Chapter 11: Implementing File Systems

Thursday, November 08, 2007
9:55 PM

- File system = a system stores files on secondary storage.
- A disk may have more than one file system.
- Disk are divided into blocks, the smallest contiguous fragment of the disk that can be read at a time. Typically, a block is 512 bytes.

- File system layers
  - Application programs -> logical file system -> file-organization module -> basic file system -> I/O control -> device
  - I/O control layer
    - Device drivers
    - Outputs bit patterns at specific ports.
  - Basic file system layer
    - Outputs high level commands such as "read block 123."
    - Is there to issue command to appropriate devices.
  - File-organization module
    - Maps logical blocks (indexed 0 to N) to physical blocks.
    - Manage free space.
    - Manage how to store blocks.
  - Logical file system
    - Manages metadata = things about the file except the actual data.
      - Name
      - Length
      - Date
      - Type
    - Metadata are stored in **file control blocks** (FCB)
    - Manages symbolic file names.
    - Responsible for protection and security.
    - Manage directory structures.

- Examples
  - Unix file system (UFS)
  - Extended file system (EFS) --- Linux
  - FAT, FAT32, NTFS --- Windows
  - ISO 9660 --- CD-ROMs

- File system implementation.
  - On-disk structures
    - Boot control block (per volume)
      - Contains information needed to boot an operating system contained in the volume.
        - Where the kernel resides on the disk.
      - Called **boot block** in UFS, **partition boot sector** in Windows.
    - Volume control block (per volume)
      - Contains volume details
        - Number of blocks
        - Free block counts
        - Free block pointers
        - FCB counts
        - FCB pointers
      - Called **super block** in UFS, and **master file table** in NTFS.
    - FCB (per file)
      - Metadata = permission, size, date, location on disk, etc.
      - In UFS, this is called the **inode**.

        - NTFS stored these information in the master file table.
    - Directory structure
        - UFS treats directories as files.
        - NTFS treats file and directories separately. Directory structure is stored in the master file table.
- In-memory structures
    - Mount table
        - Which volume is mapped to which directory?
    - Directory structure cache
        - Directory information of recently accessed directories.
    - System-wide open-file table
        - A copy of FCB for each open file.
    - Per-process open-file table
        - Per-process file-location pointer
        - Access mode.
        - Link to the corresponding entry in system-wide open-file table.
- What happen when a user creates a file?
    - The file system allocates an FCB.
    - The file system reads the directory the file is in into memory.
    - The file system updates the directory in memory.
    - The file system writes the directory back to the disk.
- What happen when a user opens a file?
    - The file system searches the system-wide open-file table for the file with the given name.
    - If the entry is not found, the file system searches the directory structure for the file of the given name, loads the FCB of the file from disk, makes new entry in the system-wide open-file table, and stores the FCB there.
    - The file system makes an entry in the per-process open-file table, and stores the pointer to the system-wide open-file table there along with some useful information.
    - The file system returns the pointer to the entry of the per-process open-file table as the return value of the open() system call.
        - This pointer is called the **file descriptor** in Unix.
        - It is called the **file handler** in Windows.
    - The process may now discard the name of the file, as it now can work with the file through the file descriptor instead.
- What happen when a user closes a file?
    - The per-process open-file table entry for the file is removed.
    - The reference count of the corresponding entry in the system-wide open-file table is decremented. If the reference count is zero, the entry is removed as well.
- Partitions
    - A partition in the disk that does not contains a file system is called **raw.**
    - Raw partitions is used for
        - Swap partition = only used as temporary storage of data
        - Boot partition = contains programs that used to boot the OS. Cannot have a file system because, at booting time, the OS and the file system is not loaded yet.
        - Database engine may use a raw partition so that it can manage the storage by itself.
    - Root partition = a partition that contains operating system kernel. Mounted at boot time.
- Mounting
    - The OS maintains an in-memory mount table.
    - The mount table tells which file system is mounted at which location.
    - The mount table for Windows is very simple. It associates drive letters to volumes.
    - The mount table for Unix is more complicated because a file system can be mounted at any directory.
        - Unix sets a flag in the in-memory copy of the inode of the mounted directory to indicate that the directory is a mount point.
        - A field in the inode then points to the mount table, which in turn contains

pointer to the superblock of the mounted volume.

- Virtual File System
  - OOP in action
  - Two roles
    - Provides a layer that decouples file-system-specific implementation from general file system interface.
    - Provides a mechanism for uniquely representing files across networks and volumes. VFS represent files using **vnode**s that contains unique numerical IDs for files across networks
  - VFS in linux
    - Four objects
      - **inode object**
      - **File object**
      - **Superlbock object**
      - **Dentry object** = directory entry
    - Each object has a number of methods.
    - Implementations are left to specific file system.
  - Examples: Gmail file system, NFS.

- Directory Implementation
  - Linear list
    - O(n) search --- very slow
    - Mitigate the slowness by software cache --- cache recently used directory information.
    - Or we can use sorted list, but insertion is expensive.
    - We can make insertion and searching in sorted list fast by using B-trees. But this is very complicated.
  - Hash table
    - Linear list is used to store directory list as usual.
    - A hash table is used to map file name to the appropriate entry in the list.
    - Problems: It has fixed size. Must reorganize disk storage when we grow the table.

- Allocation Methods
  - How to allocate free space for a file?
  - Contiguous allocation
    - Each file occupy a set of contiguous blocks on disk.
    - Advantage
      - Minimal number of head movements to access a file.
      - Sequential access is very fast.
      - Direct access is also fast.
    - Can be casted as a general **dynamic storage-allocation** problem as seen in memory management.
    - Suffers from **external fragmentation** --- when the largest contiguous chunk is not large enough.
    - Hard to implement because the size of a file might not be known in advance.
    - Extent system = modified contiguous allocation scheme
      - Contiguous space allocated initially.
      - If the space is not enough, another contiguous space, called **extent**, is allocated.
      - The location of the file then contains the beginning address, the block count, and the link to the extent.
      - If the extent is too large, then the file system suffers **internal fragmentation**.
      - External fragmentation is still a problem.
    - Example: UFS
  - Linked allocation
    - File = linked list of disk blocks.
      - Directory contains the pointer to the first and last block.
      - Each block contains a pointer to the next block.

        - The system must keeps a linked list of free space, but this is easy.
        - Pros
            - No **external fragmentation.**
            - No need to know the file size in advanced.
            - Sequential access is okay.
        - Cons
            - Scheme is inefficient if we were to implement direct access on top of it.
            - A file is larger that it should be because some space is used as "next" pointers.
            - Reliability: what happen if the pointers are lost of damaged?
        - Clustering
            - Collect contiguous blocks into **clusters.**
            - Make linked list of clusters rather than blocks.
            - Pros
                - Improves disk throughput.
                - Decrease space needed to manage free blocks.
            - Cons
                - Internal fragmentation.
            - The approach is used in most systems because it greatly improves performance.
        - Example: FAT
            - A section of disk at the beginning is used to contain the file allocation table.
            - The table is indexed by block numbers, and used as linked list. Each entry contains the next block in the file.
            - Directories contains file name and the start block.
            - The last block of a file contains a special end-of-file value in its table entry. This value is 0.
            - Improve performance by caching FAT in the main memory.
        - Indexed Allocation
            - Brings all pointers to the file blocks to a single location.
            - Each file has its own **index block**, an array of disk block address.
            - The directory contains the address to the index block.
            - Pros
                - Supports direct access.
                - Without suffering from external fragmentation.
            - Cons
                - Wasted space: overhead larger than linked allocation.
            - How large must the index block be?
                - Too large = large wasted space.
                - Too small = cannot represent large files.
            - Index block schemes
                - Linked scheme: index block contains link to the next index block.
                - Multilevel index: First level of index block points to index blocks that point to actual file blocks.
                - Combined scheme: Used in UFS. Keeps 15 pointers in the file's inode.
                    - The first 12 pointers point directly to file blocks (**direct blocks**). The next three pointers point to **indirect blocks.**
                    - The first indirect block is the **single indirect block**, a block containing pointers to file blocks.
                    - The second indirect block is the **double indirect block**, a block containing pointers to single indirect blocks.
                    - The third indirect block is the **triple indirect block**.
                    - Files can be as large as 4GB.

- Free Space Management
    - **Free-space list** keeps track of all free disk blocks. Need not be implemented as a list though.
    - Bit vector implementation
        - A huge bitmap in the superblock.
        - Each block is represented by 1 bit.

        - 1 = free, 0 = allocated.
        - Pros
            - Simplicity
        - Cons
            - Inefficient if the bitmap if not kept in main memory.
            - Bitmap can be really large, especially for hard disks nowadays.
    - Linked list implementation
        - Each free block has a pointer to the next free block.
        - Keeps the pointer to the first free block in emory.
        - Cons
            - Inefficient to traverse the disk. However, traversing does not happen frequently.
    - Grouping
        - Stores the address of n free blocks in the first free blocks.
        - The first n-1 blocks are actually free. The last free block is actually contains the address of another n free blocks, and so on.
        - A large number of free blocks can be found quickly. This is an improvement over the linked list implementation.
    - Counting
        - Stores address of the first free block, and then the number of free contiguous blocks following that block.
        - Each entry in free-space list = address of first block, and a count.
        - The list can be really large if the hard disk is really fragmented though.

- Caching
    - Disk controller has on-board cache that is large enough to store entire tracks.
        - Disk head reads all the sectors into the cache.
        - The request sector is transferred to memory from this cache.
    - Some OS maintain **buffer cache** in main memory.
        - Buffer cache stores blocks.
    - Some OS maintain file data using **page cache**.
        - Maps file data to virtual memory.
        - Store as pages rather than blocks.
        - User program interfaces with virtual memory rather than file system blocks.
        - Using page caching to cache both process pages and file data = **unified virtual memory**.
    - LRU works well as page replacement algorithm.
    - Two types of writes
        - **Synchronous write** = write occurs in the order which the disk subsytem receives them. Calling routine must wait for data to reach the disk before it proceeds.
        - **Asynchronous write** = data to write is stored in cache. The actual write happens much later, and the order is not the same as the subsystem receives.
        - In asynchronous write, the subsystem can arrange the pages to write so as to minimize seek time.

- Recovery
    - Consistency checking
        - The file system might suffer from system crashes while the data is being written to the disk.
        - The **consistency checker** (fsck in Unix, and chkdsk in MS-DOS) finds inconsistencies between the directory structure and the actual blocks on disk, and tries to fix them.
        - Very effective in linked allocation file system, but not so in indexed allocation system.
    - Backups

- Log-structured file system
    - Known as **log-based transaction-oriented** or **journaling** file system.
    - Example: NTFS and Veritas file system.

    - All metadata changes are written sequentially to a log.
    - **Transation** = a set of operations to perform a single task.
    - Once the changes are written to the log, they are said to be **committed**. The caller must wait until a change is committed before it can proceed.
    - Asynchronously, the log is played, and the changes are performed one by one. Once the change has been performed, the log entry is deleted.
    - The log is stored in a circular buffer.
    - When the system crashes, the log may contain some transactions. Transactions that are committed but has not completed is continued. Changes made by any transactions that has not been committed is undone.
    - Log file structure is fast because we can arrange changes to be made to the disk so that they can be performed efficiently. Moreover, writing to the log is fast because it's sequential.

- NFS
    - Client machine explicitly mounts a directory on a server.
    - If B mounts a directory in A, and C mounts a directory in B containing the directory in A that B mounted, then C doesn't see A's directory.
    - However, **cascading mount** is allowed. A client might mount a directory on server A, and, may mount a directory on server B to a subdirectory of the directory on server A.
    - NFS protocol is implemented using RPC primitives built on top of an external data represention (XDR) that is independent of particular file systems. As such, NFS can operate in heterogeneous environments.
    - Mount protocol
        - Server has an **export list** = a list of directories clients can mount.
        - When the server receives a mount request, it checks whether the directory the client wants to mount is in the export list. It also checks other credential information.
        - If the client is allowed to mount the directory, the server sends back a file handle for the client to use for further accesses.
    - NFS protocol
        - RPCs
            - Searching for a file within a directory.
            - Reading a set of directory entries.
            - Manipulating links and directories.
            - Accessing file attributes
            - Reading and writing files.
        - No open() and close(). NFS is **stateless**. Each request has a full set of arguments -- file identifier and offset.
        - Why? --- No special procedure is needed if the server crashes amidst of an operation.
        - File operations are **idempotent**.
        - An NFS request has a sequence number. This enables the server to determine if a request is duplicated or if anything is missing.
        - Statelessness implies synchrony. Every changes must be written to disk before the server replies to the client.
        - Server crash and recovery is invisible to the client.
        - Performance is not that good. Can be improved by using nonvolatile cache though.
        - NFS write procedure call is guaranteed to be atomic, and is not intermixed with other write calls. However, a write() system call might consist of many NFS write RPC. So users can still get their data intermixed.
    - Caching
        - File-attribute cache and file-block cache.
        - Cached file blocks are used only if the corresponding file attributes are up to date.
        - Cached attributes are discarded every 60 seconds.
        - Clients do not free delayed-write blocks before the server tells it that the block has been written to disk.
        - Caching introduces complexity in the consistency semantics of NFS.
            - Results of a write is not visible immediately to other clients. (Not UFS semantics.)

- Sessions that start later only see writes that have been flushed to the server disk. (Not AFS semantics.)