OPERATING SYSTEMS

by
Marwa Yusuf

Lecture 16
Sunday 27-12-2020

Chapter 4 (4.1 to 4.2.4)
File Systems

Why we need from long-term storage?

1) Larger size than address space.
2) Non-volatile (does not disappear with process termination or system crash)
3) Independent from processes (like databases)

- Magnetic disks: used for a long time.
- Solid State Drives (SSD): used increasingly nowadays, faster and more robust.
- Tapes and optical disks: less performance, used for backup.

A disk

- A **disk**: Linear sequence of fixed sized blocks with 2 main operations (among others):
  1- Read block k 2- Write block k
- Questions (among others):
  1)How to find info?
  2)How to provide protection?
  3)How to manage free spaces/blocks?
- Processor → process(thread), physical memory → virtual address space, storage → **file**.
  - The three abstractions are the back one of an OS.

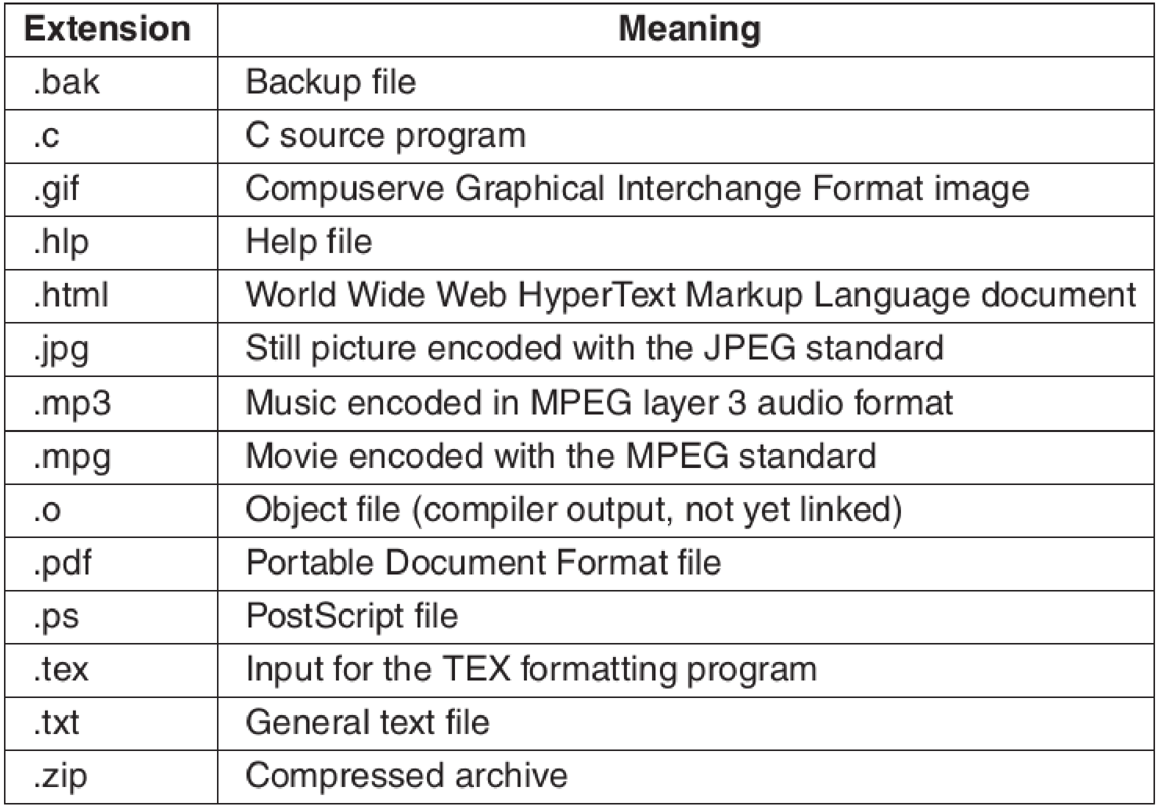
Files and File System

- Logical units of information created by processes.
- Many files exist in any given system (millions?)
- Persistent: deleted only by explicit order from the user.
- Mainly read and written (among other operations).
- Managed by OS (structure, name, access, protection, implementation … etc).
- **File System (FS)** (Fat16, Fat32, NTFS, ReFS, Ext4 … etc)
- A user is concerned by such things like naming, protection, allowed operations.
- A designer is concerned with such things as free storage management (lists or bitmap for ex.), sectors and so on.

File Naming

- Abstraction is to shield the user from lower level details.
- Naming rules vary from system to system.
  - From 8 to 255 characters, letters, sometimes digits and special characters.
  - Distinguish between lowercase and uppercase (as UNIX) or not (as MS-DOS: still in use in embedded)
- **File extension**: more info about the file
  - MS-DOS: optional, single, 1 to 3 characters
  - UNIX: optional, any length, one or more.
  - May be just a convenience (like in UNIX) to the user (not OS). A compiler may insist, but not OS.
  - Essential when a program can handle many kinds (like compiler)
  - May be a meaning and an owner is assigned to extension (like .docx in Windows)

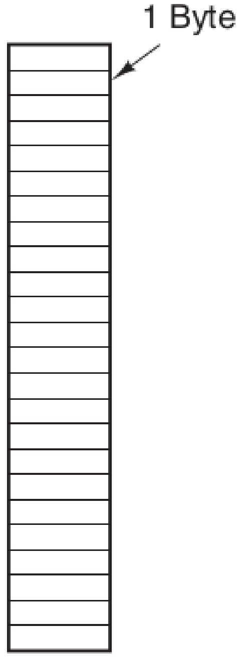
Some file extensions



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

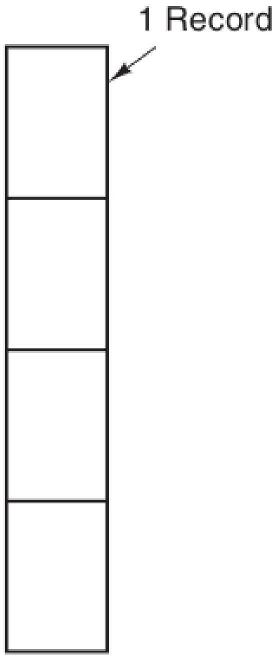
File Structure

- Files may be structured in any of several ways.
- Example: an unstructured sequence of bytes.
  - OS doesn’t know or care.
  - User-level programs impose meaning.
  - UNIX & Windows use this.
  - More flexibility.



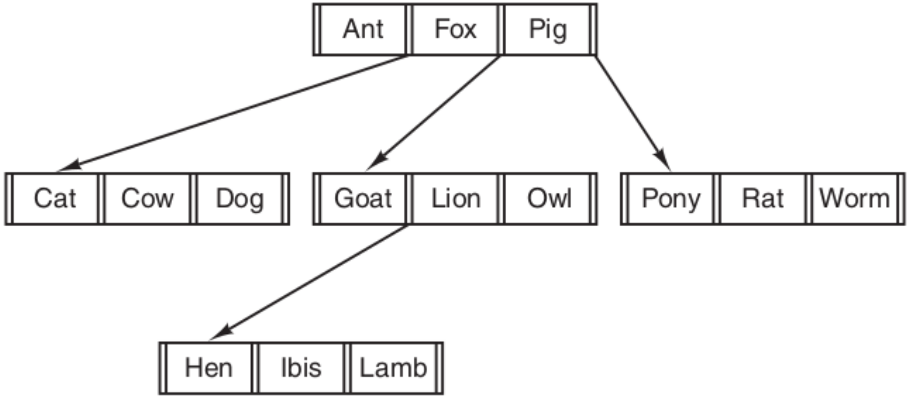
File Structure (cont.)

- A sequence of fixed-length records, each with some internal structure.
- Record is the unit of read/write.
- Historical, not used as a primary model currently in general-purpose.



File Structure (cont.)

- A tree of records, not necessarily all the same length, each containing a **key** field in a fixed position in the record.
- Sorted by **key** → rapid search.
- When adding, OS decides where to add.
- Used in large mainframes for commercial data processing.



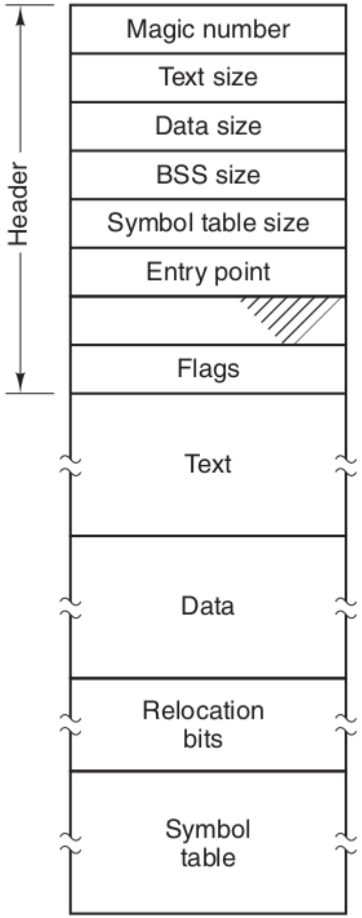
File Types

- Many OS support several types:
  - **Regular files**: for user info, like all the previous.
  - **Directories**: system files for FS structure.
  - **Character special files**: to model serial I/O devices like printers.
  - **Block special files**: to model disks.

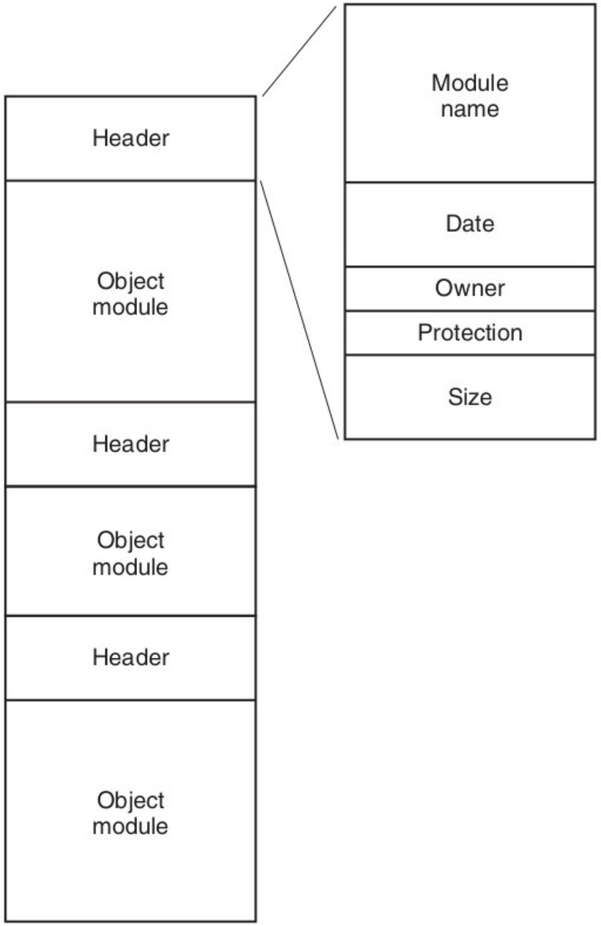
Regular Files

- Generally, **ASCII** or **binary**.
- **ASCII**: lines of text. Some systems end lines with **carriage return** or **line feed**. Lines may differ in length.
  - Can be displayed and printed as is. Can be edited with any text editor.
  - Can be pipelined between supporting programs.
- **Binary**: if printed → incomprehensible. Usually, internal structure known to using programs.

Binary File Ex. (early UNIX executable)



Binary File Ex. (UNIX archive)



File Types (cont.)

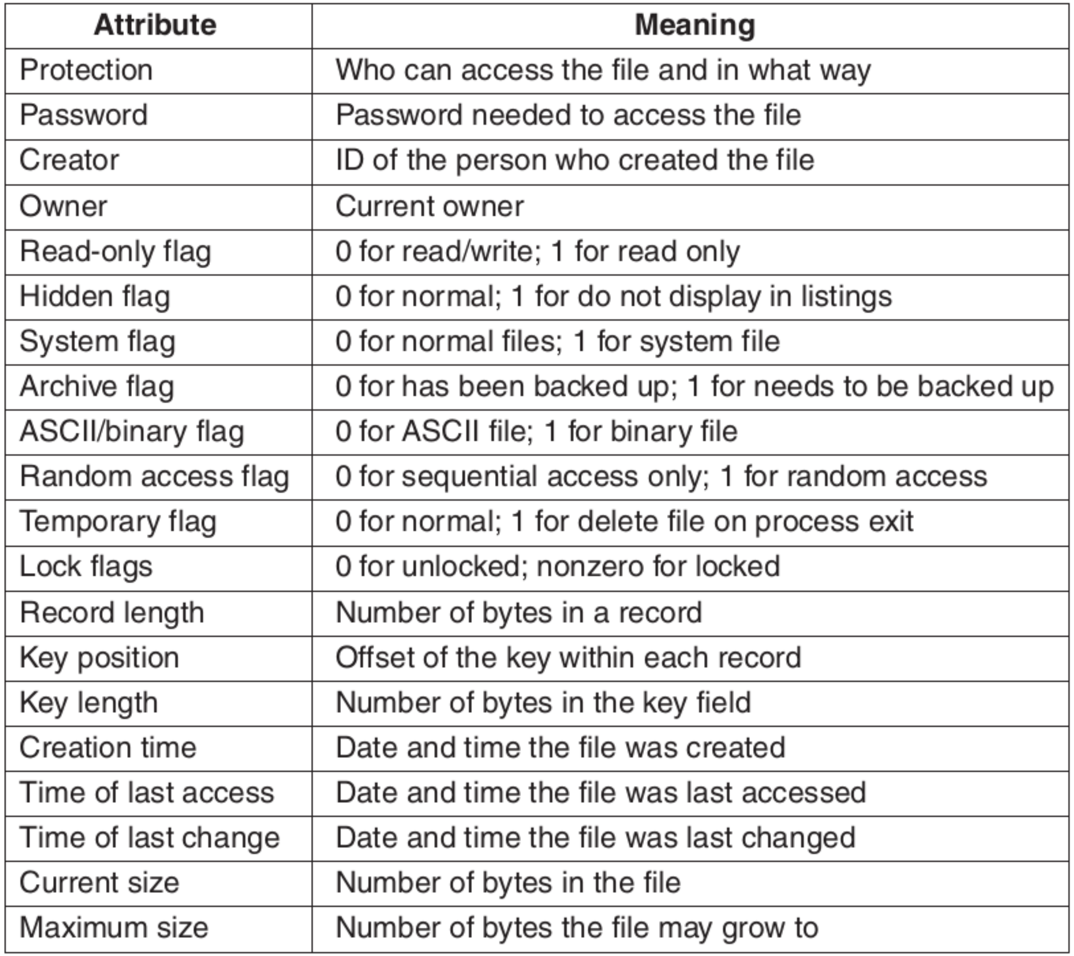
- Each OS must recognize at least its executable.
- Strongly typed files cause problems when the user tries to do something not intended in design (to protect the user).
    - Good for novices, may be frustrating for experienced.

File Access

- Earlier, all was **sequential**, read in order only.
  - Was suitable to magnetic tapes.
- **Random-access files**, out of order access, or by key.
  - Became possible with disks.
  - Required for many systems, like a database(ex: flight reservation).
  - Either **read** provides a position, or **seek** transform current position then read sequentially (Used in UNIX and Windows).

File Attributes (or Metadata)

- Archive flag: reset by backup process, set by OS.
- If last access time of source is after object, recompile.
- Some old systems required max. size at creation. Not nowadays.



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

- **Create**
  - With no data, announce, set attributes
- **Delete**
  - Free disk space
- **Open**
  - Fetch attributes & list of disk addresses into memory
- **Close**
  - Reverse open, OS may enforce max. # of open files.
- **Read**
  - Must provide size and buffer
- **Write**
  - If at the end → size increases. If in the middle → overwrite.
- **Append**
  - Restricted form of write. May not be present in some systems.
- **Seek**
  - In random access file, reposition *current* pointer.
- **Get Attributes**
  - Used by processes and users.
- **Set Attributes**
- **Rename**
  - Can copy with a new name and delete old instead.

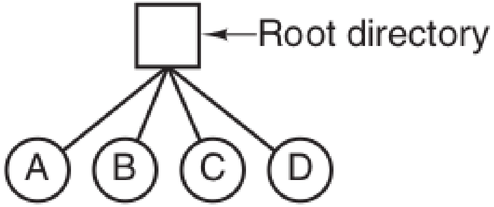
Example Program

- Code in Page 274.
- Command `copyfile abc xyz`

Directories of Folders

- To keep track of files.
- Are actually files.

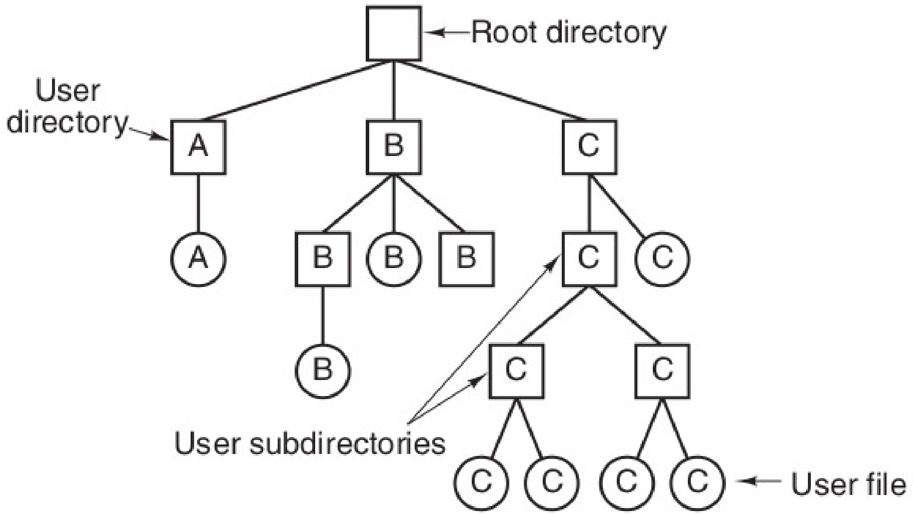
Single-Level Directory Systems



- One directory (usually called root, but whatever) contains all files.
- Early personal computers (one users).
- First supercomputer (simplicity).
- Advantages: simple, quick file locating.
- Still in use in some embedded, ex:digital cameras, portable music players.
- Not suitable for modern use (a lot of files).

Hierarchical Directory Systems

- To group related files together.
- A hierarchy: a tree of directories.
- Users can share a file server, each with his root directory.
- Nearly all modern FSs.



Path Names

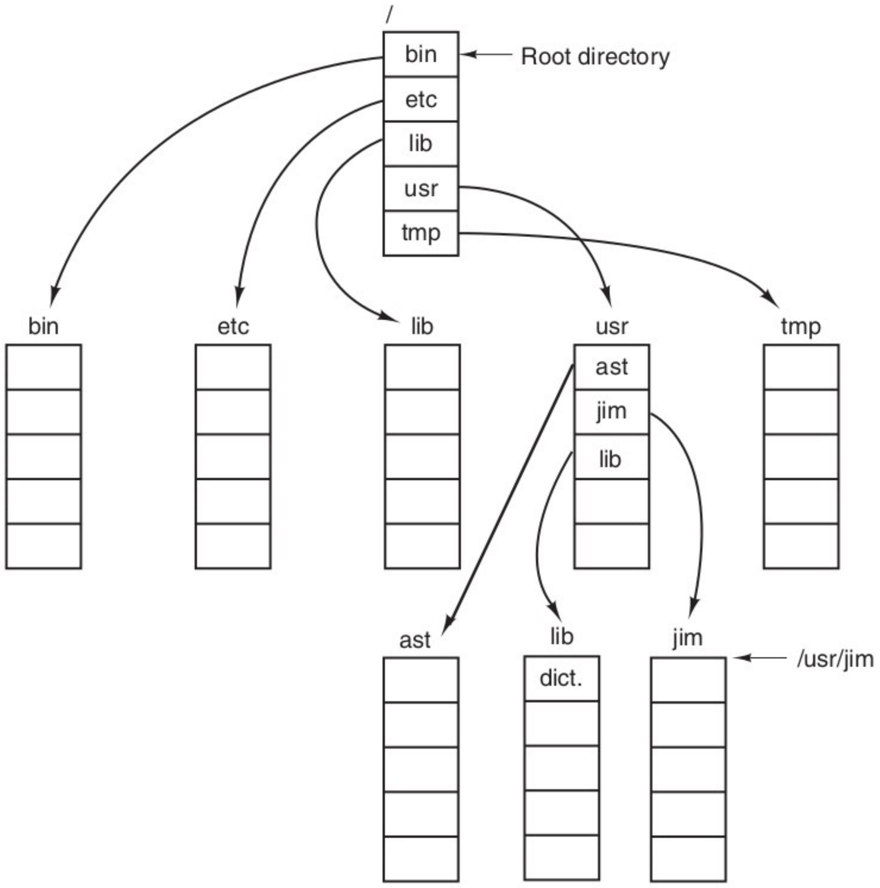
- To identify file names in a directory tree.

1) **Absolute Path Name**: path from *root* to file, unique
   - Ex: **Windows** \usr\ast\mailbox **UNIX** /usr/ast/mailbox **MULTICS** >usr>ast>mailbox
   - The 1st character is the separator → absolute

2) **Relative Path Name:** in conjunction with **working directory** or **current directory** designated by user.
   - All paths not beginning with *root* are relative to current directory.
   - Ex: if the current directory is **/usr/ast** then **/usr/ast/mailbox** can be referenced by **mailbox**
   - Each process has its own working directory. A library procedure must be careful whit changing working directory.

Path Names (cont.)

- Most OSs have special directories:
  - **“.”: dot**: current directory
  - **“..”:dotdot:** parent directory, except for root (itself).
- Ex: with working directory = /usr/ast, all the following are the same:
  - cp ../lib/dictionary .
  - cp /usr/lib/dictionary .
  - cp /usr/lib/dictionary dictionary
  - cp /usr/lib/dictionary /usr/ast/dictionary



Directory Operations

- More variation from OS to OS.

- **Create**
  - Empty except for **.** & **..**, put by OS or mkdir.
- **Delete**
  - Only if empty. **.** & **..** → empty.
- **Opendir**
  - Ex: for listing.
- **Closedir**
  - To free space.
- **Readdir**
  - Return next entry, **read** was used but forces knowing directory structure.
- **Rename**
- **Link**
  - **Hard link**: a link from a file to a pathname, so the same file may appear in multiple directories.
  - **Symbolic link**, a name for a tiny file naming another file. When opened, OS follows the path. Useful for across disks or even computers, but less efficient than hard links.
- **Unlink**
  - Remove directory entry. If present only here, delete from FS, otherwise, only this pathname is deleted. In UNIX, delete file is actually unlink.