CSCI 3150 Tutorial on Assignment 2

LI Runhui

Dept. of Computer Science and Engeering,
Chinese University of Hong Kong

November 11, 2012

Outline

- Architecture of FAT32 File System
- Directories
- Filenames
- **Recovery**

Three Areas of FAT32 File System

- **Reserved Area**
  - **Boot sector** is located at Sector 0.
  - Storing important information about the file system.
- **FAT Area**
  - Contains a number of **FAT**s.
- **Data Area**
  - Stores **root directory** and other files and directories.

Three Areas of FAT32 File System

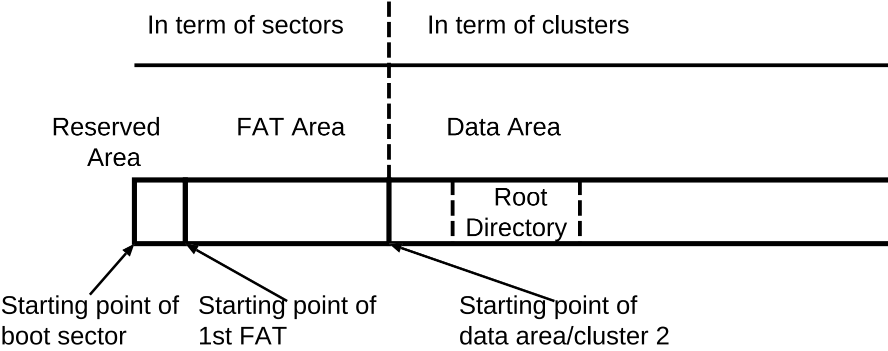

In term of sectors
In term of clusters
Reserved Area
FAT Area
Data Area
Root Directory
Starting point of boot sector
Starting point of 1st FAT
Starting point of data area/cluster 2

Figure: Three areas of FAT32 file system

Reserved Area

- Storing boot sector in this area.
    - It locates at sector 0.
    - Just read from the beginning of the file to get the boot sector.
- The boot sector is pre-defined.
    - Define a same data structure in your program.
    - Read the boot sector into your program.
    - Exact useful information.

Boot Sector

```
#pragma pack(push,1)
struct BootEntry {
 unsigned char BS_jmpBoot[3];      /* Assembly instruction to jump to boot code */
 unsigned char BS_OEMName[8];      /* OEM Name in ASCII */
 unsigned short BPB_BytsPerSec;    /* Bytes per sector. Allowed values include 512,
                                        1024, 2048, and 4096 */
 unsigned char BPB_SecPerClus;     /* Sectors per cluster (data unit). Allowed values are
                                        powers of 2, but the cluster size must be 32KB or
                                        smaller */
 unsigned short BPB_RsvdSecCnt;    /* Size in sectors of the reserved area */
 unsigned char BPB_NumFATs;       /* Number of FATs */
 unsigned short BPB_RootEntCnt;    /* Maximum number of files in the root directory for
                                        FAT12 and FAT16. This is 0 for FAT32 */
 unsigned short BPB_TotSec16;      /* 16-bit value of number of sectors in file system */
 unsigned char BPB_Media;          /* Media type */
 unsigned short BPB_FATSz16;       /* 16-bit size in sectors of each FAT for FAT12 and FAT16.
                                        For FAT32, this field is 0 */
 unsigned short BPB_SecPerTrk;     /* Sectors per track of storage device */
 unsigned short BPB_NumHeads;      /* Number of heads in storage device */
 unsigned long BPB_HiddSec;        /* Number of sectors before the start of partition */
 unsigned long BPB_TotSec32;       /* 32-bit value of number of sectors in file system.
                                        Either this value or the 16-bit value above must be
                                        0 */
```

Boot Sector

```
 unsigned long BPB_FATSz32;         /* 32-bit size in sectors of one FAT */
 unsigned short BPB_ExtFlags;       /* A flag for FAT */
 unsigned short BPB_FSVer;          /* The major and minor version number */
 unsigned long BPB_RootClus;        /* Cluster where the root directory can be found */
 unsigned short BPB_FSInfo;         /* Sector where FSINFO structure can be found */
 unsigned short BPB_BkBootSec;      /* Sector where backup copy of boot sector is located */
 unsigned char BPB_Reserved[12];    /* Reserved */
 unsigned char BS_DrvNum;           /* BIOS INT13h drive number */
 unsigned char BS_Reserved1;        /* Not used */
 unsigned char BS_BootSig;          /* Extended boot signature to identify if the next three
                                       values are valid */
 unsigned long BS_VolID;            /* Volume serial number */
 unsigned char BS_VolLab[11];       /* Volume label in ASCII. User defines when creating the
                                       file system */
 unsigned char BS_FilSysType[8];    /* File system type label in ASCII */
};
#pragma pack(pop)
```

FAT Area

- There may be more than 1 FATs.
- How to access every FAT:
  - The 1st FAT locates right after the reserved area.
  - The n-th FAT locates right after the $(n-1)$-th FAT.
  - The size of reserved area and FAT and # of FATs can be found at boot sector.

Data Area

- How to access the data area
  - Locates right after the FAT area
- Data is stored in terms of **clusters**.
  - Cluster is the basic unit of data storage in FAT 32 file system.
  - Cluster is a number of contiguous sectors.
  - The # of sectors per cluster is defined in boot sector.
  - The first cluster in data area is **Cluster 2**.
  - There are **NO** Cluster 0 and Cluster 1.

Root Directory

- **Root directory** is located in data area.
- The position of root directory can be looked up in the boot sector.

Directory

- **Directory** contains a number of **directory entries**.
- **Directory Entry** is a **pre-defined** data structure.
  - Each directory entry corresponds to a file/sub-directory.
  - It contains information like filename, file size, file location, etc.

Directory Entry

```
#pragma pack(push,1)
struct DirEntry
{
unsigned char DIR_Name[11];    /* File name */
unsigned char DIR_Attr;        /* File attributes */
unsigned char DIR_NTRes;       /* Reserved */
unsigned char DIR_CrtTimeTenth;/* Created time (tenths of second) */
unsigned short DIR_CrtTime;    /* Created time (hours, minutes, seconds) */
unsigned short DIR_CrtDate;    /* Created day */
unsigned short DIR_LstAccDate; /* Accessed day */
unsigned short DIR_FstClusHI;  /* High 2 bytes of the first cluster address */
unsigned short DIR_WrtTime;    /* Written time (hours, minutes, seconds */
unsigned short DIR_WrtDate;    /* Written day */
unsigned short DIR_FstClusLO;  /* Low 2 bytes of the first cluster address */
unsigned long DIR_FileSize;    /* File size in bytes. (0 for directories) */
};
#pragma pack(pop)
```

Filename (DIR_Name[])

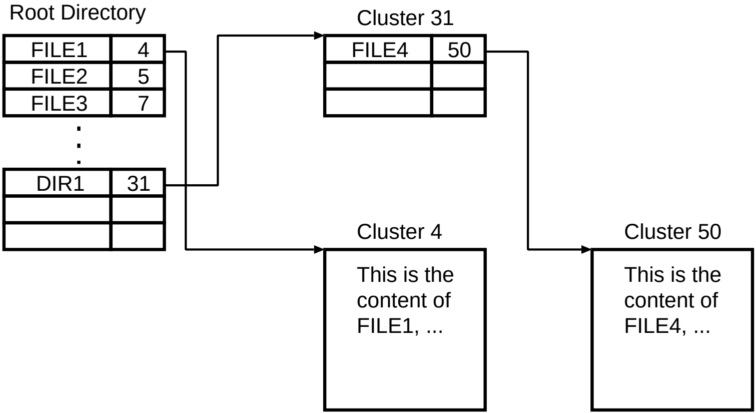
- 8 bytes for **filename**, 3 bytes for **file extension**.
- All letters are capital letters.

filename	DIR_Name[]
NAME.EXT	NAME␣␣␣␣EXT
NAME	NAME␣␣␣␣␣␣␣
NAME.A	NAME␣␣␣␣A␣␣
LONGNAME.EXT	LONGNAMEEXT
.EXT	illegal!

DIR_Attr

- There are 8 bits in **DIR_Attr**, every bit has its own meaning:
    - **0000 0001 0x01 Read only**
    - **0000 0010 0x02 Hidden**
    - **0000 0100 0x04 System**
    - **0000 1000 0x08 Volumn label**
    - **0001 0000 0x10 Directory**
    - **0010 0000 0x20 Archive**
    - **0000 1111 0x0f Long filename**
- A file can have multiple properties:
  e.g. a read-only hidden directory:
  DIR_Attr$= 0x01|0x02|0x10 = 0x13 = (00010011)_2$

Directory



Root Directory

FILE1	4
FILE2	5
FILE3	7
⋮	
DIR1	31

Cluster 31

FILE4	50

Cluster 4

This is the content of FILE1, ...

Cluster 50

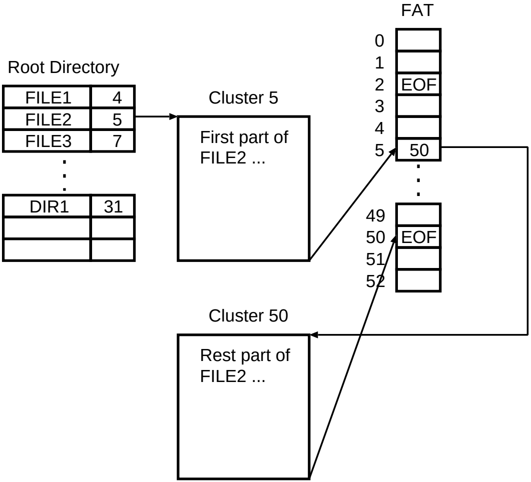
This is the content of FILE4, ...

Figure: An example of directory and directory entries

Large File

- The file size may be larger than the size of 1 cluster.
- Use **FAT** to address this problem.
  - FAT can be viewed as an integer array.
  - Each entry is a 32-bit (i.e., 4-byte) unsigned integer.
  - The value of FAT[i] is the index of the next cluster.
    - e.g. FAT[10]=15 means the cluster after cluster 10 is cluster 15.
  - Although data area starts from cluster 2, there are still FAT[0] and FAT[1] in the FATs.

Large File



Root Directory

FILE1	4
FILE2	5
FILE3	7
⋮	
DIR1	31

FAT

0	
1	
2	EOF
3	
4	
5	50
	⋮
49	
50	EOF
51	
52	

Figure: File with length more than the size of a cluster

After A File is Deleted

- After a file is deleted, there will be 2 mainly 2 changes:
- In the corresponding **directory entry**:
  - The first character in DIR_Name[] is changed to **0xe5**.
- In the **FAT**:
  - For every Cluster i, the value of FAT$[i]$ will be changed to **0**.
- Everything else is unchanged:
  - File length
  - Starting cluster
  - DIR_Attr, etc.

Before Deletion

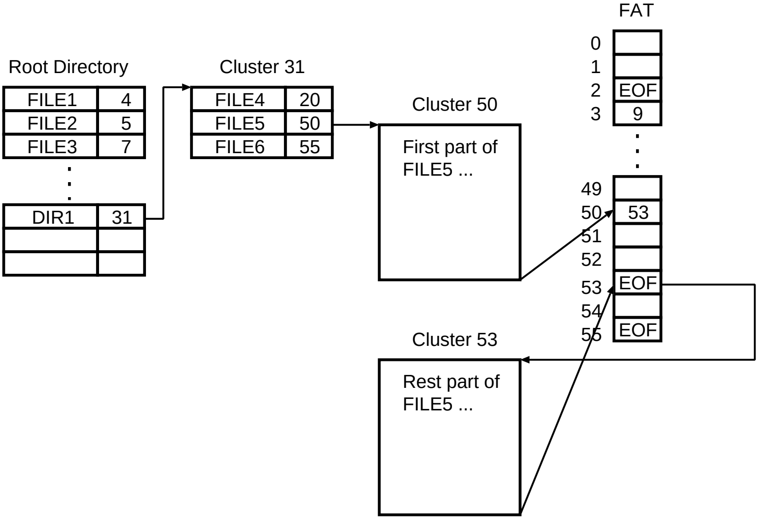
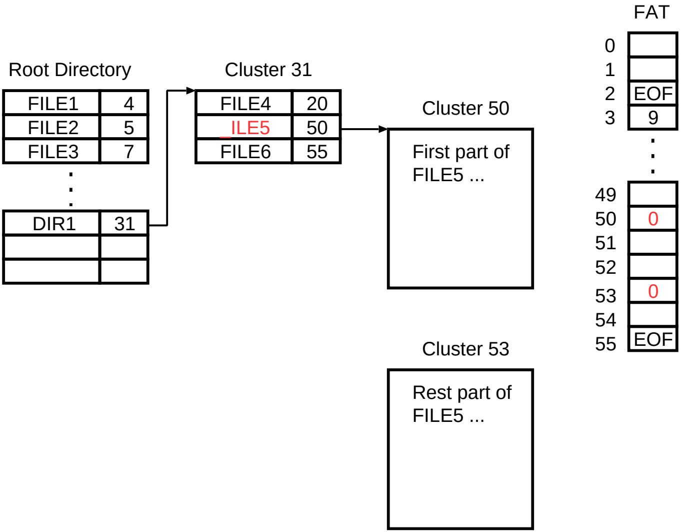


Figure: Before Deleting FILE5

After Deletion



Root Directory

FILE1	4
FILE2	5
FILE3	7
⋮	
DIR1	31

Cluster 31

FILE4	20
_ILE5	50
FILE6	55

Cluster 50

First part of FILE5 ...

Cluster 53

Rest part of FILE5 ...

FAT

0	
1	
2	EOF
3	9
⋮	
49	
50	0
51	
52	
53	0
54	
55	EOF

Figure: After Deleting FILE5

Recovery

BASIC IDEA: UNDO the changes.

1. Change back the first character of the **DIR_Name[]**.
    - The first character of DIR_Name[] is changed to 0xe5, change it back!
    - How can I get to know the filename before deletion?
        - The filename before deletion is supposed to be provided by the user.

Recovery

2. Change back the **FAT entries** which are changed to 0.
   - The related FAT entries are changed to 0, change them back.
   - The starting cluster can be find in the **directory entry**.
   - To simplify your task:
     - We assume that all the deleted file are stored in **contiguous** clusters.
     - Suppose we are recovering a file. Then for every Cluster i, set FAT$[i] = i + 1$. As for the last cluster Cluster j, set FAT$[j]$ =EOF.

Recover Files with Long Filename (LFN)

- There will be more than 1 directory entries describing a file with long filename.
  - **One** of them show the useful information about the file, file length, attribute, starting cluster, etc.
  - Others with DIR_Attr$= 0\mathrm{x}0f$ contain the real filename of the file.
- We just consider the following type of long filename:
  - The filename obeys the **8.3** format.
  - All characters except the dot **must** be numbers and lower/upper case letters.

Recover Files with Long Filename (LFN)

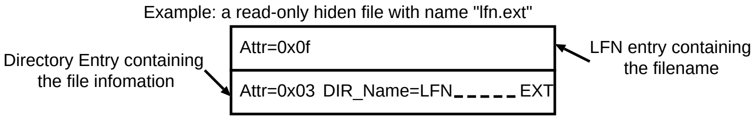

Example: a read-only hiden file with name "lfn.ext"
Attr=0x0f
Attr=0x03 DIR_Name=LFN_____EXT
Directory Entry containing the file infomation
LFN entry containing the filename

Figure: Entries of file with long filename

Q&A

Thank You!