



# *Introduction to File Systems*

## CS-3013 Operating Systems Hugh C. Lauer

(Slides include materials from  
Slides include materials from *Modern Operating Systems*, 3<sup>rd</sup> ed., by Andrew Tanenbaum  
and from *Operating System Concepts*, 7<sup>th</sup> ed., by Silberschatz, Galvin, & Gagne)



# *Discussion*

*(laptops closed, please)*

What is a *file*?



# *Reading Assignment*

- Tanenbaum, §4.1 – 4.2
- Note:– there is a lot more in this chapter than can be covered in these lecture topics!
  - Mostly deferred to CS-4513



## *File (an abstraction)*

- A (potentially) large amount of information or data that lives a (potentially) very long time
  - Often *much* larger than the memory of the computer
  - Often *much* longer than any computation
  - Sometimes longer than life of machine itself
- (Usually) organized as a linear array of bytes or blocks
  - Internal structure is imposed by application
  - (Occasionally) blocks may be variable length
- (Often) requiring concurrent access by multiple processes
  - Even by processes on different machines!



# *Review — Four Fundamental Abstractions*

- **Processes & threads**
  - This course (Tanenbaum, Chapter 2)
- **Virtual memory**
  - This course (Tanenbaum, Chapter 3)
- **Files & persistent storage**
  - CS-4513 with intro in this course (Tanenbaum, Chapter 4)
- **Sockets & connections**
  - CS-3516 or CS-513 (Computer Networks)



# *Review — Four Fundamental Abstractions*

- Processes & threads
  - This course (Tanenbaum, Chapter 2)
- Virtual memory
  - This course (Tanenbaum, Chapter 3)
- **Files & persistent storage**
  - CS-4513 with intro in this course (Tanenbaum, Chapter 4)
- Sockets & connections
  - CS-3516 or CS-513 (Computer Networks)



# *File*

- The third major abstraction of almost all operating systems
- An *organizing* abstraction
  - The way *information* is organized, stored, kept for a long time, and updated



# *File Systems and Disks*

- User view
  - File is a *named, persistent* collection of data
- OS & file system view
  - File is collection of disk blocks — i.e., a *container*
  - File System *maps* file names and offsets to disk blocks





# *Fundamental Ambiguity*

- Is the *file* the “container of the information” or the “information” itself?
- Almost all systems confuse the two.
- Almost all people confuse the two.



# Example

*Suppose that you e-mail me a document*

- Later, how do either of us know that we are using the *same version* of the document?
- *Windows/Outlook/Exchange/MacOS:*
  - Time-stamp is a pretty good indication that they are
  - Time-stamps preserved on *copy, drag and drop*, transmission via e-mail, etc.
- *Unix/Linux*
  - By default, time-stamps *not* preserved on copy, ftp, e-mail, etc.
  - Time-stamp associated with *container*, not with *information*



## *Rule of Thumb*

- Almost always, *people and applications* think in terms of the information
- Many *systems* think in terms of containers

Professional Guidance: *Be aware of the distinction, even when the system is not*



# Attributes of Files

- **Name:**
  - Although the name is not always what you think it is!
- **Type:**
  - May be encoded in the name (e.g., `.cpp`, `.txt`)
- **Dates:**
  - Creation, updated, last accessed, etc.
  - (Usually) associated with container
  - Better if associated with content
- **Size:**
  - Length in number of bytes; occasionally rounded up
- **Protection:**
  - Owner, group, etc.
  - Authority to read, update, extend, etc.
- **Locks:**
  - For managing concurrent access

Many systems co-opt container dates to be used in *lieu* of content dates



# *Definition — File Metadata*

- Information *about* a file
  - Maintained by the file system
  - Separate from file itself
  - Usually attached or connected to the file
    - E.g., in block # -1
  - Some information visible to user/application
    - Dates, permissions, type, name, etc.
  - Some information primarily for OS
    - Location on disk, locks, cached attributes



# *Observation*

- Some attributes are not visible to user or program
- E.g., *location*
  - Location is stored in metadata
  - Location can change, even if file does not
  - Location is not visible to user or program



# Question – Is Location a File Attribute?

- Example 1:–

```
mv ~lauer/project4.doc ~cs3013/public_html/c12
```

- Does *location* of file on disk change?

- Example 2:–

- System moves file from disk block 10,000 to disk block 20,000 (e.g., during defragmentation)
- System restores a file from backup

- May or may not be reflected in metadata



# *Question – Is Location a File Attribute?*

- Answer: It is an attribute of the *container*
- Not an attribute of the *information!*





# *File Name Attribute*

- *Not attached to file in most modern systems*
  - Stored in directory (see below)
- Unix/Linux – file may have multiple names
  - I.e., hard links
- Windows – file normally has only one name
  - Still stored in directory
  - May be changed without touching the file!

# File Types

*File type* may be tattooed on file as an attribute

| file type      | usual extension                | function  |
|----------------|--------------------------------|---|
| executable     | exe, com, bin or none          | read 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, rrf, doc              | various word-processor formats  |
| library        | lib, a, so, dll, mpeg, mov, rm | libraries of routines for programmers   |
| print or view  | arc, zip, tar                  | 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 |
| media          | mpeg, mov, rm                  | binary file containing audio or A/V information                                     |

Or it may be embedded in *file name* by convention



*Questions?*



# *Traditional Operations on Files*

- *Open, Close*
  - Gain or relinquish access to a file
  - OS returns a *file handle* – an internal data structure letting it cache internal information needed for efficient file access
- *Read, Write, Truncate*
  - *Read*: return a sequence of  $n$  bytes from file
  - *Write*: replace  $n$  bytes in file, and/or append to end
  - *Truncate*: throw away all but the first  $n$  bytes of file
- *Seek, Tell*
  - *Seek*: reposition *file pointer* for subsequent reads and writes
  - *Tell*: get current *file pointer*
- *Create, Delete*:
  - Conjure up a new file; or blow away an existing one



# *File – a very powerful Abstraction*

- Documents, code
- Databases
  - Very large, possibly spanning multiple disks
- Streams
  - Input, output, keyboard, display
  - Pipes, network connections, ...
- Virtual memory backing store
- Temporary repositories of OS information
- ...
- Any time you need to remember something beyond the life of a particular process/computation



# *Methods for Accessing Files*

- *Sequential* access
- *Random* access
- *Keyed* (or indexed) access
  - Hardly ever used any more!



# *Sequential Access Method*

- Read all bytes or records *in order* from the beginning
- Writing implicitly *truncates*
- Cannot jump around
  - Could possibly rewind or back up for some media
- Appropriate for certain media or systems
  - Magnetic tape or punched cards
  - Video tape (VHS, etc.)
  - Unix-Linux-Windows *pipes*
  - Network streams



# *Random Access Method*

- Bytes/records can be read in any order
- Writing can
  - Replace existing bytes or records
  - Append to end of file
  - *Cannot insert data between existing bytes!*
- *Seek* operation moves current file *pointer*
  - Maintained as part of “open” file information
  - Discarded on close
- Typical of most modern information storage
  - Data base systems
  - Randomly accessible multi-media (CD, DVD, etc)
  - ...





# *Keyed (or indexed) Access Methods*

- Access items in file based on the contents of (part of) an item in the file
- Provided in older commercial operating systems
  - IBM ISAM
- (Usually) handled separately by modern database systems



*Questions?*



# *Directory – A Special Kind of File*

- A tool for users & applications to organize and find files
  - User-friendly names
  - Names that are meaningful over long periods of time
- The data structure for OS to locate files (i.e., containers) on disk



# *Directory structures*

- Single level
  - One directory per system, one entry pointing to each file
  - Small, single-user or single-use systems
    - PDA, cell phone, etc.
- Two-level
  - Single “master” directory per system
  - Each entry points to one single-level directory per user
  - Uncommon in modern operating systems
- Hierarchical
  - Any directory entry may point to
    - Individual file
    - Another directory
  - Common in most modern operating systems



# *Directory Considerations*

- *Efficiency* – locate a file quickly
- *Naming* – convenient to users
  - Separate users can use same name for separate files
  - The same file can have different names for different users
  - Names need only be unique within a directory
  - Preferably in user's own language!
- *Grouping* – logical grouping of files by properties
  - e.g., all Java programs, all games, ...



# *Directory Organization – Hierarchical*

- Most systems support idea of current (working) directory
- Absolute names – fully qualified from root of file system
  - /usr/group/foo.c, ~/kernelSrc/config.h
- Relative names – specified with respect to working directory
  - foo.c, bar/bar2.h
- A special name – the working directory itself
  - “ ”  
.



## *Directory Organization (continued)*

- Modified Hierarchical – Acyclic Graph (no loops) and General Graph
- Allow directories and files to have multiple names
- Links are file names (directory entries) that point to existing (source) files



# Links

Substitute this string for the name you were looking for

- **Symbolic (soft) links:** uni-directional relationship between a file name and the file
  - Directory entry contains *text* describing *absolute* or *relative* path name of original file
  - If the source file is deleted, the link exists but points to nowhere!
- **Hard links:** bi-directional relationship between file names and file
  - A *hard link* is directory entry that *points* to a source file's metadata
  - Metadata maintains *reference count* of the number of hard links pointing to it – *link reference count*
  - Link reference count is decremented when a hard link is deleted
  - File data is deleted and space freed when the link reference count goes to zero





# *Unix-Linux Hard Links*

- File may have more than one *name* or *path*
- `rm`, `mv` —*directory* operations, not *file* operations!
  - The *real* name of a Unix file is internal name of its metadata
    - Known only to OS!
- Hard links are not used very often in modern Unix practice
  - *Exception*: Linked copies of large directory trees!
    - *When building your Linux kernel in OS course*
  - (Usually) safe to regard last element of path as *name* of file



# *Path Name Translation*

- Assume that I want to open “/home/lauer/foo.c”  

```
fd = open("/home/lauer/foo.c", O_RDWR);
```
- File System does the following
  - Opens directory “/” – the root directory is in a known place on disk
  - Search root directory for the directory `home` and get its location
  - Open `home` and search for the directory `lauer` and get its location
  - Open `lauer` and search for the file `foo.c` and get its location
  - Open the file `foo.c`
  - Note that the process needs the appropriate permissions at every step
- ...



## *Path Name Translation (continued)*

- ...
- File Systems spend a lot of time walking down directory paths
  - This is why `open` calls are separate from other file operations
  - File System attempts to *cache* prefix lookups to speed up common searches –
    - “~” for user’s home directory
    - “.” for current working directory
  - Once open, file system *caches* the metadata of the file



See Tanenbaum §4.2.4

# Directory Operations

- **Create:**
  - Make a new directory
- **Add, Delete entry:**
  - Invoked by file create & destroy, directory create & destroy
- **Find, List:**
  - Search or enumerate directory entries
- **Rename:**
  - Change name of an entry without changing anything else about it
- **Link, Unlink:**
  - Add or remove entry pointing to another entry elsewhere
  - Introduces possibility of loops in directory graph
- **Destroy:**
  - Removes directory; *must be empty*



## *Directories (continued)*

- **Orphan:** a file not named in any directory
  - Cannot be opened by *any* application (or even OS)
  - May not even have name!
- **Tools**
  - FSCK – check & repair file system, find orphans
  - *Delete\_on\_close* attribute (in metadata)
- **Special directory entry: “..” ⇒ parent in hierarchy**
  - Essential for maintaining integrity of directory system
  - Useful for relative naming



# *Directories — Summary*

- Fundamental mechanism for interpreting file names in an operating system
- Widely used by system, applications, and users



# *Reading Assignment*

- Tanenbaum, §4.1 – 4.2



*Questions?*