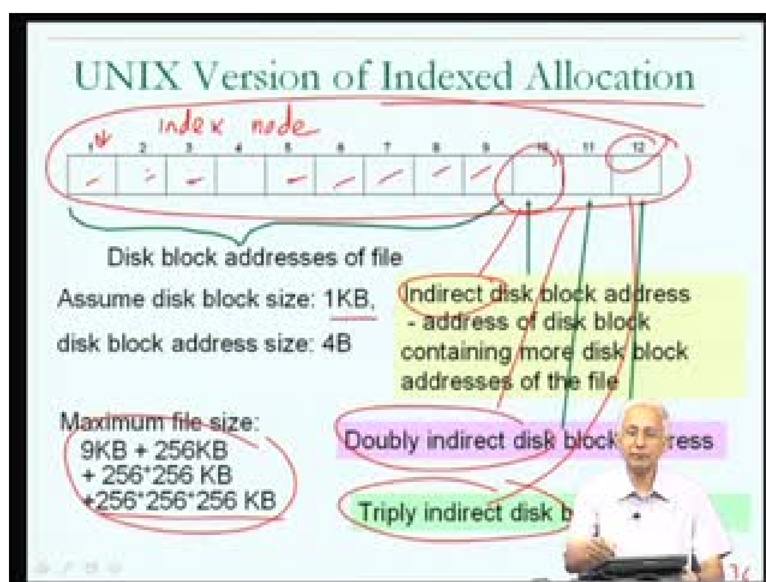
High Performance Computing
Prof. Matthew Jacob
Department of Computer Science & Automation
Indian Institute of Science, Bangalore

Module No.# 08
Lecture No. # 36

Welcome to lecture thirty-six of the course on High Performance Computing. In previous two lectures, we have started looking at the file system which is an important part of the software side of any computer system, particularly if your programs are dealing with data, there must survive beyond the execution time of your program.

This is a new kind of data, in the sense that, until now, when we talked about the data of a program, we were referring to data that was created and used by the program for parts of the execution time of the program, where as, in reality, there could be situations where you want the data to survive beyond the execution time of the program; in which case, files and the file system become important.

(Refer Slide Time: 00:54)



Now, we were looking at some of the design issues, the mechanisms that are taken into account by people who design operating systems, in designing the file system part of the operating system. And one of the…I have talked about three main issues we were going to discuss-one, relating to disk block management; the management of the disk which is the important persistent on nonvolatile secondary storage device, which we are taking into account in discussing file systems. And we saw that, the prevalent mechanism that is used for managing the disk in terms of allocating disk blocks to files is something known as indexed allocation; in which, an array of disk block addresses is associated with the each file. The first disk block address would refer to the first disk block containing the data pertaining to this file. The subsequent data and the file would be contained in the subsequent disk block, as mentioned in the index.

Now, it is important in designing such a mechanism, that one takes into account, the frequently occurring kinds of access patterns to files. In addition to this, one must take into account, the need for files should be able to grow and shrink, potentially to fairly large sizes. And in addition, the designer of the file system must ensure that the size of the index of the data structure that is used to keep track of a file, should not be too large.

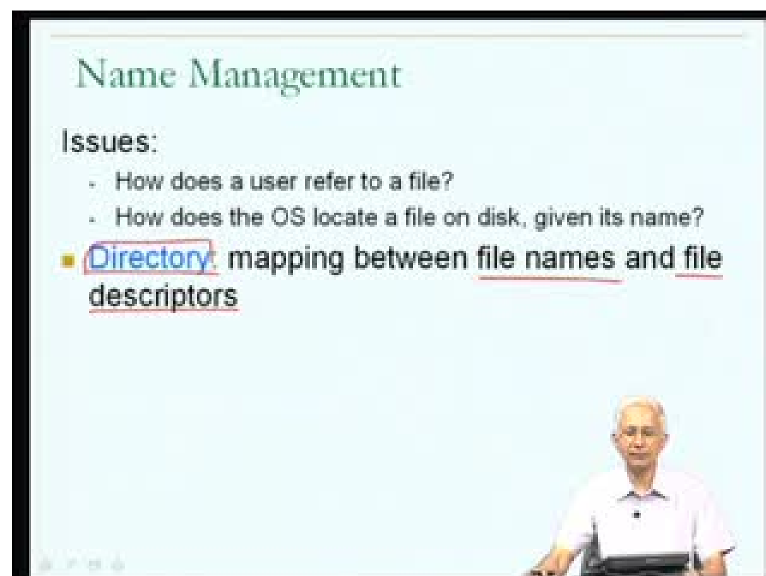
And among the various alternatives that we looked at, this form of indexed allocation, as used in Unix or Linux systems was found to be the best. So, in short, there are, in this particular example, I am showing you a index or index node or I node for short, which contains just twelve disk block addresses, but could actually allow you to represent a file which is many gigabytes in size due to the use of not only in direction, but double indirection. In other words, not just indicating the addresses of the disk blocks containing the data of the file, but in addition, using disk blocks to contain the disk block addresses of subsequent blocks, pertaining to the file. And doing this not only at one level of indirection, but at two or may be even three levels of in indirection to allow for fairly large files, as I said, up to gigabytes in size with assuming that the size of a single disk block or sector is 1 kilobyte even if there are only twelve indices at twelve addresses included in the index.

(Refer Slide Time: 03:24)

Now, we will move on to the second file system design issue, in other words, name management. Recall that, we understand a file to be a named sequence of data on a secondary storage device, such as a disk.

So, the management of the names of the files is important. The question of how can a user refer to a file and how will the operating system associate the correct name with a correct collection of disk blocks, in order to provide the data that the user actually requires.

(Refer Slide Time: 03:53)



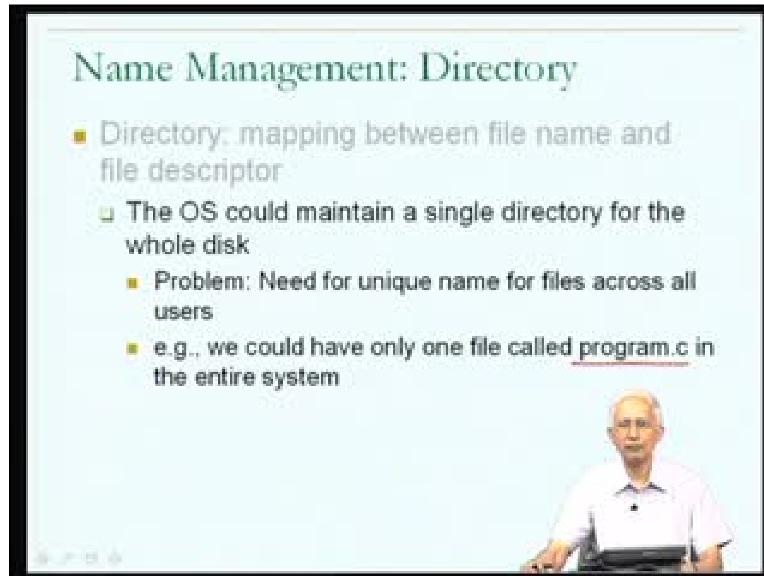
Now, the issues in name management, first of all, we have to have a clear understanding of how a user can refer to a file. So, how were the names constructed? And given a name, how does the operating system locate the file on the disk and subsequently read the associated disk blocks to provide data as requested by the program in execution?

Now, the primary structure which is used by operating systems today, for name management is something known as a directory. And the directory is a mapping between file names and the corresponding file descriptors. You will recall that, I had used the term file descriptor in the previous lecture, to refer to, in the day, operating system data structure that is used keep track of which disk blocks correspond to a particular file.

So, corresponding to any one file, they would be one file descriptor, which would contain the information regarding the various disk blocks associated with that file, along with possibly other information. But what we are learning today is that, the directory or the mapping between the names and the actual descriptors could be maintained, not within the file descriptor, but rather in a separate structure called a directory.

In other words, it is not typically the case that the name is part of the file descriptor. As we saw in the previous class, the information in the file descriptor was typically something like the index, in other words, the specification of which disk blocks contain the data associated with this file. And there could be other information as we are shortly going to see. So, in short, then the mechanism for name management, the primary mechanism for name management is the directory and mapping between file names and file descriptors. So, the user can refer to a file by a name, the operating system can refer to that name within the corresponding directory and hence get the file descriptor using which, the disk blocks of the file can be accessed.

(Refer Slide Time: 05:45)



Now, the issue which arises now is, how could the operating system maintain this directory information efficiently. Now, one possibility which you could immediately think of is that, the operating system could maintain a single directory for the entire disk. So, for each of the disks which is, which contains files, there could be a directory and that directory could contain, for each of the files on that disk, the name and the corresponding file descriptor. And you could immediately see that, this is somewhat restrictive in that, if there is only one directory for the entire disk, then there has to be a unique name for every file on the disk and therefore, it would have to be ensured that, no two files have the same name. So, this may not sound like much of a problem, but if you think about it a little bit, this could be a severe problem.

Imagine, there is a situation where there is a very large disk. Let say, 300 or 500 gigabytes in size. Such a disk could contain many thousands of files. And the files may potentially be owned by different users. Now, if it is required that each user makes up unique name across the entire file system for each of his files, and you will see that, this would end up being a problem. Every time that I wanted to create a name for a file that I intend to use, I would first have to, in some sense, ensure that this name had not been used before by anyone else, any of the user or in fact, in fact, in fact, myself, including myself for other files on the same file system.

So, this is definitely constrained that be too much to cooperate. So, for example, I mean, we have talked about many times about a C program called program dot c. And if the situation was one way, the operating system was maintaining a single directory for the entire disk, then this what imply that, there could be only one file called program dot c in the entire system among all the users on the system. There could be only one file called program dot c and very clearly this would be far too restrictive.

Now, there may be certain situations where you could have seen this as not being too much of a problem. For example, let suppose that it is a situation where there is only one user. Simple example of that is a personal computer. There may only be the one user, in fact, on certain early personal computers, this kind of a directory may have been adequate.

But today, very clearly we must understand that, even on a personal computer, there could be many users and this is too much of a restriction which leads us to the second possibility. Rather than maintaining a single directory for all the files on the disk, the operating system could maintain a separate directory for each user.

(Refer Slide Time: 08:16)

So, if there are ten users on a system, there could be ten directories, one for each of the users. And the user of the directory of a given user will contain all the files, would contain information about the mapping between names and descriptors for all the files

created and used by that particular user. Now, once again, we need to, we understand now that, under this mechanism, there could actually be many files called program dot c on a particular disk. Each user could now have a file called program dot c. And when a program which I have written, tries to open a file called program dot c, the operating system would know that, it has to look for the mapping information relating to program dot c in the directory associated with me, as a user. You will recall that, we understand that the operating system maintain or differentiates between users by having a different user ID for every user.

So, the user that, the directory associated with my user ID is one that would be refer to, when my program tries to open a file called program dot c. Now, you may imagine that, this too could be a little bit restrictive. Because now, while every user on the system could have their own file called program dot c, there could only be, as far as I am concerned, as myself being one user, I could only have one file by a given name. So, I would have to have unique names for all the files which I create. And this is the simple example of how this might not might be a problem.

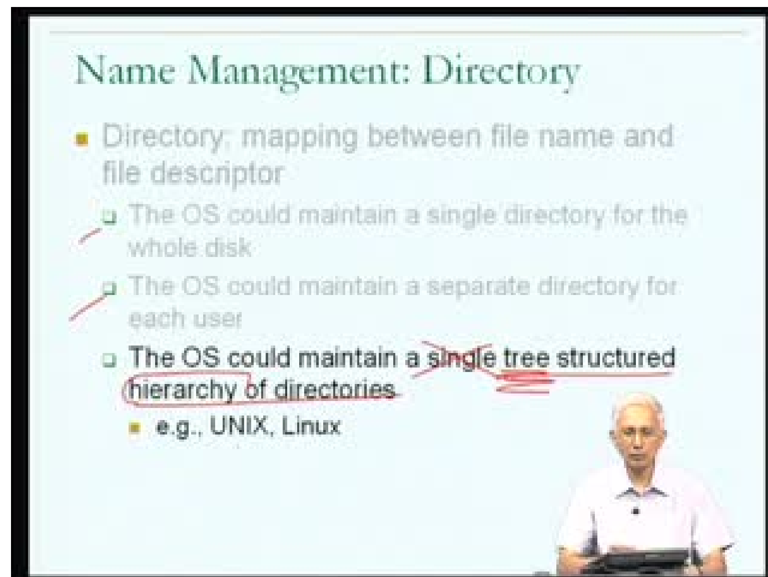
Now, it might be the case. I would be happy to have only one file called program dot c, but I may want to have many files called README .And those of you who have done programming or have written documentation may have made use of the file name README. You will put a lot of information relating to a particular project that you have done, a particular assignment that you worked with or something like that, into a file called README.

And hence if you have only one directory for you as a user, then there could be only one file called README and this in turn, might be a constraint. Many of you are students, you may be taking multiple, many courses and in each course, you may have many programming assignments to do and typically may want to have many files which actually have the same name such as README or even program dot c. For each of the assignments and each of the courses that you do, if you want to have the same name for the file, program dot c, there should still be possible.

In other words, many different files, because there is one file for each of the different programs that you wrote, but you may want to call each of the files, program dot c .This may not be a good habit in general, because could lead to confusion. But it should not be

ruled out by the directory mechanism of the computer system. And possibly, the need for multiple files called README would be a more appealing kind of an example.

(Refer Slide Time: 11:17)

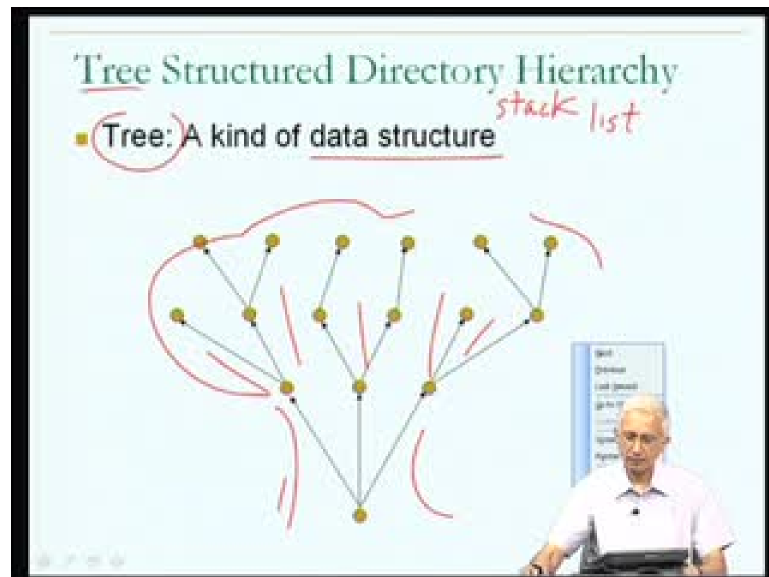


Hence we need to look beyond a single directory for the whole disk and potentially even beyond a single directory for each user to something which will allow users to have multiple files of the same name. And what this leads to is a much more general idea; the idea of the operating system actually maintaining a somewhat complicated structure for directories, and I will describe it as, what we have come to is, a Unix or a Linux type idea. Rather than maintaining a single directory across the entire disk or a single directory for each user, the operating system maintains a tree structured hierarchy of directories. And remove the word single from there. A tree structured hierarchy of directories. In other words, not a single directory even for users, but a hierarchy of directories.

We have seen the word hierarchy before, and we understood that whenever the word hierarchy was used, we were referring to some kind of a multileveled structure. And hence we are talking here about something more complicated than either of the earlier to, and must therefore be much more general and allow for much more aggressive kinds of names to be used, such as, many files called program dot c; even for a given user.We first need to understand what a tree is. We have not seen this word before in this course,

but as I had mentioned, the idea of this tree structured hierarchy of directories is what one finds in Unix and Linux systems today.

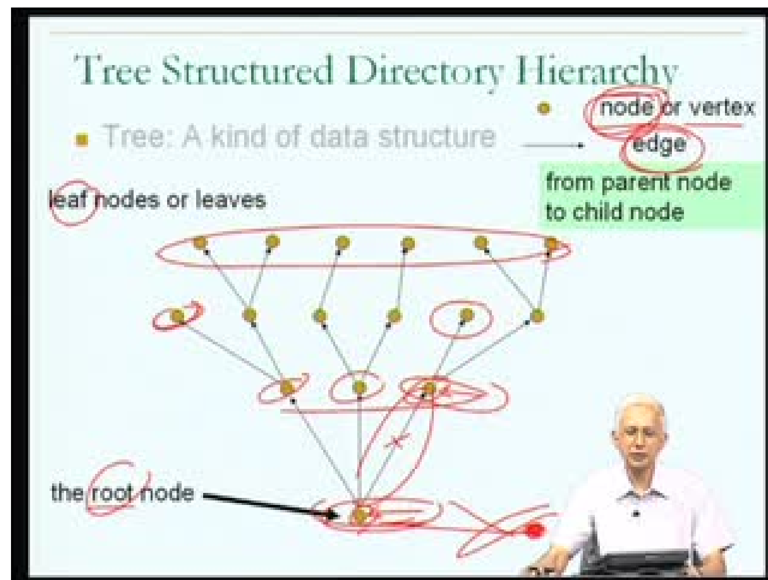
(Refer Slide Time: 12:30)



So, what then is a tree. To properly understand what a tree structured directory hierarchy is, we need to understand what a tree is. The other terms we have seen before. Now, tree should be viewed as being a kind of a data structure and you will recall that, we have seen other data structures. The data structures that we have seen so far, were things like the stack, we have seen other data structures too.

We saw the idea of the list; link list as being a data structure and so on. So, the tree is to be viewed as being another kind of a data structure or collection of data and the corresponding operations on the data. Now, typically when you think of a tree, you think of the tree in nature and this is a diagram sort of like a tree in nature. This is the bottom part, the trunk of the tree.

(Refer Slide Time: 13:39)



Then, these are the branches of the tree and leaves all over the tree. So, the diagram that I have drawn is one, which corresponds to the picture of tree that you have, when you think about nature. But more specifically, what we need to understand from this diagram is that, this is a structure which may get its name from the similarity, from the parent similarity to the tree in nature. But very clearly, it is showing some structure that we need to understand better. So, in terms of some terminology, you will notice that I have used two structures in creating this; two kinds of objects in creating this tree in this drawing.

First, there are the brown circles and there are the arrowed, I am sorry, the lines, the directed lines are arrows. And in general, you could talk of the brown circles as the nodes or the vertices of the tree. The singular of the vertices is vertex. So, I use the word vertex or node. Still, a tree is made up of nodes or vertices and the nodes or vertices, these are synonyms, I will use the word node from now on. So, the nodes of the tree are connected to each other by these directed arrows, by these direct lines which are called edges. So, we will talk about the nodes and the edges of a tree.

Now, the thing to note is that, a particular edge goes from one node to another node. For example, this particular edge goes from the node at the bottom to the node just above it on the right side, and the terminology which we will use is to talk about an edge going from a parent to a child node.

So, in this particular example, as far as this edge is concerned, I would refer to the edge of the node at the bottom as the parent of the node to which the arrow points. So, the tree is describing this parent-child relationship between nodes.

Notice that, a single parent can have multiple children. For example, the node at the bottom has three children and there could be some nodes which do not have any children. For example, this node over here, does not have any children.

So, they are certain properties associated with the tree. But the tree is showing us some kind of hierarchy and the hierarchy that is shown here, relates very much to the kinds of hierarchies we talked about before; you could view this tree as being capable of describing a business hierarchy, for example, this could be the CEO, this could be the three vice-presidents and so on. So, in some sense, the structure of the trees, not something that we are that the similar unfamiliar with. Now, in sense of further terminology, one property of the tree is that the tree has a unique node or a single node which does not have a parent. And if you look at our particular tree, the node at the bottom is the only node in this tree that does not have a parent.

In other words, it does not have an arrow pointing at it. If it had a parent, then there would have been other node which had an arrow pointing at it and I would refer to the new node as being the parent of this particular node. But in the node that we have, there is no such situation and the node which does not have a parent, the single node in the tree which does not have a parent is called the root node. So, I could, we could refer to it as the root, in general.

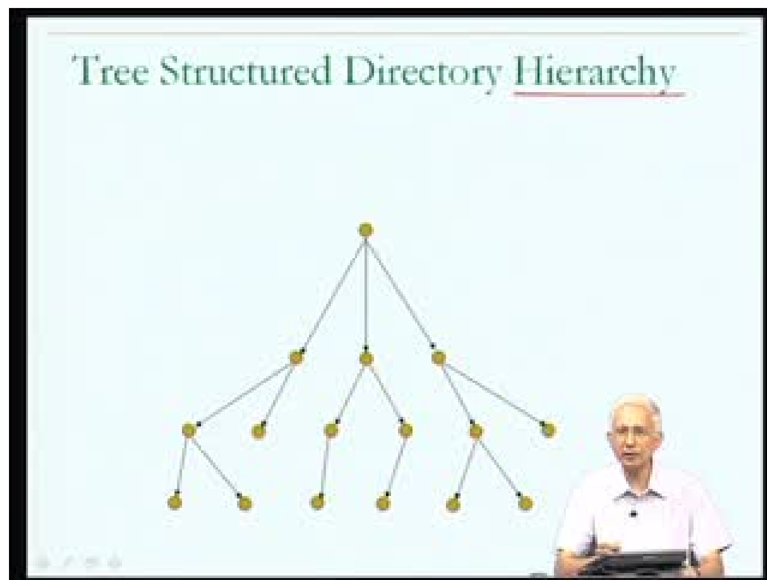
So, the kinds of trees that we are talking about have a single root and then they have multiple other nodes; each of which has a parent-child relationship with one of the other nodes in the tree. In terms of further terminology, they are, as I said, there are some of the nodes in the tree which do not have any children such as this node or in fact, all of these nodes, none of them has a child. And this is another node which does not have a child. So, all of those nodes, what they have in common is that, they do not have any children and we would refer to such nodes as the leaf nodes or the leaves of the tree.

And once again, in terms of our understanding of the biological relationship between the terminology and the tree in nature, the node at the bottom, in some sense, it is easy to

understand why it is called the root node because, the tree in nature has roots which go into the ground and that is why the tree starts. So, referring to that, the tree, the node which does not have any parent as the root node makes a lot of sense. By the same token, referring to the nodes which are at the very top and which do not have any more children as the leaf nodes make sense. Because there are branches in a biological tree in nature, but the branches may have leaves on them. But beyond the leaf, there is nothing corresponding to a structure belonging to the tree.

So, the idea of leaf nodes and the root node, the idea of the tree being made up of nodes and edges is what we will work with. Now in practice, when drawing trees in the context of computer related phenomenon, in other words, in talking about the use of this kind of a tree hierarchy in computers, one typically draws the tree the other way around. In other words, rather than showing the root at the bottom and the leaves at the top which is what one see is a nature, its more often done to draw the root at the top and the leaves at the bottom. And once again, this relates better to our understanding of the use of hierarchy.

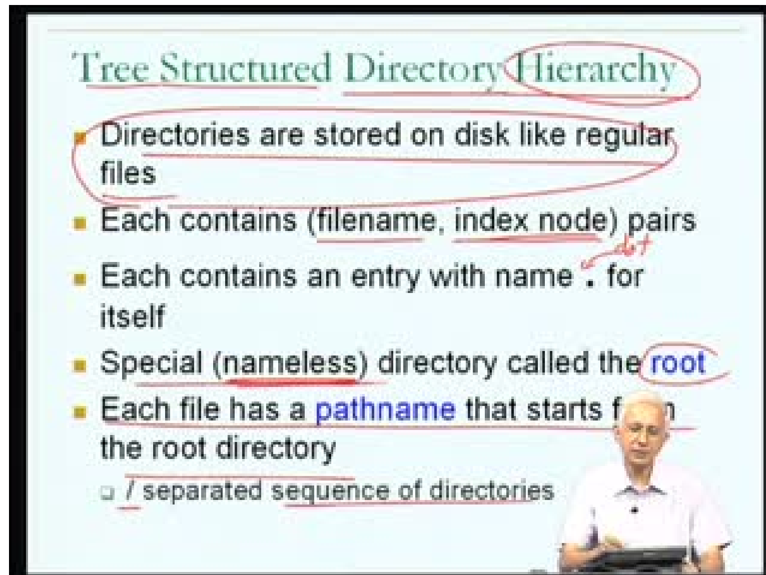
(Refer Slide Time: 18:28)



We logically start by looking at something from the top and since the root that the tree starts from the root, it makes more sense to draw the tree having the root at the top. So, here the analogy with nature sort of breaks because, one does not in nature, find trees which have their roots at the top and the leaves at the bottom. Let us perfectly, we will continue. From this point on, we will assume that will see trees which are drawn in this

way; the root at the top and the leaves lower down. The other nodes of the tree, by the way, could be referred to as, intermediate nodes of the tree.

(Refer Slide Time: 19:12)



Now, what is the idea of a tree structured directory hierarchy ? We understand what a tree is. So, we understand what it means to be structured like a tree and we see that the tree is defining some kind of a hierarchy. But how do the directories enter the picture? Remember that, the term directories here, referring to the mechanism that is used by the operating system, the file system in particular, to keep track of the mappings between file names and file descriptors.

So, directory is something which maps file names to file descriptors. Now, the idea which would be used in the tree structured directory hierarchy is that the directories themselves may be stored on disk, may be similar to the way that regular files are stored on the disk. Remember that, a directory has to contain information. What information it contains; information about a name to file descriptor mapping. This is the form of data just like a file, a text file which you create or program dot c file that you create contains information such as the text of a program or at the numbers that have to be read in by a program.

A directory too contains information. The information happens to be for each name, the corresponding file descriptor and this may not be information that a user program is

going to be is going to used. But it is information that could be stored in something like a file. So, that is an interesting idea; the idea that the directories themselves, in other words, the mappings between file names and file descriptors could be stored on disks, may be very much like regular files. So, what does the directory file contain? I could refer to it as a directory file now. The directory as a file, would now contain pairs of file names and file descriptors. And here, I am referring to the file descriptor by the term index node. You will remember that, in the case of Unix or Linux, the file descriptor takes the form of an index and hence the term index node. So, each of the directories could contain many filename, index node pairs.

Now, in addition. in the case of the Unix and Linux type file tree structured directory hierarchies, each directory contains an entry with the name dot. So, this is just a dot and that entry stands for the directory itself.

So, in order for a directory to keep track of information about itself, given that the file, the directory itself is something like a file and entry is maintained within the directory for itself as well. And that directory, that entry need not have a name since a directory itself has a name. But within the directory, its name is not relevant. Therefore, it is just represented by an entry with a name dot. In addition to this, since we are talking about a tree structured directory hierarchy, suggestion is that, there would be one node in the directory hierarchy which is right at the top and that node is going to be the root of the directory hierarchy. But other than that, that need not have a name. And therefore, in Unix and Linux systems, the root of the directory hierarchy tree is actually a nameless directory. And being a nameless directory, it still has to be referred by a name.

One cannot just refer to it as the nameless directory, but it is typically referred to as the root. Standing for the root of the directory hierarchy. So,that is actually a functional name; it has to be the root of the directory hierarchy. Its name is not root. in other words, one will not find the name r o o t root present in a directory referring to a particular directory. Therefore, in general, one should understand that the nameless directory which at the top of the directory hierarchy, we will refer to it as the root.

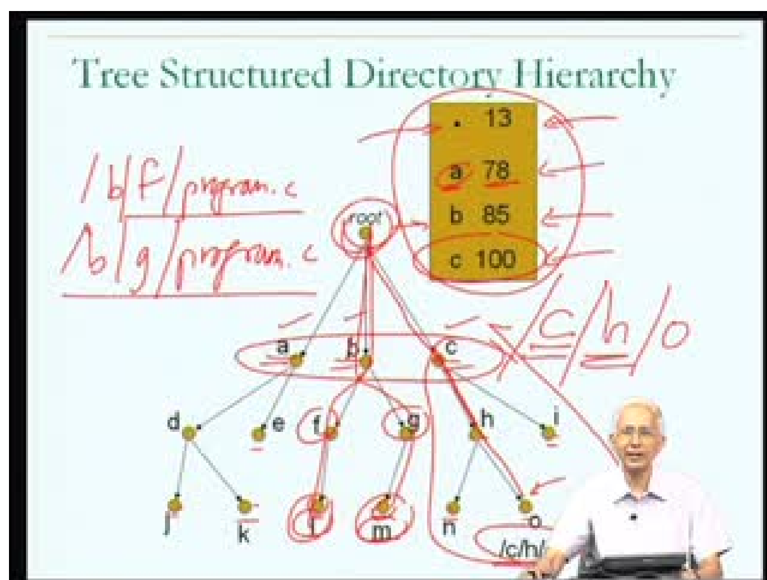
Now, what is the hierarchy of directories that we are talking about? Very clearly, we are talking about a situation where (( )) itself is a mapping between names and file descriptors and a directory is stored as a file, something like a file.

Therefore, there will be an entry for any particular director in another directory which happens to have a parent relationship with a directory that we are talking about. Hence this notion of a directory hierarchy that have directory which contains an entry for another directories name, in turn the directory, the second directory could have an entry for another directories and hence there is a tree structured relationship between the different directories.

So, what this boils down to then is, in order to refer to any particular file by name, one could actually start with the root directory, the nameless directory at the very root of the entire directory hierarchy and then go down the different directories until one comes to the file that one is referring to, and the pathway through which one goes through the directory hierarchy could be referred to as the pathname of the file.

The file itself would be a leaf node in the directory hierarchy. Let us just look at an example to understand what this means. Typically, in constructing the pathname, one separates the sequence of directories by slashes and so, I would refer to a pathname is being a slash separated sequence of directory names. Remember, we talke about a directory is having a name, simply because a directory itself stored on a disk like a regular file and must therefore have a name other than of course, special nameless directory which is called the root.

(Refer Slide Time: 24:43)



So, going back to a picture of a tree, let us try to understand how this could relate to a directory hierarchy. Now, the situation that we should have here is, we should clearly understand that each of the nodes in this tree is going to correspond to a file.

Now, some of these nodes are going to correspond to files that are in fact directories, in other words, files that contain mapping is between names and file descriptors or I node numbers. Some of the other files in the hierarchy are going to be files containing data that user can use. And as you would imagine, any of the intermediate nodes in the directory is going to be a directory in that. I am sorry, any intermediate node in the hierarchy it is going to be a directory and it is the leaf nodes in this hierarchy which are going to be the data containing files. So, if I associated a name with everyone of these nodes and I am using simple names just for ease of a typing, this particular node, I am calling node a. Remember, every file could be a data containing file or it could be directory, must have a name.

So, the name of this particular file is a. I have used names going form a to o for this particular example. At the top, we have that special nameless node of the directory hierarchy tree which is why I do not have a name associated with it. But we will refer, we have to have a name to refer to it in discussing it. So, I will just include the word root there. But just note that, that is not the name of that node, it just happens to be what we call that node. Hence, I have put it in italicize font. It is not technically a name associated with that node, it is just what we are going to refer to that node as. So, let us just consider, the directory corresponding to that root node. Remember what I mean by a directory; this is something like a file and what that file contains is mapping between file names or directory names and file descriptors.

So, let us look at the directory corresponding to the root node. What would we expect to see in that directory. We might expect to see something like this. So, it is a file or its it is a short on disk and contains information and the information happens to be in the form of pairs of names and file descriptors or file descriptor numbers in this case.

So, the file descriptors themselves may be stored somewhere else and these are referring to the names or the numbers of the corresponding file descriptor. So, in this particular directory, we find out that there is an entry for each of the files or directories which is in the next level of the hierarchy. So, there is an entry for a. There is an entry for b and

there is an entry for c. And the entry corresponds, the corresponding entry contains the name of that file or directory, as well as information about where that file or directory is contained on the disk in the form of (( )) the number of the index node corresponding to that file. In addition on the previous slide, every directory will contain an entry called dot for itself.

So, the entry called dot will correspond to the root directory and it tells me that, the information containing the root directory is contained in index node number 13. Therefore, the actual information about where this information is contained is available in index node number 13.

So, this is the idea of the tree structured directory hierarchy and just to make sure we understand what the pathname is, if you refer to, let say the file down at the bottom which I called o; file o down at the bottom, then if I try to trace its complete path from the root directory, then I would say that, if at the root directory, I had to take the branch going through c.From c, I had to take the branch going to h; I separate the branches by slashes and from h, I have to take a branch which takes me to o.

I started this whole thing from root which is a nameless directory. So, there is nothing before the first slash. In order to go to this particular file, I look into the root directory, I find the entry called c which tells me that, the information relating to this particular file must be further persued in the directory called c.

So, I open up the directory called c. If I am trying to find out the information about where the disk blocks of the file o are contained, I open up the file called c and look into that for another entry relating to the directory h and so on. It is the pretty much what the file system has to do. And this provides a mechanism through which, complete pathnames can be used to identify files, starting from the unique root directory. Now, there are many possibilities as to how this could be used. For example, the person who is administering your system might provide different users with directories at the second level. So, a might be the first user, b might be the second user and c might be the third user.

Subsequently, each of the users could create their own subdirectories. In other words, within their region of the directory hierarchy, they can create directories to their liking.

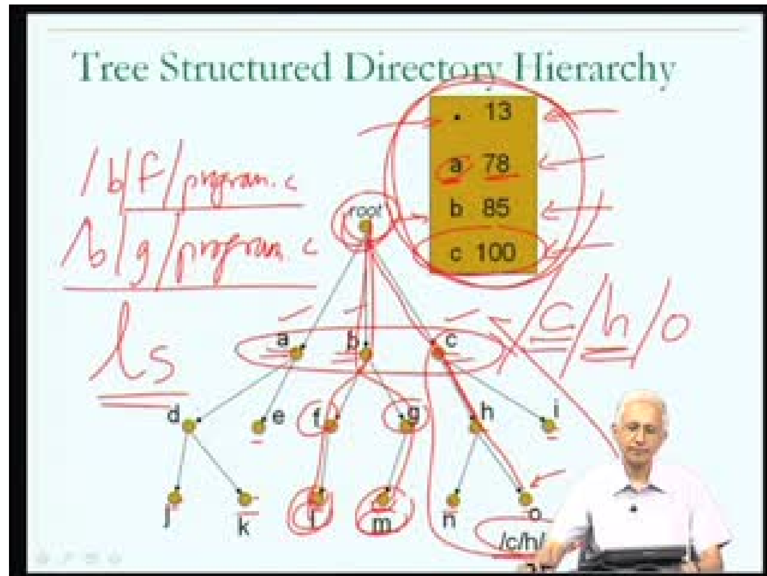
For example, user b may have created a subdirectory called f and a subdirectory called g, corresponding may be to the two courses that that user is doing. And in each of these two subdirectories, I could be a file and the two files could actually have same (( )) both of these files. I have called 1 l and I have called 1 m.

But both of them could just as well have been called program dot c. How would the operating system differ between these two program dot c is by the fact that, one of them has a pathname, slash b slash f slash program dot c; that is this one. And the other one has a pathname, slash b slash g slash program dot c. So, even though both of the files they seem to have the name program dot c, they are unique in that if I look at the entire pathname. The pathnames are different. And with this mechanism, it is possible for many of the problems that we saw with a two earlier directory hierarchy ideas,I am sorry,the directory ideas to be overcome.

(Refer Slide Time: 31:01)

So, this is a commonly used idea in Unix and Linux style maintenance of directory information in Unixand Linux. Now, with this, we have seen pretty much what is happening in Unix and Linux systems from disk management. The idea of the indexed allocation using a fixed size index node, we have seen that there is the idea of the tree structured directory hierarchy and the important concept of pathnames and how there could be directories which contain information about the mapping between file names and file descriptors.

(Refer Slide Time: 31:27)

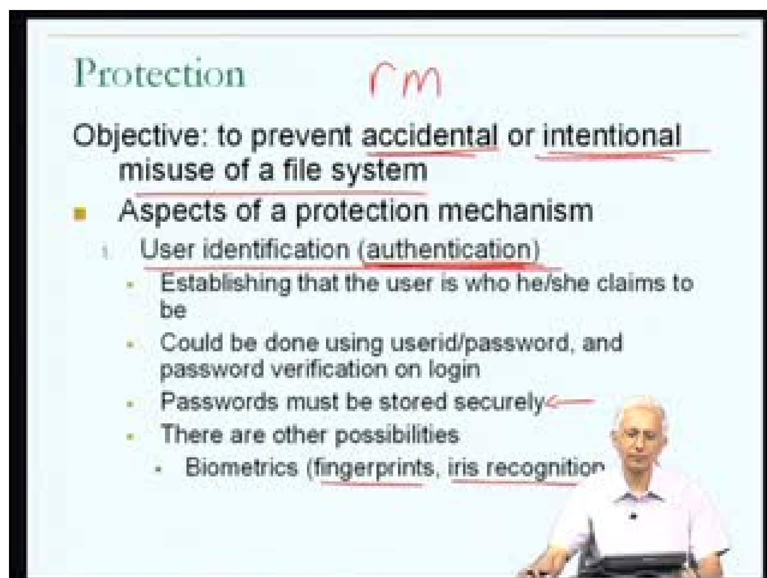


And just as a small aside, I should mention that, now that we know that directories are stored on disk very much like files, it may be interesting to try and see whether, you as a user, can open a directory just as you can open a file, whether you can open a directory and read the contents of the directory. It might be too much to expect to be able to modify the contents of the directory. But it may be possible for you to try to write the program that tries to open a directory and read the contents of the directory. And if you can do this, then you should be able to write a program, such as the l s program that we heard about, one or two lectures ago. What the l s program does is, it produces a listing of the contents of the current directory

(Refer Slide Time: 32:20).

Therefore, it should be conceivable to write such an l s program, if one can open a directory and readout its contents and just print them out on the screen. Now, we can now move on to the third important aspect of file system design. And that relates to protection. And specifically, by talking about what mechanisms are available to protect files from users; to protect the files of one user from another user.

(Refer Slide Time: 32:38)



Now, why it does one need to protect the files of one user from another user? Now, the reason that one is to do this is to prevent any misuse of the files within the file system

and the misuse could be accidental or could be intentional. But I mean by accidental is, if I, as a user, and I am able to access files of yours, then I could accidentally delete such a file. Many of you are familiar with the Unix r m command to remove a file and some of you may even have been in the situation where you accidentally removed one of your own files by just typing r m on the file.

Now, things could be even worse if I could accidentally remove files of other users. Similarly, there is a danger that there could be some intentional attempt to misuse files. Example, to either steal the contents of somebody's file, misuse them in that form or to modify the contents of somebody else file and this is clearly something that a file system should be designed to prevent.

Now, any protection mechanism will have a few common aspects to it. I am going to talk about each of these aspects a little bit. The first thing is, we have this clear understanding that, we are going to associate a file with a user and one talks about a file belonging to a user. Hence, the first step in any kind of a protection mechanism must be to identify or to figure out who a particular user is. When a program is being run, one assumes that the program is associated with a particular user.

But the operating system must somehow associate the execution of the program with a particular user and must make sure that, the programmers in fact, started or the execution of that program was initiated by that particular user.

Therefore there is a need to do something called user identification or authentication. Actually, verifying that the user claiming to be, let say, user one is in fact, user one. Now, essentially the objective of authentication is to establish that the user is, whoever he or she claims to be. If one did not have user authentication then, a person could just come to a computer system and start claiming to be the administrator of the system and do all kinds of things on the system. Therefore, authentication is very clearly an important part of any protection mechanism.

Now, you may ask how could authentication be done and some of you will be familiar with one of the commonly used mechanisms for authentication; and that is the use of user ids and passwords. So, the idea here is that, whenever a new account is created for a user, the user is asked to provide a password,password code that will be remembered by

the computer system and used by the user; in return, the user tries to log in to the computer system. Subsequently, when the user does login to the computer system, you will be, he or she will be asked to provide the user id as well as the password. And the computer system can then verify. The computer system will have to store the passwords securely.

But the computer system can then check whether the password has typed in by the user is the same as the password as remembered in the computer system. If they match, this can be used as confirmation that the user is who he or she claims to be. Now, it is of course important that, the passwords be stored securely. You will note that the passwords himself will probably have to be stored in a file. They cannot be stored in main memory. Because they must survive beyond the life time of one program. And if that is going to be stored in a file in a haphazard way, then any of the users currently on the system would be able to open and read the password file and therefore, all the passwords would then become common knowledge and so on.

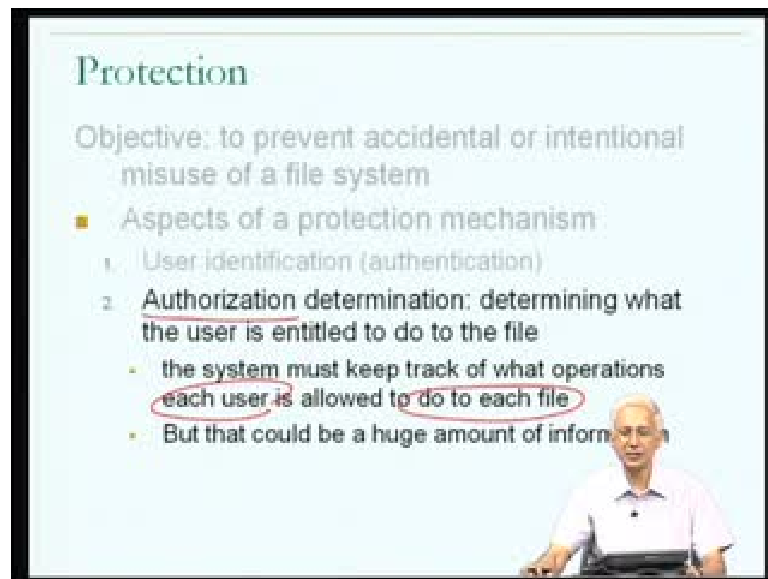
So very clearly, some mechanisms for making sure that the passwords are stored securely will have to be ensured such as encryption or something along those lines. Now, there are other more recently, there have been other different kinds of ideas for doing user authentication based on other ideas other than password. And you may have heard of some of these which are known as biometrics. For example, many of the laptops that you may be using, may actually have mechanisms through which one can record and display ones fingerprint and fingerprint information could actually be used for authentication. If this kind of a biometric is used for authentication, then when you attempt to login into the computer system, you would be asked to put your, one of the finger which fingerprint had been recorded, onto the fingerprint reader, on the on that laptop.

It would be then compared with the stored version of your fingerprint and if there was adequate correspondence, this would be viewed as authenticating you as the user who you claim to be. So, this is similar to the password, in that there, it is a secret stored on the computer and available only to the user. But unlike the password which is something that one has to remember, the fingerprint is biological reality as far as the individual is concerned and very difficult for somebody else to replicate.

Other ideas which are use today are based on the iris. The iris of the eye and iris device which could be used to check the iris properties of the individual who is currently trying to login to the computer system, compare those with these stored facts known about the iris properties of that particular user and using this for authentication.

So, this of course is important to note that, if passwords are being used, one must of course, make sure that, the ones password is adequate strength, in order to make it difficult for other users or people who are trying to break into a system to misuse ones user id by guessing ones password. So, user of the, authentication and identification of the users is an important property of any protection mechanism.

(Refer Slide Time: 38:13)



Now, in other important aspect as far as find the system protection is concerned is what is called authorization determination. And the problem here is, of the many users on the computer system, for a particular file that we are interested in, we have to track; the operating system to keep track of what the different users or what a particular users is entitled to do as far as a particular file is concerned. And that is what we might refer to as determining what the user is authorized to do in connection with a particular file.

Now, this itself may seem like a trivial problem. But if you think about it a little bit, you will realize that, in order to do this, the system, operating system, the file system will have to keep track of what kinds of operations each user do for each file and therefore,

the amount of information that would have to be stored in connection with authorization determination could be substantial. Remember that, authorization information would have to be stored for each file. First of all, it will have to be stored for each user, but it will have to be stored for each file and there could be thousands, millions of files on a computer system. For each of those files, information about what privilege, what operations each user could do on that file must be stored.

So, for each of the million files on a computer system, there would have to be information about each of the thousand users on the computer system, it leaving us with a huge amount of information that would have to be stored. And therefore,this is not a trivial problem, it is a problem of a sizable dimensions in terms of the amount of information that might have to be stored unless it is dealt with, in a careful fashion.

(Refer Slide Time: 39:54)

UNIX: 9 access permission bits

- Divides the universe of users into 3, in connection with a given file
  - The Owner of the file
  - The associates of the owner of the file
    - "users in the same group as the owner of the file"
  - The other users
- Associates 3 permissions with each of these for that file
  - Read permission
  - Write permission
  - Execute permission

Now, the way that authorization information is stored in the case of the Unix and Linux styles systems is interesting and therefore I will go through that. The idea here is, rather than storing information about how each of thousands of users would be able to, at the different operations, each of thousands of users would be allowed to do on each of millions of files. Unix or Linux manages with just nine bits of information and the way that it does this, is by first of all, not maintaining information about each user for each file, by dividing the universe of users into three sets in connection with the given file. The three sets are first of all, the owner of the file and we understand now that,

associated with each file, there is going to be a particular user who is considered to be the owner of the file. Second group, the second set of users that the operating system maintains information about is the associates of the owner of the file. In other words, the other users on the system who associate with and collaborate with or interact with the owner of the file and this is typically referred to as the users who are in the same group as the owner of the file.

The keyword there is group. The keyword in the first grouping was owner,the keyword in the second grouping is group and the third, a set of users that the operating system maintains information about, in connection with each file, is what is known as the other users or other. So there is owner, there is group and there is others. So, the operating system does not maintain information on the basis of each of the thousand users, the operating system just maintains information on the basis of these three groupings.

One set of information for the owner of the file, an other set of information use, as in, the group as the file as the owner of the file and third set of information for everyone else. What about the actual permission, the actual operations that the user would be allow to do on a particular file. The way that Unix or Linux maintains this information is by distinguishing between three kinds of operations; the read operation, the write operation and the execute operation. Now, you will remember here that, we are talking about operations on files and we clearly understand that it may be necessary to read data from a file. It may be necessary to write data from a file and to write data to a file. And hence, these are clearly well specified operations that we can relate to. But this whole notion of having an execute operation on a file is new.

Up to now, we were thinking a files as being data containing, but we should bear in mind that, some of the files that are present on a computer system such as an a dot out file will typically not be opened and read or written by a user. On the other hand, they are more likely to be executed by a user. And therefore, the execution or running a program has to be viewed as being an operation and hence in Unix and Linux systems, there is a separate, a third operation for which authorization bits are kept track of, known as the execute operation.

So, when we put all this together, then figure out that, what the Unix linux is doing is that associated with any one file, it is maintaining nine bits of information; three of the

bits of information relate to the permissions or the operations that the owner of the file can do on the file, three of the bits of information relate to the operations that in group, as the owner, can do on that particular file and the last three bits of information relate to the kinds of operations that any other user on the system could do to that particular file. Here, we are talking about the file, a dot out for example. What are the three bits? The three bits are permission to read, write or execute the file which are typically indicated by r w and x.

(Refer Slide Time: 39:54)

So, for a particular file, if the owner has the permission to read or write or execute the file, one would find those three bits set in the collection of information that the operating system is maintaining about that particular file. If the members of the same group as the owner are allowed to read the file, but not to execute, or to write it, then, one might find only the read bit set and the execute and the write bits not set. And if the others, the other users on the system are not allowed to do any operations on the file, one might not find any of the others bits set.

So, among the nine bits in this particular example, we find only the first four bits are set. The other five bits are not set and subsequently, the operating system could use this storage of authorization information relating to the file a dot out, in allowing or not allowing any particular access.

Now, there are certain other niceties of this which one could think about. For example, in addition to the fact that some files contain data, and other files are executable, we have this additional complication from our discussion of name management. Some of the files in a computer system are actually directories. For example, there is a root directory, there are subdirectories associate with each user and so on. And a directory is essentially, as we see something like a file, but contains information about the location of other files. But a directory in itself, will have to have these protection bits associated with it.

So, there will be the idea of the owner of a directory and the operations that in owner could do on a file which are going to be represented by the read write and execute bits; just as for any other file and one could interpret or try to understand with the operating system means by reading a directory, writing a directory or executing a directory. It make sense to view reading a directory as being able to open and access the entries in the directory.

So, for example, in order to execute the l s command, one would have to have read permission on the corresponding directory. Typically, when one talks about writing as referring to modifying the contents of a file, in connection with a directory, therefore one must assume that one would require the write permission if one was going to modify the contents of a directory. Now, one would modify the contents of the directory, if one needed to remove a directory entry or to add a new directory entry and therefore to do either of these operations, it may be necessary to have the write permission on a directory. Similarly, for execute, one could argue what the meaning of the execute permission might have on a directory and so on.

So, with this, we understand that, while, in order to specify the different authorizations that millions of, on millions of files by thousands of users in the first case, it look like a huge amount of information might have to be remembered. With this simple scheme, unix systems are able to manage with just nine bits of information for each file in the file system.

.

.

(Refer Slide Time: 46:45)

So subsequently, once the information about what operations can be done by each class of users on a particular file, the next question is how is this going to be enforced. So, the access enforcement. The first issue in protection mechanism was authenticating the users, the second issue was actually having (( )) track of the information about what operations each user could do on each file. The third aspect is enforcing those authorizations, making sure that the user who does not have permission to read a file, does not when that user attempts to write a program that does read the file, making sure that the access is not allowed.

So, that is the objective of access enforcement. And as you can well imagine, this is the important final step of the protection problem and it involves preventing users from doing unauthorized accesses to files and would clearly involve whenever user attempts to open a file to do, let us say, read access, the operating system is checking to make sure that, this particular user with that particular user id has the permissions and this can be done by referring to the access bits, the nine access bits associated with the file.

(Refer Slide Time: 47:50)

UNIX: 9 access permission bits

- Divides the universe of users into 3, in connection with a given file
  - The Owner of the file
  - The associates of the owner of the file
    - "users in the same group as the owner of the file"
  - The other users
- Associates 3 permissions with each of these for that file
  - Read permission
  - Write permission
  - Execute permission

(Refer Slide Time: 48:12)

Protection

Objective: to prevent accidental or intentional misuse of a file system

- Aspects of a protection mechanism
  - User identification (authentication)
  - Authorization determination: determining what user is entitled to do to the file
  - Access enforcement

Now, one piece of information which I did not mention, I talked about these nine bits of information (( )) with a file and the question will of course, arise to where the nine bits of information be stored. You will recall that, associated with each file, there is a directory entry and the directory entry contains a mapping between the file name. So, directory entry contains mapping between the file name and the I node number. For example, might be I node number thirteen, the file descriptor, the I node which contains the information about that location of the different data blocks of the file. Where then are the

read, write, execute the nine bit stored as far as file a dot out is concerned? Where is the information about who the owner of the file a dot out is concerned is stored?

Now, when we talked about the directory, clearly said that, directory was the mapping between file names and file descriptors. I never said that, the directory contains other information. And there may well be reasons for not wanting to put additional information inside the file directory. Because, as we have, as I have suggested, a directory is like a file on the disk and can actually be opened by any user who has read permission to the directory and the contents of the directory can be read. Therefore, it is probably not a good idea for the operating system to keep a lot of additional information inside the directory.

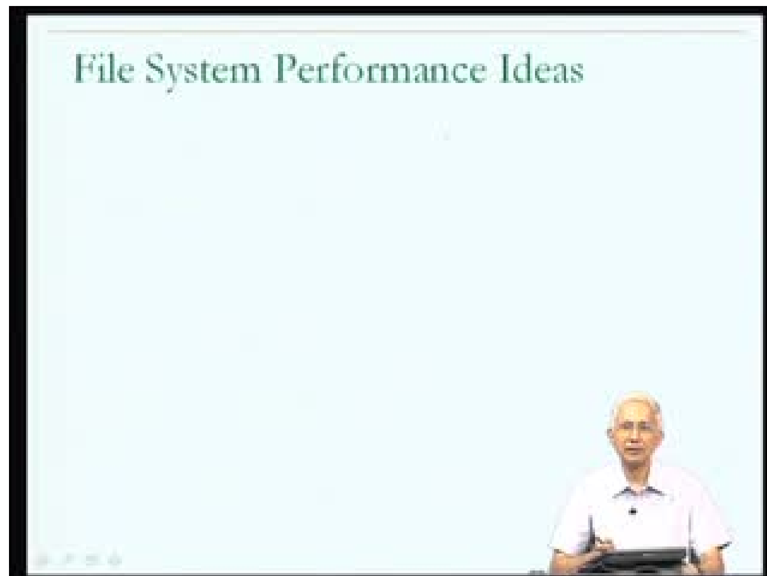
Where then, is this additional information contain and the additional information that I am talking about are the read write execute bits associated with that file, as well as information such as, who is the owner of the file, who are the members of the group associated with that file and so on. So, there are lot of other pieces of information associated with a file, then operating system will have to keep track off .And as far as much of this information is concerned, the information will in fact, be stored inside the structured, which until no,w I have been calling - the I node,the index node. So, the index node, in addition to containing the index corresponding to the file, will also contain other information conceivably the read write execute permissions permission bits on the file, as well as information about who the owner of the file is and there could be other piece of information such as the date and time on which the file was created, the date and time at which the file was last modified and there is other information that might be relevant as far as a file is concerned.

So, all of this additional information about a file will typically not be stored in the directory. But will actually be stored inside the I node or the index node, corresponding to the file. And the index node contains the index, in addition to all this information. We understand why it is called the index node rather than being referred as to the index. Now, putting all this together, we realize that, as far as the operating system is concerned, the disk contains a lot of information. Let me just draw the disk over here, we have a perspective of the disk as being linearly addressed from disk block 0 up to some maximum disk block address and parts of the disk are used for the file system. So, there could be regions of the disk which contains the blocks of the different files.

But in addition to this, there is a lot of information regarding the file system such as, all the I nodes that also will have to be stored on the disk. Therefore, we suspect that parts of the disk are going to be reserved by the operating system for things like remembering the I nodes of all the files which are stored inside the file system. And once we learn a little bit more about the way that the disk is used, we would have been able to understand how to access the I nodes.

Now, one final comment on these, the aspect of protection. I have talked about how one could imagine writing an l s program. l s program is the program which you invoke l s and it will list on the screen, the contents that the names of the various files which are present in the directory- the current directory. And I suggested that, this could be done by opening the directory as a file and reading and writing its contents. But we now understand that, the contents of the directory include the name of the file, but a lot of the information that we know, the l s program is capable of giving to us is not contained in the directory itself. It is actually contained inside the index node or the I node,corresponding to the file. And therefore, writing the l s program may require a little bit more information that what we actually have at this point in time.

(Refer Slide Time: 52:15)



Now, with this comments, I will close our discussion of the fundamental issues in file system design. And in the next lecture, we will move on to another and very important topic from the perspective of our programming, that is file system performance.

Now, we understand that the files are all stored on disk and that we have programs which will access the files by opening them, reading the contents of the file. Therefore, we would like to know something more about files from the perspective of understanding how we can make our programs which access files more efficient in terms of time. We will show in the next lecture. Thank you.

.

.