Chapter 10: File System

Thursday, November 08, 2007
11:46 AM

- File
  - Named collection of related information.
  - Recorded on secondary storage.
  - Smallest unit of logical secondary storage.
  - A sequence of bytes, basically.

- File Attributes
  - Name = human readable name
  - Identifier = unique number = non-human readable name
  - Type
  - Location = pointer to the device and the location of the file on the device
  - Size
  - Protection = who can read, write, or execute the file?
  - Time, date
  - User identification = which user and which group of users own the file

- Directory
  - A list of directory entries.
  - Directory entry = file name, and unique identifier
  - File identifier is used to locate other file attributes (in a centralized table, perhaps)
  - Directory is stored on secondary storage as well. So it may be treated as a file.
  - Can be nested to form a hierarchy of directories and files.

- Basic Operations
  - Creating a file
    - Find a space for it.
    - Make new entry in the containing directory.
  - Writing a file
    - Input = file name, write pointer, content to write
    - Search for the file in the directory structure.
    - Write to it.
    - Update write pointer.
  - Reading a file
    - Input = file name, read pointer, pointer to where to put the data read
    - Search for the file in the directory structure
    - Read data to the location pointed to by the pointer
    - Update read pointer
  - Repositioning (seek)
    - Since a process normally either reads or writes a file, there's usually no need for separate read and write pointer.
    - Instead, we have a per-process **file-position pointer.**
    - Input = filename, new position to seek to
    - Search for the file in the directory structure
    - Update file-position pointer
  - Delete a file
    - Release all file space
    - Update directory
  - Truncate a file
    - Remove all file content, but let directory entry survive.

- Open
  - To avoid having to search for the file again and again, some system require user to **open** a file before reading and writing it.

- Information about opened file is kept in **open-file table**.
- When a file is no longer being actively used, it is **closed**, and the corresponding entry in the open-file table is removed.
- Create and delete operation do not require opening a file.
- When many processes access a file at the same time, the system usually keeps two level of open-file table.
  - Per-process open-file table: Keeps file-position pointer of the process, access rights, accounting information, and pointer to the corresponding entry in the system-wide process.
  - System-wide table: location of file on disk, access date, file size, and open count.
- Information associated with an open file
  - File pointer = where in the file a process is reading or writing.
  - File-open count = Add one when a process opens the file. Subtracts one when a process closes the file. When zero, the system-wide open-file table entry is removed.
  - Disk location
  - Access rights = who can do what to a file.

- File Locking
  - Locking a file allows one process to prevent other processes from doing something to a file.
  - Types of locks
    - **Shared lock**: Several process can acquire the lock together. Usually, it is the read lock. When a process has the lock, other processes cannot write to the file.
    - **Exclusive lock**: Usually a writer lock. When one process has it, no process can read or write the file.
  - Lock enforcement
    - **Mandatory lock** = OS enforces the lock.
    - **Advisory lock** = The system does not prevent other processes from gaining access to the locked file. User program must be written so that it acquires the lock before gaining access.
    - Windows implements mandatory locks. Unix uses advisory locks.
  - User must be careful of deadlocks.

- File Types and Structures
  - A design decision: Should the OS support some types or structures of file?
  - Why?
    - To prevent common mistakes
      - Running a text file as executable.
      - Printing an executable file.
    - Relieve programmers of tedious work.
      - OSX expects every file to have **resource fork** and **data fork**.
      - In an executable file, user can change resource fork to change the label of a button in the program.
      - The data fork is the traditional file content.
  - Why not?
    - OS becomes large and complex if it supports a lot of file types or structures.
    - MS-DOS and Unix thinks that files are sequence of byte, and leave user programs to deal with the file content themselves.
  - How?
    - File extension.
    - Embedding file type information in file information kept by the file system.
      - OSX keeps name of the program that created the file.
    - **Magic number** stored at the beginning of the file.
      - Unix uses it to identify executable, batch file, PostScript file, etc.
  - All operating system must support one file type though: executable file.

- Access Methods
  - **Sequential Access**
    - Information is accessed one record after another.

        - Most common.
        - Tape model.
    - **Direct Access**
        - A file is made up of fixed length **logical records.**
        - Programs can read and write records in no particular order.
        - Disk model: disk contains blocks, and allows random access to blocks.
        - User normally provide **relative block number** relative to the beginning of the file to address a particular block.
    - Simulating sequential access on direct access file is easy. The other way around is inefficient.
    - **Indexed Access**
        - Usually used in database.
        - The data is indexed using some keys. The system keeps a small index table in memory.
        - To search for a data item, the given key is used to binary search the index. The index table then contains the pointer to the location of the corresponding record on the disk.
        - For large files, the index file itself may be too large to fit in the disk. The solution is to keep the index as a file, and make index for the index itself.
        - Example of such a system is IBM's **indexed sequential-access method** (ISAM)

- Storage Structure Vocab
    - **Partition** = a contiguous slice of the disk space.
    - **Volume** = parts of disks (can be more than one disk) that contains a single file system.
    - **Device directory** or **volume table of contents** keeps information of all files in the system such as location, size, type, and name.

- Directory (cont.)
    - Symbol table: Maps file name to the directory entry, which contains file information.
    - Operations
        - Search for a file
        - Create a file
        - Delete a file
        - List the entries
        - Rename a file
        - Traverse the file system
            - Scanning for virus, for example.
    - Organization
        - Tree Structured Directories
            - A directory contains a set of files of subdirectories.
            - A directory is just another file, but is treated in a special way.
                - All directories have the same internal format.
                - One bit in each directory entry specifies whether the file is a regular file or a directory.
                - System calls are used to create and delete directory.(Well, in monolithic kernel, that is.)
            - Each process has **current directory**.
                - Changing it is done through a system call.
                - The current directory of a subprocess is usually the one of the parent process.
            - When the user make reference to a file, the file is searched in the current directory first. If the file is not found, then the **search path** (a sequence of directories specified by the user) for the process is used to search for the file.
            - Path names
                - **Absolute path name**: Begin at the root.
                - **Relative path name**: Defines a path from the current directory.
            - User can impose structure onto his/her files.
        - Directed Acyclic Graph Directories
            - Permits directories to share of files and directory. Same file can be in one or

two directories.
        - When one user makes change to the shared file, other users see the change as well.
        - Implementation
            - Hard Links
                - A new type of directory called a **link**, which is a pointer to another file.
                - Links to a file are counted.
                - Files can only be deleted when the last link to it is removed.
                - Used in Unix.
                - Unix prohibit links to directory to ensure acyclic nature of the directory structure.
            - Symbolic Links
                - The link is just an absolute or relative path name.
                - When the user program references a link, the link is **resolved** by using the information contained therein to find the real file.
                - When the file system is traversed, this links are typically ignored.
                - Used in Unix as well.
            - Duplicating directory entries.
                - Two directories may contain the same directory entries.
                - Have to maintain consistency of information between directories when the file is modified.
        - Problems
            - When traverse a file system, we may operate on a file more than once.
            - Have to be careful when deleting a file.
                - Very problematic with the duplicate directory entry approach.
                - Deleting the file from one directory leaves dangling pointers.
                - What if the dangling pointer points inside another file?
                - With symbolic link, it's a lot easier. Dangling links doesn't hurt anyone.
    - Problems with cycles in file system.
        - Traversing becomes difficult.
            - Needs to keep track which parts have been traversed. Very expensive when the size of the graph is as large as the disk.
        - Safe deletion becomes difficult.
            - Have to use garbage collection to reclaim space. Very expensive again.
- File System Mounting
    - A file system must be **mounted** before it becomes available to the processes.
    - Mounting procedure
        - Input consists of a device name, and a **mount point**, which is location within the file directory structure where the directory structure on the device will be attached.
        - Typically, a mount point is an empty directory.
        - OS inspects that the device has a directory structure in the specified format.
        - OS records that a file system is attached to the mount point.
    - Mounting in some OS
        - When the Macintosh OS discovers a new disk, it mounts the disk on the root level, using the device name as the mount point.
        - Windows discovers disks at boot time and mount the file system using drive letters as mount points. It maintains a two level directory structure, with the top level being devices, each with a number of drive letters for the volumes contained therein.
        - The mount command is used explicitly in Unix. There's a file called fstab that tells Unix which devices to mount and where to mount file systems at boot time.

- File Sharing
    - Handling multiple users.
        - Notion of file owner and file group.
        - Owner can do anything to a file.
            - Read, write.
            - Change attributes.

        - Grant accesses.
    - Group = users that can share access to a file (can only do a subset of operations to it.)
    - All other users can perform another subset of operations.
    - What operations are allowed are determined by the owner.
    - Owner and group IDs are stored with the other file attributes.
- Distributed file system (DFS)
    - User from another machine can mount file systems or directories on other machines.
    - Machine containing the files = server. Machine seeking access to the file = client.
    - Must have a mechanism for granting access to appropriate user, and prevent spoofing of names. This can be handled by Distributed Information Systems.
    - One other important thing DFS has to deal with is failures of network. Consider the scenario when a network goes down while one client is reading or writing a file, for example.
    - To implement recovery, the DFS has to maintain some **state information** at both the client side and the server side.
    - Sun Network File System (NFS) takes the **stateless** approach.
        - NFS protocols carry all information about the file being operated, the file pointer, etc, so that operations can be performed without keeping information about which file is opened.
        - NFS is very easy to implement.
        - NFS thinks that all requests are valid. So it may be possible to forge read or write requests.
        - Newer versions of NFS is made stateful to improve security and performance.
- Consistency Semantics
    - Specify when modifications to file by one user is visible to other users.
    - **File session** = everything between matching open() and close().
    - Unix semantics
        - Writes to an open file by a user is immediately visible to other users.
        - One mode of sharing allows user to share pointer to current location of a file. So a write by one user updates the file pointer other users see as well.
    - Session semantics (Andrew File system (AFS))
        - Writes to an open file by a user is not visible immediately.
        - When a file is closed, changes made to it are visible only to file sessions started later.
        - Almost no constraints are enforced!
    - Immutable-shared-files semantics
        - Once a file is declared "shared," its content may not be altered, and its name cannot be rebound to anything else.

- Protection
    - Protect = prevent improper access
    - Types of Access
        - Read = can read
        - Write = can write, or rewrite the file
        - Execute = load the file to memory and execute it
        - Append = write to the end of the file
        - Delete = delete the file
        - List = list the names and attributes of the file
        - Typically, protections of these accesses are implemented. Other types of accesses can be thought of combinations of these accesses. Therefore, protections of the above accesses extend to other types of accesses as well.
    - Access control
        - Who has which type of access to which file?
        - Access-control list
            - A list of users and the type of accesses he has to the file
            - Enable complex access control
            - But the list can be long and hard to maintain.
            - Also complicates disk space management.

- Owner-group-universe access control
  - Three classifications of users
    - Owner = can do everything to a file
    - Group = A set of users sharing the file, typically designated by the owner.
    - Universe = All other users.
  - In Unix, implemented with rwx bits for owner, group, and universe. So 9 bit each for a file.
- Modern OS combines the two approaches.
  - Every file has the 9 rwx bits.
  - Users can also add access-control list.
  - Access-control list overwrites rwx bits.

- Other protection approach.
  - Passwords for reading, writing, and executing files.