Digital Computer Organization

Prof. P. K. Biswas

Department of Electronic & Electrical Communication Engineering

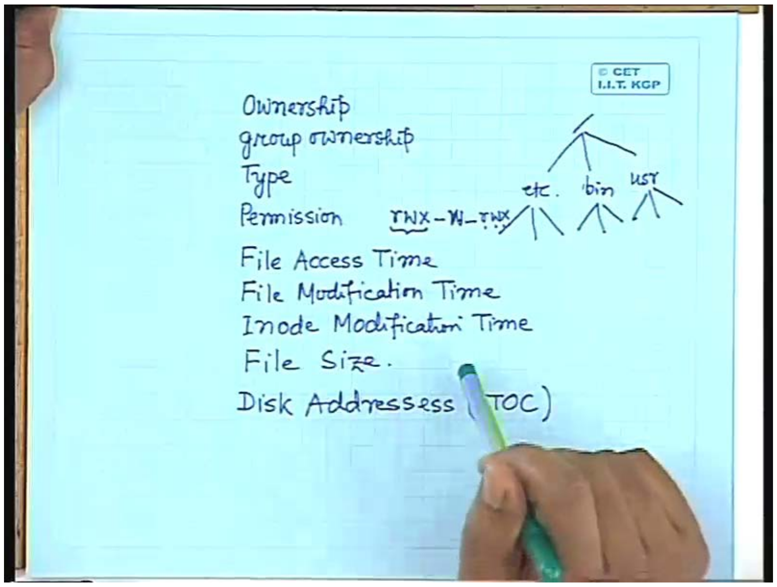
Indian Institute of Technology, Kharagpur

Lecture No. # 25

Secondary Storage Organization- II

So, what we have discussed about the fields of an inode? These are the fields which are stored on the secondary stores. So following the same logic, why you have come to this inode is that whenever a process puts a request for a particular data from a particular file, the first thing that has to be done is you have to access the inode of the file. Then in the inode, you have to come to the table of content TOC part of the inode. From the TOC part depending upon which byte offset of the file is interested in, you have to go to the corresponding data block may be either following the direct pointer or single indirect pointer or double indirect pointer or triple indirect pointer, using some of the pointers one of the pointers you have to come to a particular data block then you have to access that data block.

(Refer Slide Time: 00:01:03 min)

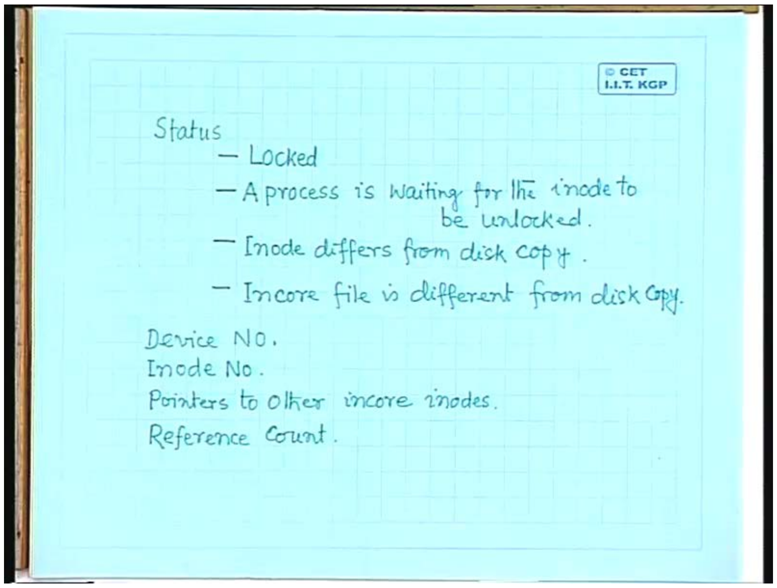

© CET I.I.T. KGP
etc.
bin
usr

Ownership
group ownership
Type
Permission rwx-w-rwx
File Access Time
File Modification Time
Inode Modification Time
File Size.
Disk Addressess (TOC)

So for every read operation for every write operation on the file, you have to access the TOC part of the inode to get the corresponding block identification number. Then once you have the block identification number then you can check within the buffer whether that particular block is present in the buffer or not. If it is present in the buffer you don't have any problem, you can simply transfer that from the buffer area to the user area. If it is not present in the buffer then I have to physically read that particular block from the secondary storage and put it into one of the buffer cache. Now because you have to access the inode for every file operation whether it is read or write whatever it is that means once you open a file, it is quite expected that the inode of that file will be accessed a number of times. Now for accessing the inode, if every time I have to go to the disk then obviously the throughput of the file system will be very poor.

So to improve the throughput what is done is the same concept that we have used for the buffer cache that is we put the data block in the buffer cache so that your physical access to the disk, the frequency of physical access to the disk is reduced. Using the same concept whenever you open a file, the first time you access the inode of that file from the disk and copy that inode into main memory. So that as long as the file is open, every reference to the file will try to refer to the inode of the file and now the inode is available in the main memory so your access time will be quite less. So this is a copy of the inode in the main memory which is called an incore inode and the incore inode is maintained almost in the same way as we have maintained the buffer cache that is we will have a free incore inode list. We will also have a number of hash lists, the hash lists consist of a number of inodes on that particular hash list. So to maintain the hash node in the main memory in addition to these fields we need a number of additional fields, one of the field is obviously the status field that we have also used in case of buffer cache.

(Refer Slide Time: 00:04:14 min)



© CET I.I.T. KGP

Status
- Locked
- A process is waiting for the inode to be unlocked.
- Inode differs from disk copy.
- Incore file is different from disk Copy.

Device NO.
Inode No.
Pointers to other incore inodes.
Reference Count.

So the first field in the incore copy that will be the status field and as before the status field will contain similar kind of information that is the first information that has to be contained in the status field is whether the inode is locked because it may so happen that more than one process opens the same file simultaneously for reading purpose. In that case when one process say p_1 wants to read a data from that file then before reading the data from the file, it has to access the inode to find out what is the block number of the data that is requested. Once it gets that block number then it can release the inode. So during this process the inode should remain locked, so that during the same time if the second process also wants to access the same inode it should not permit it.

Once the first process releases the inode, the second process can now access the inode. So one of the field, one of the information that you have to maintain in the status field is whether the inode is locked or inode is unlocked. Here as we have seen there is a difference that as we have seen in the buffer cache that if the buffer is not locked that means it is free but in case of inode it is not so.

Even if the inode is not locked that does not mean that inode is free, I will come to that later. So the first information that you have to maintain in the inode is whether the inode is locked then the next information that you have to put in the inode is similar to as you have used in case of buffer cache that whether a process is waiting for this inode to become unlocked or not, inode to be unlocked. Then other information that you have to maintain is whether the inode is same as the disk copy or there has been some modification in the inode. So this information is whether inode differs from disk copy and the other information is, one is inode differs from the disk copy and the second information that has to be maintained is whether the file data is different or the file data has been modified after the file has been opened. So here we have to put whether incore file is different from disk copy, so these are the various information that you have to maintain in the status field of the incore copy of the inode. Then obviously the other information's will be the device number and the device number that contains the file that we are interested in.

Then the inode number itself, you find that in case of disk copy of the inode we didn't need this inode number because in that case the position of the data structure in the inode blocks tells you that what is the number of the inode. So just for example what you said is that if the inode needs a 50 bytes for representation, if the data structure for inode needs 50 bytes in that case first 50 bytes in the inode block will belong to the first inode. Next 50 bytes will be for the second inode, next 50 bytes will be for the third inode and so on. So it is the position of the data structure in the inode blocks tells you that what is the inode number but incase of incore copy, we may not need all the inodes in sequence and a particular inode may appear anywhere in the hash list or anywhere in the free list.

So I need a specific field which will tell me that what is the number of the inode of that incore copy whether it is the inode number 1 or inode number 2 and so on which is not needed in case of disk copy. Then obviously I have to have a set of pointers, pointers to other incore inodes and here again we can have two sets of pointers, one set of pointers will be for the free inode list and other set of pointers will be for pointers to other inodes in the same hash queue, just as we have done in case of buffer cache. And here I have to have an additional field which is not present in case of buffer cache that is what is called reference count and it is the reference count which makes it different from unlocked in case of buffer and unlocked in case of inode.

In case of buffer cache we have said that whenever the buffer is unlocked that means the buffer is free and it can be on the free list as well in addition to its existence on the hash queue. In case of inode because there is the possibility that the same file may be opened simultaneously by more than one process and we expect that as long as the file is open, its inode should be present in the main memory. I can remove the inode from the main memory only when the file is closed but it may so happen that two processes has opened a particular file.

The first process which opened that file first and because of that file open command, the corresponding inode has been read from the secondary storage from the inode block and it has made an incore copy of time. Now the second process wants to open the same file, so it will get the inode number of the file and after getting the inode number the first operation that will be done is to check the incore copies of the inodes in the corresponding in the particular hash queue to see whether the inode already exists in the main memory or not. If it is existing in the main memory that means some other process had already opened it.

So I need not get the same inode from the disk once more because already it is present in the main memory but what has to be done is this difference count field has to be incremented by one. Now when the first process closes the file but not the second process, the file is closed for the first process but it is still open for the second process. That means though the first process will not need the data from the file anymore but the second process will need the data from the file. So first process has unlocked the inode, may be the second process has also unlocked the inode because inode will be needed only to get the block address not for other purpose. So may be the inode is unlocked for both the processes, first process and second process but the second process has not closed the file but the first process has closed the file that means file is still open for the second process and the second process is lightly to access the data from the same file a number of times. So I cannot simply remove this inode from the incore copy, I can remove the inode from the incore copy only when the second process also closes the file so that is what is maintained in this reference count field.

When the first process opens the file, the inode corresponding to that file is read from the secondary storage and put into main memory and the reference count field is set to one. When the second process also tries to open the file, it finds that the inode is already present in the main memory so it need not be read from the secondary storage but now there are two instances, two active instances of the same inode, so reference count will be incremented by one more so now reference count becomes equal to 2. So it is the reference count field which indicates that how many active instances of the same file exists in the system.

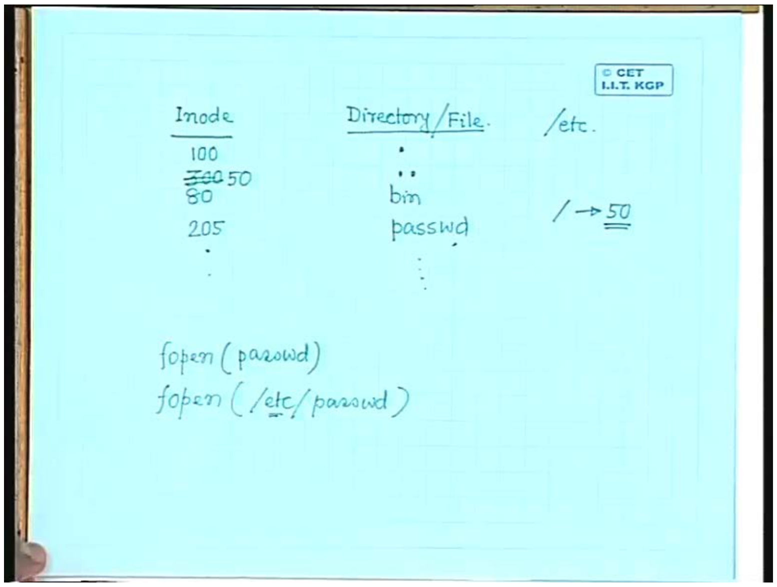
Now when the first process closes the file, it simply makes the reference count decremented by one. So earlier reference count field was equal to 2, now it becomes equal to one because it is one that indicates the file is still active for some other process. So I cannot simply remove the inode from the main memory or I cannot put this inode on the free inode list because once it goes to free inode list that indicates that this inode may be overwritten by another inode from the disk. I can put this onto the free inode list only when the reference count field becomes equal to zero that means there is no other active instance of the same file. So that can only happen only when the second process also closes the file. So after the first process, if the second process closes the file then earlier reference count field was equal to one, now it becomes equal to zero and when the reference count field becomes equal to zero, it can be put onto the free list. So that is the difference between your buffer cache management and the inode management.

In case of buffer cache we have said that whenever the buffer is unlocked, I can put it onto the free list, may be either head of the free list or at the tail of the free list but in case of inode only when the reference count field is equal to 0 even if the inode is unlocked. Only when the reference count field is equal to zero then only I can put onto the free inode list otherwise I cannot put it onto the free inode list. Other managements will be almost similar to what we have discussed in case of buffer cache.

Now the major operation is as we have seen that for getting any data from the inode, I have to access the inode, get the table of contents, from the table of contents I can get the block number and once I have the block number then using that block number I can check in the buffer cache whether it is present. If it is not present I have to access that block from the secondary storage, put it into buffer. But what is most important is that we have not said till now is how to know the inode number. Because when I execute a program or I want to open a file, I simply give the file name or at best I can give the path name starting from the root leading to that file. But I never specify the inode number or even the user need not know what is the inode number, so there comes the concept of directory. Directory plays a very very important role in finding out the inode number of a requested file or the inode number of a requested subject. Now what is a directory? As we said the directory is a special kind of file, so just like file, in case of files we have to have unique inode, every file will have to have an unique inode. Similarly every directory or every subject will have to have an unique inode. So I just demarcate between the inode given to a file or an inode given to a directory only through the type field. The type field indicates whether the inode is unallocated. I will not say free, inode is unallocated that means it has not been allocated to any of the files or any of the directories or the inode has been allocated to a file, regular file or the inode has been allocated to a directory or there are some special kinds of files which are maintained whether the inode has been allocated to any special kind of file. I will forget about this special kind of file now. Let us consider about only directory, file and the free inode.

So whenever a process wants to open a file, usually we give the file name, just the file name if the file exists in the current directory or we simply give the path name from the root, if the file is not in the current directory or even if it is in the current directory then also we can specify the path name from the root. Now what is the directory? As we said that directory is a special kind of file which contains a list of inode number file subject pair. So if you open a directory just like a file, you will find that it will contain two fields.

(Refer Slide Time: 00:17:39 min)


© CET I.I.T. KGP

Inode	Directory/File
100	.
~~300~~ 50	..
80	bin
205	passwd
.	⋮
.	

/etc.

$/ \rightarrow \underline{\underline{50}}$

fopen (passwd)

fopen (/etc/passwd)

One field is for inode and the other field is for directory or file. Suppose say this is the directory of say slash etc. Under slash etc we can have a number of sub directories or a number of file. For example under slash etc I can have a directory say bin, this directory bin, sub directory bin will have a specific inode number let us say inode number 80. Under this I can have a file say password, this password again will have an unique inode allocated to it and suppose this inode is let us assume any numbers say 205.

Now the sequence which particular inode will be allocated to a particular file or to a particular directory that depends upon what are the inodes unallocated at the time when this file or this directory has been created. So, one of the inodes which is still unallocated will be allocated to the newly created file or the newly created directory. So similarly it can have many other entries and the corresponding inodes. So a directory is nothing but a list of inode directory file pair along with this there may be other attributes just to identify that whether this is a file or this is a directory, of course that is also available in this particular inode. So 80th inode will say that bin is a directory.

Similarly 205th inode we will say that password is a file, it is not a directory and that will be specified in the type field of this particular inode. So just like this when I say that etc is a directory that etc itself has been allocated a particular inode, a unique inode which is not allocated to any other file or any other directory. Then along with this there are two special kind of entries, one entry is dot, the other entry is a dot dot. This entry dot means that it gives you the inode number of current directory. Suppose this number is 100 then as I said that this directory etc, this is the current directory is etc, the directory etc itself is assigned an inode number 100, dot dot gives you the inode number of the parent directory of this directory. So may be the parent directory of this has been allocated an inode number of 300. So you find that from a particular directory using dot, I can find out what is the inode number of the current directory or using this dot dot I can identify, I can find out what is the inode number of its parent directory.

For any other entries whether it is a file or a directory, I can always find out what is the inode number of the corresponding file or the inode number of the corresponding sub directory under the current directory. Now in addition to these directories there is a root inode, there is a root directory for every file system. Root directory is also assigned an unique inode number. Suppose the root directory is assigned an inode number let us say any inode which is not allocated to this say 50. So for getting the inode number of any file or any directory that is specified I need one of the two components, either the inode number of the current directory or the inode number of the root directory because when you specify a path, the paths are specified from the root directory itself or I simply specify the file name or the file directory which is under the current directory. So, to get the inode number of the target file or the target directory if I have either the root directory inode number, or the current directory inode number that is sufficient.

The root directory inode number is usually stored in a global variable which is accessible to all the process but this cannot be modified. Similarly current directory inode number is within the directory itself. So if I give a command like say open, fopen syntax may not be correct. So if I give a command say fopen password, when presently I am under this directory etc then how this will be converted to the corresponding inode number? I know that because I have not specified any path name, so these have to be found within current directory only. You simply search the current directory, get the entry corresponding to this file name password, extract what is the inode number of that. So once I get the inode number of this password, the next operation what is to be done is you read this inode number. First you check whether this inode number is present in the main memory or not or whether I have an incore copy of this inode number 205.

If I have an incore copy of this inode number 205, what I do is I simply increment the reference count field of the incore copy of inode number 205 by 1 but if it does not exist in the main memory, if I don't have an incore copy of this inode number 205 what I have to do is I have to read the block from the secondary storage which contains this inode number 205 and that is very simple because in case of inode blocks all the inodes are put in sequence. The first inode appears first, second inode appears next, third inode appears after that and so on. So from this inode number I can always find out that what is the block if I have more than one blocks containing the inodes, I can always find out that which block contains this inode number 205 and what is the byte offset within the block for this inode number 205 because I always know that what is the starting block number of the inode blocks, what is the size for every block, what is the size of every inode all these things are known.

So, I can always find out that what is the starting byte of this inode number 205 and which particular block. So what I do is I simply read that block, for reading the block again I have to go to that buffer cache management because this block has to be read and put into the buffer. Then once it is in the buffer then only I can filter out this inode number 205 and get the inode number and put this inode into the incore copy of the inode. On the other hand if the command is given like this fopen slash etc slash password, so if I give a command like this then what I have to do is because this is the path name starting from the root, I know what is the inode number of the root because that is stored in a global variable. I have to take out this inode number, read it from the disk, put it into incore copy or may be while booting this inode number is already present in the incore copy.

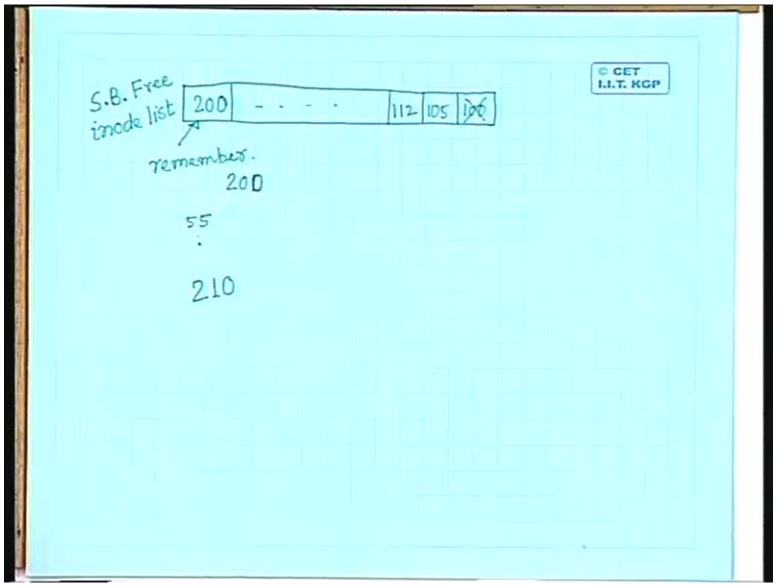
I have to check that this is a directory not a file because there are other components on this. So if this is a file then this path name is not valid, it has to be a directory. So I have to check that this is a directory, on verifying this is a directory I also have to check that I have the write permission on this directory that I have to have the search permission on this directory. In case of directory, search permission is equivalent to execute permission. A file can be executed but a directory cannot be executed. So in case of a directory, the search permission is equivalent to execute permission, so if you have execute permission on the directory then you can search that directory. So if you have the search permission then what I have to do is the next component is etc, I have to check that directory. So this etc, this root stored directory will be some sort of listing like this, I have to search for this component etc in that listing.

Once I get an entry for etc I have to take out the inode number of that. So once I get inode number of etc and in this case the inode number of etc is 100. So I have to get inode number 100. Again read it from the disk, put that into main memory then from that inode number I get the disk blocks from the table of content which contains the data of the directory and data of the directory is this list. So I have to read this list then search within this list for this entry password. So I get this entry password, inode number of the password 205, I have to extract this 205 then on getting this 205 inode in the incore copy, I can go to the table of contents of this inode. Then once I have access to the table of contents of this inode, I have access to different data blocks containing the data of this file password. So this is the very very important step and always whenever I want to access a file, I want to open a file or close a file or whatever I do, I an user process and application specifies the file name or the path name leading to that file. It may be file name or even the directory. So first operation that has to be done is you convert the file name or convert the path name to the target inode number and that can be performed in this manner. So once you have access to the inode, you have access to the inode table of contents. Once you have access to the inode table of contents, you have access to the data blocks containing data of that file or containing data of that directory and data of the directory means simply this listing. Sir, the parent directory of the slash etc is the root directory. Parent directory of slash etc is the root directory. Parent directory of slash etc is root directory. Yes, I remember. I see it should be 50th it should be 50. For a root directory, the parent directory will be root directory itself.

So now it is clear that given a path name or given a file name, how you can convert that to an inode number. Once you have an inode number, you have access to everything in the file. Instruction will take more time. Yes, I think more time because this has to be passed. Now next thing is so far whatever we have done, we have assumed that an inode is already allocated to a file or inode is allocated to a directory. The next important question is how do you allocate the inode to the directory or how do you allocate inode to the file? For that what we need to know is what are the free inodes when the file is created or free means unallocated inodes because free we are using for some other purpose.

What are the unallocated inodes in the file system which are not yet given to any other file or any other directory, when this new file or the new directory is been created? So for that again the super block plays an important role. In addition to the information that are maintained in the super block that we have said that the super block maintains information about the file system itself that is how big is the file, how big is the file system, how may files it can store and all those things. The super block itself contains the list of unallocated inodes that may not be an existing list that is the sub list of the total number of unallocated inodes of the file system.

(Refer Slide Time: 00:31:11 min)



So in the super block we have a list of inodes which are unallocated at a particular instant of time or a sub list of the inodes which are unallocated at a particular instant of time. Now let us assume that at a particular instant of time, I have several inodes which are still unallocated on the file system. The inodes may be say 100, 105, 112 like this, may be inode number 200 and in this case the super, this is called a super block free inode list or unallocated inode list. So this list is simply the serial number of the inodes, this is not the inode itself or it is not the data structure, inode data structure it is simply the serial number of the inode. to access this inode number 100, I have go to the inode block and from the byte offset of this inode number 100, I have to get that inode that is the data structure of the inode.

Now it says that inode number 100 is not yet allocated to any other file or any other directory, 105 is not allocated to any other file or any other directory and so on. So up to this inode number 200 is maintained in the super block free inode list. There may be other inodes free on the file system but those are not yet maintained in the super block free inode list but they will be brought in the super block free inode list, when this super block free inode list becomes empty. Now this being the situation whenever a process puts a request for creating a particular file or creating a particular directory, creation means this is being created for the first time. So it has to be allocated an inode which is not yet allocated. So what it has to do is it has to change the super block free inode list. On checking it finds that the first inode that appears in the super block free inode list is inode number 100 that means inode number 100 is not yet allocated to any other process.

So this inode number 100 will be allocated to the process or allocated to the file or directory which is being created. On allocating this inode number 100, inode number 100 will be removed from this super block free inode list. Now once you allocate this inode number 100, this inode number 100 has to be located on the inode block, make an incore copy of that inode so for the inode was free that is type field was set to 0. Now the inode is no more free that means the type field has to be set to either regular file or directory depending upon to which it is being allocated. Reference count has to be made to one because file has been open for creation and all this updated information of this inode has to be written on to the disk inode. With this allocation of this inode to the file or to the directory is complete. Then other operations from the file or directory will continue as it is.

Next time another process tries to open a file or a directory, this inode number 105 will be allocated so that way it will continue. Last when all these entries are empty, the last inode which is maintained in the super block free inode list is inode number 200. So when the situation comes that this inode number 200 is to be allocated to the requesting process then with allocation of this inode number 200, the super block free inode list becomes empty. So when it is going to become empty then what is done is the super block free inode list is filled up with other free inodes or unallocated inodes on the file system. For that we have to search for the inodes which are unallocated and the search operation will start from inode number 201. So this is an inode, that is the last inode on the super block free inode list is called a remembered inode. That means you have to remember this inode number for searching for other unallocated inodes on the file system. So starting from inode number 201, you search for other unallocated inodes on the file system and whatever inode number you get sequentially, this search is sequential, you fill up those inode numbers in the super block free inode list and finally setting this remembered inode in the super block free inode list. So there can be a situation that suppose the super block free inode list can contain list of 100 inodes and I am starting my search operation from say inode number 201, there are two situations that I have starting from 201, I have more than 100 inodes free unallocated or I have less than 100 inodes unallocated. So filling up of this super block free inode list will stop when you find that you don't have any other free inode on the super block free inode list or the super block free inode list is filled up. So at the most I can put 100 inode numbers or less than that if I don't have sufficient number of inodes which are unallocated on the file system. So this is the situation I mean this is the way in which the inodes will be allocated to different new files or new directories when they are created. Now the reverse process is unallocating a particular inode. How do you unallocate a inode? An inode will be unallocated, you find that inode becoming free whenever you close a file and the reference count becomes zero but that is not unallocation of an inode.

In that case what you are doing is you are simply removing the inode from the main memory, you won't have an incore copy of the inode but the inode is still allocated to the file and inode has to be made free or unallocated, when the file is deleted or the directory is deleted. So when you delete a file or you delete a directory, in that case I have to put this one in the super block free inode list if possible. So for doing that what is done is you simply check; what is the status of the super block free inode list. If you find that I have some empty space, an empty entry in the super block free inode list, I can simply write the inode number which is being unallocated into that empty entry that is one way.

If there is no space in the super block free inode list, empty then what has to be done is the inode number which is being unallocated, if you find that inode number is less than the remembered inode because this remembered inode tells me that from which point I have to search for new inodes, unallocated inode. Then you replace this remembered inode by this new inode number sorry here the search operation will starts from inode number 200 itself. So if you find that suppose the new inode which is being unallocated is inode number 55 and at that point, I don't have any free space in the super block free inode list. So what I will do is I will simply replace this 200 by this inode number 55.

So that next time I search for free inodes unallocated inodes, this 55 will be encountered. Otherwise if this new inode which is becoming free is more than the remembered inode then I simply forget it, don't make any changes in the super block free inode list. Suppose the inode number which is being made free is the inode number 210, this is being unallocated. My earlier remembered inode number is 200, next time I will start searching from this inode number 200. so 210 will any way be searched, so I will have these are the different possibilities whenever an inode is unallocated that is if I have an empty space in the super block free inode list, I simply put this new inode number which is unallocated into that empty space. In case super block free inode list is full and the inode number that is being unallocated is less than the remembered inode, you replace the remembered inode by this new inode number. If it is more than the remembered inode number then I don't make any change in the super block free inode list. I simply mark the inode as unallocated by setting the type field equal to 0 on the disk copy of the inode. So that next time, the search operation starts from this remembered inode that inode in any case be searched. So with this I hope that allocation of an inode or unallocation of inode is clear. So next day we will talk about management of disk blocks.