CS310 Operating Systems

Lecture 32 : File System - Introduction

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

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

We will study..

- File System – Introduction
- File System – Architecture – top level view
- File Naming
- File Structure
- File Types – Regular Files
- Binary Files – ELF format

File System - Introduction

Abstractions with Operating Systems

- Does a user need to know about processor ?
  - No. We have Process abstraction
- Does a user need to know about physical memory?
  - No. We have Process Address Space abstraction
- Does a user need to know about details of disk operations and configuration etc
  - No. We have File system abstraction

- Three Important Abstractions in OS
  - Process Abstraction
  - Address Space Abstraction
  - File Abstraction

Disk Storage

- Non-volatile Storage – for files
- Disk or Flash Memory – medium to store files
- Information is stored in blocks on the disks/Flash
- We can read and write blocks

- Henceforth – we will use term Disk to mean non-volatile storage including both Hard disks and Flash

Why Files ?

- Many important applications need to store more information then have in virtual address space of a process
  - Example: banking, Corporate record keeping, etc
- Process's lifetime is limited
- The information must survive the after termination of the process that was using it
  - For many application loss of data is not acceptable
- If a process gets killed because of some reason, information should not be lost forever
- Multiple processes must be able to access the information concurrently
  - This is possible by making information independent of address space of a single process

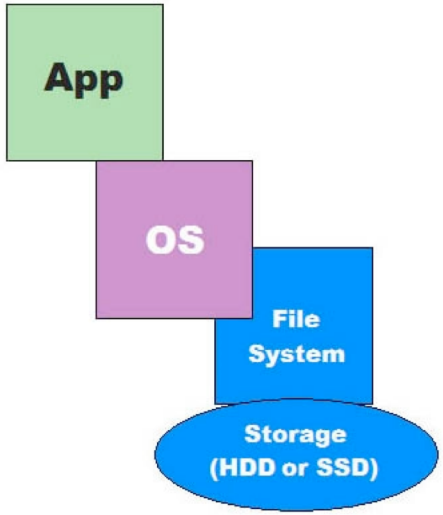
Disks - Reading and Writing

- Disk can be considered as a linear sequence of fixed-size blocks with two basic operations
  - Read block k
  - Write block k
- In a multiuser system, we can face following issues:
  - How do you find information?
  - How do you keep one user from reading other user's data?
  - How do you know which blocks are free?
- Different disks have different capacities, different speeds etc – how do you handle these differences?
- How does a user know all these info ?

- **Solution: File System**

File System

- Files are logical unit of information – Created by Processes
- A Disk contains 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 Systems – User's perspective

- What constitutes a file?
- How files are named?
- How files are protected?
- What operations are allowed on files?

File System Architecture – Top View

A new abstraction: File System

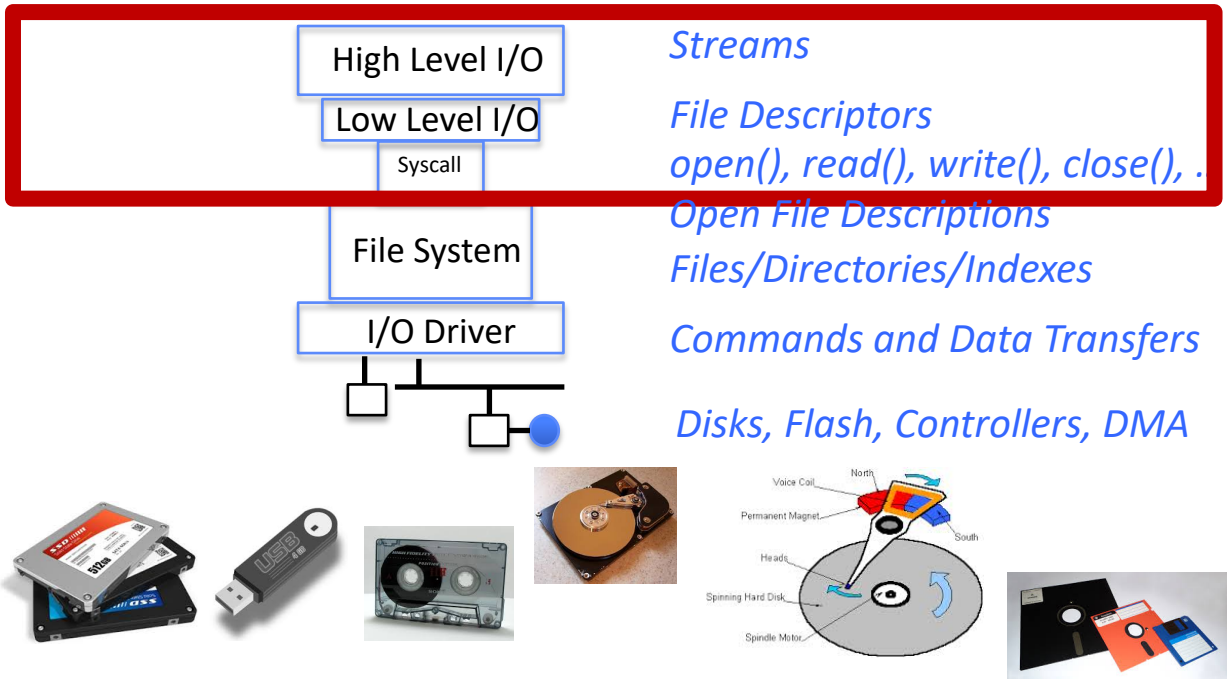
An OS abstraction that provides persistent, named data

File systems are a common operating system abstraction to allow applications to access non-volatile storage

I/O and Storage Layers

Application / Service

High level concepts



File Naming

File Naming

- File is an abstraction mechanism
  - A way to store info on disk and later read it back
- Writing and Reading file information should be done in a way that disk operation (where information is stored etc) details are not visible to users
- The most important characteristic of any abstraction is
  - Naming of objects

- When a process creates a file – it gives it a name
- When the process terminates, the file remains
  - It can be used by other processes

File Naming

- Naming conventions vary from system to system
- One to 8 letters in all current OS's
- Many file systems allow up to 255 characters

- Many OS support
  - Two part names
    - Exp: prog.c
    - File extension: part after period - Tells something about the file

Aside - Different File Systems

- FAT 16: MS-DOS, Windows 95
- FAT 32: Windows 98, Windows NT, Windows 2000, Windows XP
  - Obslete now
- NTFS (Native File System)
  - Latest Windows: 11, 10, 8.1 and MS Servers
  - Linux
- The Resilient File System (ReFS)
  - Microsoft’s newest file system
- BlueStore / Cephfs
  - Linux
- APFS
  - Apple
- EROFS
  - Android

Suffix Examples

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 Name – More info

- In Unix, the size of the extension is up to the user
  - File may have two or more extensions
    - Example: *homepage.html.zip*
- In Unix, extensions are just conventions
  - Not enforced by the OS
- A C compiler may actually insist that file to be compiled must end in .c
- Windows OS is aware of extensions
  - Assigns meaning to the extensions
  - → Which program must be executed when you double click on a file
    - Double click on *file.docx* will start Microsoft Word

File Structure

File Structure – Three categories

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

In use

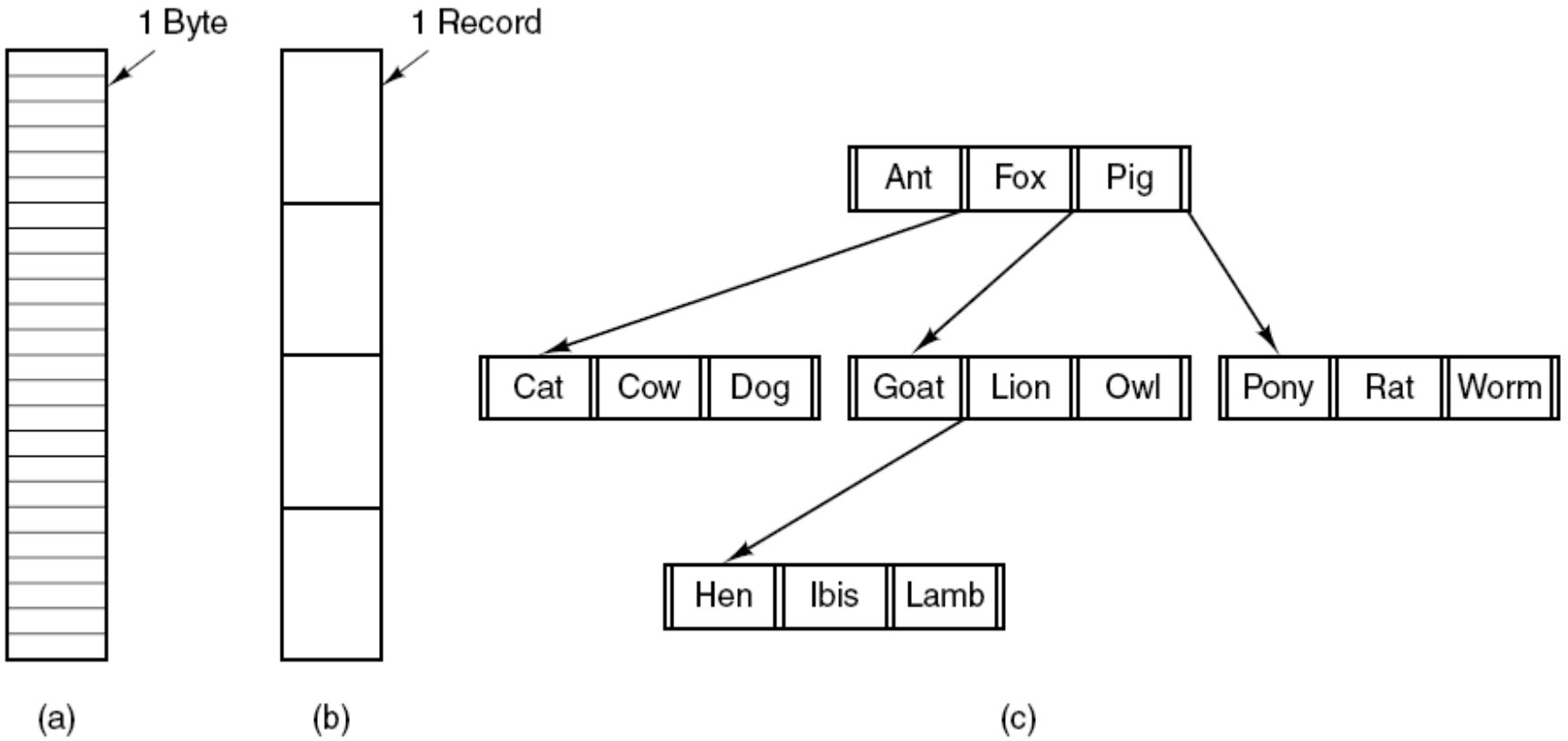
- Fixed length records (card images in the old days)
  - Read/Write operations are done on one record
    - Example: Based on punch cards (80 columns) – record 80 characters
    - Not in use
  - VMS OS provides highly structured file – with records

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

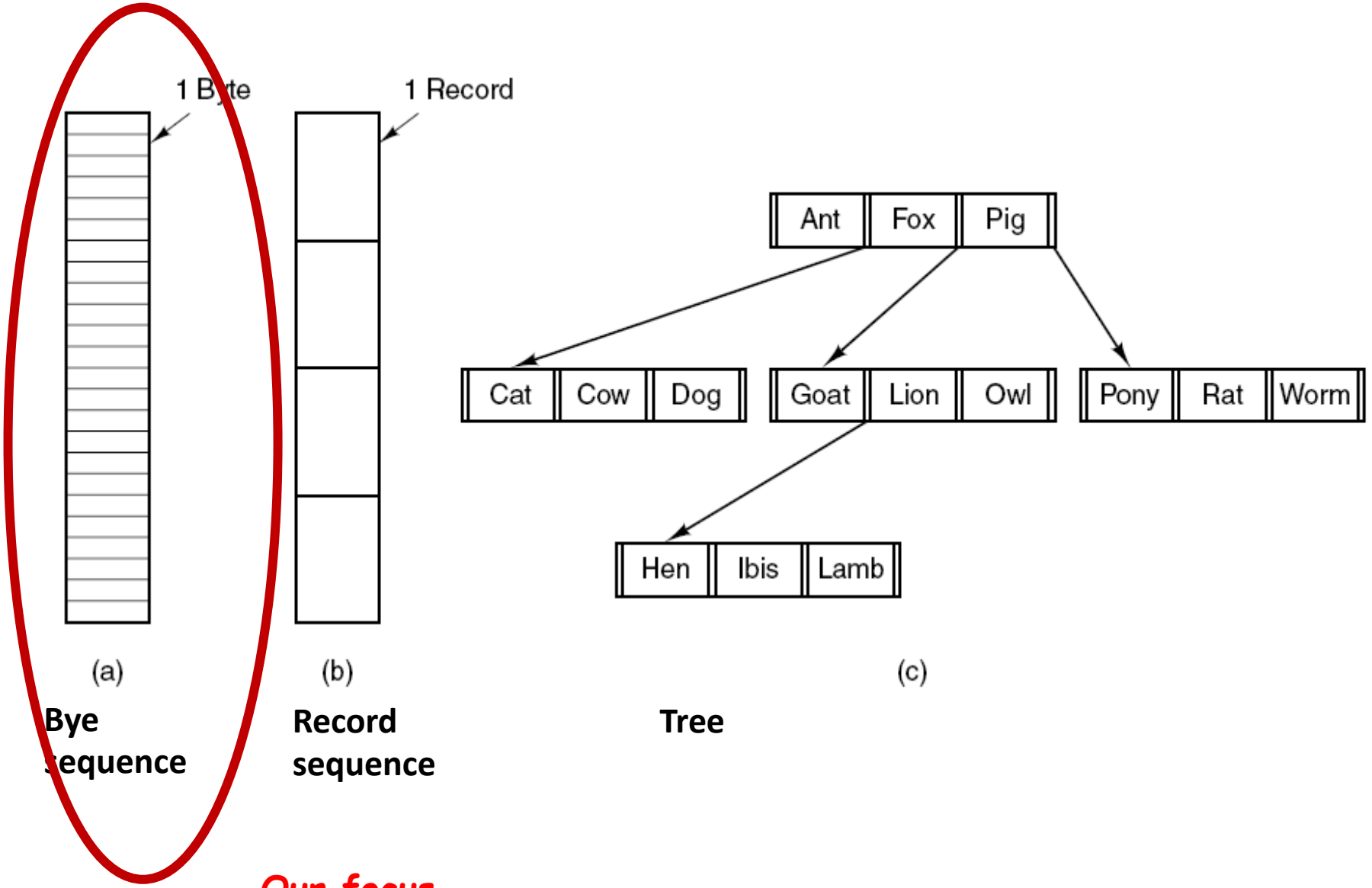


Bye sequence

Record sequence

Tree

File Structure – Three Types



Bye sequence

Record sequence

Tree

Our focus

File Structure – Byte Sequence

- Unstructured sequence of bytes
  - OS doesn't know what is in the file
  - Used in Unix , Windows, Linux
- Extremely flexible way of designing files
- User programs can put anything they want in their files and name them – as they want
  - OS does not enforce semantics of file contents

- Remember analogy of the Internet
  - It's just a pipe – it doesn't enforce any semantics to data being carried in packets

File Types

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)

- On file reading or writing 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)

- File position's maximum value
  - 64 bit in modern computers
  - OS specific
- Size of a file: length in bytes
- A file's length can be changed by truncation
  - Truncation to size smaller than original: Causes bytes removed from the end
  - Truncation to size larger than original: Causes bytes added at the end – filled with zeros
- A file may be empty – contains no valid bytes

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)

- *i-node number*
  - Stores metadata associated with the file
    - Owner, modification timestamp, type, length, location of the file's data (but no file name)
- *i-node* is
  - A physical object – located on disk (in Unix style filesystem)
  - Conceptual entity – represented by a data structure in the linux system

File Types – Regular Files - Text

- ASCII files
  - Lines of text
  - Lines are terminated with a carriage return character or line feed character (or both – as in Windows)
  - Lines need not be of the same length
  - ASCII files an be edited, displayed and printed
  - Can be used for input or output of a program
  - Can use pipes to connect programs if they produce/consume ASCII

File Types – Regular Files - Binary

- Binary files have some internal structure
  - Known to programs that use them
- Unix - Examples of Binary Files
  - Executable File Formats Library procedures (modules) that are compiled but not linked

- ELF: Executable and Linkable Format
  - Standard le format for executables, object code, shared libraries, and core dumps
- There are other formats as well: a.out, COFF, PE, Mach-O, COM, ...

Lecture Summary

- File system (User Perspective)
  - a collection of files and directories
  - Operations
- Files can be
  - Created, deleted, Read, written
- Files are named
  - Extension after period – tells us about the file
- Files types
  - Regular: Text and Binary
  - Directories
  - Block Special Files
- Binary Executable and Linkable file: ELF file format