CSE451 Midterm Review and More File Systems Spring 2001

Gary Kimura
Lecture #20
May 9, 2001

Today's Topics

- Midterm Review
- Concentrate on the file system
- General features and semantics presented in a file system
- How does a file system impose structure on a disk
- How does a file system keep track of things in memory

Midterm Review

- Synchronization in particular deadlocks
- Memory Management Hardware Support
- Memory Management Overview
- Paging
- Page Replacement

The File System

- The File System's task is to present to the rest of the operating system a model of directories and files.
- It can only use the tools provided by the disk driver
- Besides reading and writing sectors to the disk, the disk driver also allows for a few control operations such as
  - Check state of media (write protected, removed, etc)
  - Query size of media
  - Etc.

Files

- A file is simply a named collection of data.
- The structure and interpretation of that data is typically defined by its creator and unknown to the file system.
- In some systems, though, the file type is known to the system, to prevent improper file manipulation.

Directories

- Directories support file naming and location.
- Most systems (like UNIX and Windows) support multi-level directories, where a file name describes its path from a root through the directories to the file at the leaf.
- Most systems have a *current* directory, from which names can be specified relatively, as opposed to absolutely from the root of the directory tree.

Directory Contents

- Conceptually is a collection of file headers each describing the logical information about a file, e.g.:
  - File name
  - File type
  - File size
  - Location on the disk
  - Protection
  - Creation, last access, and last modify time
  - Etc.

Typically User Level Interface

- The typically file system related API's are
  - Open file
  - Create file
  - Read File
  - Write File
  - Delete File
  - Open Directory
  - Enumerate Directory
  - Query File Attributes
  - Set file attributes (rename, protection, etc.)
- State is also kept for each opened file
  - Current file position
  - Sharing mode
  - And more

Kernel Specific Features

- In addition to the user level commands the file system must also deal with special kernel level request that deal with handling page faults.
  - This is necessary so the file system can distinguish which data structures need to reside permanently in memory and which can be paged
  - Typically Open Page File is the only API really necessary to do this
- Additionally the file system will expose volume managements calls such as
  - Mount Volume
  - Dismount Volume
  - Etc

Common terminology

- In discussing disks and file systems we often need to distinguish where the sector is located
  - Physical sectors are how we identify the sector by its location on the disk
  - Logical Sector Numbers (LSN) are used to identify the continuous stream of sectors presented by the driver driver
  - Virtual Sector Numbers (VSN) are used to identify the continuous stream of sectors presented by the file system

More terminology

- Besides location there is size
  - Clusters: is the minimum unit of allocation used by the file system (it can vary for each volume). A multiple of sectors (LCN, VCN)
  - Sectors: the minimum read/write granularity for a disk (LSN, VSN)
  - Bytes: the byte offset within the appropriate measuring scheme (VBO, LBO)

Disk Structure

- There is no structure to a disk except cylinders and sectors, anything else is up to the File System and Disk Driver.
- The File System imposes some structure on disks.
- Each disk contains:
  - 1. data: e.g., user files
  - 2. meta-data: File System info describing the disk structure
- For example, the free list is a data structure indicating which disk blocks are free. It is stored on disk (usually) as a bit map: each bit corresponds to one disk block.
- The File System may keep the free list bit map in memory and write it back to disk from time to time.

Meta-Data

- How the meta-data is represented is an File System issue, e.g., the free list could be a bit map, or a linked list (each free block points to next one), or something else.
- Disk storage (files) can be allocated in different ways:
  - contiguously on disk
    - it’s fast and simplifies directory access
    - it’s inflexible, causes fragmentation, needs compaction
  - linked structures
    - each block contains a pointer to the next
    - good only for sequential access
  - indexed structures
    - store index to all blocks in 1 index block
    - good for random access.

Storing Files

- Files can be allocated on disk in different ways, e.g.:
  - 1. contiguous allocation
    - like memory
    - fast and simplifies directory access
    - inflexible, causes fragmentation, needs compaction
  - 2. linked structure
    - each block points to next block, directory points to first
    - good for sequential access (bad otherwise)
  - 3. indexed structure
    - an “index block” contains pointers to many other blocks
    - better for random access
    - may need multiple index blocks (linked together)

On-Disk Structures

- Most volumes have a starting point where at a predefined location on the disk is the volume information. This is not usually LSN = 0. More like LSN = 1
- Think of this as the root directory of a volume it contains
  - volume size information
  - Creation information
  - Protection information
  - Label information
  - Free space and used space information

Example: FAT File System

- The DOS file system (also called the FAT file system) is a simple structure with plenty of bad examples but easy to understand. As we dissect it I’ll point out some of the short comings
- On a FAT volume there are four areas of interest
  - BIOS Parameter Block (BPB): identifies the volume as a fat file system
  - File Allocation Table (FAT): used to control the allocation and lookup of clusters for each file
  - Root Directory: contains entries for each file and directory on the root of the volume
  - File Data Area: used by the file system to store files and additional sub-directories

The BPB

- For small volume the BPB contains
  - BytesPerSector: size of a sector on the volume
  - SectorsPerCluster: number of sectors per cluster
  - ReservedSectors: number of sectors skipped before the first FAT
  - Fats: number of FATs on the volume (typically 2 for some bizarre reason)
  - RootEntries: number of files/directories we can have in the root directory
  - Sectors: number of sectors on the volume
  - SectorsPerFat: number of sectors needed to store each copy of the FAT

The FAT

- The volume is divided up into clusters each one with a number (i.e., LCN).
- The FAT is logically an array containing LCN values. The size of the FAT is the number of clusters on the volume.
  - For example, FAT[10] corresponds to the 10th cluster on the volume
- If a cluster is free then its FAT entry is 0.
- If a cluster is in use then its FAT entry the LCN of the next cluster in the file, or if is is the last cluster in the file then its value is –1
- Bad clusters also have a distinguished value

Directory Entries

- On FAT files could only be “8.3” (i.e., file names are at most 8 characters and extensions are at most 3 characters). This limits the maximum size of individual directory entries

```
typedef struct _PACKED_DIRENT {
    FAT8DOT3       FileName;             //  offset =  0
    UCHAR          Attributes;           //  offset = 11
    UCHAR          NtByte;               //  offset = 12
    UCHAR          CreationMSec;         //  offset = 13
    FAT_TIME_STAMP CreationTime;         //  offset = 14
    FAT_DATE       LastAccessDate;       //  offset = 18
    union {
        USHORT     ExtendedAttributes;   //  offset = 20
        USHORT     FirstClusterOfFileHi; //  offset = 20
    };
    FAT_TIME_STAMP LastWriteTime;        //  offset = 22
    USHORT         FirstClusterOfFile;   //  offset = 26
    ULONG32        FileSize;             //  offset = 28
} PACKED_DIRENT;                         //  sizeof = 32
```

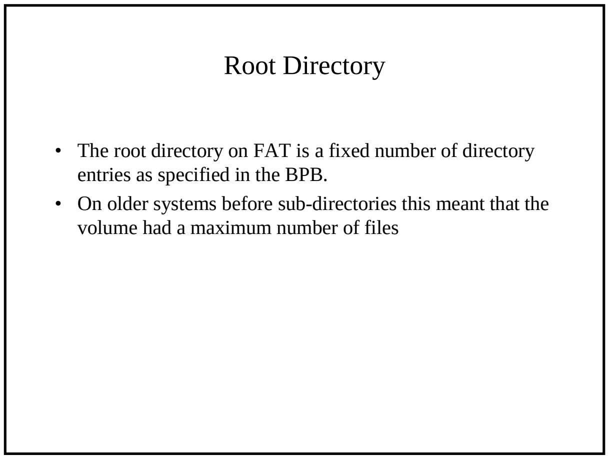
File Attributes

- The first byte in the directory entry tells use quite a bit about it

```
FAT_DIRENT_NEVER_USED            0x00
FAT_DIRENT_REALLY_0E5            0x05
FAT_DIRENT_DIRECTORY_ALIAS       0x2e
FAT_DIRENT_DELETED               0xe5
```

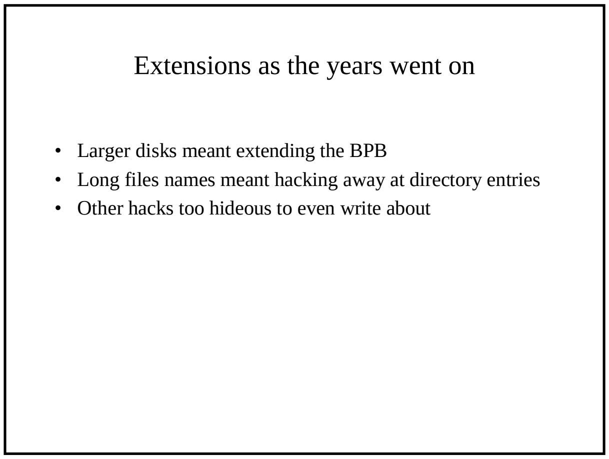
- The attribute byte tells us more

```
FAT_DIRENT_ATTR_READ_ONLY        0x01
FAT_DIRENT_ATTR_HIDDEN           0x02
FAT_DIRENT_ATTR_SYSTEM           0x04
FAT_DIRENT_ATTR_VOLUME_ID        0x08
FAT_DIRENT_ATTR_DIRECTORY        0x10
FAT_DIRENT_ATTR_ARCHIVE          0x20
FAT_DIRENT_ATTR_DEVICE           0x40
```



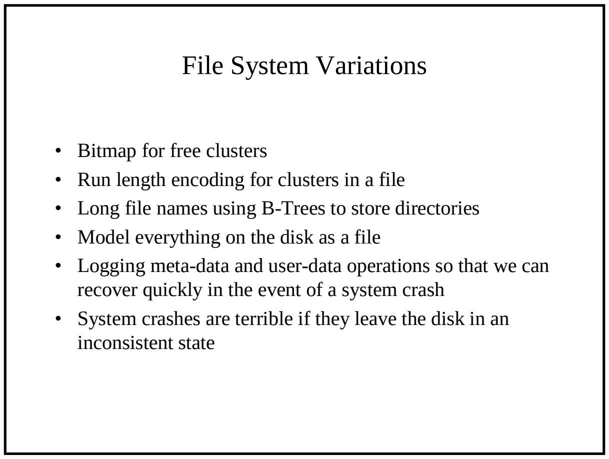
Root Directory

- The root directory on FAT is a fixed number of directory entries as specified in the BPB.
- On older systems before sub-directories this meant that the volume had a maximum number of files



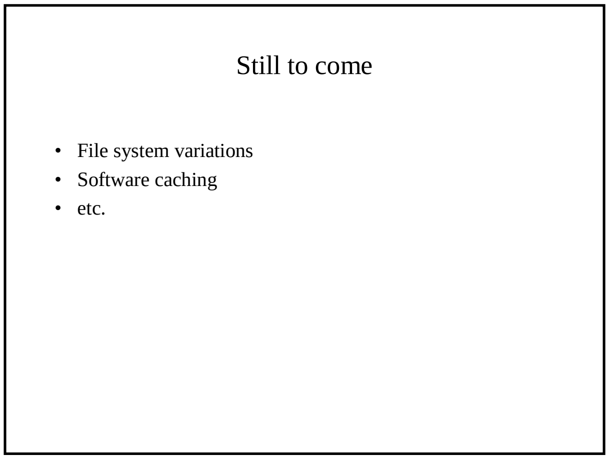
Extensions as the years went on

- Larger disks meant extending the BPB
- Long files names meant hacking away at directory entries
- Other hacks too hideous to even write about



File System Variations

- Bitmap for free clusters
- Run length encoding for clusters in a file
- Long file names using B-Trees to store directories
- Model everything on the disk as a file
- Logging meta-data and user-data operations so that we can recover quickly in the event of a system crash
- System crashes are terrible if they leave the disk in an inconsistent state



Still to come

- File system variations
- Software caching
- etc.