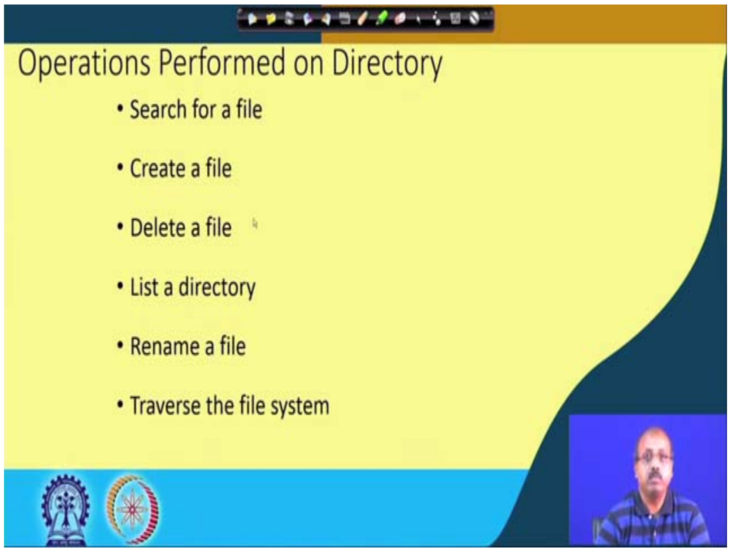
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 60
File System and Secondary Storage (Contd.)

(Refer Slide Time: 00:31)



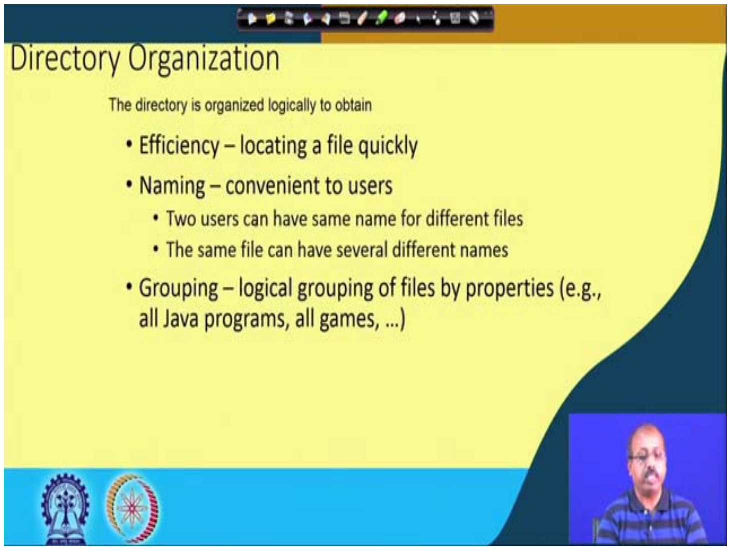
Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

So, operation that we can do on a directory is the most frequent operation is the search operation; we are searching for a file in a directory so, that is one operation. We create a file so, when we create a file then the entry has to be made in the directory that this file has been created. Similarly, we can we may want to delete a file so, when a file is deleted; the directory entry has to be deleted. We may want to get a listing of the directory so that shows all the files and directories that are there within this directory.

Sometimes we want to rename a file so if we rename the file then the directory entry will change and also you want to traverse the file system so it may be recursively if going through all the sub directories that we have within a directory and that way. So these are the operations that are performed on the directory.

(Refer Slide Time: 01:11)



Directory Organization

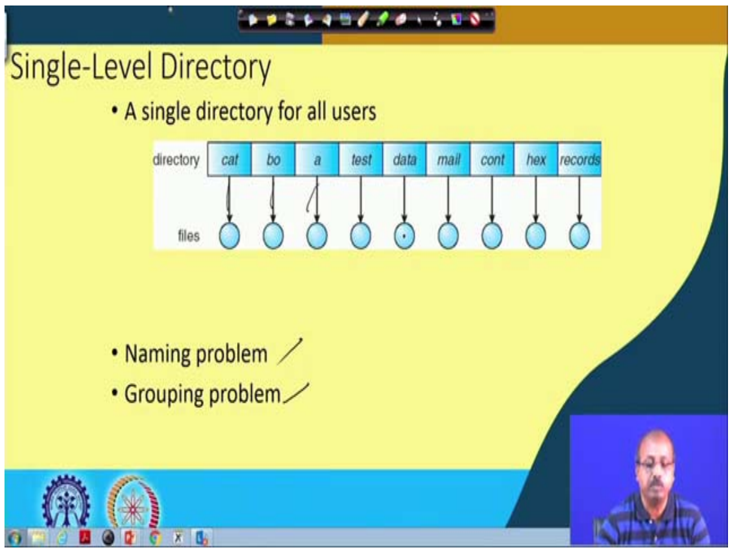
The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties (e.g., all Java programs, all games, …)

So, how to organize this directory structure? So, directory is organized logically to obtain efficiency because, we have to locate the file quickly, naming so that it is convenient to the users. So, two users can have same name for different files ok. So, that way they should is if they are on the same directory then that will create confusion. If they are on two different directories then we can conveniently locate them and separate them.

The same file can have several different names definitely so, from different users, it may be that they mean the same file, but they give different names to the files so that should also be possible. So, we have to have both way like two files given the same name and same file given two different names so both are to be supported. Then grouping so, it is the logical grouping of files by properties. For example, all types of all Java programs, all games so they maybe we may like to group them into some format some listing.

(Refer Slide Time: 02:12)


Single-Level Directory

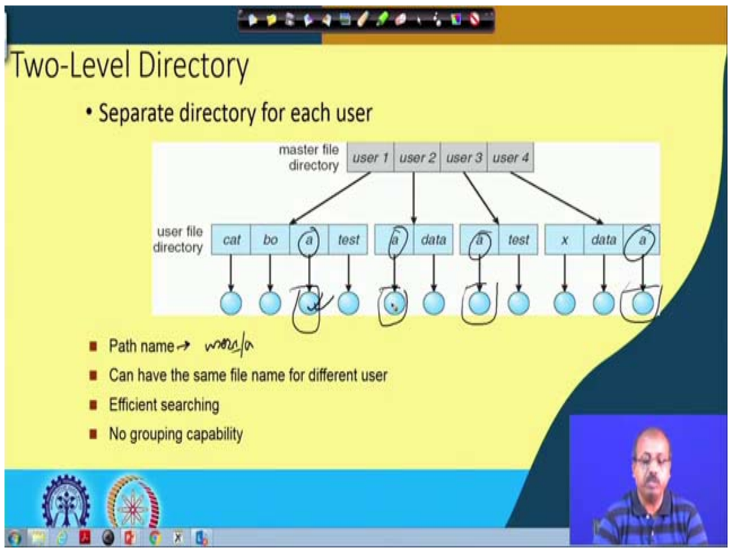
- A single directory for all users

- Naming problem
- Grouping problem

So, single level directory so, here this is the this is the simplest possible organization that we can think about, we have got a list of all the files that are there in the system and the for all users and then for each entry is nothing but a it contains the corresponding pointer to the so this is the single level directory. The problem is that then there is a naming problem because I cannot have another file with the same name as data because there is already a file called data so, naming problem is there.

Grouping problem is also there because I cannot very easily find out which files are say C programs so that is not possible because everything is in the same directory structure.

(Refer Slide Time: 03:00)


Two-Level Directory

- Separate directory for each user

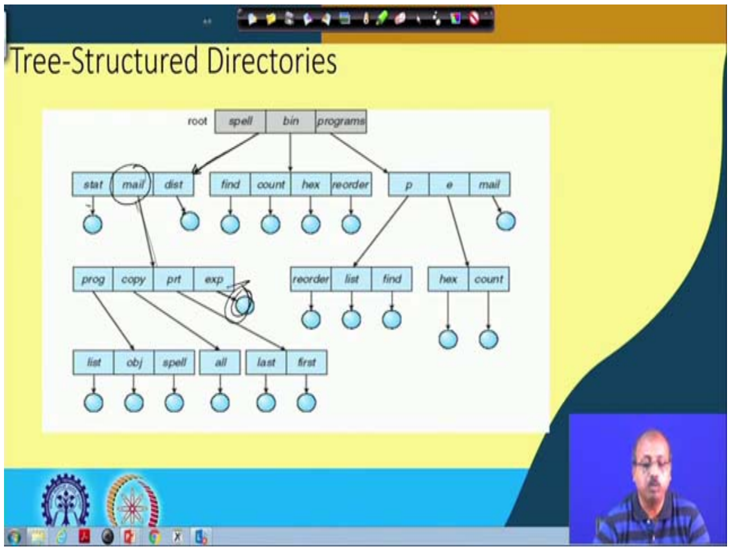
- Path name → user/a
- Can have the same file name for different user
- Efficient searching
- No grouping capability

So, to do something better, we can have two level directory. So, in a two level directory maybe at the top level at the system level for each user, we create a subdirectory. So, user 1, user 2, user 3, user 4 so, these four entries are there in the master level directory or master file directory. And, in the user file directory for user 1 maybe this there are four files that are there so, it is a given similarly where user 2 there are two files.

Now, you see that this name a is repeating at many places so, all the users they have got a file named a, but physically the files are all different so they are not the same file. So, once you have this hierarchical structure, we can have same name for you can have same name for different files given by different users and we can for getting a particular files so, we can we can talk about the path name. For example, for getting this user 1 slash a so that actually refers to this file, similarly, user 2 slash a refers to the second file so like that.

So, can have same file name for different users, searching becomes efficient because once the path name is given from the top you can just search by those path name and come to the particular entry. And then no grouping capability so, again you cannot group the files as per their types so that is not possible, but anyway so this is much better than that single level hierarchy.

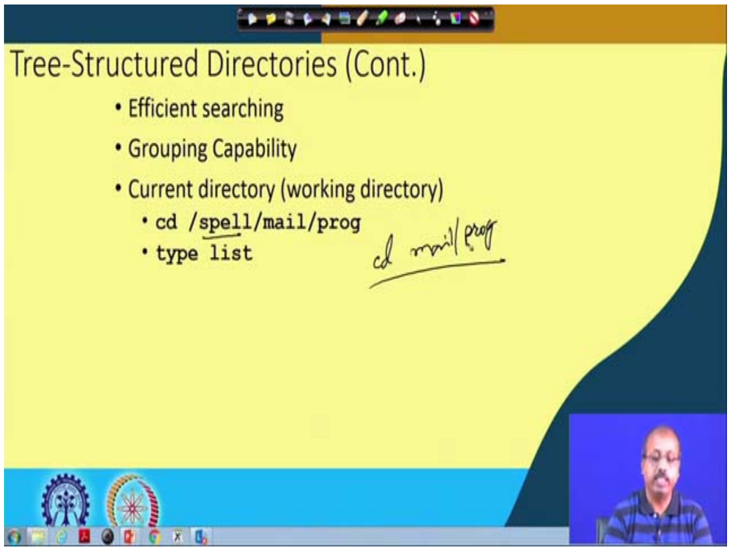
(Refer Slide Time: 04:33)


Tree-Structured Directories

You can have a tree structured directory also. So, like so instead of two only two level, I can have multiple level and at any level so, we can have simple file or we can have a subdirectory like you see that this one in this case so, this under this directory spell. So, we have got two files stat and dist and there is another directory mail. So, under this mail we have got say these four entries out of that this exp is a file so this is a file and these are again subdirectories.

So, this way, I can have this tree structured organization. So, there are there are multiple levels and at each entry in the tree structure, it can be another directory or it can be a simple file so both are possible.

(Refer Slide Time: 05:26)



Tree-Structured Directories (Cont.)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - `cd /spell/mail/prog`
  - `type list`

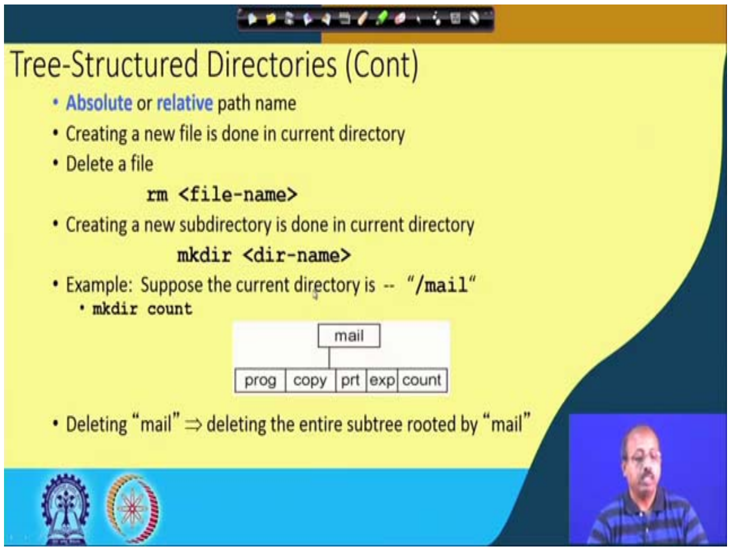
cd mail/prog

So, that way this tree structure directory is the most common one that we have in any operating system that that helps us in finding this thing. So, tree structured directories they are efficient for searching, we have got grouping capability because we can just we can so all the programs may be kept under this prog subdirectory, all experiments we kept under the experiment subdirectory. So, like that we can keep them under separate subdirectory.

So, we have got the grouping capability. So, we have got the current directory or the working directory concept. So, we can change the current directory by the command like cd, most many operating system will accept the command cd for change directory and we can specify a path for the live directory to which you want to change. So, this slash spell slash mail slash prog so, it will take me to the prog in this case so, it is starting a spell, mail and then this prog.

So, it is taking me to this directory so now, if I do now if I do a type list. So, it will list town all the files so it will be typing this file lists there. So, whatever so, this file will be printed, the content will be printed so, type list. So, list is the file so, it will searched in the current directory that we have.

(Refer Slide Time: 06:42)



Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

```
mkdir <dir-name>
```

- Example: Suppose the current directory is -- "/mail"
    - mkdir count

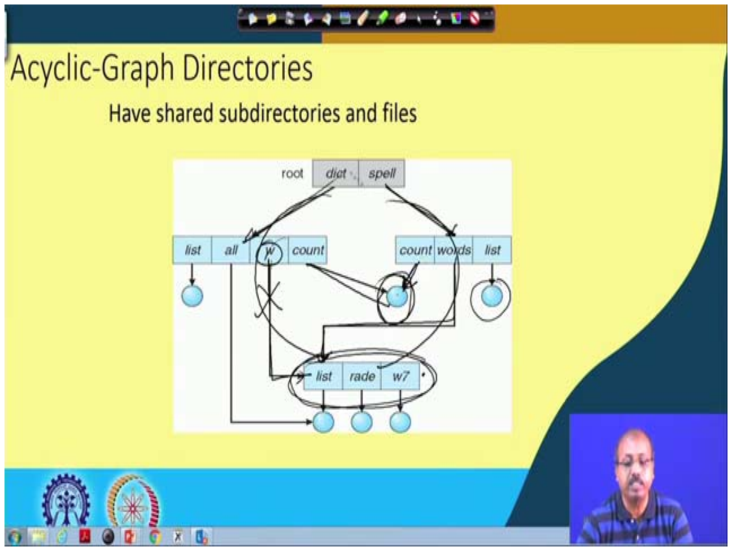
mail
prog
copy
prt
exp
count

- Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

So, we can have absolute path and relative path. So, absolute path means you are specifying the entire path from the beginning of the file system like say this one. So, this is an absolute path, but assuming that at present we are in the spell directly. So, if we say that it is if we simply write like cd so, if we. So, if we simply write like cd mail slash prog. So, assuming that we are at present in the spell directory so, we can write like this so, the means the initial slash is missing.

So; that means, that with respect to the current directory, we are talking about the path. So, that way we have got this relative path also. So, we can have absolute path, we can have a relative path for specifying the filename. So, creating a new file is done in current directory and deleting a file so that is also done from the current directory. So, name is or Unix, it is the command is rm, for other operating system it will be different. Similarly, you can create a new subdirectory by making using a command something like mkdir ok. So, again this exact commands may vary from operating system to operating system but we can do that.

(Refer Slide Time: 07:58)


Acyclic-Graph Directories

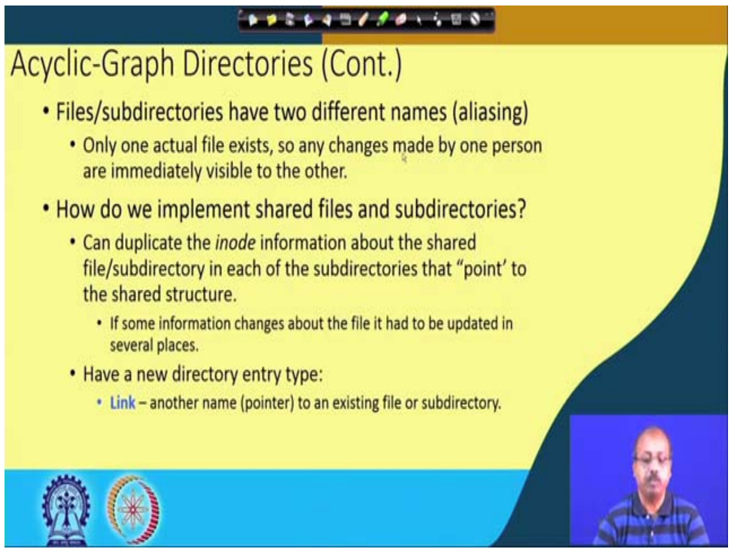
Have shared subdirectories and files

Now, sometimes we use the some graph type of directory. So, it is required because the some file has to be shared across number of directories, like here you see this particular file so this is shared from this part as well as this part. So, this is required to have a better view of the organization so that as a user, when you are coming from spell you see that I have got this file count as well as this directory words and this file list.

On the other hand so this one when you are coming by this dictionary so, you find that you have got this file count and then directories and also this w from here and the words from here they point to the same directory. So, as a result you can come to this position either from this path or from this path. So, that actually gives a better view of the file organization, but of course, it is difficult because if you are say deleting say or suppose you are at present in this path and you are deleting this w.

So, if you are deleting this w, it should not delete this entry only this link should go so, that should happen so this is very important. So, we can have the shared subdirectories and file so that makes the sharing easy in acyclic graph directory but of course, we have to be careful because some files may get some files, shared file should not be deleted once the link is gone.

(Refer Slide Time: 09:24)



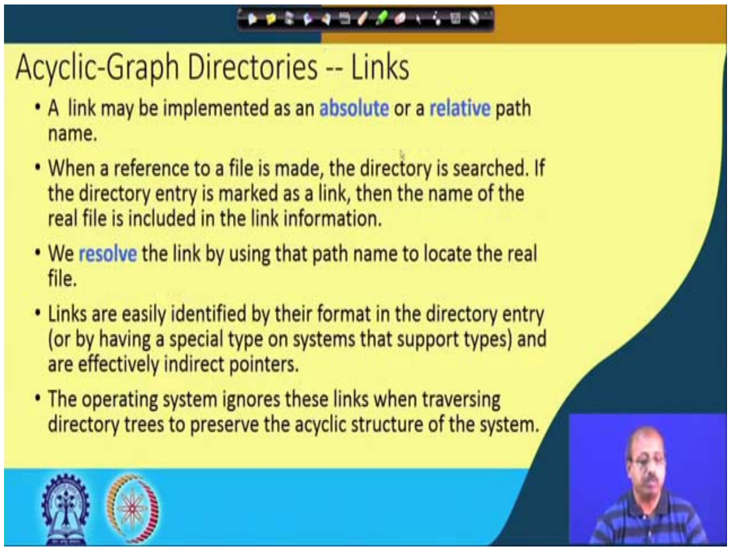
Acyclic-Graph Directories (Cont.)

- Files/subdirectories have two different names (aliasing)
  - Only one actual file exists, so any changes made by one person are immediately visible to the other.
- How do we implement shared files and subdirectories?
  - Can duplicate the *inode* information about the shared file/subdirectory in each of the subdirectories that "point' to the shared structure.
    - If some information changes about the file it had to be updated in several places.
  - Have a new directory entry type:
    - Link – another name (pointer) to an existing file or subdirectory.

So, files or subdirectories have two different names or aliasing, only one actual file exists. So, any changes made by one person are immediately visible to the other and how we implement the shared files and subdirectories, we can duplicate the inode information about the shared file or subdirectory in each of the subdirectories that point to the shared structures so, we can copy the inode structure. If some information changes about the file it had to be updated in several places.

So, one inode has been copied so if one of the processes they change the file then all the places they need to be modified. Have a new directory entry type which is link. So, if you look into this operating systems like Linux and all so you will find this file type link so, which is not physically the file, but it is actually a link to the actual file. Say, other operating system like windows also they have got this feature so we got this link file. So, whenever you are deleting a file. So, if there are links coming to that file then only the link will be deleted not the actual file. So, that way it is useful.

(Refer Slide Time: 10:33)



Acyclic-Graph Directories -- Links

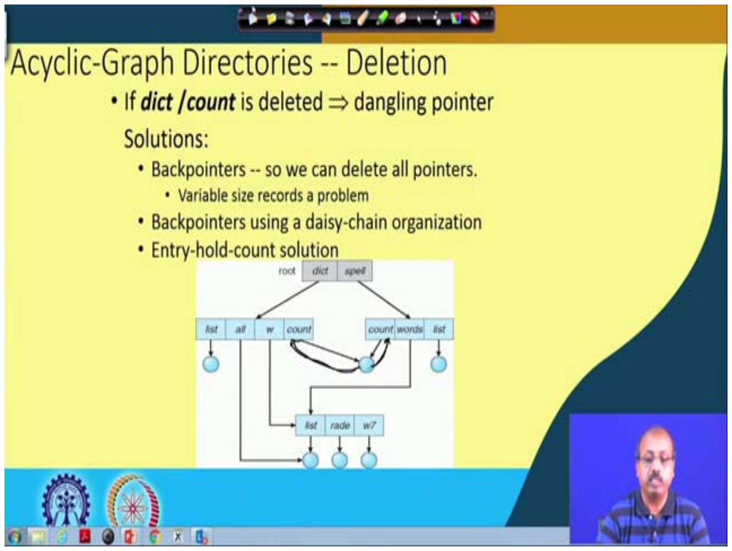
- A link may be implemented as an absolute or a relative path name.
- When a reference to a file is made, the directory is searched. If the directory entry is marked as a link, then the name of the real file is included in the link information.
- We resolve the link by using that path name to locate the real file.
- Links are easily identified by their format in the directory entry (or by having a special type on systems that support types) and are effectively indirect pointers.
- The operating system ignores these links when traversing directory trees to preserve the acyclic structure of the system.

A link may be implemented as an absolute or a relative path name. So, that is there just like absolute and relative pathname for file so you can have for the links also these absolute and relative pathname. When a reference to a file is made, the directory is searched. If the directory entity is marked as a link then the name of the real file is included in the link information.

We resolve the link by using that path name to locate the real file that. So, so once it once it is found that it is a link so, we find out the actual file name and then the search starts with the actual file name. So links are easily identified by their format in the directory entry. So having some special character in the name of the in the type name by having a special type on system that support types and effectively by means of indirect pointers.

Operating system ignores this links when traversing directory trees to preserve the acyclic structure. So, when you are if you traversing the entire tree and if you have got this type of links then these those links are avoided because then the same thing will be repeated again and again. So, that are they are avoided for a listing of the entire file system.

(Refer Slide Time: 11:44)



Acyclic-Graph Directories -- Deletion

- If ***dict /count*** is deleted ⇒ dangling pointer

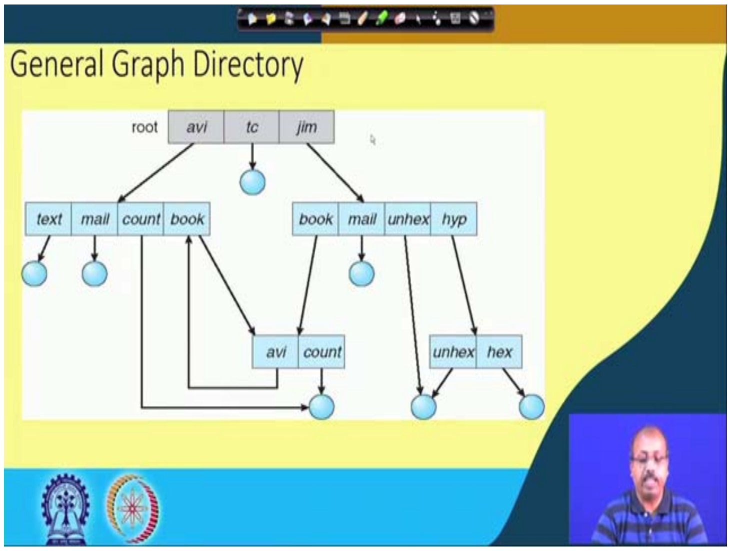
  Solutions:
  - Backpointers -- so we can delete all pointers.
    - Variable size records a problem
  - Backpointers using a daisy-chain organization
  - Entry-hold-count solution

If you want to delete a file so say is under this directory. So, dict count so, this file is deleted. So, naturally there will be dangling pointers that will come. So, solution is to have some back pointers so that we can delete all pointers and. So, variable size records is that is a problem and back pointers using a daisy-chain organization. So, this can also be used and we can entry hold count solution. So this is the best one basically what we try to do is that we try to see like how many how many such pointers are there and when that count becomes zero then only the file will actually be deleted.

Otherwise, what will happen is that if you delete this file. So, there so, from this side it is deleted. So, from this side this pointer becomes a dangling pointer so that is that is a problem. So, if we have got back pointer so that we can delete all pointer. So, if I have got a pointer also like this. So, apart from this one so, if I also have a pointer from the file to like this then once this file is deleted by this link then if we go by this back pointer to figure out like what are the duplicate entries and delete those duplicate entries also.

So, this is a problem, but naturally this size becomes variable. So it can be used as daisy-chaining structure that is there for interrupt processing and also not go into that because that will be very complex.

(Refer Slide Time: 13:11)

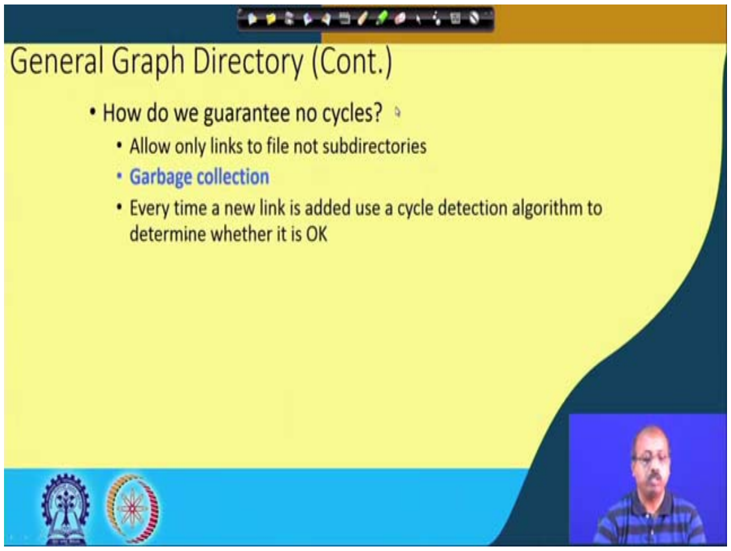


General Graph Directory

And we can we can have the general graph directory structure. So, there we can have the cyclic structures also like here in this case, you see that this avi so, this book is pointing to avi and then this avi is in turn avi is in turn pointing to this book so that way we have got a cyclic structure.

So, this is the most general form whether we follow it or not so that is a question. So, it is this is most general graph for the directory structure so that is not followed because in most of the cases what happens is that we go up to this acyclic graph structure and stop at that point. So, this directory structure, it uses the acyclic graph structure only does not going to cyclic graph.

(Refer Slide Time: 13:54)



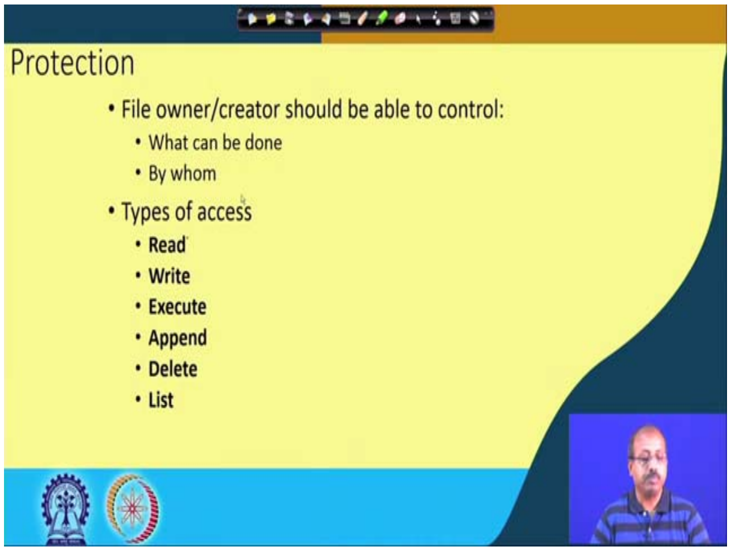
General Graph Directory (Cont.)

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - Garbage collection
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

So, how do we guarantee that there are no cycles? So, we can allow only links to the file not subdirectories so that is one possibility like in this previous case. So, from this we were pointing to a subdirectory so that way it was a problem. So, if that is not allowed then a cycle can be avoided or we can do some garbage collection is something is deleted then after sometime, we do we check all the pointer to see that they are all valid.

If something is not valid then we can remove that pointer so that way we can do some garbage collection and rectify the directory structure. Every time a new link is added, use a cycle detection algorithm to determine whether it is or not. So, that is very very costly operation I should say, but if you are really looking for this general graph directory structure then we have to do it like this.

(Refer Slide Time: 14:45)



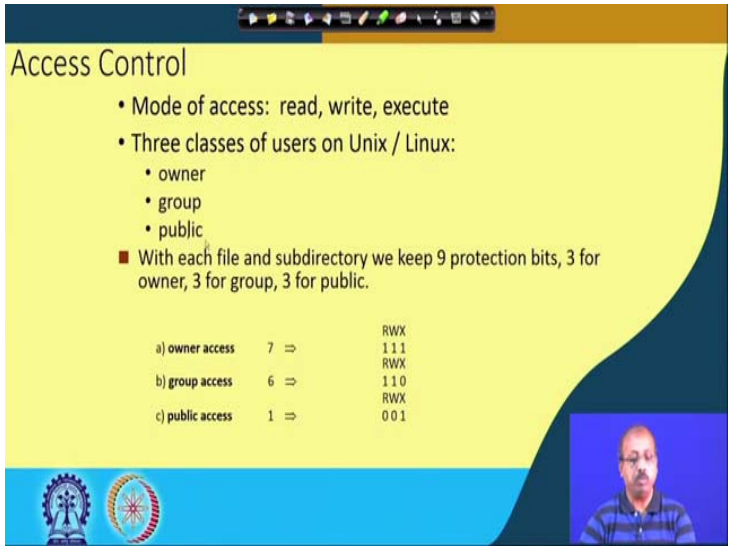
Protection

- File owner/creator should be able to control:
  - What can be done
  - By whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

Then to provide the protection so, file owner or creator should be able to control what can be done and by whom. So, what can be done so, that is the type of access read, write, execute, append, delete, list. So, I may want to read the content of the file, I may want to modify the file by during the write, I may want to execute a file, I may want to write something at the end of the file that is happened, I may want to delete the file or I may want to just list the content of the particularly the directory files so, list the content of that.

So, these are the different types of accesses that we have for a file considering both the general files and directory files. Now, out of these accesses for a particular file or directory, what are the accesses given to individual users so what can be done and who can do these things. So, these are the two things that we have to answer.

(Refer Slide Time: 15:40)



Access Control

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux:
  - owner
  - group
  - public
- With each file and subdirectory we keep 9 protection bits, 3 for owner, 3 for group, 3 for public.

a) owner access	7 ⇒	RWX 111
b) group access	6 ⇒	RWX 110
c) public access	1 ⇒	RWX 001

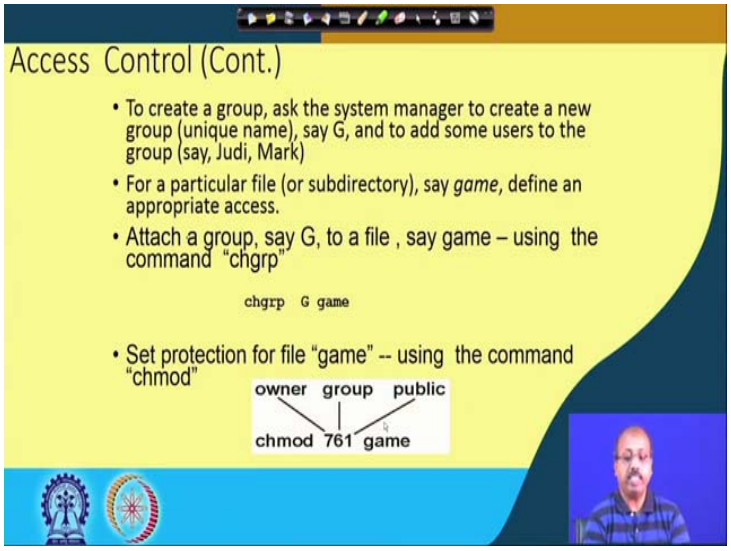
So, if you divided into this access into different groups the mode of access. So, there is a read, write and execute. So, these are the three different modes by which we can we can access a file. There are three classes of users particularly on Unix or Linux operating system. So, these users are divided into three groups owner, group and public.

So, owner actually who created the file and he owns the file and then the people who are close to that owner may be so they may be a development team and also they be able to they form a group and outside that all other so, they are the public. So, we may want to set different type of access rights for different classes of people. So, with each file and subdirectory, we can keep nine protection bits three bits for owner, three bits for group and three bits for public.

So, owner access so, they are called RWX bits in Unix operating system. So, if it is if the three bits as the octal number 7. So, these bits are all 111. So, 6 is octal number 110. So, this is so, I can say that for the owner, I give all read, write, execute permissions so, all the bits are 1. So, the octal number is 7.

So, group it is I want to give only the read and write, but no execute permission. So, that way the permission become 6 and for the public I want to give only the execute permission not read write permission so that way, it becomes 1. So, I give 7 to owner, 6 to group and 1 to public and that way I can set the access bits.

(Refer Slide Time: 17:21)



Access Control (Cont.)

- To create a group, ask the system manager to create a new group (unique name), say G, and to add some users to the group (say, Judi, Mark)
- For a particular file (or subdirectory), say *game*, define an appropriate access.
- Attach a group, say G, to a file , say game – using the command "chgrp"

```
chgrp  G game
```

- Set protection for file "game" -- using the command "chmod"

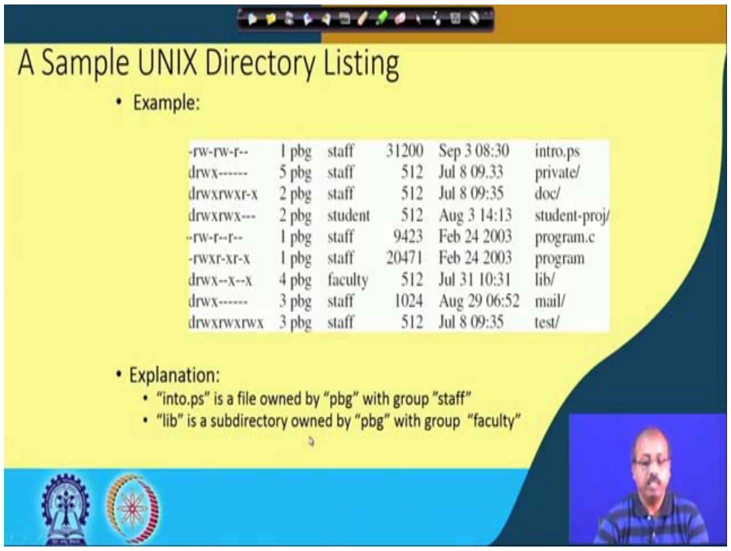
owner
group
public
chmod 761 game

To create a group, we can ask the system manager to create a new group in terms of name for example, G and add to some and add some users to the group. Say, maybe we have got two users Judi and Mark so they are added to say group G. For a particular file, say game, define an appropriate access. So, attach a group G to a file.

So, can we can we can we can we can by the commands chgrp. So, this is this comments are all taken from say Linux operating system. Change the group G games so, for the game file so, it is a group id is changed to G. So, this g becomes the group of this file.

Now, we can set the protection bits like this so, by the by this command chmod. So, chmod command is change mode. So, 761 so, 7 so, this is an octal number as I was explaining. So, the owner has got read, write, execute permission group has got only read and write permission and public has got only execute permission by means of this chmod command so, we can do that.

(Refer Slide Time: 18:28)



A Sample UNIX Directory Listing

- Example:

```
-rw-rw-r--   1 pbg  staff    31200  Sep 3 08:30    intro.ps
drwx------   5 pbg  staff      512  Jul 8 09.33    private/
drwxrwxr-x   2 pbg  staff      512  Jul 8 09:35    doc/
drwxrwx---   2 pbg  student    512  Aug 3 14:13    student-proj/
-rw-r--r--   1 pbg  staff     9423  Feb 24 2003    program.c
-rwxr-xr-x   1 pbg  staff    20471  Feb 24 2003    program
drwx--x--x   4 pbg  faculty    512  Jul 31 10:31   lib/
drwx------   3 pbg  staff     1024  Aug 29 06:52   mail/
drwxrwxrwx   3 pbg  staff      512  Jul 8 09:35    test/
```

- Explanation:
  - "into.ps" is a file owned by "pbg" with group "staff"
  - "lib" is a subdirectory owned by "pbg" with group "faculty"

If you look into a directory listing or Unix operating system. So, they may look like this. So, first few bits so they are actually identifying the read, write, execute permissions for the people users of the system. So, the first bit which is kept here as a dash and the for the few other entries, it is equal to d tells that whether the entry is a normal file or a directory. If the bit is shown as d; that means, it is a directory list directory file and if it is shown as a dash; that means, it is an ordinary file.

Now, for all types of file I can have read, write permission so, rw dash. So, owner has got only read write permission. So, the file is not executable as you can understand that this is a postscript file into dot ps the postscript file so that cannot be executed. So, execute permission is not set for group also it is read write and this for others public so, this is only read.

So, similarly, we can have other this thing. So, for directory structure so I can have read, write, execute permission. So, that will allow me to list down the directory also. So, so lib for example, is a subdirectory. So, lib is a subdirectory owned by the group pbg and the group faculty so, that way we can have this lib as a group. So, we can have this directory listing like that.