# CS310 Operating Systems

## Lecture 34 : File System - 2

Ravi Mittal
IIT Goa

# Acknowledgements !


- Contents of this class presentation has been taken from various sources. Thanks are due to the original content creators:
  - Book: Modern Operating Systems by Andrew Tanenbaum and Herbert Bos,
    - Chapter 4
  - Book: Linux System Programming: talking directly to the kernel and C library, by Robert Love
  - Book: Computer Systems, A programming Perspective, Bryant and O'Hallaron
  - Class presentation: University of California, Berkeley, CS162
  - Book: Linux – The Textbook, by Sarwar, Koretsky

# Read the following:

- Book: Modern Operating Systems, by Andrew Tanenbaum and Herbert Bos
  - Chapter 4
- Book: Linux System Programming: talking directly to the kernel and C library, by Robert Love
- Book: Operating Systems: Three Easy Pieces, Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

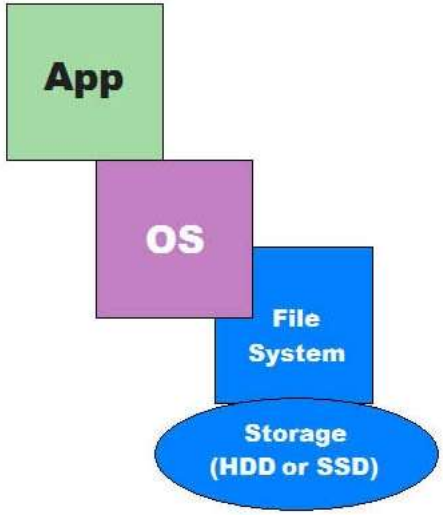
# Today's Class

- File System – Introduction (previous class)
- Directories
- Hard and Soft (Symbolic) links - Concepts

# Previous Classes

# File System

- Files are logical unit of information – Created by Processes
- A Disk may contain thousands of files – usually independent of each other
- Processes can read existing files and create new files
- Files remain in existence not affected by process creation and termination
- File must disappear only when the owner explicitly deletes it
- File are managed by Operating system
- File System deals with how files are
  - Structured
  - Named
  - Accessed
  - Used
  - Protected
  - Implemented, and
  - Managed



# File Naming

| 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 Structure – Three categories

- Byte sequences
  - Maximum flexibility-can put anything in
  - Unix and Windows use this approach
  - Byte stream

In use

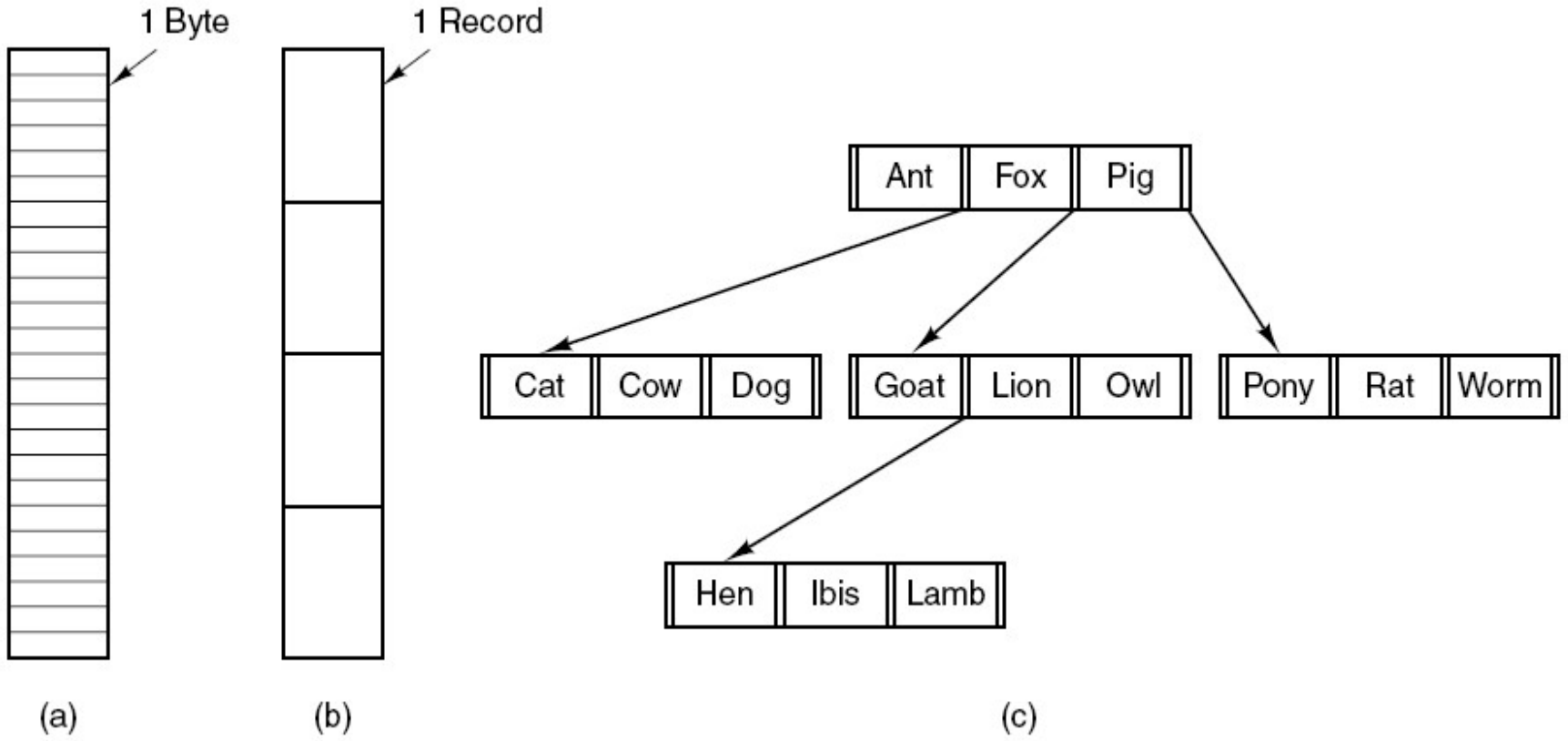
- Fixed length records (card images in the old days)
  - Read/Write operations are done on one record
    - Example: Based on Punch cards (80 columns) – record 80 characters
    - Not in use
  - VMS OS provides highly structured file – with records

Obsolete

- Tree of records- uses key field to find records in the tree
  - Tree is sorted on the key field
  - Used in some large mainframes

Obsolete

# File Structure – Three Types




**Bye sequence**

**Record sequence**

**Tree**

# File Types

- Regular Files (most common)
  - Text
  - Binary
- Directories
  - A file that contains the names and locations of other files
- Character Special and Block Special Files
  - Terminals (character special) and disks (block special)
- FIFO (named pipe)
  - A file type used for inter-process communication
- Socket
  - A file type used for network communication between processes

# Regular Files

- What we mostly call as files – are *regular files*
- *Regular files* contain bytes of data organized into linear array called a *byte stream*
  - Linux has no further organization or formatting specified for a file
- Any byte  within a file can be read or written to
- Read or write operation start at a specific byte
  - Conceptual location
    - *File position* or *file offset*
      - Essential metadata
- At file opening – *file position* is zero

# Regular Files (Linux)

- File reading or writing is done byte by byte
  - The file position increases
- File position may be set manually

- File position starts at zero
- Writing a byte to the middle of a file overwrites over the previous byte
- It is not possible to expand a file by writing into middle of it

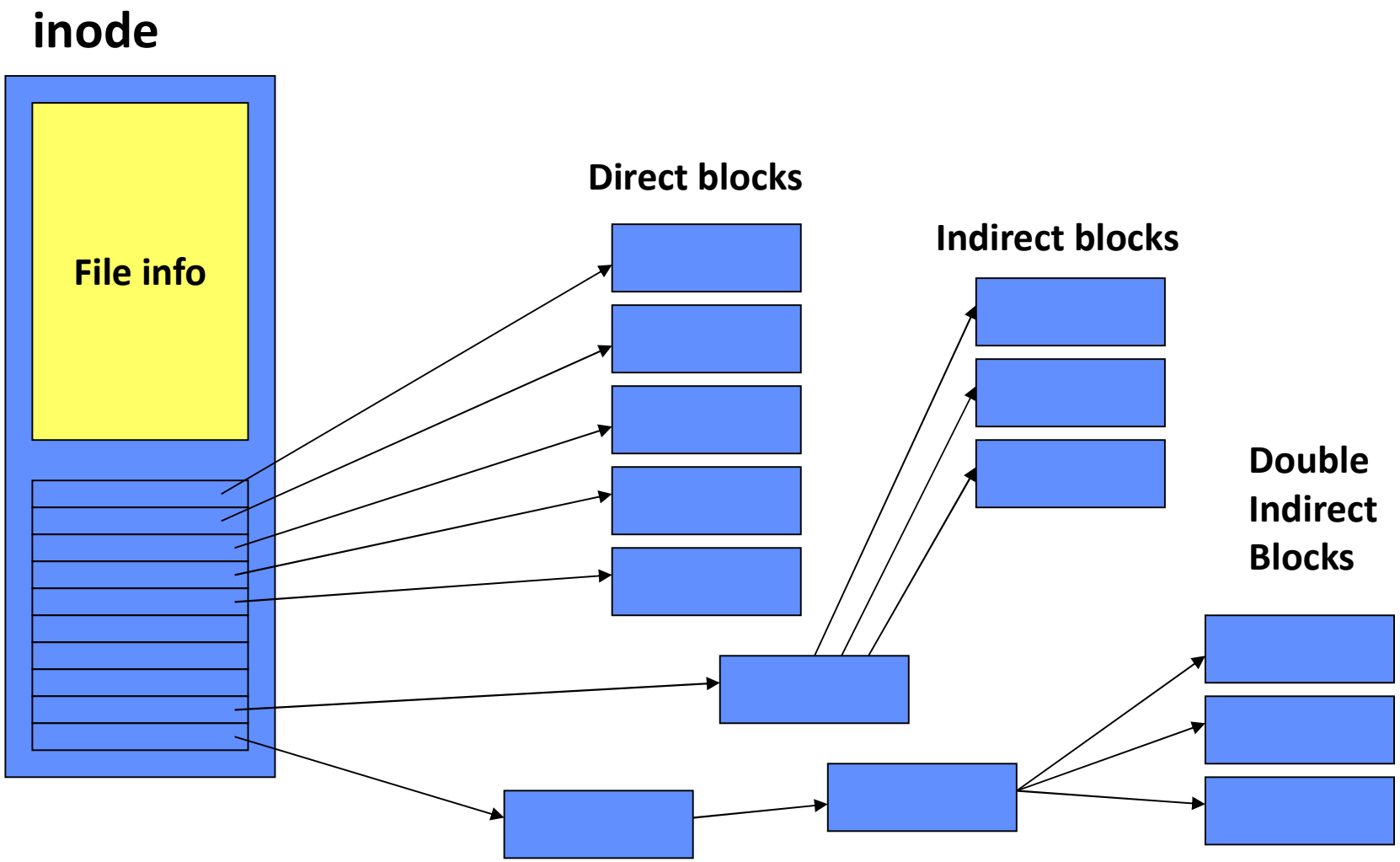
# Regular Files (Linux)

- A single file may be opened more than once
  - By the same process or different processes
  - Each open instance of a file gives a unique file descriptor
- Processes can share their file descriptors
- No restriction on concurrent file accesses by multiple processes
- If concurrent reading and writing is happening
  - Result may be unpredictable
  - So user processes must coordinate
- Files are associated with file names but they are referenced by *inode* – information node
  - *Inode* is assigned an integer value unique to the file system – *inode* number or *i-number*

# Regular Files (Linux) - *inode*

- When a file is created
  - System allocates a unique *inode* to the file
  - It creates an entry in the directory in which the file is created
    - Directory entry: an ordered pair (**inode number**, **filename**)
- The *inode* number for a file is used to access its attributes, including its contents on the disk for reading or writing (changing) them
- *An Inode* stores metadata associated with a file
  - Modification time stamp / access time
  - Owner (UID, GID)
  - Type (file, directory, device, pipe, …)
  - Length
  - Pointers to data blocks that store file's contents
- Example: You created a file Chapter 3 in your current working directory
  - System allocates *inode* number **53472**
  - Directory entry: (**52473**, **Chapter3**)

# *Inode* diagram



# Directories

# Directories - Introduction

- Directories are handled as normal files
  - However, marked in *inode* as directory
- Accessing a file via its *inode* number is cumbersome and unsecure (security hole)
- Files are always opened from user space by a name
  - Not by *inode* number
- *Directories* are used to provide the names with which to access files
- A directory provides mapping of human-readable names to *inode* numbers
- A name and *inode* pair is called a *link*
- Directory is implemented as a simple table or a hash
  - Implemented and managed by Kernel
- The kernel directly uses this mapping to perform name-to-*inode* resolutions

# Directories - Introduction

- A user application requests a file name to be opened
  - Kernel opens the directory containing the filename and searches for the given name
  - From the filename, the kernel obtains the *inode* number
  - *Inode* number points to *inode*
  - The *inode* contains metadata associated with the file and location of file's data on the disk
- Initially, there is only one directory on the disk, the *root directory*
  - Donated by /.
- There are many directories in a file system
- Links inside of directories can point to the *inode*s of other directories
- Directories nest inside of other directories – hierarchy of directories

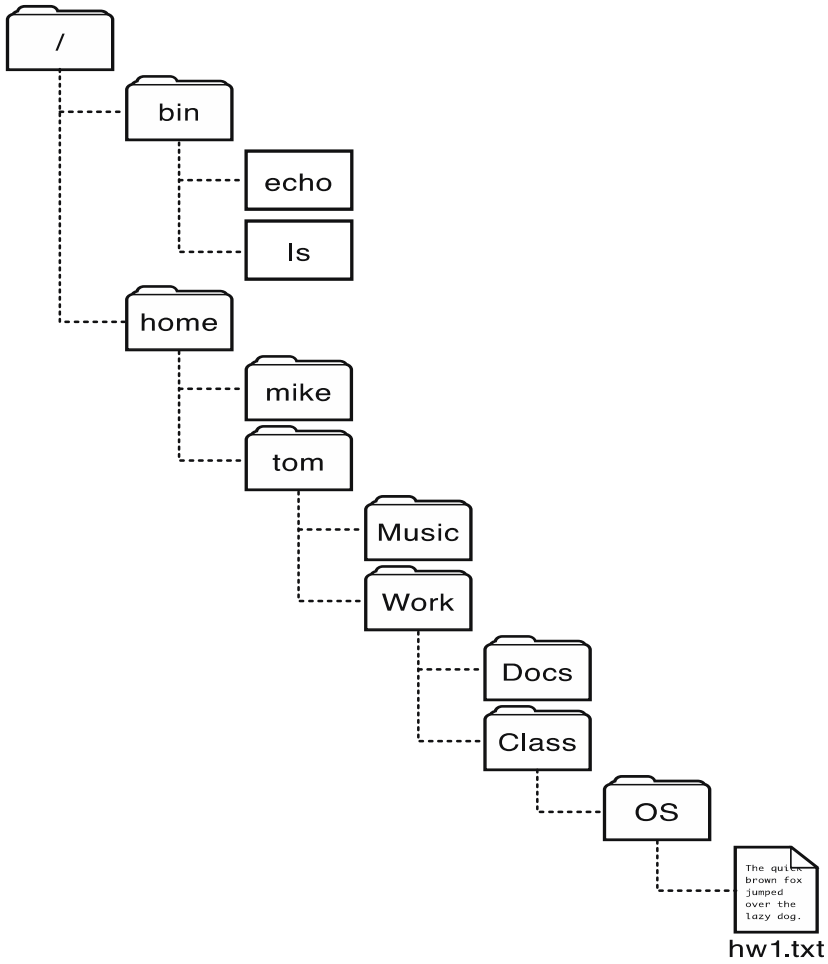
# Directories - Introduction

- Users identify files via pathnames example
  (*/home/blackbeard/concorde.png* )
- To open a given path, kernel walks each *directory entry (dentry)* in the pathname to find the *inode* of the next entry
- *dentry cache in Linux*
  - to store the results of directory resolutions
- Path
  - String that identifies a file or directory
    - Example: /home/tom/Work/Class/OS/hw1.txt or /home/tom
- Absolute Path
    - if it stats with "/", the root directory
    - Example: /bin/ls
  - Relative Path
    - relative to a process's current working directory
    - Example: Work/Class/OS if the current directly is /home/tom
      - Equivalent to absolute path: /home/tom/Work/Class/OS

# Directories - Introduction

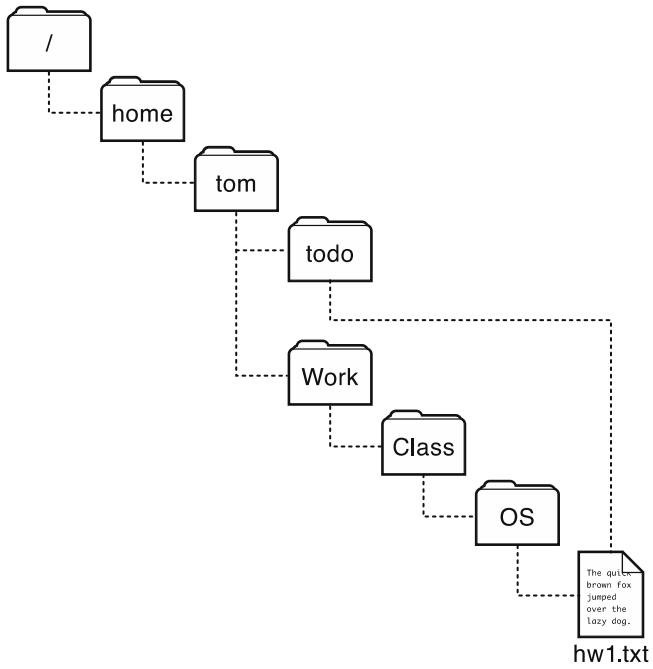
- Paths are resolved from left to right and the respective name is looked up in the directory
- As long as the current name is not the last in the path, it has to be a directory
  - Else it terminates with an error

# Hierarchical organization of files using directories



# Directory is (not always) a tree

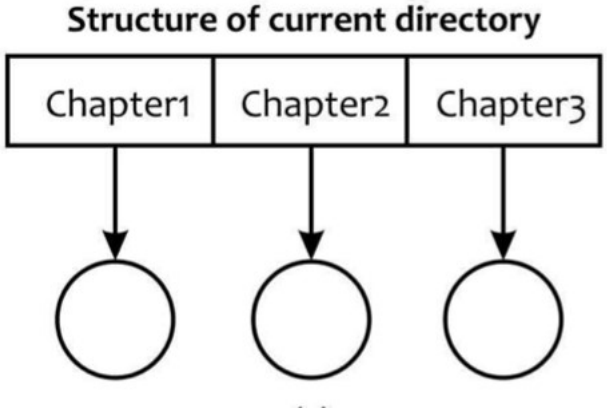
- Sometimes it is convenient for a file to appear in two directories



# Directory Structure

- Consider a current directory containing three files: Chapter 1, Chapter 2, and Chapter 3

Files



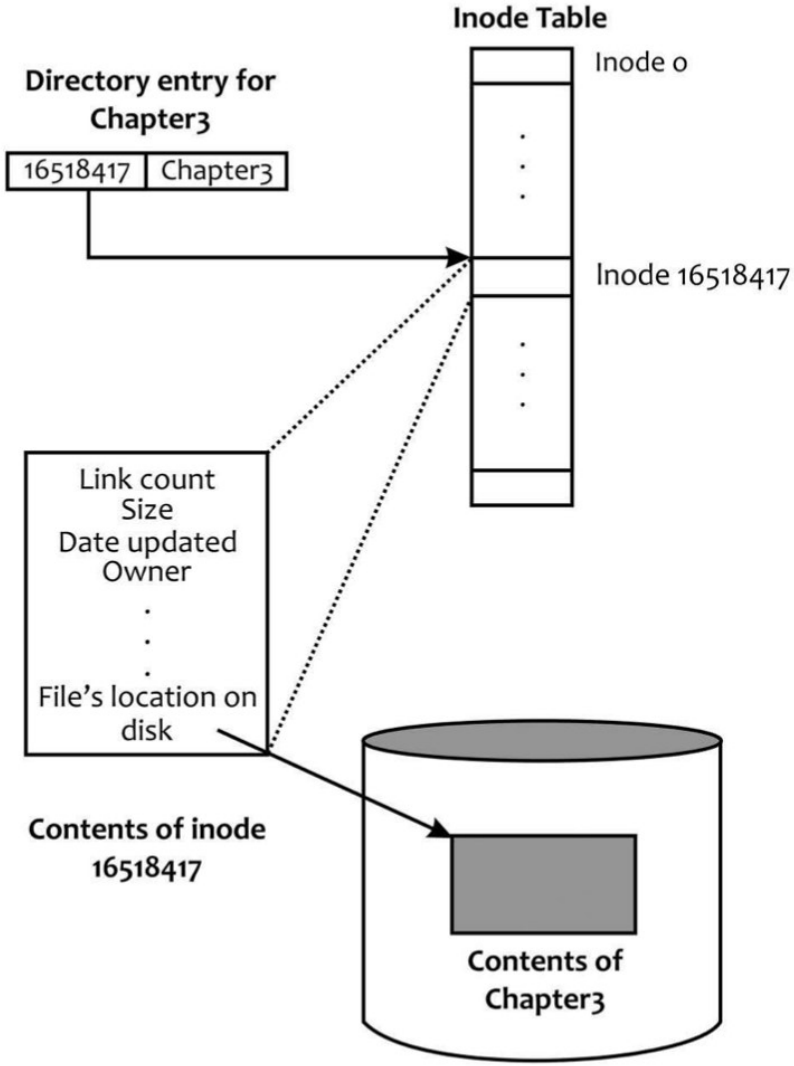

Logical Structure

Contents of current directory

| Inode # | File |
|---|---|
| 16517779 | . |
| 16517772 | .. |
| 16518415 | Chapter1 |
| 16518416 | Chapter2 |
| 16518417 | Chapter3 |

Contents of the disk block containing current directory

# Directory Entry, inode, and file contents




Adapted from: Linux: The Textbook by Sarwar and koretsky

# ls –il command

```
(base) Ravis-MacBook-Pro-2:sample ravimittal$ ls -il
total 29504
4391804511 -rw-r--r--@ 1 ravimittal  staff   2830690 Jun 18 16:53 Chapter 1.pptx
4391921390 -rw-r--r--@ 1 ravimittal  staff   1016998 Jun 22 18:04 Chapter 2.pptx
4393535828 -rw-r--r--@ 1 ravimittal  staff  11250437 Aug  8 15:34 Chapter 3.pptx
(base) Ravis-MacBook-Pro-2:sample ravimittal$
```

# Hard and Soft (Symbolic) Links

# File Sharing Via links

- Attributes of a Linux file are stored in its *inode* on disk
- When a file is opened, its *inode* is copied into the main memory
  - Speedy access to its contents
- How the use of an *inode* that results in a mechanism that allows you to access a file by using multiple pathnames?
  - To allow access to some files and directories through various other directories
- A link is a way to establish connection between
  - Files to be shared with directory enteries
- A file has *N* links, we mean that the file has *N* directory entries somewhere in the file system hierarchy
- Linux supports two types of links:
  - Hard links
  - Soft/symbolic links

# Hard Links

- A hard link is a *pointer* to the *inode* of a file
- A hard link gives additional file name
- All hard links point to the same *inode*
- As long as the link counter is not 0, the file survives
- Hard links can be created for files in the same logical file system

# ln command

**Syntax:**

```
ln [options] existing-file new-file
ln [options] existing-file-list directory
```

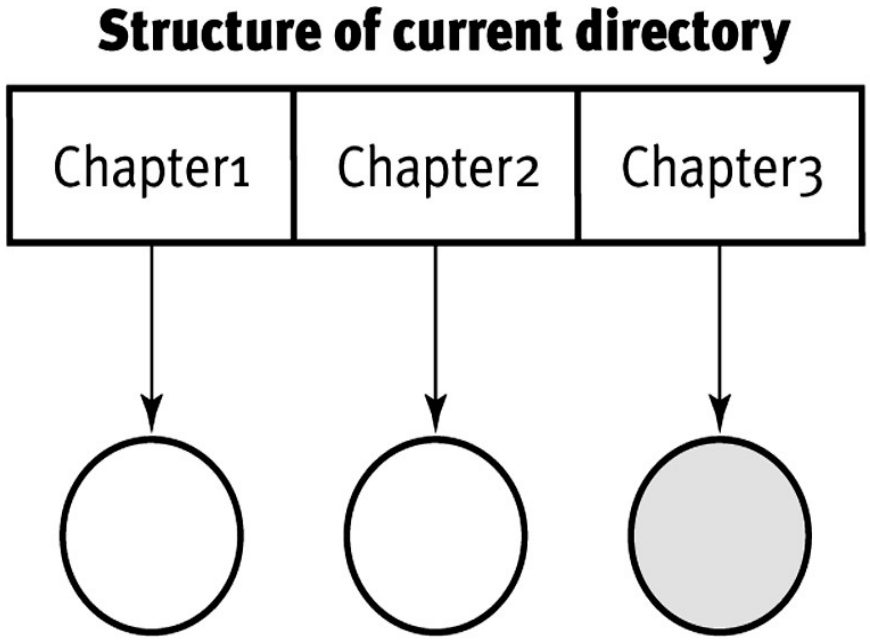
**Purpose**: First syntax: Create a link to **existing-file** and name it **new-file**

Second syntax: Create links to the ordinary files in **existing-file-list** in **directory**; links have the same names as the original file

**Commonly used options/features:**

- **-f** Force creation of link; don't prompt if **new-file** already exists
- **-n** Don't create the link if **new-file** already exists
- **-s** Create a symbolic link to **existing-file** and name it **new-file**

```
(base) Ravis-MacBook-Pro-2:sample ravimittal$ ls
chapter1.pptx   chapter2.pptx   chapter3.pptx
(base) Ravis-MacBook-Pro-2:sample ravimittal$ ln chapter3.pptx chapter3.hard
(base) Ravis-MacBook-Pro-2:sample ravimittal$ ls -il
total 31496
4391804511 -rw-r--r--@ 1 ravimittal  staff   2830690 Jun 18 16:53 chapter1.pptx
4393535828 -rw-r--r--@ 1 ravimittal  staff  11250437 Aug  8 15:34 chapter2.pptx
4391921390 -rw-r--r--@ 3 ravimittal  staff   1016998 Jun 22 18:04 chapter3.hard
4391921390 -rw-r--r--@ 3 ravimittal  staff   1016998 Jun 22 18:04 chapter3.pptx
(base) Ravis-MacBook-Pro-2:sample ravimittal$
```



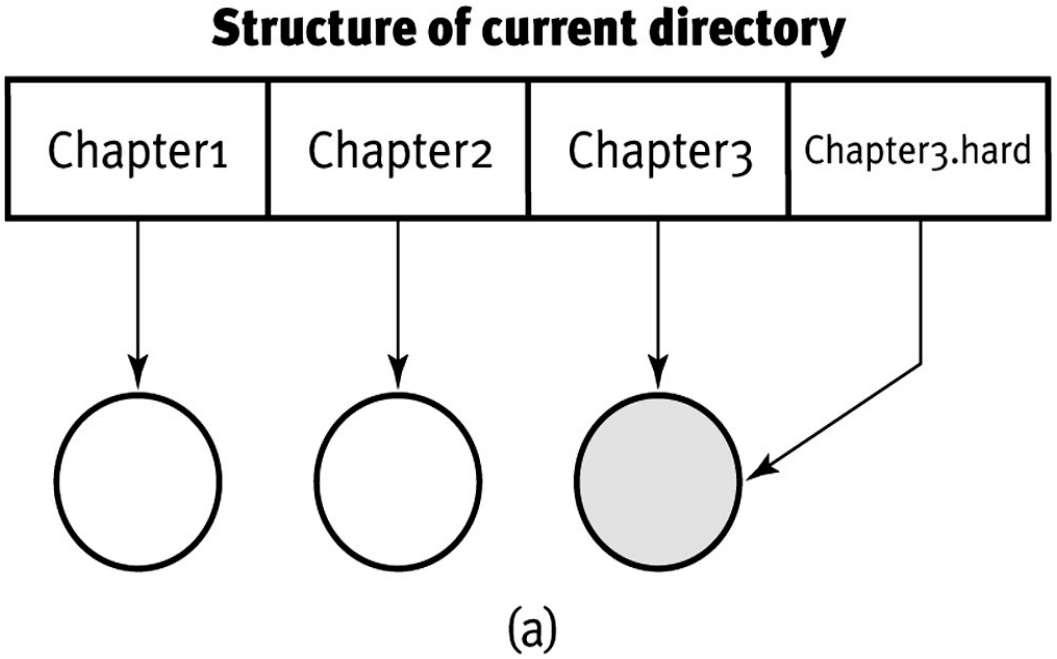

**Contents of current directory**

| Inode # | File |
|---|---|
| 1076 | . |
| 2083 | .. |
| 13059 | Chapter1 |
| 17488 | Chapter2 |
| 52473 | Chapter3 |

(b)

# Establishing a hard link

## Chapter3  Chapter3.hard



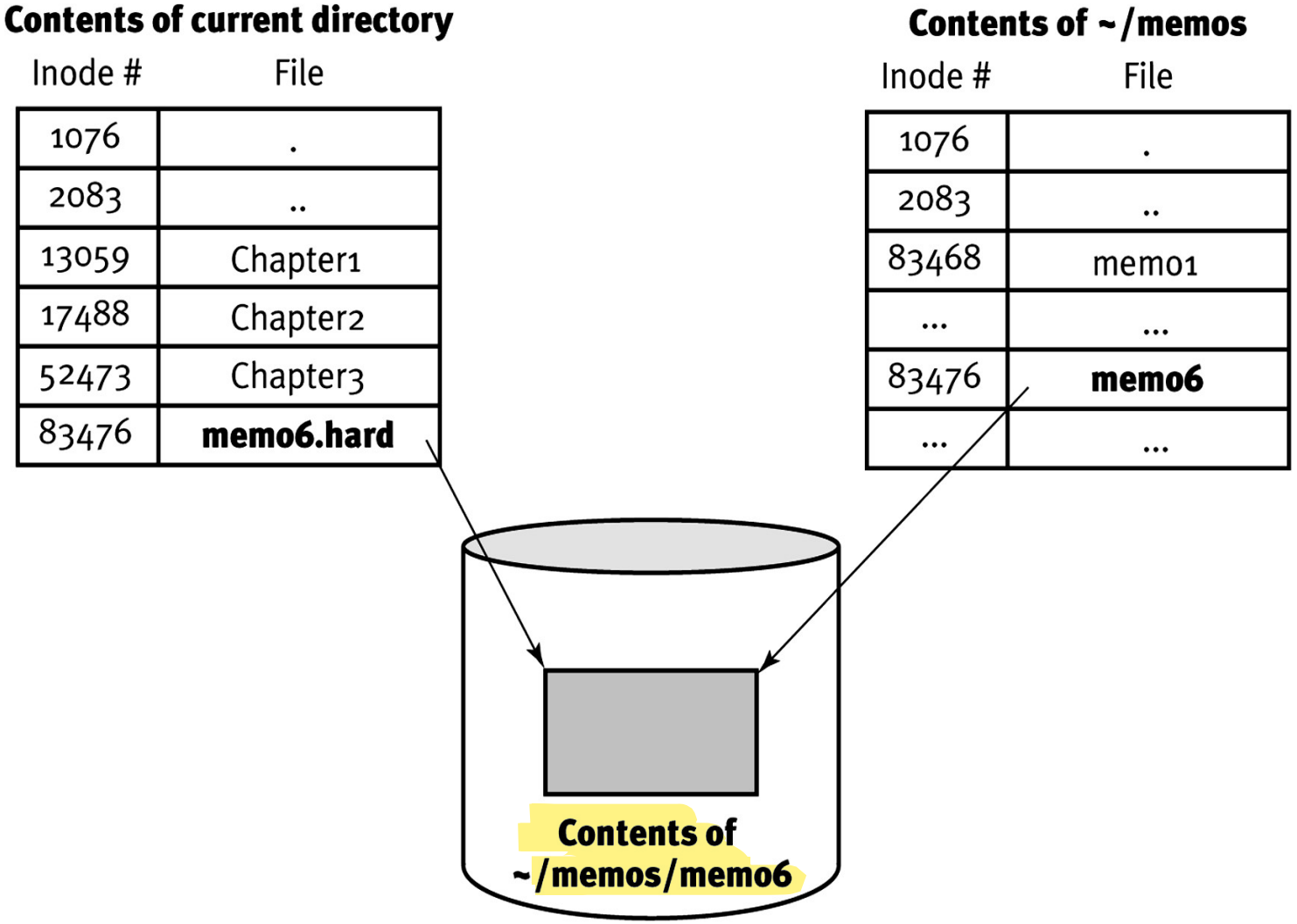

**Contents of current directory**

| Inode # | File |
|---|---|
| 1076 | . |
| 2083 | .. |
| 13059 | Chapter1 |
| 17488 | Chapter2 |
| 52473 | Chapter3 |
| 52473 | Chapter3.hard |

(b)

# Hard Link across directories

## ln memo6.hard memos/memo6



# Soft link or Symbolic link

- Many systems allows to use multiple names to refer to the same file
- A symbolic link is a directory mappings from a file name to another file name
- To open a file is opened via a symbolic link
  - File system first translates the name in the symbolic link to the target name and then uses the target name to open the file
- Example:
  - Create /a/b
  - Create a symbolic link from /c/d/ to /a/b
    - Unlinking /a/b, the file is no longer accessible and open("/c/d") will fail
- A MacOS file alias is similar to a symbolic link

# Lecture Summary

- When a file is created
  - System allocates a unique *inode* to the file
- *Inode* stores metadata of the file on disk
- Directory stores file name and *inode* number
- Hard and Soft (Symbolic) links provide a way to access a file using multiple pathnames