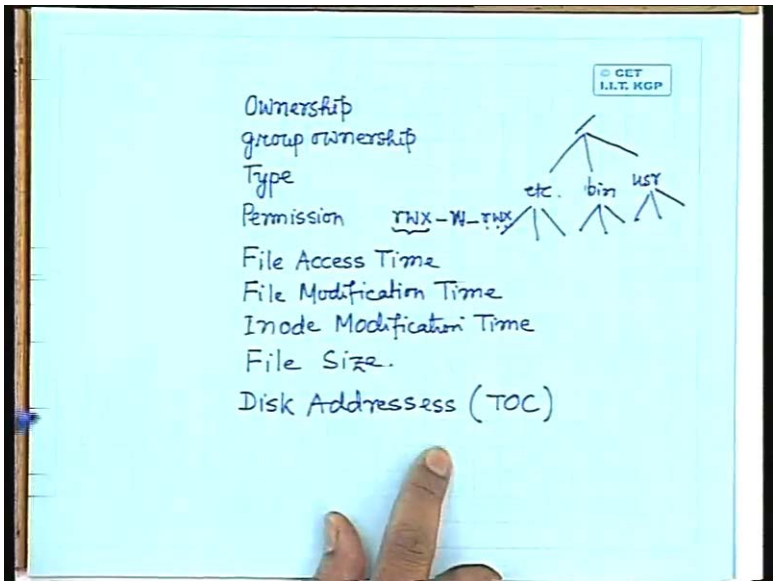
So earlier the file was needing 3 blocks, now it will need 4 blocks but once three block file was created, 3 kilo byte file was created after that there might have been some more files created. So the block which is next to that file has been allocated to some other file. Now if I wanted to change the size of the file, if I want to increase the size of the file to 4 kilo bytes so it needs 4 blocks I find that the next block, the fourth block in the sequence is no more available. So I cannot change the size of the file. So to avoid that problem what is done is, what can be done is even these blocks can be maintained in a form of a link list or I will have a chain of blocks may not be contiguous which will contain the file data.

So in such case what will happen is this block number which is there in the file allocation table or the file map table, it points to the first block containing the file data. Then address of the next block which contains the same file data will be say put as a pointer in this first block itself. So this first block can have a pointer which points to the next block containing the file data. Again this can have a pointer which points to the next block containing the file data. This can also have a pointer which points to the next block in the file data, containing the file data and the last block will have a null pointer. So you find that even this chained or linked list concept can also be used for storing the file data. So now whenever a page fault interrupt occurs and the process needs a particular block, always it has to access the first block then from the first block it gets the link to other blocks. Then finally it will come to the desired block and treat the data but the disadvantage with this process which is the disadvantage of all linked list implementation is that access is fully sequential. I cannot go to say these block directly to read this particular block. I always have to come to first block then only sequentially I have to come to the desired block.

Now disadvantages can be avoided if I modify this kind of file allocation that is in a second case what I will have is say this block number which is specified within the file allocation table, this points to a particular block. Now this block instead of becoming the first block in the chain of blocks containing the file data, these blocks can contain a table. So now the concept is something like this say I come to a particular block, this block contains a table or table of pointers. These pointers point to different blocks which contain the file data so this block number points to a particular block, these block actually contains a table of block numbers or list of block numbers and these blocks pointed to by this table entries actually contain the file data. So now we find that unlike in the previous case where this access was strictly sequential, here I can have random access but the first thing that I have to access is this table. So once I have an access to this table then following any arbitrary pointer, any random pointer I can come to random block to access

the data contained in that block. So the disadvantage that we had in this case has been removed in this particular case. And it is a modification of this concept, the second concept which is used in case of UNIX operating system and there the file map table or file allocation table is contained in what is called an inode or index node. Now coming to what is an inode, for a particular file system every directory or every file in that file system will have an unique inode allocated to it. An inode is nothing but a data structure which contains along with the file map table various other information of the file.

(Refer Slide Time: 00:31:43 min)



So what are the different information's that you need? The first information that is needed is the information regarding ownership of the file that is who is the owner of that file, the user name. The second information that is needed is group ownership. Those who are working on UNIX operating system, you might be knowing that whenever an user account is created the user belongs to a particular group. So for every file or every directory, we have to have these two information's that who is the ownership, what is the ownership of the file, who is owner of the file and the group to which that owner belongs. The other information that is to be maintained in the inode is the type because as we said that for every file or every directory on the file system, I will have an inode.

So I have to have some indication about that these inode is the inode of a file or inode of a directory because the files will have some characteristics, directories are having some other characteristics and we can say that directories are nothing but special kind of files which gives the file system a tree like structure. Is it not? Because in case of this file system we usually have a root directory. Under root directory we can several subdirectories say for example etc, we can have a subdirectory bin, we can have a subdirectory usr and so on. Under this subdirectory we can have again a number of subdirectories or number of files. Here also we can have again a number of subdirectories or again a number of files.

So each of these nodes are basically directories and this directory is nothing but a special kind of file which gives the file system a tree like structure like this. So I have to identify, I have to have a demarcation about that the inode that I am talking about whether this inode is the inode of a file or it is the inode of a directory. Not only that the type field also contains information about whether this inode is already allocated or inode is not yet allocated. Among the other information we have to have the permission fields and if you give the directory listing, simply by LS comment you find that you get this kind of directory, something like this. What does it mean? So this permission is divided into three fields, I can three kinds of permissions on any directory or any file Unix operating system r indicates read permission, w indicates side permission, x indicates execute permission.

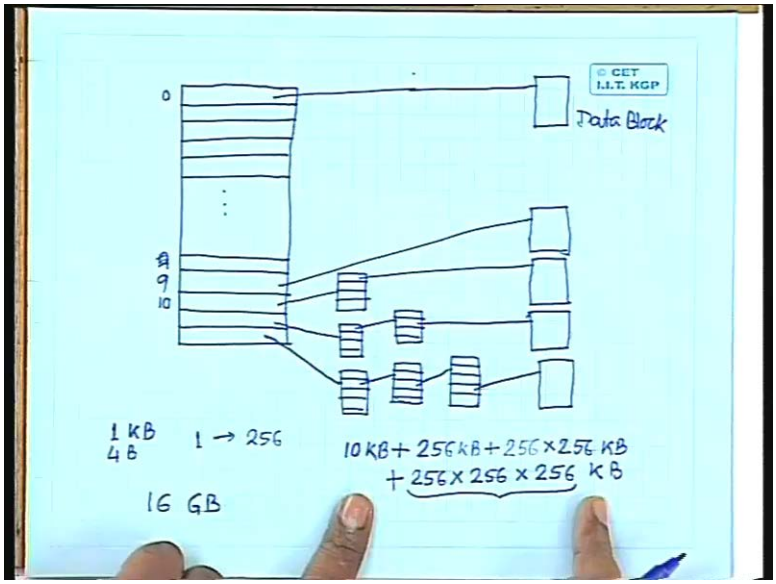
Now first three bits indicates that what is the permission given to the owner of the file, second three bits again it can be rwx, read write and execute it indicates that what is the permission that is given to any member of the group to which the owner belongs. So this indicates the group permission and the third three fields, this read write and execute this indicates the permission given to any other user of the system. The user may belong to some other group. Of course except the super user because super user will have all the permissions read write execute. So this is the permission which is given to the owner, this is the permission given to any other user belonging to the same group to which the owner belongs and this three bits, this three fields indicates the permissions given to any other user of the system who does not belong to the same group.

Now among other information's we have to have information about file access time that is when the file was accessed last, we have to have information about file modification time that is when the file was modified last and the other information is inode modification time that is when the inode was modified last. Then another crucial information that is of course needed is file size, what is the size of the file or the size of the directory, how many bytes are needed for this file or how many bytes are needed to store this directory and the third one that comes is what we are talking about the file map table or file allocation table that is the disk addresses of different data blocks containing the file data and this is what is normally referred to as TOC or table of content. So this is a data structure, TOC table of content, this last field this actually contains the addresses of the disk blocks or the numbers of the disk blocks which contains the file data that is like this.

See here what we have said is I can have a global table, in the global table I have a block field. The block field points to a particular block which contains the table of disk addresses or block addresses, those blocks contain the file data. In case of Unix operating system, this table is part of the inode. So for every file or every directory in the file system, an inode is allocated, the inode has this fixed structure. One of the fields in the inode is this table which contains the list of block addresses which contain the file data and this table is called the table of contents or TOC. So now we find that because the inode has got fixed number of fields and every field is constant, I mean not constant of fixed length. This ownership field it contains a fixed length, group ownership field contains a fixed length, type field contains is of fixed length, permission field is again of fixed length. So every field in the inode, inode data structure is of fixed length so that means the number of bytes needed to store the inode is also fixed. So since the number of bytes needed to store the inodes is also fixed, the simple way we can store this inodes on the inode blocks is just by sequence of bytes.

So if I know that suppose I need 50 bytes to store all these fields. Then first 50 bytes in the inode blocks will be the first inode, second fixed 50 bytes will belong to the second inode, next 50 bytes will belong to the third inode and so on. So all these inodes can simply be put in the form of series of bytes on the disk, a particular inode will be accessed by its byte offset. Disk address field is also of fixed type. That is taken care of by the TOC table of contents. Table of contents is a table which contains the block addresses containing the file. We will see how big it can be, I will see that later. Disk contents, these different entries in this table of content has different meaning, let me talk about that.

(Refer Slide Time: 00:40:57 min)



So the format of this TOC table of table of content is something like this, in Unix file 5 system five the TOC has got 13 entries, so there are 13 entries. The first 10 entries starting from entry number zero to entry number 9, they contain the block number of the block which contains file data. So there is a 9, so these are direct pointers to blocks which contain the file data, so this is the data block.

Similarly this also points to a block which contains the file data, this is also a data block. Entry number 10 is a single indirect block or single indirect pointer that means it points to a block, this block itself contains a number of block addresses. In the first case in case of direct entries, each of these entries are block numbers, these blocks contain the file data. This entry again contains a block number, it points to a block. Now this block does not contain the file data, this block again contains a set of pointers, a number of pointers or block addresses each of these blocks so each of these entries points to a block which contains the file data. This entry is a second indirect entry that means these points to a block which contains a sequence of block numbers each of these points to a block which again contains a sequence of block numbers, each of these points to a block which contains the file data. This is third indirect. This point to a block which contains a set of block numbers, each of these points to a block which in turn contains a set of block numbers, each of them points to a block which in turn contains a number of block numbers, each

of them points to a block which contains the file data. So this is the structure of the inode table of contents in case of Unix system five.

Now we find that if I wanted to find out that what is the file size that can accessed by such an inode table of contents, let us assume that every block is of size 1 kilo byte. And suppose for addressing a particular block, I need 4 bytes. Every block is of 1 kilo byte, for addressing a particular block I need 4 bytes that means one block can contain 256 number of block numbers. Now assuming this you will find that using this direct pointers I have 10 direct pointers, entry number 0 to entry 9 they directly point to data blocks. So using this direct pointers, I can access a file of size 10 kilo bytes. So this is what I can access using direct pointers. Using single indirect pointer, I come to a block which contains 256 entries or 256 block numbers and these entries point to blocks which contain the file data that means using this 256 entries I can access 256 kilo bytes of file data.

So using the direct pointers plus single indirect pointer, I can access 10 kilo bytes plus 256 kilo bytes of file data. Then come to double interact, you find that this is 256 into 256 kilo bytes of file data plus using triple interact, it will be 256 into 256 into 256 kilo bytes of file data. So this is the total size of a file that I can access using this structure of inode. Now if you compute this, this is nothing but 2 to the power 24, 2 to the power 24 means 16 megabytes so 16 mega kilo bytes that means 16 gigabytes, so just this triple indirect block gives you a file size of 16 gigabytes. So a size of file which can be accessed by this inode structure is more than 16 gigabytes and I think this is sufficient for most of the applications that we can think of. So let us take a break after that we will continue.