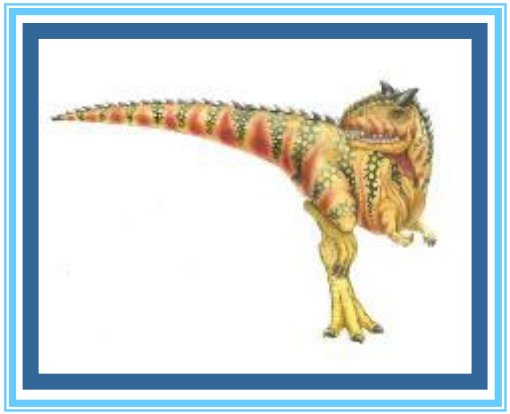
Chapter 10: File-System

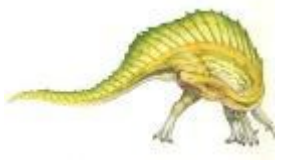


Chapter 10: File-System

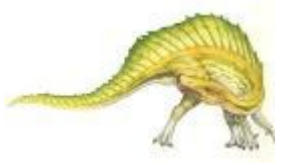
- 10.1 File Concept
- 10.2 Access Methods
- 10.3 Directory and Disk Structure
- 10.6 Protection
- 10.7 Summary

Objectives

- To explain the function of file systems.
- To describe the interfaces to file systems.
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures.
- To explore file-system protection.



10.1 File Concepts



File Concept

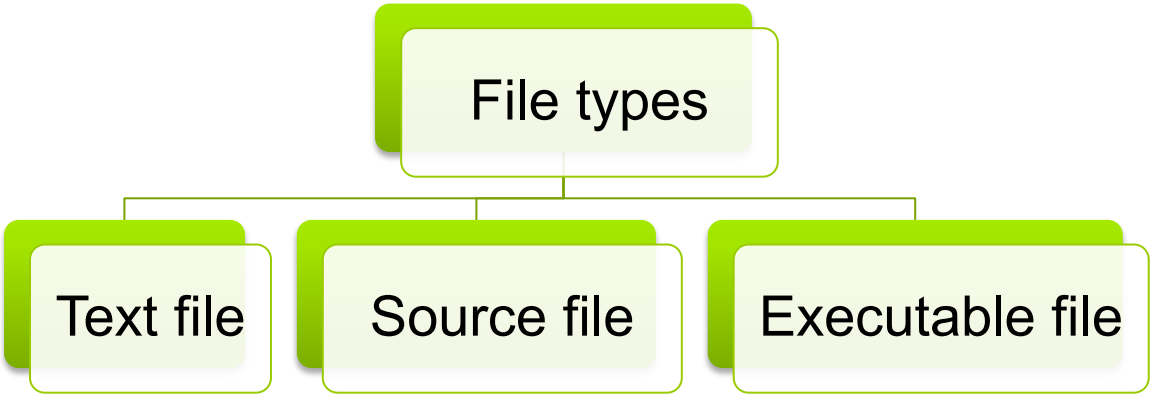
Overview

- Computers can store information on various storage media:
  - *e.g.* magnetic disks, magnetic tapes, and optical disks.
- Operating system:
  - provides a uniform logical view of stored information.
  - abstracts from physical properties of storage devices to define a logical storage unit → **File**.
  - maps the file onto physical devices.

A **file** is a named collection of related information that is recorded on secondary storage.

File Concept

- Contents (many types) is defined by file's creator.



- **Text file** : A sequence of characters organized into lines (and possibly pages)
- **Source file** : A sequence of functions the followed by executable statements.
- **Executable file** : A series of code sections can be loaded in memory and execute.

File Concept

Definitions

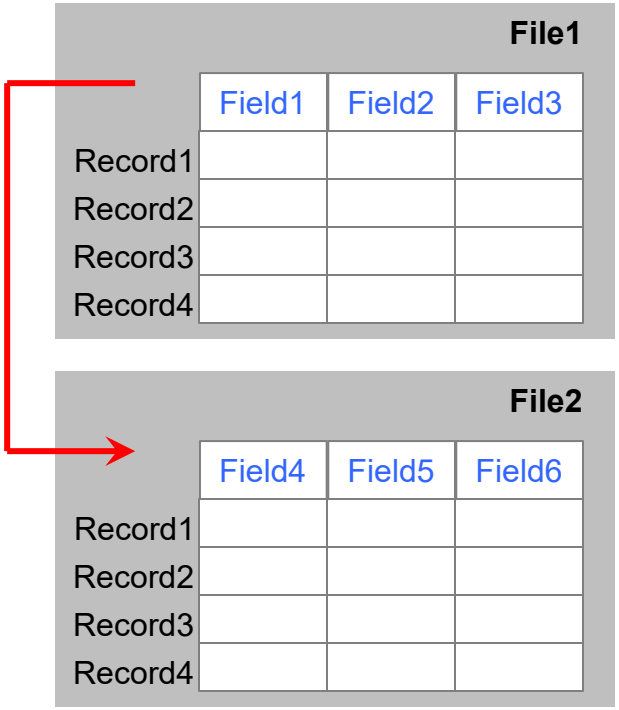
- **Field**
  - Group of related bytes.
  - Identified by user (name, type).
- **Record**
  - Group of related fields.
- **File**
  - Group of related records that is stored in secondary storage.
  - Contain information used by specific application programs to generate reports.
  - Sometimes known as *flat file* because it has no connections to other files.

File1

	Field1	Field2	Field3
Record1			
Record2			
Record3			
Record4			

File Concept

- **Databases**
  - Groups of related files that are interconnected at various levels to give users flexibility of access to stored data.



- **Program files**
  - Contain instructions.

- **Directories**
  - Listings of filenames and their attributes.
  - Attributes are information related to a files.
  - Organize files into a logical structure.

File Concept

File attributes

- File attributes are stored in a directory.
- Attributes that are commonly used are:
  - **Name** – only information kept in human-readable form.
  - **Identifier** – unique tag (number) identifies file within file system.
  - **Type** – needed for systems that support different types.
  - **Location** – pointer to file location on the device (disk).
  - **Size** – current file size.

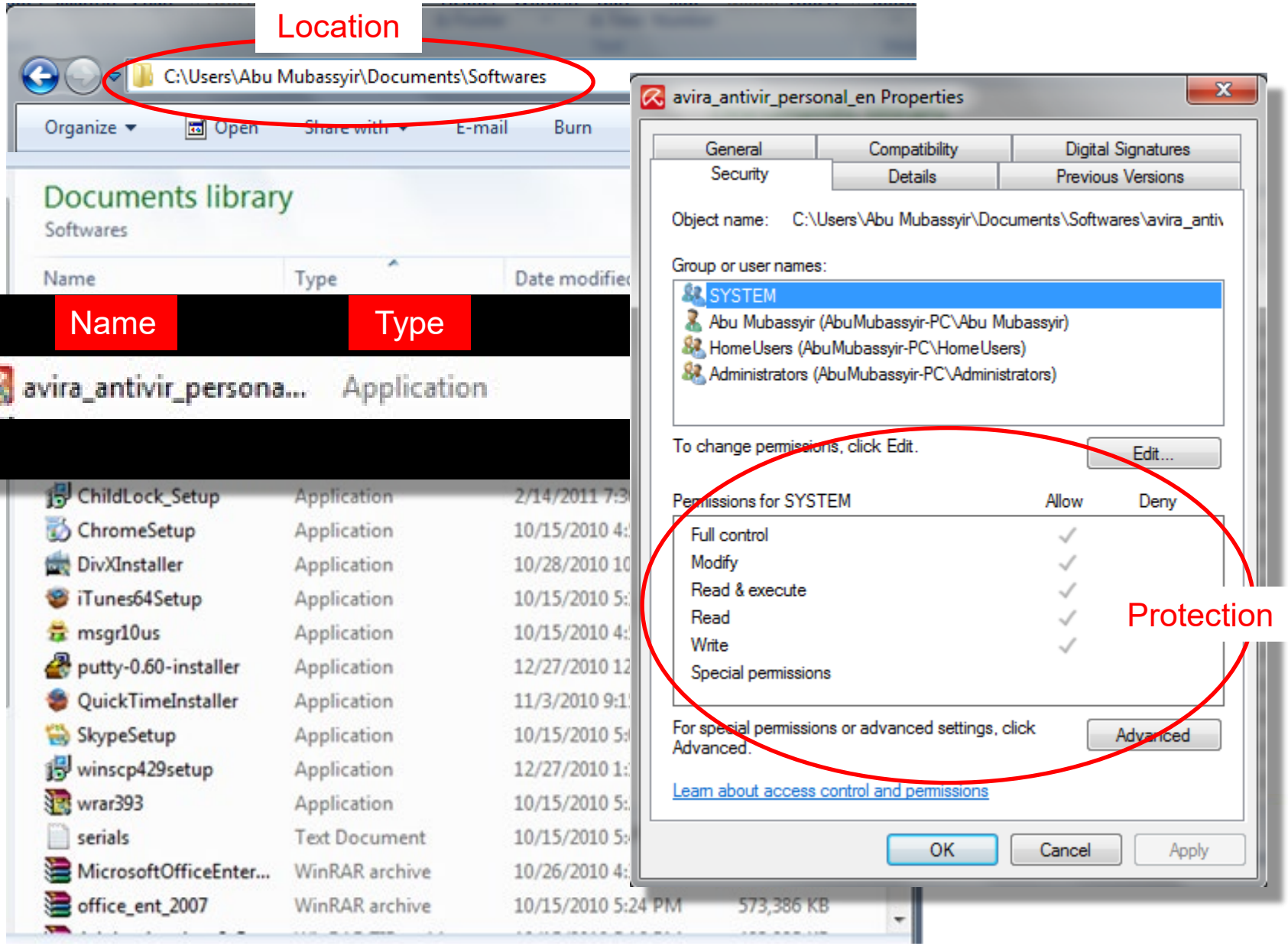
File Concept

- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – information kept for creation time, last modification time, and last use time.
  - Useful for data for protection, security, and usage monitoring.

- Many variations, including extended file attributes such as file checksum.
- Information kept in the directory structure (on disk), which consists of “inode” entries for each of the files in the system.

10.1

File Concept



File Concept

File operations

- **Create**
- **Write –** at write pointer location
- **Read –** at read pointer location
- **Reposition within file -** seek
- **Delete**
- **Truncate**
- `Open` (F_i) – search the directory structure on disk for inode entry F_i, and move the content of the entry to memory
- `Close` (F_i) – move the content of inode entry F_i in memory to directory structure on disk.

File Concept

File types (name, extension)

- **Extensions:**
  - Appended to relative filename.
    - Two to three characters.
    - Separated by period.
    - Identifies file type or contents.

- Example: `TUNE.MP3,Menu.doc, slide.ppt`
- Unknown extension:
  - Requires user intervention.

File Concept

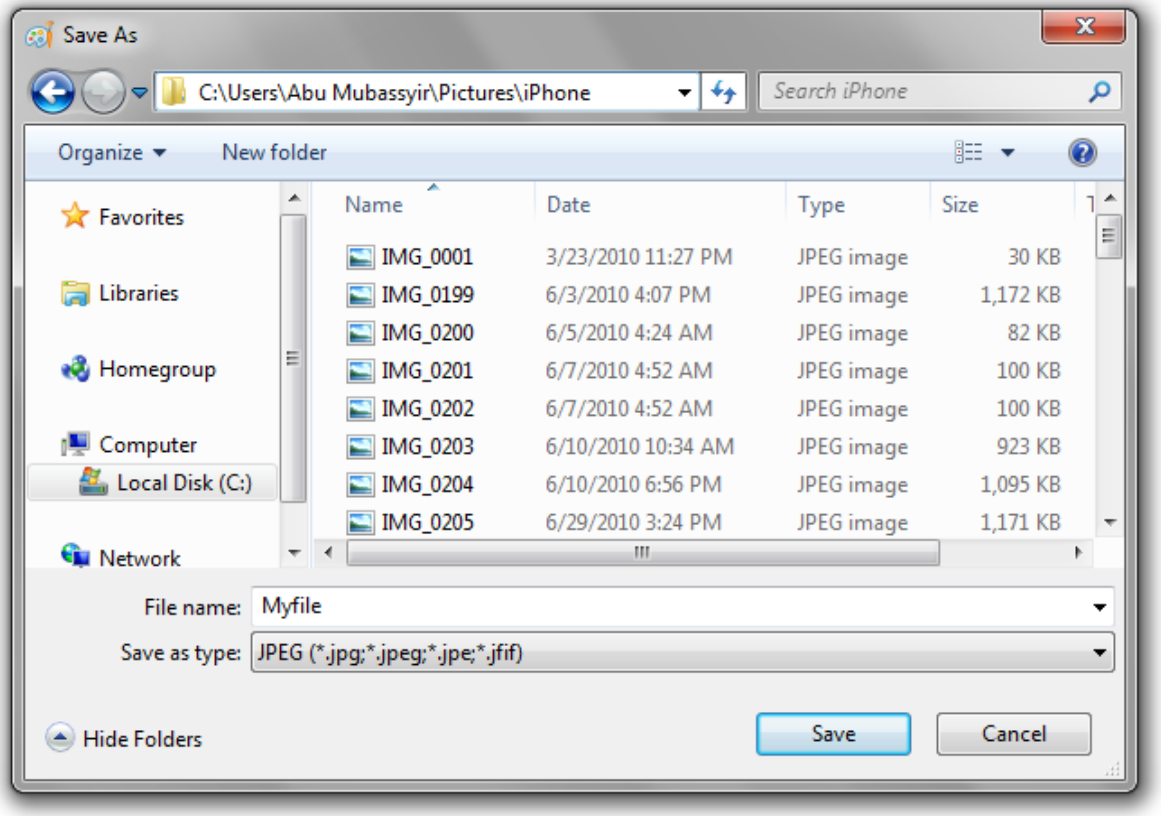
File type	Usual extension	Function
Executable	*exe, com, bin or none*	ready-to-run machine-language program
Object	*obj, o*	complied, machine language, not linked
Source code	*c, p, pas, 177, asm, a*	source code in various languages
Batch	*bat, sh*	commands to the command interpreter
Text	*txt, doc*	textual data documents
Word processor	*wp, tex, rrf, etc.*	various word-processor formats
Library	*lib, a*	libraries of routines for programmers
Print or view	*ps, dvi, gif*	ASCII or binary file
Archive	*arc, zip, tar*	related files grouped into one file, sometimes compressed.
Multimedia	*mpeg, mov, mp3, mp4, avi*	binary file containing audio or A/V information

Figure 10.3 Common file types

File Concept

Example 1a :

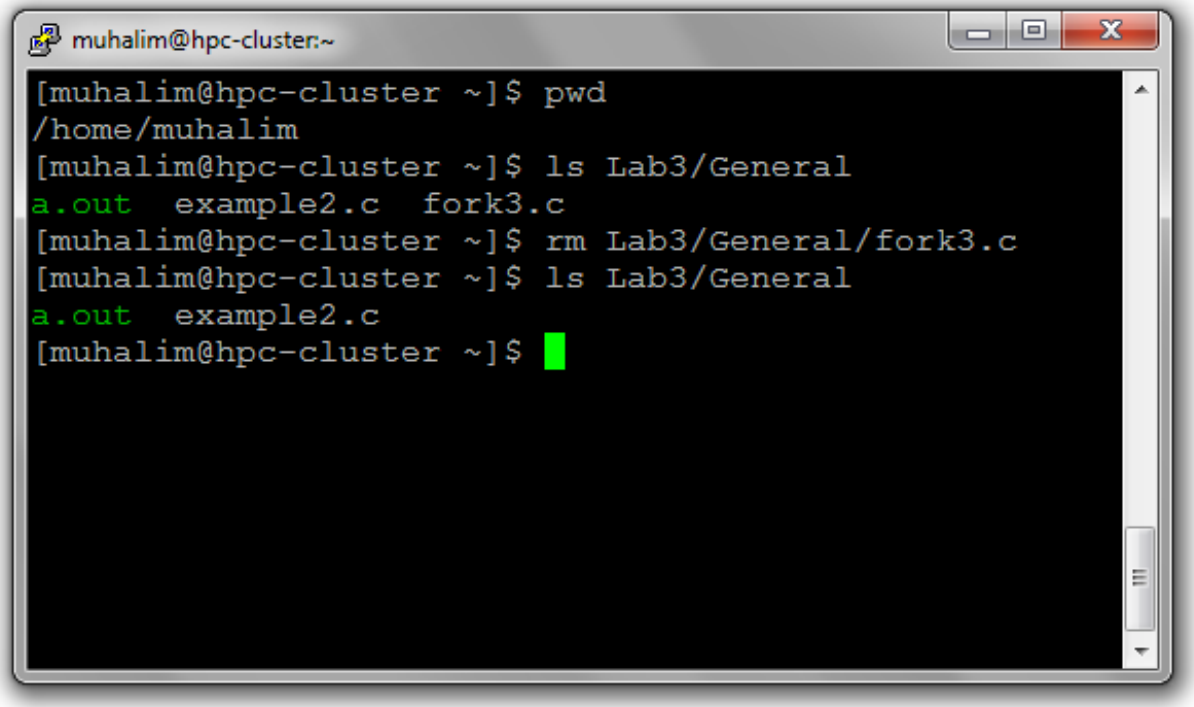
- Operating system specifics.
- **Windows :** Drive label and directory name, relative name, and extension.



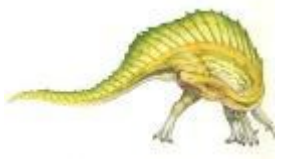
File Concept

Example 1b :

- Operating system specifics.
- **UNIX/Linux :** Forward slash (root), first subdirectory, sub-subdirectory, file's relative name.



10.2 Access Methods



Access Methods

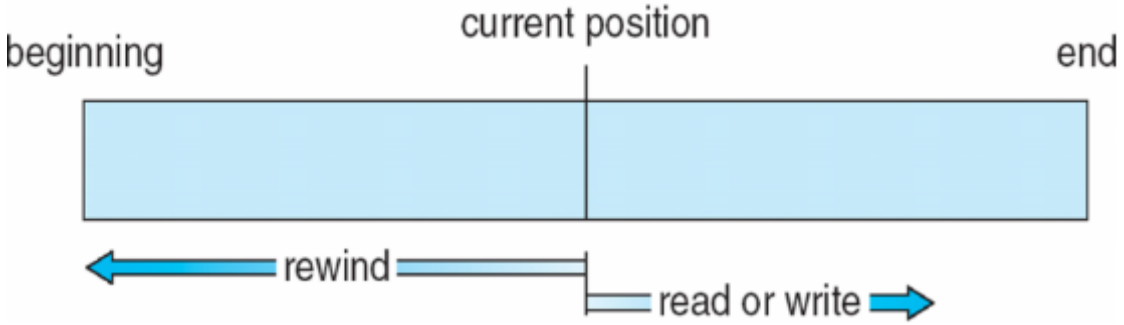
Overview

- Files store information that will be accessed and read into computer memory.
- The files can be accessed in several ways for a particular application → major design problem.

Access Methods

(a) Sequential access

- General structure:



- Operations:
  - `read_next()` – reads the next portion of the file and automatically advances a file pointer.
  - `write_next()` – append to the end of the file and advances to the end of the newly written material (the new end of file).
  - `reset` – back to the beginning of the file.

Access Methods

(b) Direct access

- File is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- File is viewed as a numbered sequence of blocks or records. For example, can read block 14, then read block 53, and then write block 7.
- Operations:
  - `read(n)` – reads relative block number n.
  - `write(n)` – writes relative block number n.
- Relative block numbers allow OS to decide where file should be placed.

Access Methods

(c) Other access methods

- Can be built on top of the base methods.
- Generally → involve creation of an **index** for the file.

- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item).
- If **too large**, keep index (in memory) of the main index (on disk).

Access Methods

- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index.
  - File kept sorted on a defined key.
  - All done by the OS.
- VMS operating system provides index and relative files as another example (see next slide).

Access Methods

Example 2 :

(Page: 468)

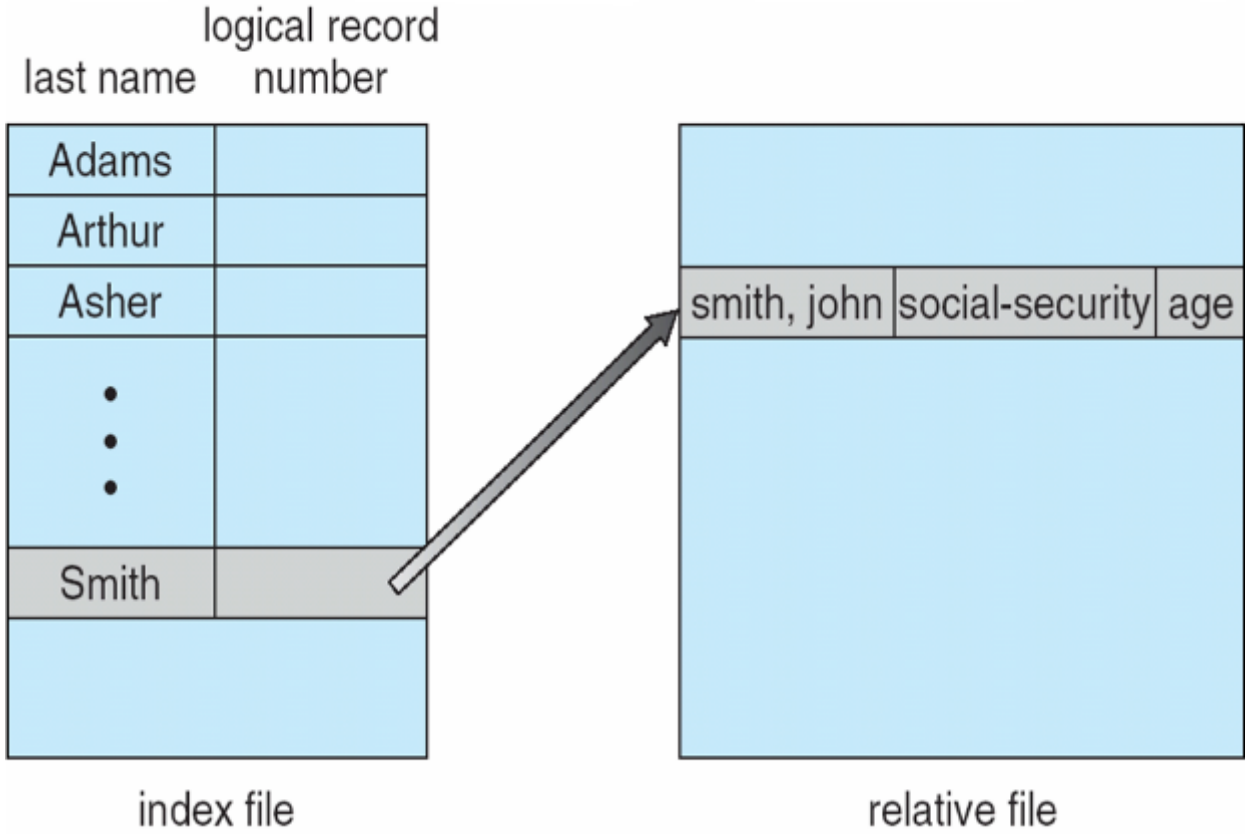
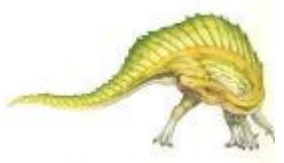


Figure 10.6 Index and Relative Files.

10.3 Directory and Disk Structure



Directory and Disk Structure

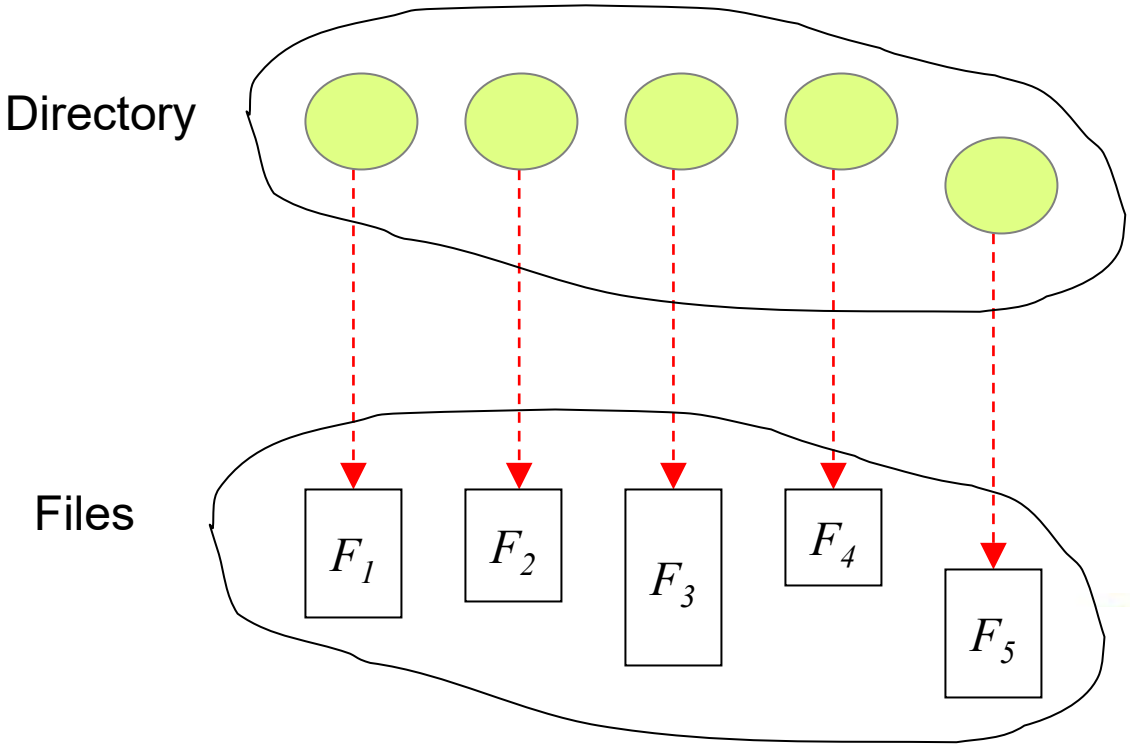
Directory structure

- Files are stored on random-access storage devices (hard disks, optical disks, solid-state disks).
- A storage devices:
  - can be used in it entirety for a file system,
  - can be subdivided into partition that hold a separate file system.
  - can be collected together into RAID sets.
- Any entity containing a file system is known as **volume**.

10.3 Directory and Disk Structure

- Each **volume** contains a file system and information about each files in the system.
  → **Device directory** / volume table of contents.

- A collection of nodes containing information about all **files**.
- Both the **directory structure** and the **files** reside on disk.



Directory and Disk Structure

Typical Volume Configuration

Volume:

- **Secondary storage unit**
  - Removable – *e.g.* CD, DVD, USB
  - Nonremovable – eg.hard disks
- **Multi-file volume**
  - Contains many files.
- **Multi-volume files**
  - Extremely large files spread across several volumes.

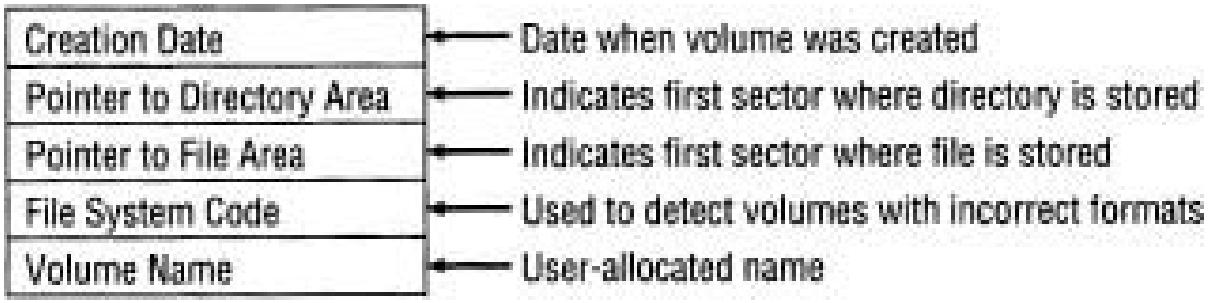
Removable storages unit

Non-Removable storages unit

10.3 Directory and Disk Structure

Volume Name:

- File manager writes the **volume name** and other descriptive information on **easily accessible area** on each unit.
  - ✓ *e.g.* Innermost part of CD, beginning of tape, first sector of outermost track.



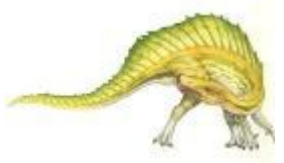
(figure 8.3)

The volume descriptor, which is stored at the beginning of each volume, includes this vital information about the storage unit.

10.3 Directory and Disk Structure

Operation performed on directory

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

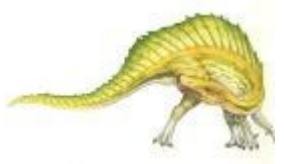


10.3 Directory and Disk Structure

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, …).



10.3 Directory and Disk Structure

Introducing subdirectories

- File Managers create an **MFD** (**Master File Directory**) for each volume.
- MFD is stored immediately after volume descriptor.
- It lists:
  - Names and characteristics of every file in volume.
    - File names (program files, data files, system files)
  - Subdirectories.
    - If supported by file manager.
  - Remainder of volume.
    - Used for file storage.

FAT (File Allocation Table)
NTFS (New Technology File System)

10.3 Directory and Disk Structure

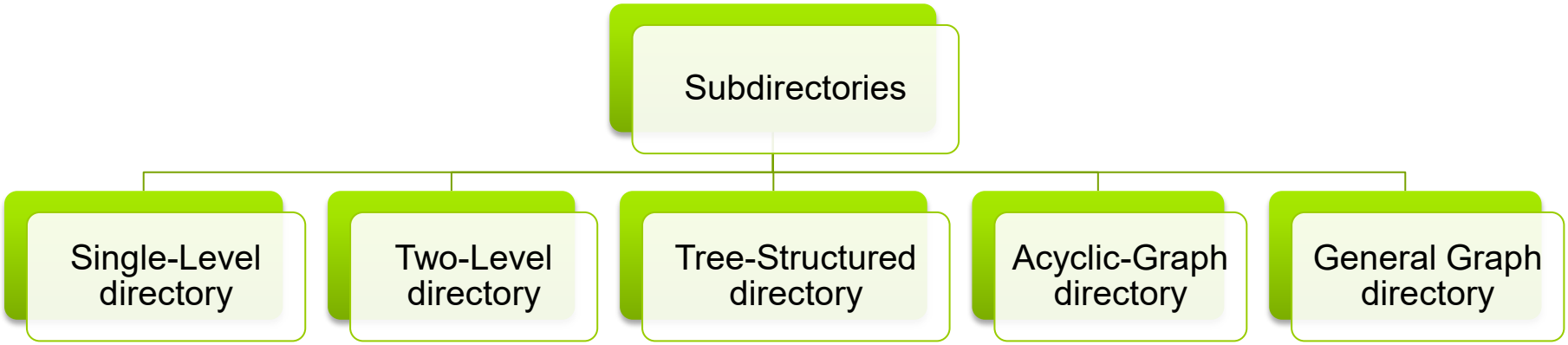


Figure The most common schemes for defining the logical structures of directory.

10.3 Directory and Disk Structure

(a) Single-Level directory

- A single directory for all users.

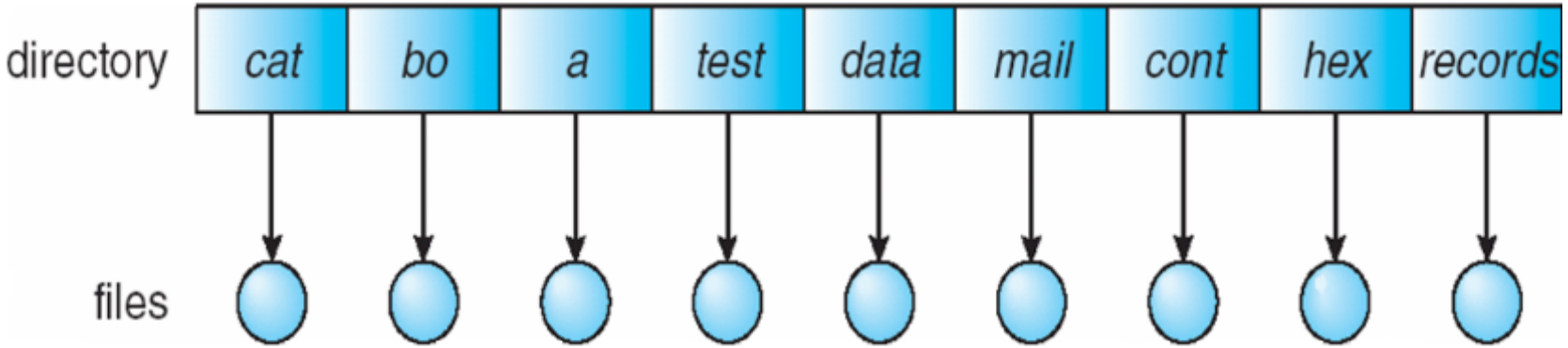
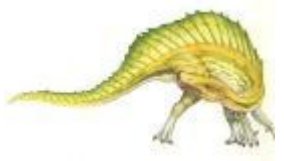


Figure 10.9 Single-level directory.

10.3 Directory and Disk Structure

Disadvantages:

- Long search time for individual file.
- Users cannot create subdirectories.
- Users cannot safeguard their files because entire directory was freely available to every user.
- Each program needs unique name.



Directory and Disk Structure

(b) Two-Level directory

- Separate directory for each user.

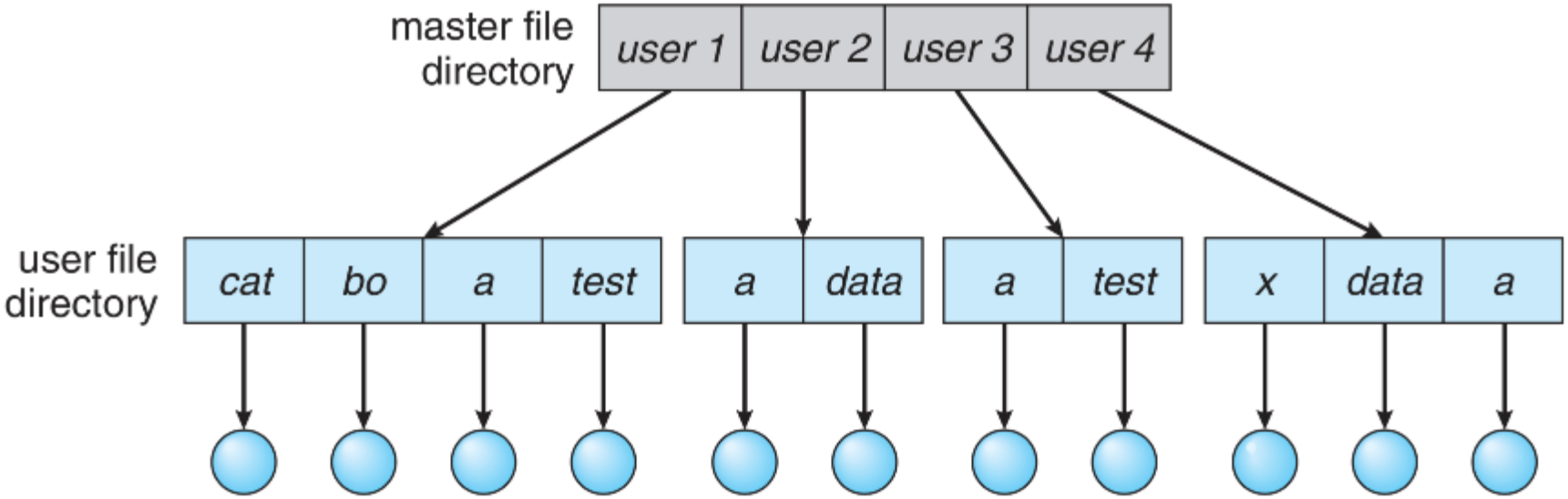


Figure 10.10 Two-level directory.

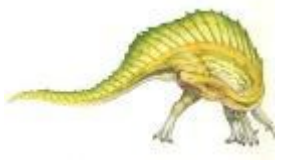
10.3 Directory and Disk Structure

- **Advantages**:
  - ✓ Path name.
  - ✓ Can have the same file name for different user.
  - ✓ Efficient searching.
- Disadvantage:
  - ✓ Problem remain → No grouping capability

10.3 Directory and Disk Structure

(c) Tree-Structured directory

- File managers today.
- Users create own subdirectories (**folders**).
  - ✓ Related files are grouped together.
- Implemented as **upside-down tree.**
  - ✓ Efficient system searching of individual directories.
  - ✓ May require several directories to reach file.



10.3 Directory and Disk Structure

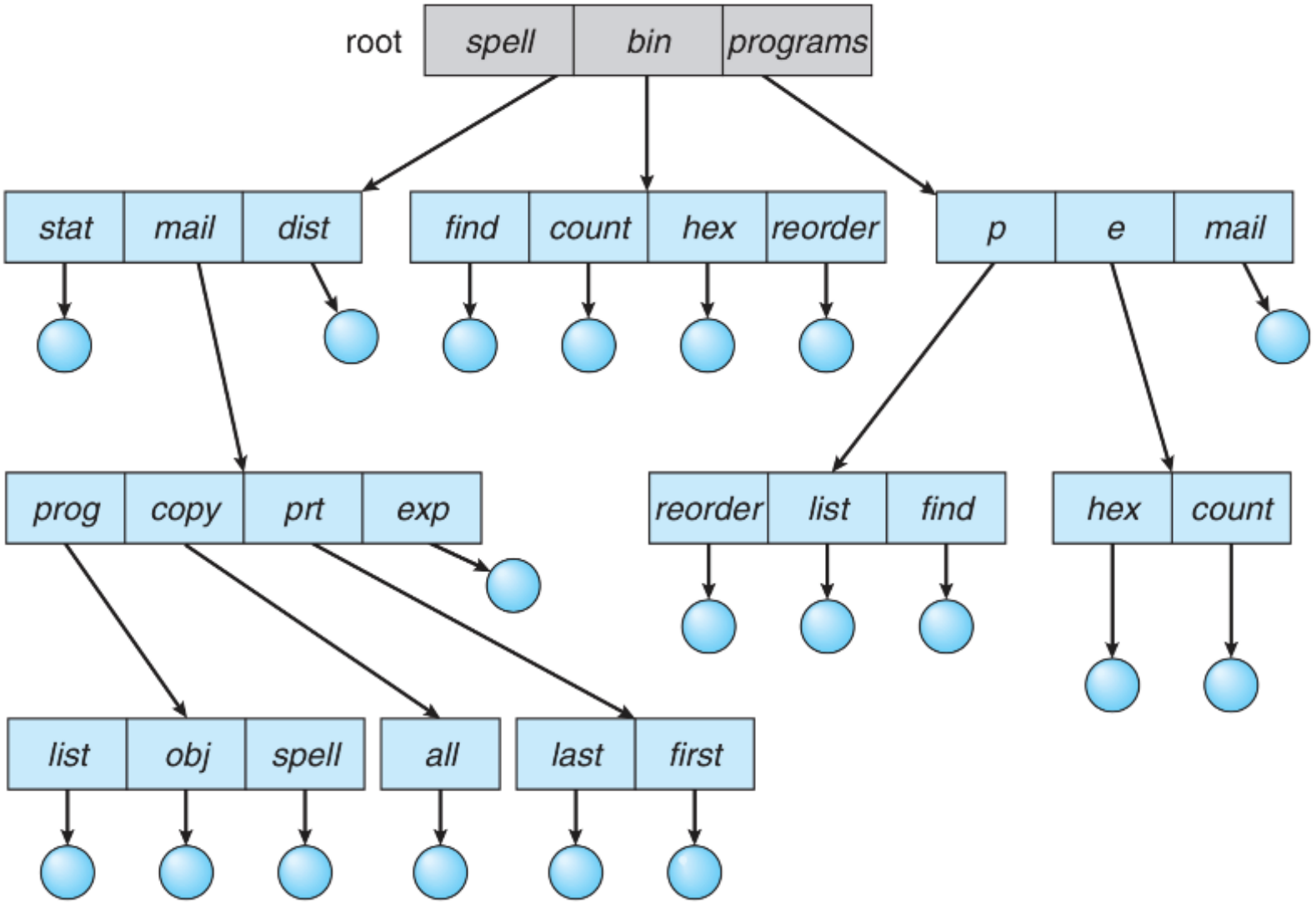


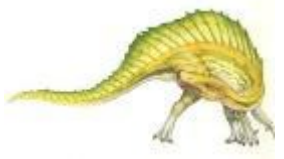
Figure 10.11 Tree-structured directory structure.

10.3 Directory and Disk Structure

- **Advantages**:
  - ✓ Efficient searching.
  - ✓ Grouping Capability.

- Current directory (working directory):

```
cd /spell/mail/prog
type list
```



10.3 Directory and Disk Structure

(d) Acyclic-Graph directory

- Have **shared** subdirectories and files.
- A tree structure prohibits the sharing of files or directories.
- An **acyclic graph** — a graph with **no cycles** that allows directories to share subdirectories and files (Figure 11.12).

- 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.

10.3 Directory and Disk Structure

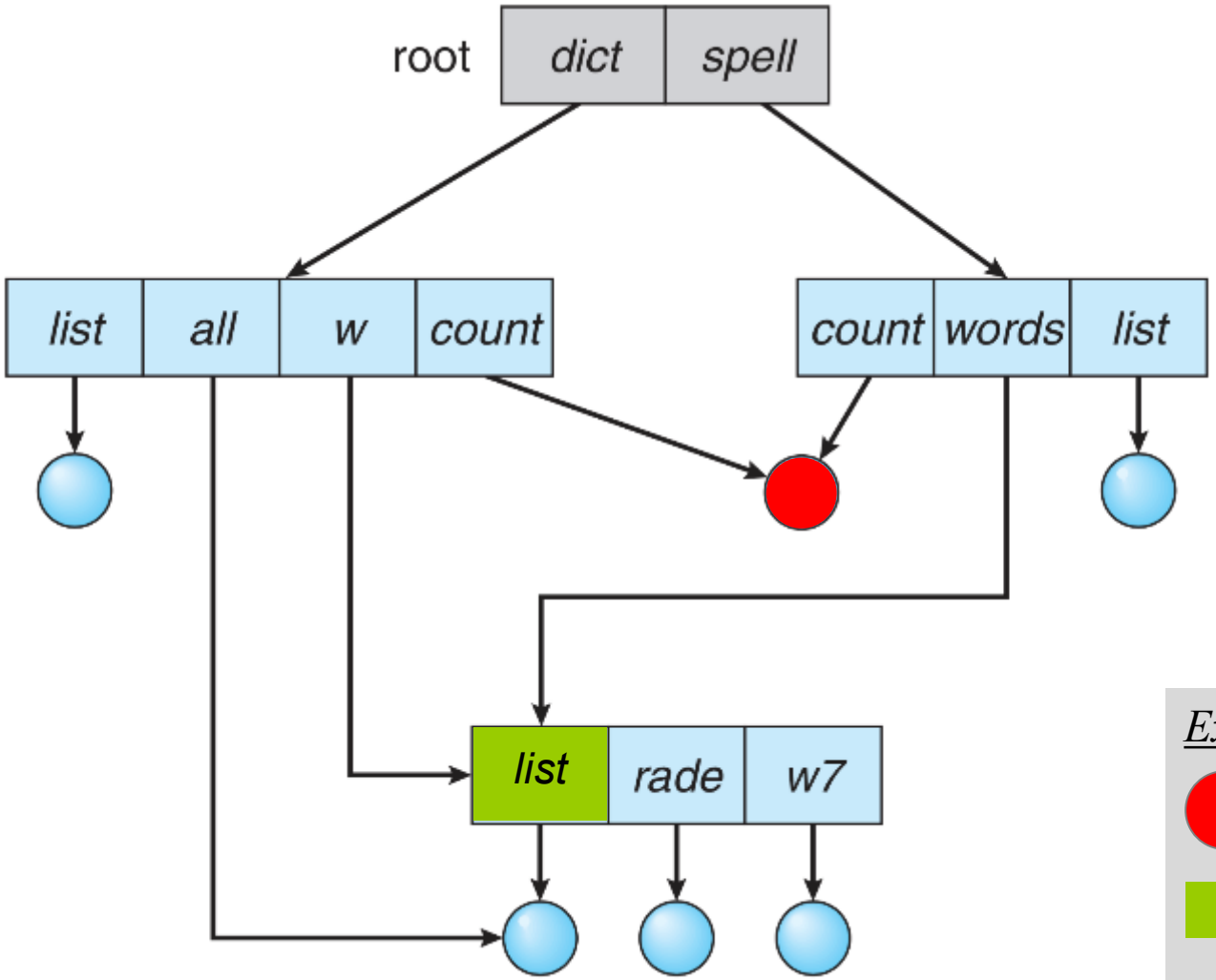


Figure 10.12 Acyclic-Graph directory structure.

Directory and Disk Structure

(e) General graph directory

- A serious problem with using an acyclic-graph structure is ensuring that there are no cycles.

- How do we guarantee no cycles?
  - Allow only links to file, but not subdirectories.
  - Use a **garbage collection** : to determine when the last reference (cycle) has been deleted and the disk space can be reallocated.
  - Every time a new link is added, use a **cycle detection algorithm** to determine whether it is OK.

10.3 Directory and Disk Structure

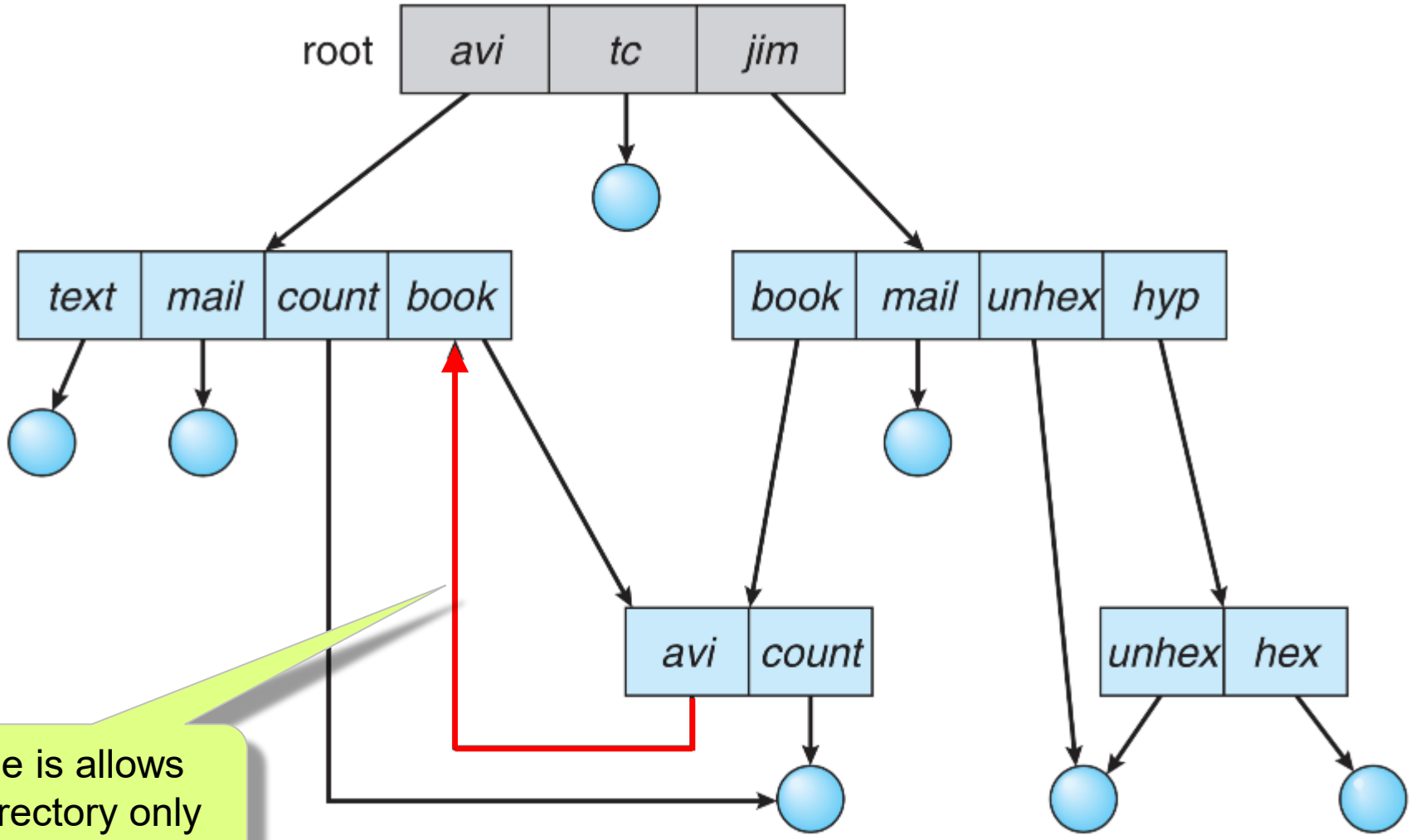
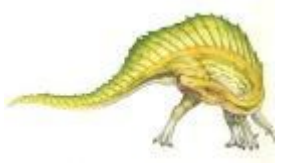


Figure 10.13 General graph directory structure.

10.6 Protection



Protection

Overview

- File owner/creator should be able to control:
  - What can be done?
  - By whom?

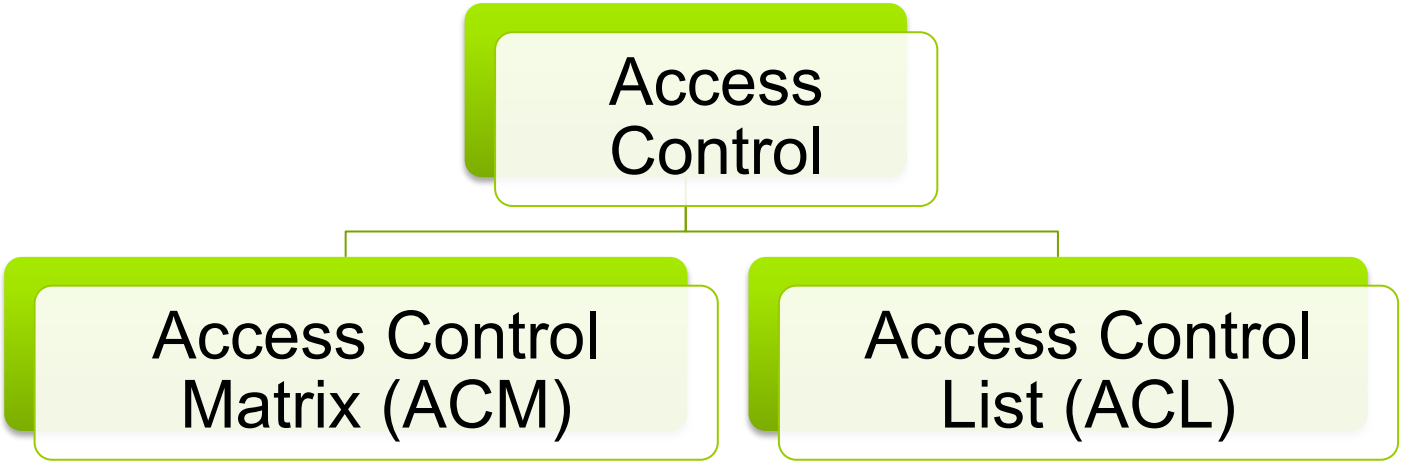
- Types of access:
  - `read`
  - `write`
  - `execute`
  - `append`
  - `delete`
  - `list`

Protection

Access control verification module

- **File sharing**
  - Data files, user-owned program files, system files.
  - **Advantages**
    - Save space, synchronized data updates, system's resource efficiency (I/O operations can be reduced).
  - **Disadvantage**
    - Need to protect file integrity → **access control**

Protection



(a) Access Control Matrix (ACM)

- An **access control matrix** abstracts the state relevant to access control.
- This method shows the access rights for each user for each file.

Protection

Example 6a :

- User 1 is allowed unlimited access to File 1 but is allowed only read and execute File 4 and is denied access to the three other files (File 2, 3 and 5).

R = Read Access
W = Write Access
E = Execute Access
D = Delete Access
– = Access Not Allowed

	User 1	User 2	User 3	User 4	User 5
File 1	RWED	R–E–	----	RWE–	--E–
File 2	----	R–E–	R–E–	--E–	----
File 3	----	RWED	----	--E–	----
File 4	R–E–	----	----	----	RWED
File 5	----	----	----	----	RWED

Protection

- **Advantages:** (Access control matrix)
  - Easy to implement.
  - Works well in system with few files, users.
- **Disadvantages:**
  - As files and users increase, matrix size increases
    - Possibly beyond main memory capacity
  - Wasted space: due to null entries.

Solution:
Access Code

Protection

Example 6b :

- The five access codes for User 2 from **Example 6a**.
- The resulting code for each file is created by assigning a '1' for each checkmark, and a '0' for each blank space.

R = Read Access
W = Write Access
E = Execute Access
D = Delete Access
– = Access Not Allowed

Access	R	W	E	D	Resulting Code	
R–E–	√		√		1010	*File 1*
R–E–	√		√		1010	*File 2*
RWED	√	√	√	√	1111	*File 3*
----					0000	*File 4*
----					0000	*File 5*

Protection

(b) Access Control List (ACL)

- An **access control list** showing which users are allowed to access each file.

- It is the most general scheme to implement independent access is to associate with:
  - ✓ each **file**, and
  - ✓ **directory** the specifying user names and the types of access allowed for each user.

Protection

Example 7 :

- **ACL** method requires less storage space than an **access control matrix**.

R = Read Access
W = Write Access
E = Execute Access
D = Delete Access
– = Access Not Allowed

File	Access
File 1	USER1 (RWED), USER2 (R-E-), USER4 (RWE-), USER5 (--E-), WORLD (----)
File 2	USER2 (R-E-), USER3 (R-E-), USER4 (--E-), WORLD (----)
File 3	USER2 (RWED), USER4 (--E-), WORLD (----)
File 4	USER1 (R-E-), USER5 (RWED), WORLD (----)
File 5	USER5 (RWED), WORLD (----)

Protection

- Contains user names granted file access with ACL.
  - User denied access grouped under “WORLD”
- Shorten list by categorizing users:

  - **SYSTEM**
    - Personnel with unlimited access to all files.
  - **OWNER**
    - Absolute control over all files created in own account.
  - **GROUP**
    - All users belonging to appropriate group have access.
  - **WORLD**
    - All other users in system.

Protection

Example: UNIX

- Solve the problem of directory size by classifying users into 3 classes:

  - **Owner** – A user who creates the file.
  - **Group** – A set of users who are sharing the file and need similar access.
  - **Universe/public** – All users.

- Defines 3 bits to represent each class: `R W X`
  - *e.g.* `RW- R-- ---` indicates **owner** has `RW` access, **group** has `R` access and **public** has no access.

Protection

Example 8a :

With each file and subdirectory we keep 9 protection bits, 3 for owner, 3 for group, 3 for public.

```
RWX R-- R-X
```

- Owner access 7 → RWX
  111
- Group access 4 → R--
  100
- Public access 5 → R-X
  101

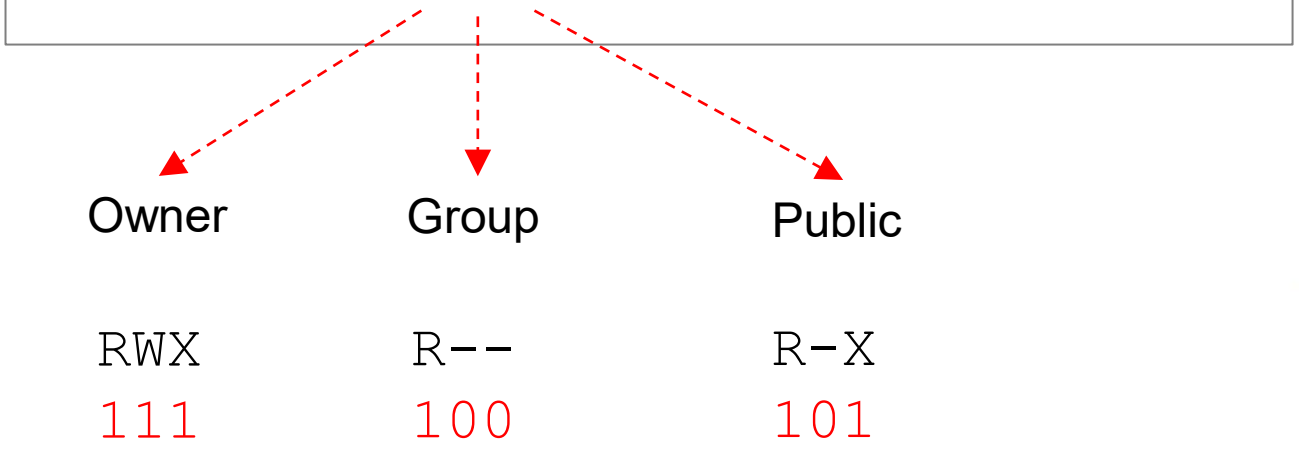
Protection

Example 8b :

UNIX: Set protection for file "`game`" using the command "`chmod`".

- ✓ All user classes can read the file.
- ✓ Only owner can write the file.
- ✓ Owner and public can execute the file.

```
> chmod 745 game
```



Protection

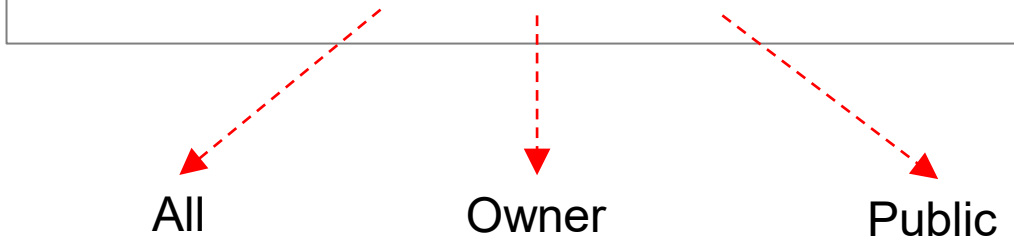
Example 8c :

UNIX: Another way to set protection for file "`game`" using the command "`chmod`".

- ✓ All user classes can read the file.
- ✓ Only owner can write the file.
- ✓ Owner and public can execute the file.

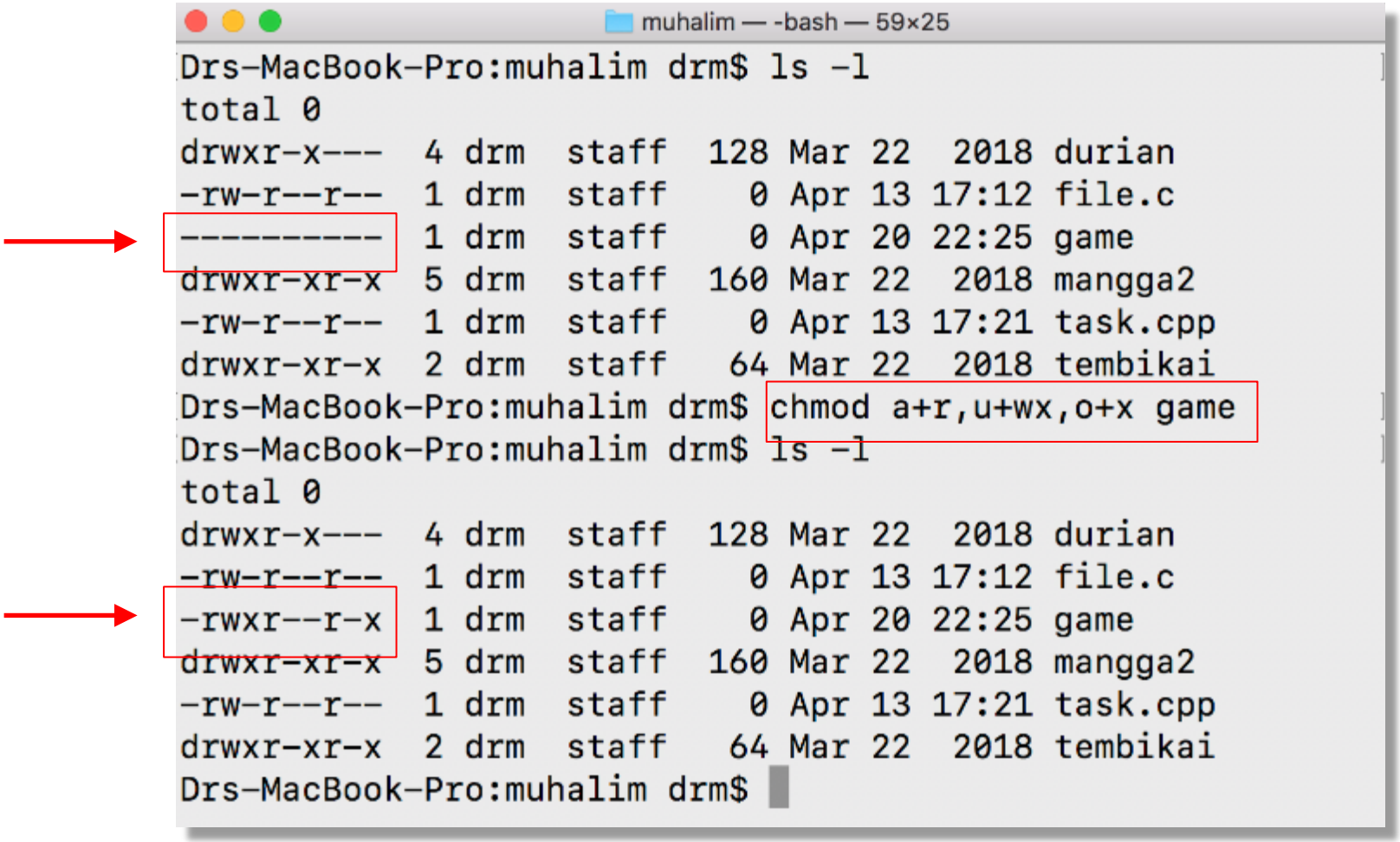
`a` = All
`u` = User / Owner
`g` = Group
`o` = Public
`+` = Access Allowed
`−` = Access Not Allowed

```
> chmod a+r,u+wx,o+x game
```



R-- R-- R-- -WX --- --- --- --- --X

Protection


muhalim — -bash — 59×25
Drs-MacBook-Pro:muhalim drm$ ls -l
total 0
drwxr-x--- 4 drm staff 128 Mar 22 2018 durian
-rw-r--r-- 1 drm staff   0 Apr 13 17:12 file.c
---------- 1 drm staff   0 Apr 20 22:25 game
drwxr-xr-x 5 drm staff 160 Mar 22 2018 mangga2
-rw-r--r-- 1 drm staff   0 Apr 13 17:21 task.cpp
drwxr-xr-x 2 drm staff   64 Mar 22 2018 tembikai
Drs-MacBook-Pro:muhalim drm$ chmod a+r,u+wx,o+x game
Drs-MacBook-Pro:muhalim drm$ ls -l
total 0
drwxr-x--- 4 drm staff 128 Mar 22 2018 durian
-rw-r--r-- 1 drm staff   0 Apr 13 17:12 file.c
-rwxr--r-x 1 drm staff   0 Apr 20 22:25 game
drwxr-xr-x 5 drm staff 160 Mar 22 2018 mangga2
-rw-r--r-- 1 drm staff   0 Apr 13 17:21 task.cpp
drwxr-xr-x 2 drm staff   64 Mar 22 2018 tembikai
Drs-MacBook-Pro:muhalim drm$

Protection

Example 8d :

UNIX: Another way to set protection for file "`game`" using the command "`chmod 745`".

muhalim — -bash — 59×25

```
Drs-MacBook-Pro:muhalim drm$ ls -l
total 0
drwxr-x---  4 drm  staff  128 Mar 22  2018 durian
-rw-r--r--  1 drm  staff    0 Apr 13 17:12 file.c
-rw-r--r--  1 drm  staff    0 Apr 20 22:25 game
drwxr-xr-x  5 drm  staff  160 Mar 22  2018 mangga2
-rw-r--r--  1 drm  staff    0 Apr 13 17:21 task.cpp
drwxr-xr-x  2 drm  staff   64 Mar 22  2018 tembikai
Drs-MacBook-Pro:muhalim drm$ chmod a+r,u+x,o+x game
Drs-MacBook-Pro:muhalim drm$ ls -l
total 0
drwxr-x---  4 drm  staff  128 Mar 22  2018 durian
-rw-r--r--  1 drm  staff    0 Apr 13 17:12 file.c
-rwxr--r-x  1 drm  staff    0 Apr 20 22:25 game
drwxr-xr-x  5 drm  staff  160 Mar 22  2018 mangga2
-rw-r--r--  1 drm  staff    0 Apr 13 17:21 task.cpp
drwxr-xr-x  2 drm  staff   64 Mar 22  2018 tembikai
Drs-MacBook-Pro:muhalim drm$
```

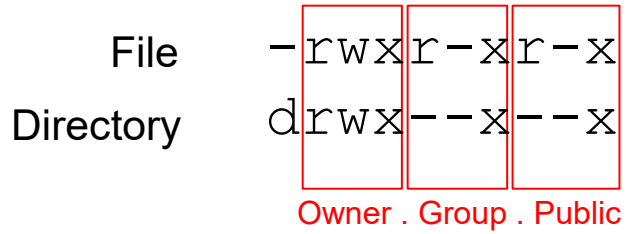
Protection

Example 9 : A sample UNIX 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/

Explanation:

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



Protection

Exercise 10.1

Given a sample UNIX 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/

(a) How many files and directories exist in the listing?

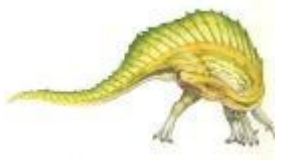
(b) Define the access control of all files.

(c) Who can access the `program.c`? Justify your answer.

(d) Change the access control of the file in (c) so that all users can execute it as well.

10.7 Summary

- A **file** is an abstract data type defined and implemented by the operating system (OS).
- A file is a sequence of **logical records**.
- An OS needs to **map** the logical file concept onto physical storage devices.
- A **tree-structured directory** allows a user to create subdirectories to organize files.
- **Access** to files can be controlled separately for each type of access (`read`, `write`, `execute`, `delete`)



End of Chapter 10