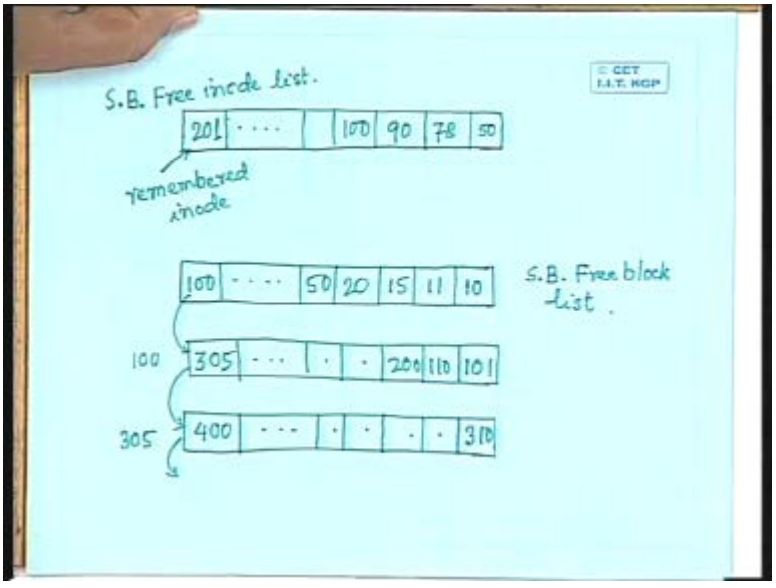
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture No. # 26
Secondary storage Organization III

So, we are talking about the file system organization with respect to the unix file system and we have said that for accessing any file, what you have to access first is the inode of the file because it is the inode which maintains every information of the file including its ownership, the access permissions, group ownership, the date of modification and even the table of contents that is which part of the file is loaded in which block in the secondary storage. So in the last class we are talking about allocation of an inode to a file or a directory that is newly created.

So whenever we want to allocate any inode to a file or a directory that is to be created, we have to first see that what are the inodes in the file system that are still free or not yet allocated to any of the existing files or any of the existing directories. And thus for such operation the super block plays a major role in the sense that super block contains a list of inode numbers which are still unallocated. And we have also said that this list is not an exhausted list in the sense that only a subset of the inodes which are not yet allocated will be contained in the super block free inode list. There can be other inodes in the file system which are still unallocated but they will be brought into the super block free inode list when the super block free inode list becomes empty. So the situation was something like this.

(Refer Slide Time: 00:02:49 min)



We can consider the super block free inode list as a list of inode numbers, so this list can be something like this. I can have an inode number say 50, inode number 78, say inode number 90, 100 so like this it can continue, may be this is the inode number 201. So this is the situation when the super block free inode list is full, so this is the super block free inode list. So whenever a directory or a file is created for the first time, you have to allocate one of the inodes to the directory or the file which has been created. So typically what is done is you take an inode from this super block inode list, so here the inode 50 will be allocated to the file that is newly created file or the directory whatever it is. And this location, once this inode 50 is allocated then this location, this entry in the super block free inode list will be empty.

Next time another file or directory is created, you have to allocate this inode number 78 to that file or directory. So I will have two empty entries in this free inode list. So this way when we have to allocate the last inode that is the inode number 201 from this super block free inode list, now with this after allocating this inode number 201, the entire blocks, entire super block free inode list will be empty. So when we have to allocate this last inode remaining in the super block free inode list in that case we have to search the file system for other inodes which are still free. So that has to be searched from the inode blocks which will contain the list of inodes and we have said that for searching this operation, we will just check the type field in the inode because if the type field is equal to zero that indicates that the inode is still free it is not yet allocated to any of the files or the directories.

So you have to search the inodes from this inode number 201 which are still free and fill up those inode numbers in this super block inode list. And this can be, filling up can be to the extent of the maximum capacity of the super block free inode list or until there is no other inode which are free on the file system. So after filing up the super block free inode list then only the last inode that is inode number 201 can be allocated to the newly created file or newly created directory and because the search operation has to continue from this location so this inode number is also called a remembered inode.

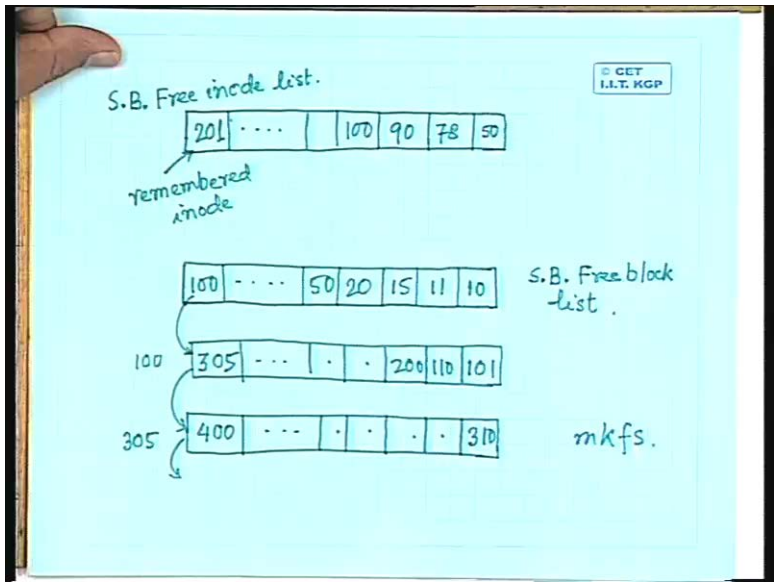
Similarly in the reverse process we have said that whenever inode is made free by deleting a file or by deleting a directory in that case we have to check what is the status of the super block free inode list. If there is any empty entry in the super block free inode list, in that case the new inode number which is being unallocated that numbers can simply put into the empty entry but in case the super block free inode list is full, there is no empty entry in the super block free inode list. Then what we have to do is we have to compare the number of the inode which is becoming free with the remembered inode number.

So if this number which is being free is more than the remembered inode number, then I don’t have to make any change in the super block free inode list. Only thing that I have to do is in the inode block that corresponding inode type field has to be set equal to zero. So next time when the canal will search for free inodes from the inode list because the remembered inode is less than the inode number which is becoming free so that inode number will be checked and it will be found that inode to be free and eventually that will come into super block free inode list and subsequently allocated. But if the newly freed inode number is less than the remembered inode number in that case we said that we will replace this

remembered inode number by this released inode number because if we don't put that in the super block free inode list because the newly released inode number is less than this, when the canal will search for other free inodes it will skip that, that will not be taken into consideration. Whereas if I replace this remembered inode by this newly released inode number, my search operation will start from the newly released inode number for free inodes and because this remembered inode is more than that so this will eventually be checked and subsequently allocated.

So that is why this special measure has to be taken when an inode is released because of deletion of a file or deletion of a directory and the super block free inode list is full. So if this newly released inode number is greater than this I don't have to bother, I simply make the type field of newly released inode in the inode block equal to zero but if it is less than the remembered inode in that case I have to replace the existing remembered inode by this new inode. So this is how the allocation and deallocation of the inodes have to be performed.

(Refer Slide Time: 14:01)



Now similarly as we said that whenever you create a file or you modify a file what may be needed is, if a file is expanded you need more and more disc blocks to contain the file content where the file content will be stored. So again you have to get new and new blocks to store the file data and those blocks must be free. If a block is already allocated to contain some file data then that block cannot be allocated to a new request. So whenever a request for a new data block is made, we have to check the file system to find out that what are the blocks which are free are not allocated to any of the existing file or existing directory and a block from that free list has to be allocated to the new file.

So earlier whatever we have discussed about is the inode allocation. Inode allocation will only contain the file information, information about the file but actually data of the file will be contained in the data blocks. For that also whenever a file is created or a file is modified

it needs more and more data blocks or the reverse process is if you delete a file then the data blocks have to be released so which has to go to the free block list. Now maintenance of a free block list is slightly different from the maintenance of inode list. The reason is in case of inode simply by checking the type field of the inode because every inode has a fixed structure, it has a fixed number of fields every field contains a fixed number of bytes.

So in case of a inode simply by checking the type field, we can say that whether the inode is free or the inode is already allocated which is not possible in case of data blocks because a block may contain any arbitrary data. So for maintenance of free data blocks our mechanism has to be different from the way we maintain the free inode list. So in case of data block what is done is all the free data blocks are maintained in the form of a link list, in case of inode we don't use a link list and the link list is something like this. In super block we have a free block list as in case of free inode list. This free block list contains a list of block numbers which are free. So suppose the block numbers are something like this 10, 11, 15 say 20, 50, 100, this is what is super block free block list. So up to this, it is similar to what we do in case of super block free inode list.

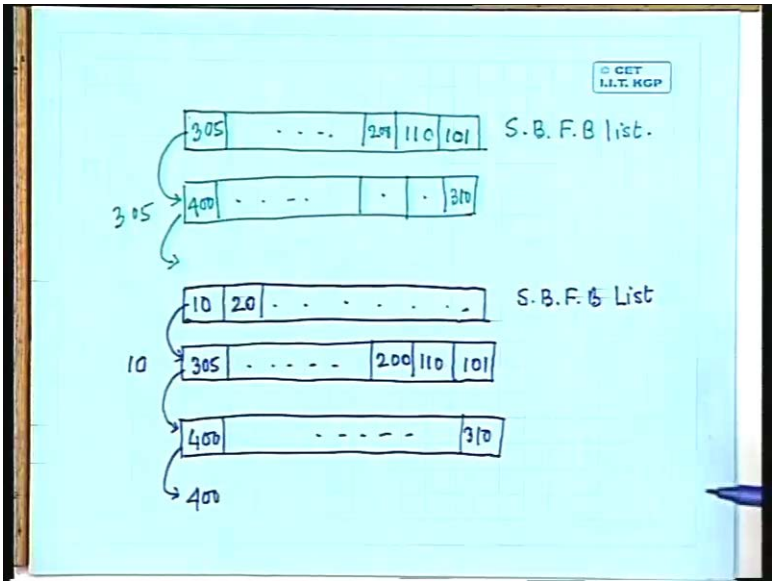
Super block free inode list contains a list of free inode numbers. Here the super block free block list contains a list of free block numbers but the difference is in case of free inode list, the last inode number was called the remembered inode. This is the number which is not a pointer but in case of super block free block list, the last number of the block is a pointer, pointed to the block number 100. So this is block number 100. So as the super block free block list contains a list of block numbers which are free, similarly this block number 100 will also contain a list of other free blocks in the file system. So like this it will continue, say may be here it is 101, 110, 200 and this way it can continue may be this is 305. Again this last entry that is 305 it is a pointer pointed to the block number 305, again this block 305 will contain a number of block numbers which are free. So this one may be say 310 and it can continue may be this is 400. So you find the difference between the free inode list and the free block list. In case of free inode list we don't have any linked list structure, simply because by checking the type field of an inode I can find out whether the inode is free or the inode is allocated which is not possible in case of blocks because a block may contain any arbitrary data.

So I don't have any specific structure for a block so that by taking a particular field I can find out whether the block is free or the block is allocated. So I have to have a special mechanism for maintaining the free block list and that is done like this. The super block contains a list of free blocks. The last block number is actually a pointer which points to the block itself. So this is block number 100, so when I come to this block number 100 by following this pointer I come to block number 100 physically which again contains a number of free block numbers and that way it continues. So when the free block list is maintained in this form, how the allocation is to be made? Whenever a process puts request for a new block you come to the super block free block list. Take a block number, if this is not the last block number in the super block free block list take a block number I find that here block number 10 is free, so block number 10 is allocated to the process which has requested for a data block. Next request can be made by assigning block number 11, next request can be made by assigning block number 15.

So this way you find that whenever I allocate a block from the super block free block list, the corresponding location in the super block free block list becomes empty. So it will continue like this. finally when it comes to the last block last block number, suppose all other locations in the super block free block list is empty, all those blocks have been allocated. The next request has to be made by allocation of block number 100 but if I allocate block number 100 without taking any precaution what will be done? I will lose the pointer information to this particular block that means though I have a number of free blocks in the file system still existing, none of those free blocks can be allocated to any further request.

So whenever any block is to be allocated, you have to check whether the block number that is to be allocated from the super block free block list is the last block or not, last block in the super block list. If it is the last block then what we have to do is we have to copy the content of block number 100, here it is 100 so copy the content of block number 100 in to the super block and after that block number 100 can be allocated to the requesting process. So in effect the situation that we will have is something like this.

(Refer Slide Time: 00:16:46 min)



Taking this same example, when block number 100 is to be allocated or situation which is something like this, super block free block list will contain block number 101, it will contain 110, it will contain 200 like this, the last entry will be 305 which is a pointer to block number 305, block number 305 contains other block numbers say 310, continues last number is say 400, this is a pointer to block number 400. So you find the situation that we have. In this case when 100 is the last block number present in the super block free block list and we have to allocate block number 100. So before allocation of block number 100, what we do is you just copy the… this is the content of block 100, you simply copy the content of block 100 into a super block free block list.

Once this is copied into super block free block list then this block 100 is free and I also saved the pointer information. So now this pointer instead of coming from block number 100, this pointer to this block 400 will come from super block free block list. So this is precisely what we have done here, this is our super block free block list. So simply block 100 has been removed from this pointer and this block 100 can be allocated to the process where it is requested for a new block and other pointer information remains intact. Next allocation can again start from 101 then to 110, then to 200 and so on. When 305 comes then the content of this block number 305, this is block number 305, content of 305 has to be copied into super block free block list.

So the remaining pointers now will point from the super block free block list and this block number 305 can be allocated to the process where it has requested for. So this is how the allocation of the free blocks from the file system can be done. Again the question is the reverse process. Whenever any process releases some block and because we have said that this is the way all the free blocks in the file system will be maintained and you know that there is an utility program in the unix operating system which is called mac file system of mkfs.

The responsibility of this software mkfs is to arrange all the free blocks in the file system in the form of such a linked list. The movement any free block goes out of this linked list structure that free block is no more accessible, I mean it cannot be allocated to any process. So to make any free block accessible to the canal, the free blocks must be part of such a type of linked list. So naturally whenever a new block, a block is made free by deletion of some directory or by deletion of some file or by truncation of some file, the block which is becoming free that has to be coupled with this linked list somehow.

So the simplest way is whenever a block is made free, I just check the super block free block list. If I find that there is an empty entry in the super block free block list, I simply put the number of the block which is made free into that empty entry in the super block free block list. Then the block becomes part of this linked list but the problem comes when I find that a block is becoming free but the super block free block list is not there. There is no room in the super block free block list. In such case what you do is you just copy the content of super block free block list in the new block which is becoming free and just put the number of that block in the super block free block list.

So in effect what I have is if I have this kind of situation say here all the entries in the super block free block list are full. I don't have any empty room in the super block free block list and in this kind of situation, if a block number say 10 becomes free so this is block number 10 which is becoming free. And I don't have any room in the super block free block list to put this block number 10. If I can put block number 10 then block number 10 is part of this free list and I don't have any problem. So in this case what will be done is simply the content of the super block free block list will be copied into block number 10. So block number 10 will contain 101, 110, 200 and so on last entry is 305. This 305 is pointer to block number 305 which contains other block number say 310. It continues, this is say block number 400 which is pointed to block number 400 and what I do in super block free block

list is just put this block number 10 as a last entry in the super block free block list. So this is super block and this is the pointer to block number 10.

All other entries are empty entries; there is no other entry in the super block. So you find that now this block number 10 has become part of this free list. So whenever a process puts request for another block in that case this block number 10 has to be allocated to that particular block but before allocation of block number 10, the content of block number 10 will be copied into the super block free block list and block number 10 will be allocated, so I effectively come back to this configuration. So whenever allocation of a block is to be made, you have to check the super block free block list, if it is not empty just take a block number and allocate that block.

If you find that after allocation of that block, super block free block list is going to be empty, you follow the pointer, come to the block which is next, copy content of that block into super block free block list and after that the block number can be allocated. Similarly while freeing the blocks, if I have an empty entry in the super block free block list, I simply put the block number in the super block free block list. If there is no empty entry, I copy the content of super block free block list in the new block and put the number of the new block in the super block free block list.

So the entire linked list structure is maintained and this special kind of data structure has to be maintained for data block which is not needed in case of inode blocks, inodes simply because inodes have well defined structure but data blocks don't have this. (Conversation between Professor and Student: Refer Slide Time: 24:40). It will be done sequentially because whenever you return the blocks, the return has to be done sequentially one block at a time not a chunk of blocks simultaneously. So what is done is whenever you put a file or when you have such a situation, there is a process called garbage collection. So what garbage collection program does is it tries to find out that what are the blocks which are reachable and makes this kind of list with only the reachable blocks and whatever is the bad sector, I mean typically the bad sector that goes out of the list.

Initially they are in the increasing order, when the first time the mac file system works they will be in the increasing order but after that order is not connected. I should not say order is not important, the order is important in the sense that it determines that what is the performance of the file system because if I can somehow ensure that different parts of the file will be stored in subsequent blocks, the throughput of the file system will be high but if the parts of the file has scattered in different blocks which is spread over all over the file system in that case the time to access the different blocks will be different. So the throughput of the file system will degrade. So what can be done? Defragmentation. When you find that the performance of the file system is becoming worse, you just run defragmenter. Defragmenter what it will do is it will reallocate the blocks and after that this free block list will be maintained in increasing order.

So defragmenter will try to put different parts of a file in consecutive blocks if possible but that is not a continuous process, it has to be done intermittently and in most of the cases it is done manually. Yes any other question? In this scheme the block which is released most

recently will be used first but that does not matter. Ya, not the best policy in the sense say block released means the block does not contain any more valid data. So whichever way we allocated that does not matter, so for us the correctness of the system is concerned but it may matter for performance that is the throughput. Quite possible. (Conversation between Professor and student: Refer Slide Time: 28:20). May be possible because their reading and writing operation is quite frequent so that is possible but so far as throughput is concerned, we always expect that all the file systems should be clustered at one end, so far as throughput of the system is concerned. So in any case we have to accept it. Then we will stop here today. Thank you.