LECTURE 3.3
FILES

COP4600

Dr. Matthew Gerber

4/13/2015

FILES

Files (11.1)

- A file is typically the primary logical, user-facing unit of secondary storage
- From the user's point of view, data typically cannot exist apart from a file
- A file is, at minimum, a **named, sequential array of bytes**
- There are many different *types* of files but they all share a named and sequential nature
- *Files* are stored in *volumes*, which are housed in *partitions*

Attributes of Files (11.1.1)

Many attributes can be associated with files, but almost all systems have at least some variation of these.

- **Name:** The human-readable, symbolic file name
- **Identifier:** The file system’s “name” for the file, usually a number
- **Type:** The type of the file in systems that support explicit file types
- **Location:** A description of where the file resides on the underlying device
- **Size:** The size, in bytes (or words, or blocks) of the file
- **Protection:** Information about who is allowed to do what with the file
- **Last-Access Information:** May include date, time, and/or user who created, last modified, and/or last read the file

Operations on Files (11.1.2)

- **Creation:** Allocating space in the file system and the directory for the file
- **Writing:** Overwriting bytes in the file, or adding bytes to the end of the file, at the file's current *position*
- **Reading:** Reading bytes from the file's current position into memory
- **Seeking:** Setting the file's current position
- **Deleting:** Removing a file from the directory and releasing its space in the file system so that it can be used by other files
- **Truncating:** Like deleting, but preserves the directory entry

Working With Files

- Note that all of these operations need us to know where the file is in the file system and hence on disk; this is implicit per-file state information
- Three of them – reading, writing and seeking – also involve **explicit** per-file, **per-process** state: the position pointer
- To maintain this state information, we require processes to *open* files they need to work with
- Processes then (hopefully) *close* files when they are done with them
- The operating system maintains an *open file table* that contains this information
- The position pointer is maintained per process per file; the location information is maintained per file
- An *open count* is maintained so that the operating system knows when all processes using a file have closed it

File Types (11.1.3)

- There needs to be some indication of what kind of data a file contains before it is opened
- In almost all systems these days, this is handled on an application basis by the file extension; all the OS does is uses the extension as a hint to the graphical shell
- MacOS was the last holdout for file types as metadata, and basically gave up with OS X
  - (It does still have an explicit field designating which application created the file)
  - (This can be more useful than it sounds)
- UNIX used to use a system called *magic numbers*
  - Basically nobody even tries any more

Logical File Structure (11.1.4)

- It was once common for operating systems to provide services for more than one type of file
- This is increasingly rare; almost all files in almost all operating systems are now simply arrays of bytes
- The logical structures of almost all files are hence defined by applications, or by support libraries dedicated to individual types of files
- Obvious exceptions to this are executable files in their various forms, including programs and libraries

Physical File Structure (11.1.5)

- On disk, files are by definition stored in sectors
- For recordkeeping purposes, sectors are almost never divided between files
- Hence the last sector of a file almost always has some wasted space
- Most operating systems store files in *clusters* or *blocks* of multiple sectors
  - For obvious performance reasons, blocks are almost always multiples of the sector size
- This lowers the amount of record-keeping that needs to be done and allows for larger reads and writes, but also increases wasted space

Directories (11.3, 11.3.2)

- A given volume needs to retain information about what files are in it
- This information is maintained in a *directory*
- The directory will, at a minimum, contain the file's core attributes – name, identifier, type, location, size, protection, and last-access information
    - Some of this may be in the directory itself; some may be in a *file control block* that the directory references
- The primary job of the directory is to allow the operating system to find files
- The secondary job of the directory is to allow the *user* to find files

Directory Operations

- At a minimum, a directory must let us:
  - **Locate a file** by either its symbolic name or its identifier
  - **Create a file** and add it to the directory
  - **Delete a file** from the directory
  - **List the files** in a the directory
  - **Rename a file** in the directory
  - **Traverse the file system** and examine every file

Directory Levels

- The simplest directory structure is a single-level directory
  - Easy to understand and implement
  - Only works with small numbers of files
  - Does not effectively support multiple users at all
- Two-level directories were the next logical step
  - Supported multiple users with a separate directory for each user
  - Users saw *only* their own directory and were unaware that other directories existed
- Some concepts from two-level directories survive as extensions to the tree system

Directory Trees

- The overwhelmingly common directory structure in modern use
- The tree has a *root directory*, which can contain an arbitrary number of files and directories
- Each directory may in turn also contain an arbitrary number of files and directories
- Each process has a *current directory* or *working directory*
- Directories can therefore be accessed by *absolute* or *relative* paths

Directory Graphs

- Allows for directories to share entries
- The most common method of providing this is links in their various forms
  - *Soft* or *symbolic* links are pointers from a directory to another directory where the file actually resides
    - They are not links to *files*, they are links to *names* – hence the term “symbolic link”
  - *Hard* links are actual, equal-footing valid references to the *file* itself
    - Systems that support hard links must implement counters to know when a file can actually be deleted
- Cycles must be avoided, or at least managed

Mount Points

- For file systems to be used, they need to be placed somewhere that the system can find them
- This process is called *mounting* the file system
  - In UNIX-like systems, one file system is designated the *root file system*, and other file systems are mounted into empty directories
  - In Windows, each file system is assigned a *drive letter*
    - Recent versions of Windows can also use directory mounts, but drive-letter mounting is still more common
- Mounting of file systems on non-removable storage is almost always done at boot
- File systems on removable storage must be mounted whenever they are attached – and it must be possible to dismount them before detachment

NEXT TIME: FILE SHARING AND SYSTEMS