

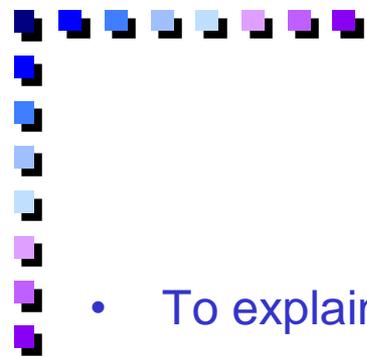


# Operating Systems ICS 431

## Week 12

### File System Interface

**Dr. Tarek Helmy El-Basuny**



## Objectives of this Chapter

- To explain the functions of the file system component of the OS,
- To describe the interfaces to the file system,
- To discuss the design issues of the file system,
- To explore the file-system sharing & protection methods.

# File-System Interface

- What is file system?
- File Concepts
  - File structure, File types, File attributes, File operations.
- File Access Methods
  - Sequential access, Direct access, Relative/Index access methods
- What is directory structure?
- Attributes of a Directory Structure.
- Operations Performed on a Directory.
- Directory Structure methods
  - Single level directory,
  - Two levels directory,
  - Tree structure directories,
  - Acyclic graph directories.
- File Sharing
  - Multiple users access and consistency semantics.
- Protection
  - Modes of Access
  - Access Control Matrix

# What is File System?

- It is a component of the operating system which consists of:
  - **An interface** that allows the users to store, retrieve and update a set of **files** and **directories**.
  - The **data structures and algorithms** used to implement that interface.
- From a user's point of view, **the file system is perhaps the most visible part** of the OS because:
  - The user wants quick access to the files,
  - The user needs to guarantee that the files will not be corrupted,
  - The user needs to secure the files from non authorized access.
- **Functions of the File System:**
  - Identifies and locates a selected file or directory,
  - Organizes the location of all files plus their attributes,
  - Supports user access control on a shared file system,
  - Allocates files to free disk space,
  - Manages free storage on the disks for available files,
  - Manages the buffering blocks in main memory,
  - Grouping files into separate **collections/folders/directories**.
  - Interacts with the end user.

# File Structure

- A **file** is a collection of data stored in one unit, and identified by a **filename**. or
- A **file** is a collection of letters, numbers and special characters with a name associated to it called "**filename**".
- **File Concepts:**
  - File Structure, File Types Implementation, File Attributes, File Operations.
- **File structure** usually defined by the **SW that creates the file**. It may be a **source code file**, a **database file**, a **document file**, a **text file**, a **media (audio or video) file**, an **image file**, a **library file**, etc ...
  - **A text file:**
    - Contains a sequence of characters, numbers and special characters organized into lines.
  - **A source code file:**
    - Contains a collection of code, possibly with comments, written using a certain programming language and specifies the actions to be performed by a computer.
  - **An object file:**
    - Contains a sequence of bytes organized into blocks understandable by the system's linker and is usually not directly executable.
  - **An executable file:**
    - Contains a series of code sections linked with the needed libraries that the loader can bring into memory for execution.

# File Type Implementation

- A common technique for implementing the file type **is to include the type as a part of the file identity**. The extension helps the user and OS to know the type of a file.
- Using this technique helps the OS to avoid some inconvenient commands.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	CompuServe Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

## File Attributes

- A file has certain attributes, which vary from one OS to another but typically include:
- **Name:** is a string of characters, some OS differentiate between upper and lower characters while others no, **file name is the only information kept in human-readable form.**
- **Type:** is needed for operating systems that support different types.
- **Location:** is a pointer to the storage device and to the location of the file on that storage device.
- **Size:** the current file size (in bytes, words or blocks), and the maximum allowed size of a file are included in this attribute. **Is there a limit of the file size?**
  - If the drive containing your file is formatted with NTFS (which is the default in Windows 7), then the maximum file size is **16 TB.**
- **Protection/access control:** this attribute controls who can do reading, writing, and executing the file.
- **Time, date, and user identification:** time and date of creation, last modification and usage. This data is used for protection, security, and usage monitoring.
- All file's attributes are stored in the "**Files Control Block**" (FCB)
- **Directory structure:** keeps information about files maintained on the disk.
- Each file has an entry consists of a file name and its unique identifier that identifies the other attributes of the file.
- The size of the **directory structure** itself **may be megabytes.**

## More File Attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

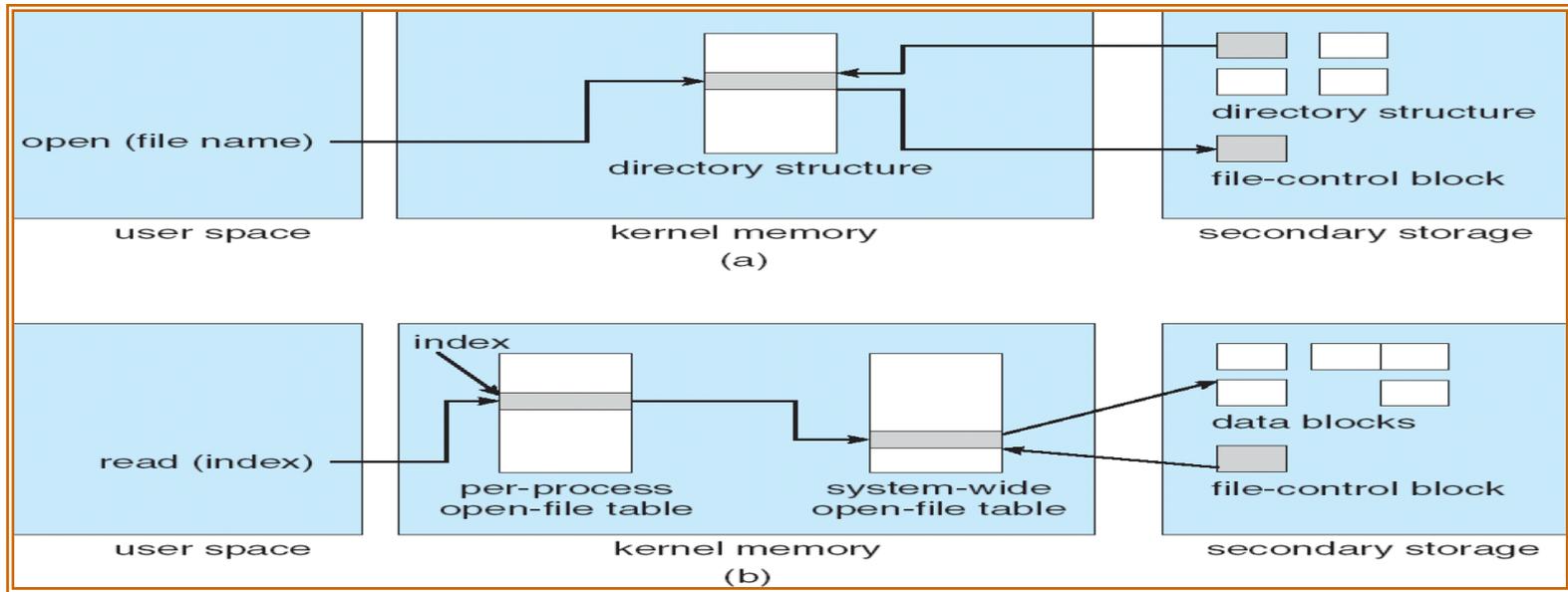
## File Operations

- The operations that can be performed by the OS on files are:
- **Create a file:** A **space** in the file system must be found for the file and an **entry** for the new file must be made in the **directory structure**.
- **Write a file:** A **system call** specifies both the name of the file and the information to be written. The OS searches the directory to find the location of the file and writes its information. **The OS needs to keep a write pointer.**
- **Read a file:** A **system call** specifies both the name of the file and where in memory the next block of the file should be put. **The OS needs to keep a read pointer.**
- **Delete a file:** Searches the directory structure for the file, having found it then release all the file space so that it can be used by other files.
- **Truncate a file:** Means delete the content of the file but keeps its attributes with the file length changed.
- **Open(F<sub>i</sub>):** Searches the directory structure on disk for the entry F<sub>i</sub>, and moves the content of that entry to memory (**open file table**).
- **Close (F<sub>i</sub>):** Moves the content of the entry F<sub>i</sub> in memory to directory structure on disk.
- Other operations like: **Renaming a file, changing the attributes, appending data, etc...**

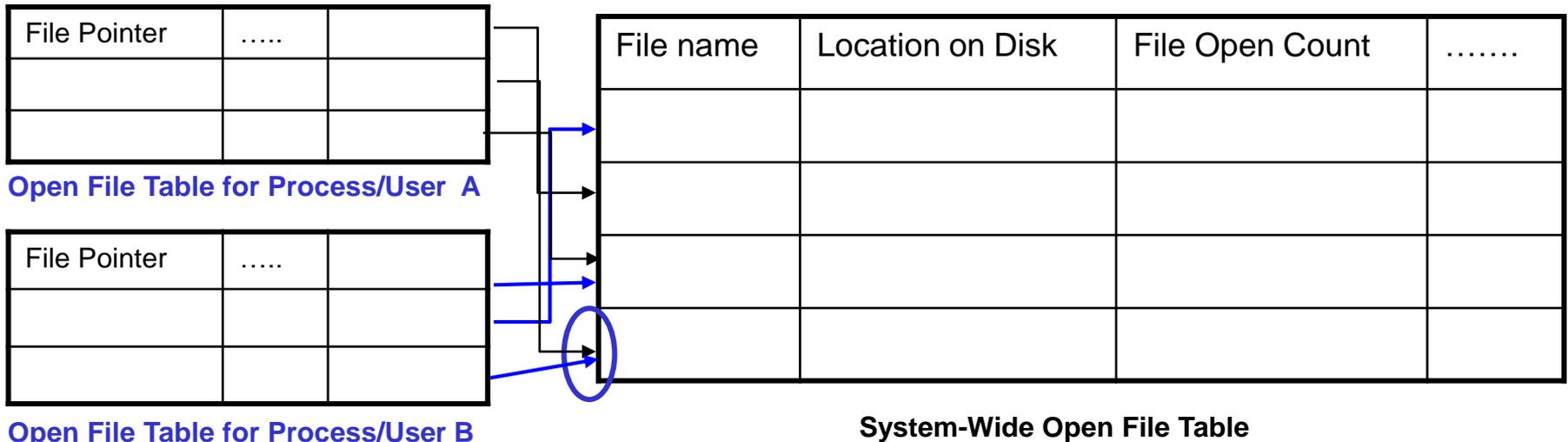
## Open File Table

- Most of the file operations mentioned involve searching the directory for the entry associated with the identified file.
- To avoid the time of searching the disk, the OS keeps in MM a small table containing information about all opened files (open-file-table).
- When a file operation is required, the file is specified via an index into this open-file-table, so no searching on disk is required.
- When the file is closed, the OS removes its entry from the open-file-table.
- The implementation of open-file-table only in multi-user or multi-processing systems like Unix is not enough. Multi-users may open the file at the same time. In this case the OS creates two levels of tables: a per-process/user open-file-table and a system-wide open-file-table.
- There is an entry for the open file in the per-process/user table and points to another entry within the system-wide table.

# Open File Table



(a) refers to opening a file. (b) refers to reading a file.



## Open File Table Contents

- Several pieces of information are associated with the open file:
- **File name:** The name of the file.
- **File pointer:** The system must track the last read-write location as a current file pointer position.
- **File open count:** As files are closed, the OS must reuse its open file entries (or it can run out of space in the table). Multi-processes/users may open a file, the OS must wait for the last file to close before removing the entry from the open file table. This counter tracks the number of opens and closes.
- **Access rights:** The access mode of each file by the process is stored on the per-process table so the OS can deny or allow the I/O requests.
- **Disk location of the file:** Most file operations require the system to modify data within the file. The information needs to locate the file on disk is kept in memory to avoid reading it from the disk every time.

index	file name	permissions	access dates	pointer to disk block
0	TEST.C	rw rw rw	...	→
1	MAIL.TXT	rw		→
2				
.				
.				
.				
n				

## Creating a File

- To create a new file, the SW application calls the file system, the file system then:
  - Allocates a new File Control Block (FCB),
  - Reads the appropriate directory into memory,
  - Updates the directory with new file name and a pointer to its FCB,
  - Writes it back to the disk for consistency.
- Some operating systems (UNIX) treat a directory exactly as a file by using the same system call,
- Other operating systems (Windows) implement separate system calls for files and directories and treat directories separate from files.

## Opening a File

- To open a file for any I/O operation:
  - The **open system call** passes the file name to the file system.
  - The directory structure (**usually cached**) will be searched for the given file name,
  - Once the file is found, the FCB is copied into the **system-wide open-file** table in memory,
  - An entry is made in the **per-process open-file table**, with a pointer to the **system-wide open-file table**,
  - The **open system call** returns a pointer to the appropriate entry in the per-process open-file table,
  - All file operations are performed via this pointer (**file descriptor in Unix, file handle in Windows**).

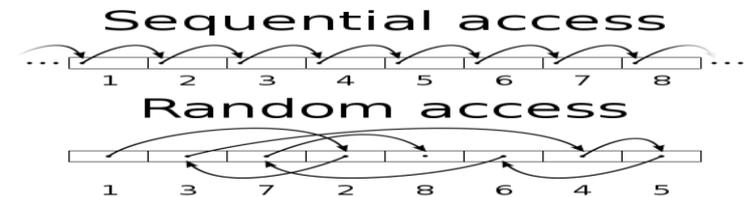
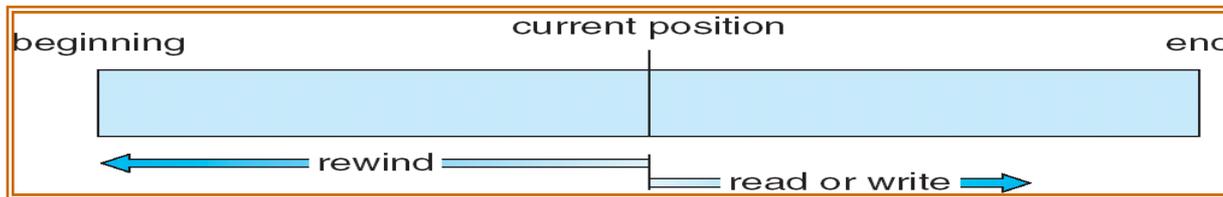
## Closing a File

- After completing all I/O operations on a file, it should be closed.
  - The **per-process table entry** will be removed and the system-wide entry's **open count will be decremented**.
  - When all users that have opened the file close it, the updated file information is copied back to **the on disk directory structure** and **the system-wide open-file table entry** is removed.
- Some systems use a caching scheme where all information about an open file, except for its actual data blocks, is in memory.

# File Access Methods

- The file's information must be read into memory when it is used. OS provides one or more access methods.
- **Sequential access**
  - File's content is read/written in sequential order, one record after the other.
  - To access 50<sup>th</sup> record in a file, it must read the record 49<sup>th</sup> first.
  - File's read/write pointer specifies file location for each read/write.
  - It is the most common mode of **compilers**.

• **Sequential Access**  
read next  
write next  
reset  
no read after last write



- **Direct (Random) access**
  - Blocks can be read/written in random order, we may read block #14 then 7.
  - It is useful for immediate access to large amount of information.
  - Application specifies file location for each read/write, i.e. **DB files**.
  - The file operation must include the **block number** as a parameter.
  - The **block number** is used as an index to find the beginning of the file.
- Accessing data sequentially is much faster than accessing it randomly because of the way in which the disk hardware works.

## Sequential Access on a Direct/Random-Access File

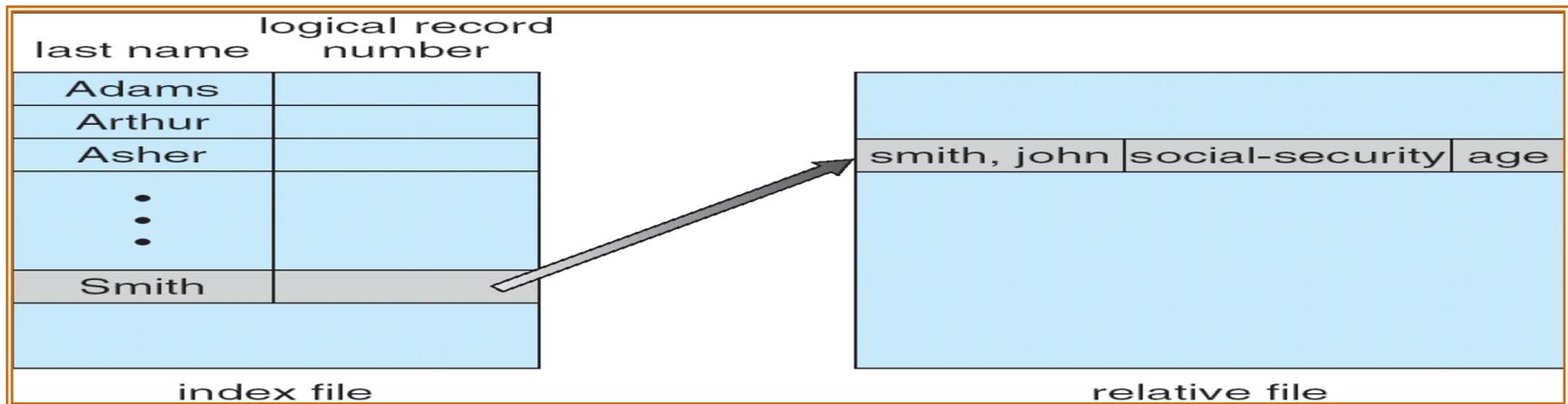
- Sequential Access to files: to reach a particular record, all the preceding records must be sequentially read.
- Direct Access to files: we can reach to a particular record in the file directly and this facilitates the operations of reading, deleting, updating and inserting records.
- OSs support both Direct and Sequential file access.
- If we simply keep a variable **Current Position "CP"** that defines the start of reading or writing, then we can simulate sequential file operations on a direct access as followings:

sequential access	implementation for direct access
<i>reset</i>	<i>cp = 0;</i>
<i>read next</i>	<i>read cp;</i> <i>cp = cp + 1;</i>
<i>write next</i>	<i>write cp;</i> <i>cp = cp + 1;</i>

- **Direct Access**  
**read  $n$**   
**write  $n$**   
**position to  $n$**   
**read next**  
**write next**  
**rewrite  $n$**

## Relative Access Files

- **Relative Access files:** in order to support random as well as sequential file access. The OS uses an index table contains records that represents the location of file relative to where the file begins.
- The **index table** contains pointers to various blocks.
- The OS first searches the **index table** and then uses the pointer to access the desired block of file direct.
  - The user enters key field
  - Disk address computed from key field
  - The record then accessed directly.

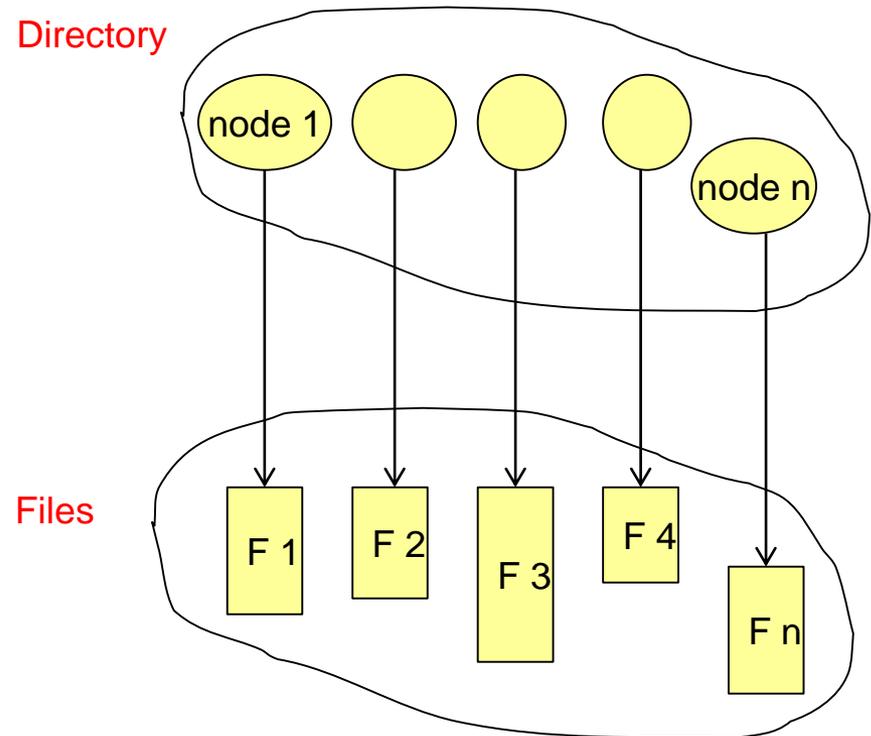


# Directory Structure

- **Directory Structure** is a collection of nodes containing information about all files in the file system.
- It describes the way in which the files are organized.

- **Attributes in a Directory Structure**

- Name
- Type (rar, zip, etc.)
- Address/Location
- Current length
- Maximum length
- Date of last access
- Date of last update
- Owner ID
- Protection information



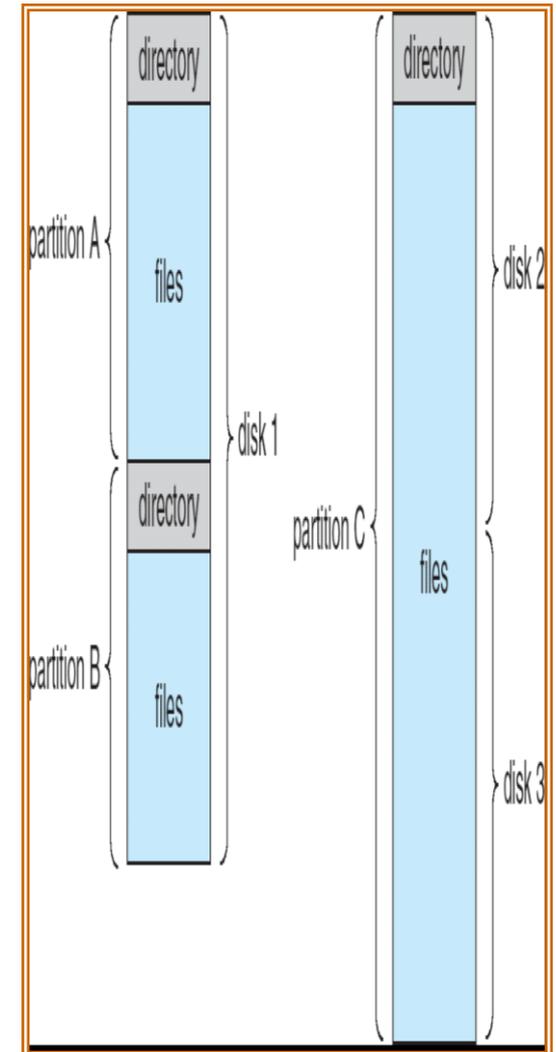
- Both the Directory Structure and the Files Control Blocks reside on disk.
- Backups of these two structures are cached on memory for fast access.

## Operations Performed on a Directory

- **Search/Navigate for a file:** Search a directory to find an entry of a particular file.
- **Create a file in a directory:** New file can be created and added to the directory.
- **Delete a file from a directory:** When a file is no longer needed, we can remove it from the directory.
- **Append a file into a directory:** Adding/Moving a file into the directory.
- **List files in a directory:** List the files in a directory.
- **Rename/resize a file:** The name/size of the file must be changeable to allow changing if the name/contents have been changed.
- **Traverse the file system:** How many files and directories are available within a directory.
- **Backup:** Save regularly the contents of the entire file system to auxiliary memory to avoid troubles in case of a system failure.

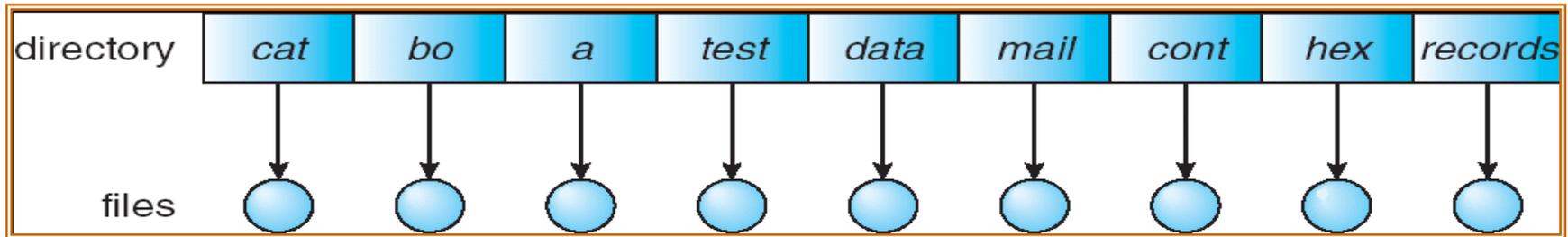
# Organize the Disk (Logically) into Partitions and Directories

- Why do we divide disks into partition?
- With multiple partitions:
  - You may install different OSs on your machine.
  - You can mount one or more of your partitions as read-only.
  - If something happen to a file system on a partitioned system you would probably only lose files on a single file system.
  - It reduces the time required to perform file system checks.
  - It minimizes the seek time.
- Why do we create directories on disks?
- **Efficiency:** Locating a file quickly.
- **Avoid naming conflict:**
  - Two users can have same file name for different files. The same file can have different names.
- **Grouping:** Logical grouping of files by properties, (e.g., all Java programs, all games, )



## Single-Level Directory Structure

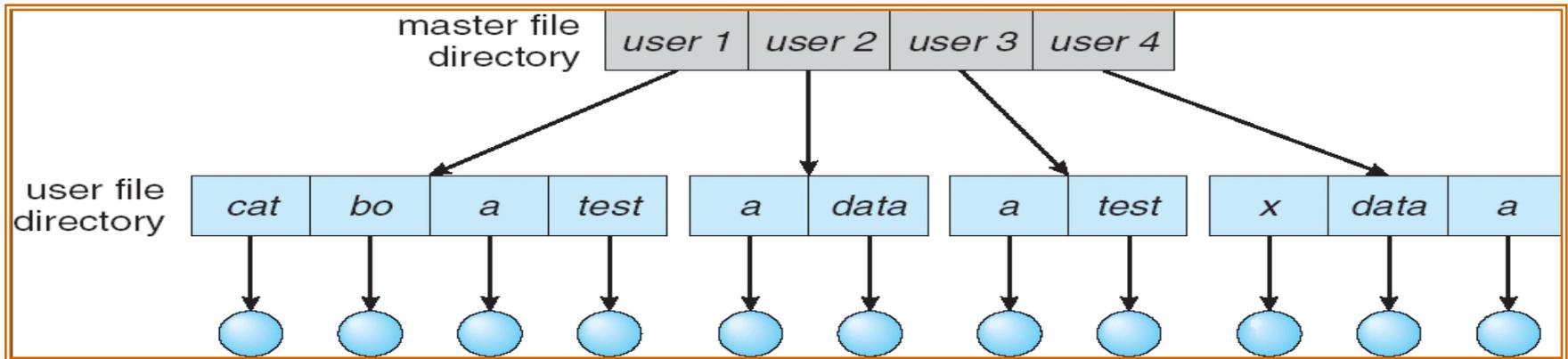
- Only files are there in the directory.
- Advantages:
  - The simplest directory structure,
  - Easy to implement, and understand.



- **Disadvantages of Single-Level Directory :**
  - Naming problem when the number of files increase **or the system has more than one user.**
  - Grouping problem: **hard to group related files**

## Two-Level Directory

- To avoid the naming confusion, a separate directory for each user should be created.
- Each user has **User's File Directory (UFD)**.
- Each **UFD** lists only the files of a single user.
- When a user's job starts, the system's **Master File Directory (MFD)** is searched.
- The **MFD** is indexed by user's name or account, each entry points to user's **UFD**.
- Files names are unique within the **UFD** only, so different users may have the same file names in their **UFDs**.
- Two-level directory can be thought of as a tree. The root is the MFD and the leaves are the UFD, the descendents of the UFD are the files.
- Each file in the system has a path consists of the user's name & file's name.
- Efficient searching: This structure effectively isolates one user from another



# Tree-Structured Directories

- Two-level directory can be extended to multi-level tree structure.
- A directory contains a set of relevant files or subdirectories.
- A directory is simply another file but treated in a special way.
- One bit in each directory entry defines the entry as a file or as a subdirectory.
- The current directory contains the files that are of current interest and will be searched first for any file reference.
- Handling (deleting/creating) directories controlled by some system calls.

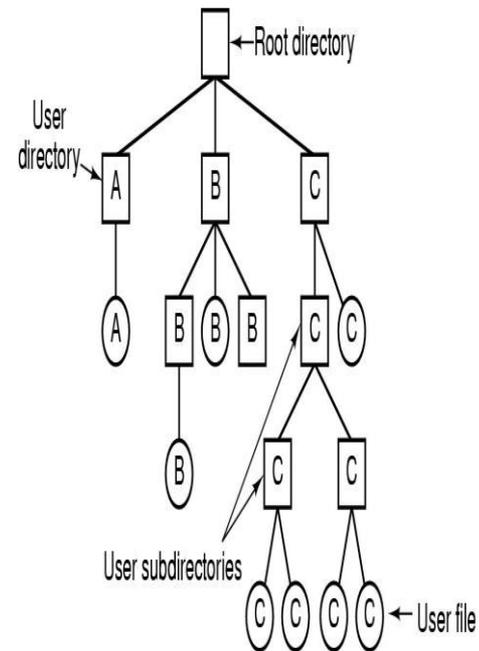
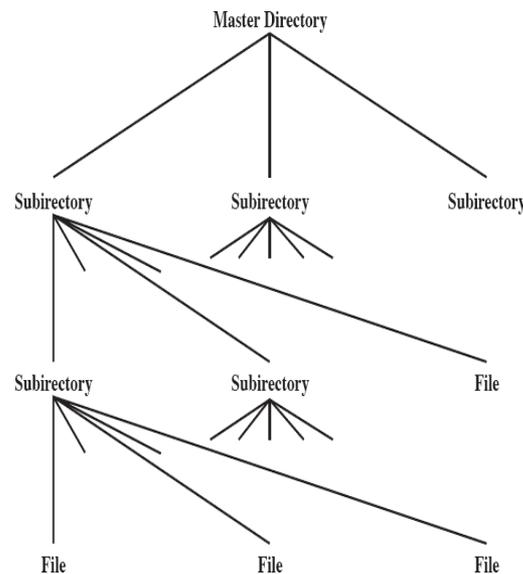
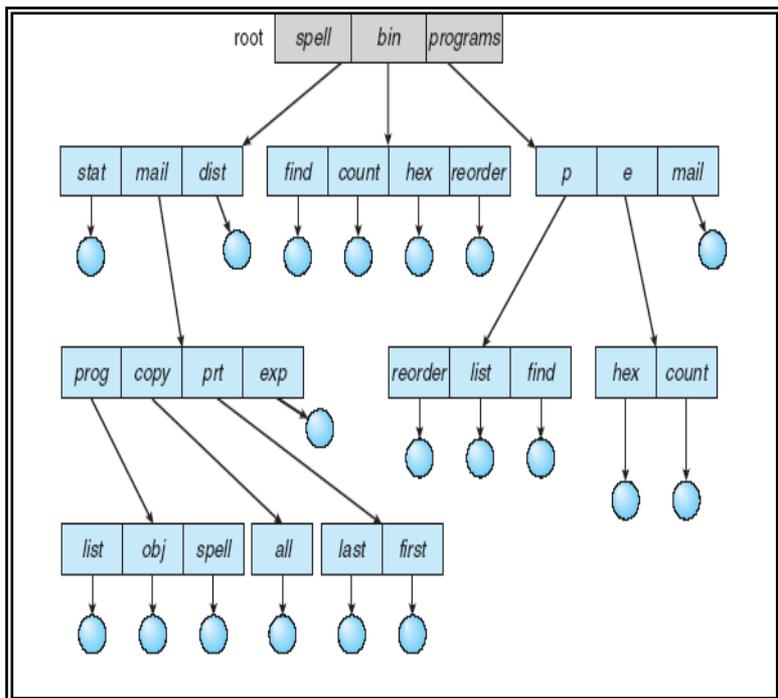


Figure 12.4 Tree-Structured Directory

# Example of Tree-Structured Directory

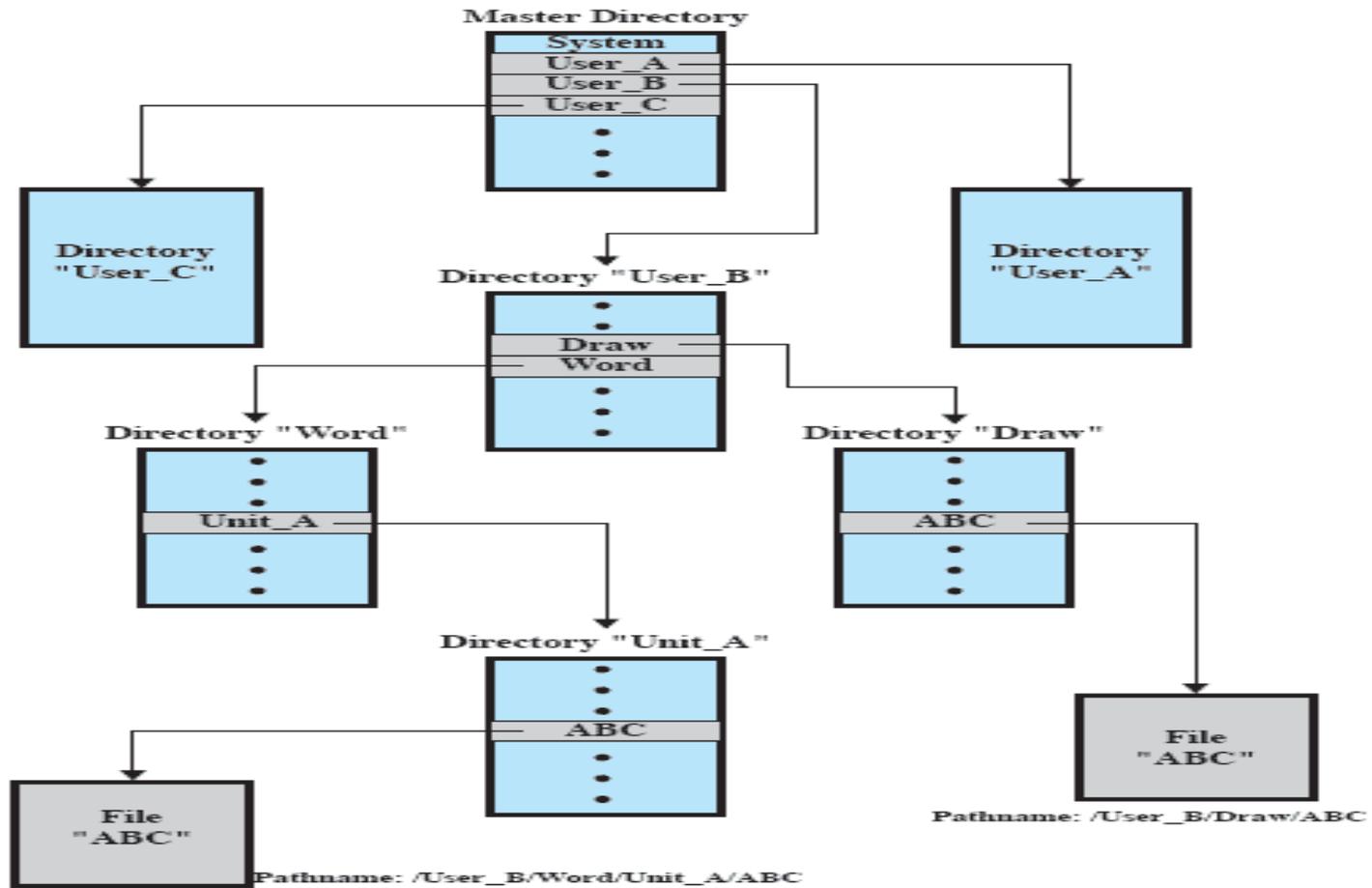


Figure 12.5 Example of Tree-Structured Directory

# Tree-Structured Directories

- To access a file, the user should either:
  - Go to the directory where file resides, or
  - Specify the **path** where the file is
- **Path names** can be **absolute** (given the path starting from the root, i.e. helmy/ICS 431/week12-FS.ppt) or **relative** (giving the directory name on the path, i.e. week12-FS.ppt if we are in the same directory).
- Creating a new file is done in current directory
- Delete a file

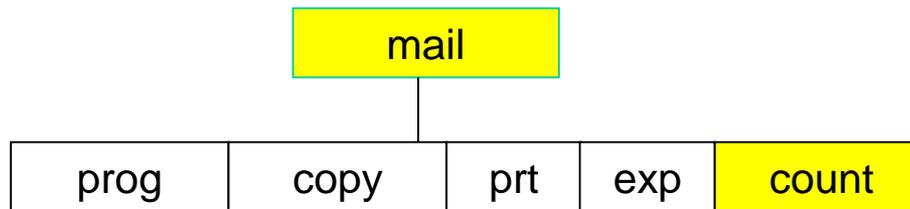
`rm <file-name>`

- Creating a new subdirectory is done in current directory

`mkdir <dir-name>`

Example: if in current directory `/mail`

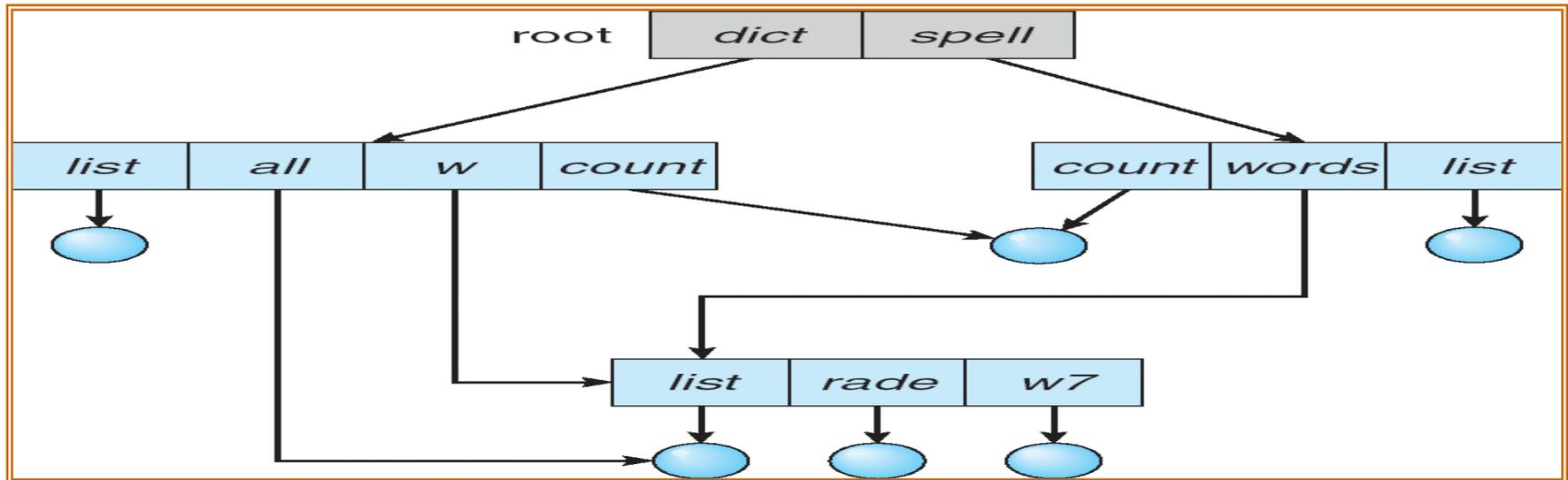
`mkdir count`



Deleting “mail”  $\Rightarrow$  deleting the entire sub-tree rooted by “mail”

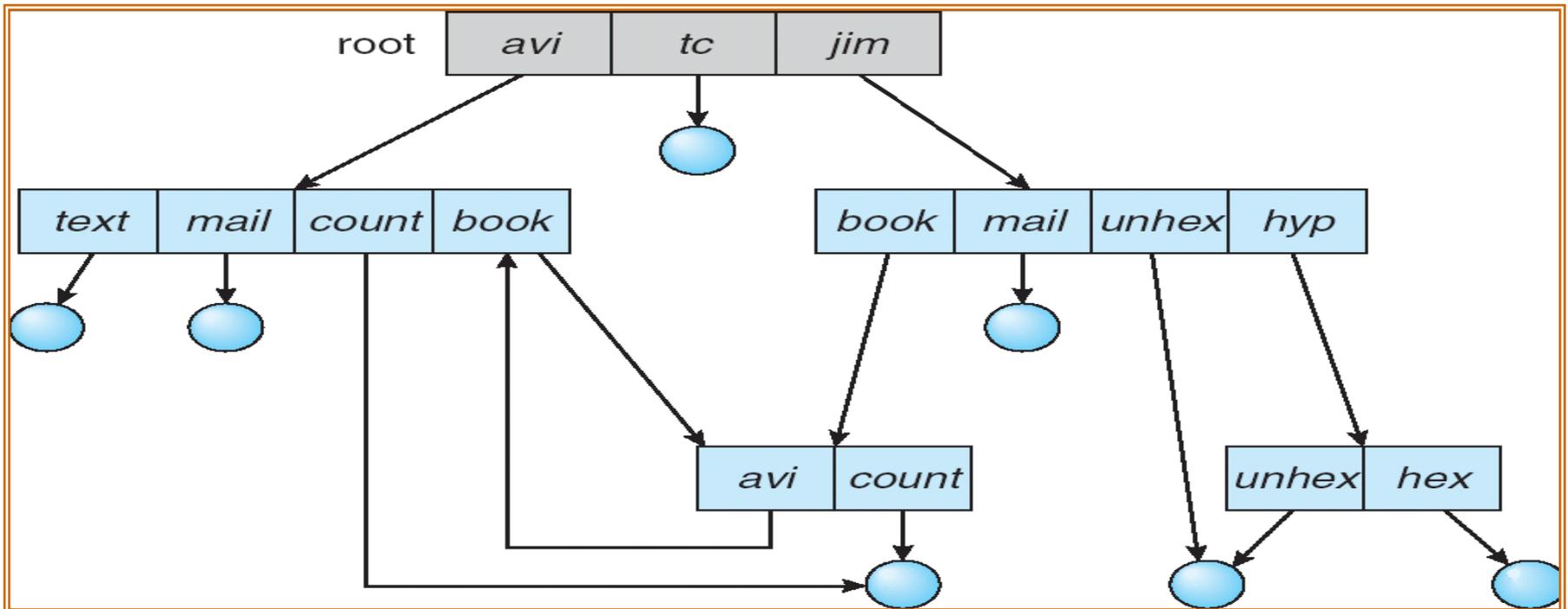
# Acyclic-Graph Directories

- A tree structure prohibited sharing of files or directories.
- Sharing is important as some users may share the same projects and want to access the same files or directories without recreating them again.
- The acyclic-graph directory allows sharing of subdirectories and files.
- **With Acyclic-Graph Directories:**
  - Only one actual copy of the file or directory is stored, so any changes made by one person will be immediately visible for the other.
  - Can be accessed through one or more paths.
  - The deletion of a link should not affect the original file; only the link is removed.



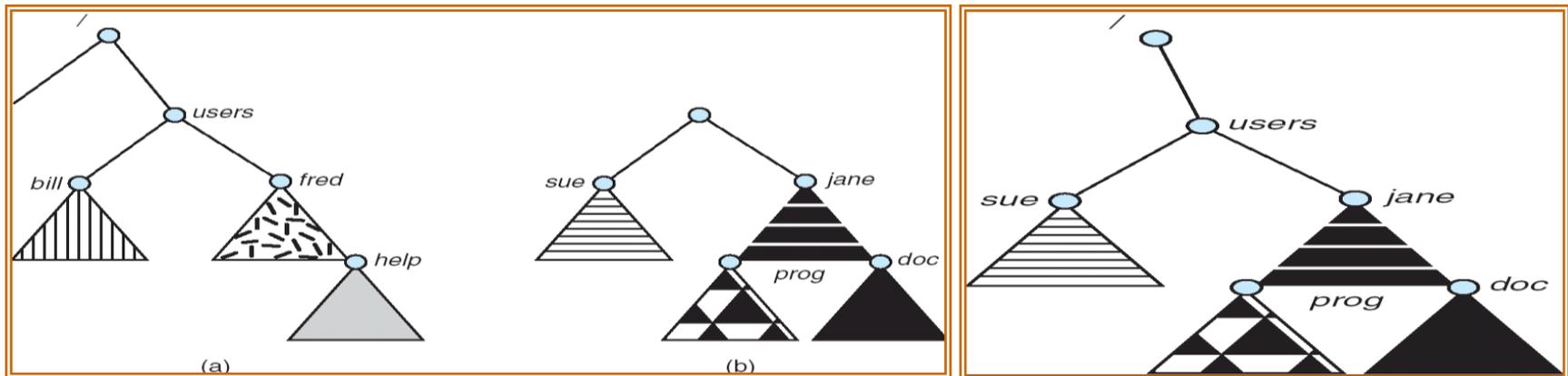
## Problem of Acyclic Graph Directory

- One serious problem: with using acyclic-graph; is having cycles.
- How do we guarantee that no cycles will be exist? to avoid wasting the time of searching again and again.
  1. Allow only links to files not subdirectories, or
  2. Every time a new link is added, use a cycle detection algorithm to determine whether cycle will be created or no.



# File System Mounting

- **Mounting** is a process by which the **OS** makes **files** and **directories** on a storage device available for user to access via the computer's **file system**.
- Mount allows two file systems to be merged into one. **i.e. when you insert a flash derive.**
- In some OSs like Unix, and Linux, a file system must be mounted before it can be accessed.
- The **mount command** will pass the kernel three pieces of information; the name of the file system, **the physical block device which contains the file system** and, **where in the existing file system topology the new file system is to be mounted.** **In Linux,**
- **Mount-type device dir** (type-device menas the drive, dir menas the directory to be mounted)
- **umount** detaches the specified file system(s) from the file hierarchy



(a) Existing.

(b) Un-mounted Partition

Mount point: /users

## File Sharing

- Sharing of files on multi-user systems is recommended.
- Sharing may be done through:
  - **User IDs** identify users, allowing permissions and protections to be per-user
  - **Group IDs** allow users to be in groups, permitting group access rights
- Networking allows file system access/sharing between systems
  - Manually through **programs like FTP**
  - Automatically through **distributed file systems**
  - Semi automatically through the **world wide web**
- **Consistency semantics: it is an important aspect in evaluating any file system that support file sharing.**
  - Unix allows any writing to the shared file to be visible immediately to other users.
  - Some OSs allow any modification to be visible in the next access sessions.
  - Some OSs allow immutable shred file, once the file declared as immutable, it can not be modified.

## File Sharing Implementation

- **A common way to implement shared files and subdirectories is to create a new directory entry called a link.**
- A link is a pointer to **another file** (indirect pointer) or **subdirectory**, it may be relative or absolute.
- When a reference to a file is made, the OS searches the directory, if the directory entry is marked **as a link**, then the name of the real file or directory is given.
  - Good for linking files on other machines and save disk space.
- **Disadvantages**
  - Takes longer to lookup as the path is traversed
  - The file pointed to can be deleted or changed, **if files have been deleted, links should be also deleted to avoid dangling pointers.**
  - Backup might do multiple copies
- **Another common way of implementing shared files is to duplicate all information about them in both sharing directories.**
- The problem here is to maintain consistency if the file is modified.

## Protection & Reliability

- File owner/creator should be able to keep it safe from physical damage (**reliability**) and improper access (**protection**).
- **Reliability**: Can be provided through duplicate copies of files regularly.
- **Protection**: Can be provided in many ways like prohibited access or access list with some operations may be controlled:
  - **Read**: read from the file
  - **Write**: write the file
  - **Execute**: load the file into memory and execute it.
  - **Append**: write new information at the end of the file
  - **Delete**: delete the file and release its space for reuse.
  - **List**: list the name and attributes of the file

## Access Lists and Groups

- **Mode of access:** read, write, execute.
- Consider an author who is writing a book, he has three co-authors to help him, the text of the book is saved in a file named **book**. The protection of the file **book** is as follows:
  - Author should be able to invoke all operations on the file.
  - Coauthors should be able to read and write but not delete the file.
  - All other users should be able to read but not write.
  - The Unix/Linux defines 3 bits R (**read**) W (**write**) X (**execute**) or – means nothing to control the access and 3 bits to define the users, i.e. O (**owner**) G(**group**) U(**universe/all**).
- Three classes of users

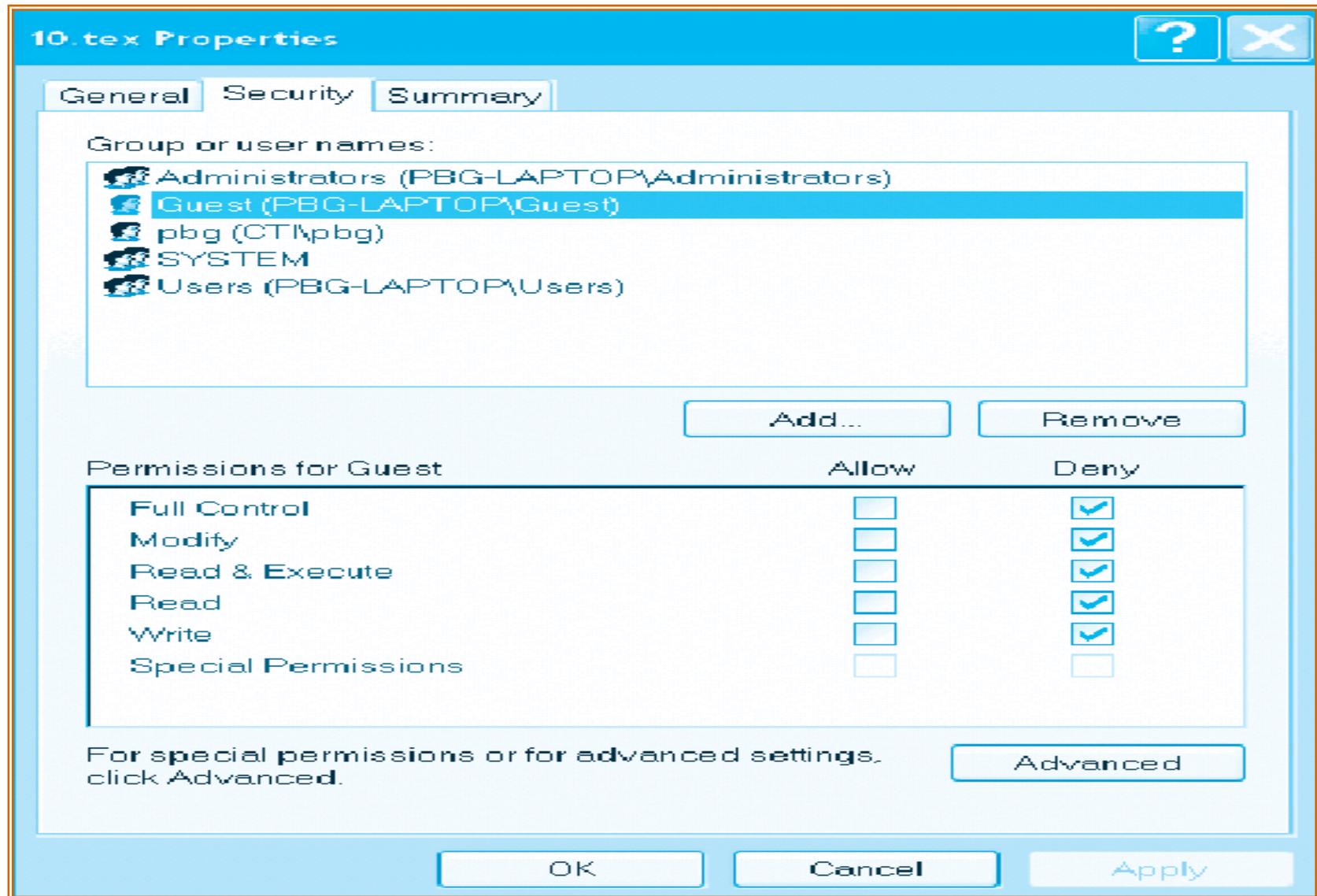
	RWX
a) <b>Owner access</b> ⇒	1 1 1
	RWX
b) <b>Group access</b> ⇒	1 1 0
	RWX
c) <b>Universe access</b> ⇒	0 0 1

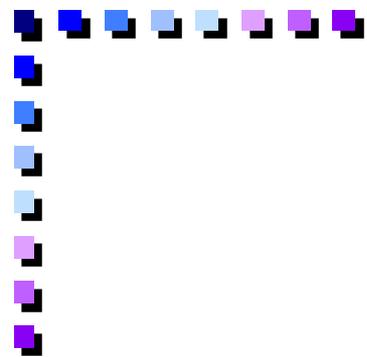
## A Sample Unix/Linux Directory Listing

-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/

- **Let us explain the meaning of each entry in the above table.**
- In the first position, **-** stands for a file while **d** stands for a directory
- **rw-** refers to the authority of the **owner**, who can read and write only. Followed by **rw-** authority of the **group** then **r--** the authority of the **universal** users who can only read. Note that permission can be changes **using chmod command**.
- **1** refers to the number of memory blocks
- **pbg** is the name of the owner followed by the **staff** as a group name. **chown** used to change the owner of a file while the **chgrp** command used to change the group name of the file owners.
- 31200 are the number of bytes used in that file.
- Next is the date of creation and the name of the file.

# Windows Access-Control-List Management





**The End!!**

**Thank you**

**Any Questions?**