

实验十三 扫描 FAT12 文件系统管理的软盘

一、实验目的

通过查看 FAT12 文件系统的扫描数据，并调试扫描的过程，理解 FAT12 文件系统管理软盘的方式。

通过改进 FAT12 文件系统的扫描功能，加深对 FAT12 文件系统的理解。

二、实验内容

2.1 准备实验

请读者按照下面方法之一在本地创建一个 EOS 内核项目，用于完成本次任务：

方法一：从 CodeCode.net 平台领取任务

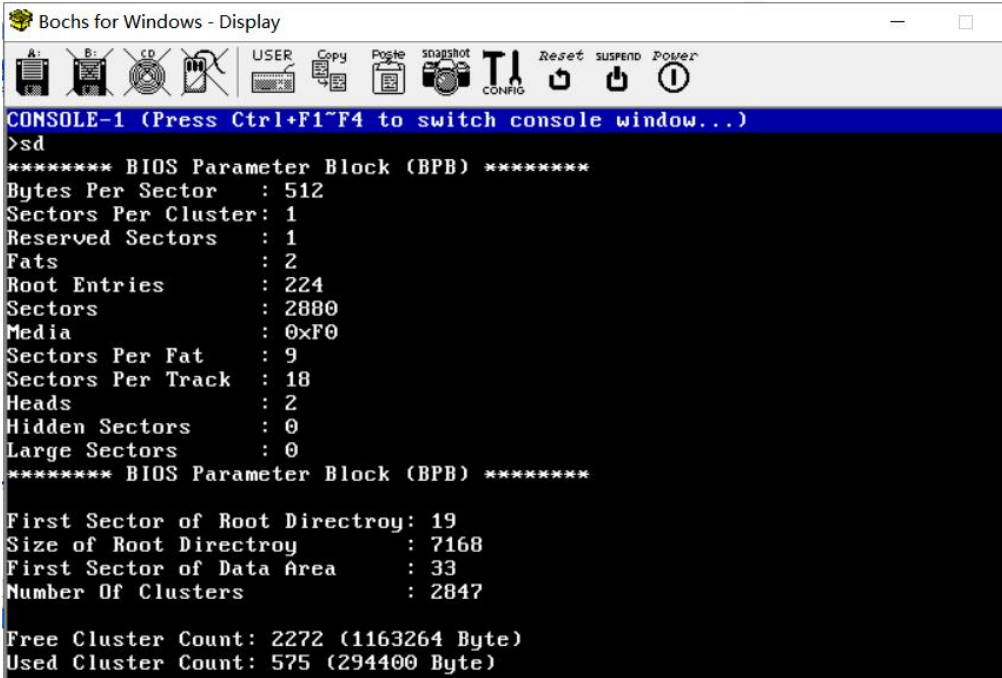
读者需要首先登录 CodeCode.net 平台领取本次实验对应的任务，从而在 CodeCode.net 平台上创建个人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将个人项目克隆到本地磁盘中。

方法二：不从 CodeCode.net 平台领取任务

如果读者不使用 CodeCode.net 平台，就需要使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将实验模板克隆到本地磁盘中，实验模板的 URL 为 <https://www.codecode.net/engintime/os-lab/Project-Template/eos-kernel.git>。

2.2 阅读控制台命令“sd”相关的源代码，并查看其执行的结果

通过执行指导书的步骤 1-3，学习到了操作系统中 FAT 的数据结构，每个表项都与数据区中的一个簇相对应，而且表项的序号也是与簇号一一对应的，在控制台执行 sd 命令以后查看到 FAT 文件系统的具体内容，如：每个分区大小为 512 字节，每个簇有对应一个分区，根目录的第一个分区标号是 19，该目录大小为 7168，数据区的第一个分区是 33 号，一共有 2847 个簇（2283 个空闲，564 个占用）等。



```
Bochs for Windows - Display
A: B: CD USER Copy Paste snapshot T1 CONFIG Reset SUSPEND Power
CONSOLE-1 (Press Ctrl+F1~F4 to switch console window...)
>sd
***** BIOS Parameter Block (BPB) *****
Bytes Per Sector : 512
Sectors Per Cluster: 1
Reserved Sectors : 1
Fats : 2
Root Entries : 224
Sectors : 2880
Media : 0xF0
Sectors Per Fat : 9
Sectors Per Track : 18
Heads : 2
Hidden Sectors : 0
Large Sectors : 0
***** BIOS Parameter Block (BPB) *****

First Sector of Root Directroy: 19
Size of Root Directroy : 7168
First Sector of Data Area : 33
Number Of Clusters : 2847

Free Cluster Count: 2272 (1163264 Byte)
Used Cluster Count: 575 (294400 Byte)
```

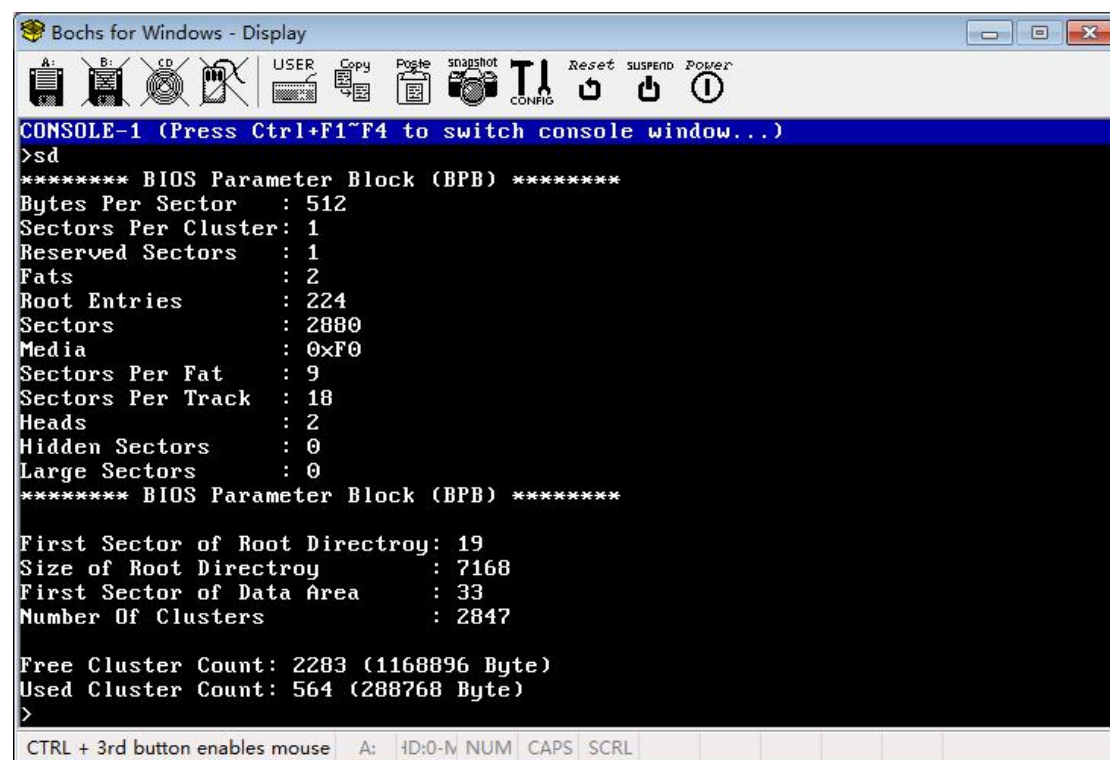
将卷控制块中缓存的 BIOS Parameter Block，以及卷控制块中的其他重要信息进行输出。

Bytes Per Sector: 区字节数
Sectors Per Cluster: 每簇的扇区数;
Reserved sectors: 保留扇区
Fats: fat 表
Root Entries: 根目录项数;
Sectors: 扇区
Media 媒体
Sectors per Fat : 每 fat 中扇区数
Sectors per track: 每磁道扇区数
Heads 磁头
Hidden Sectors 隐藏扇区
Large Sectors: 大扇区

2.3 根据 BPB 中的信息计算出其他信息

```
//printf(STDHandle, "First Sector of Root Directroy: %d\n", pVcb->FirstRootDirSector);  
//printf(STDHandle, "First Sector of Root Directroy: %d\n", pVcb->Bpb.ReservedSectors + pVcb->Bpb.Fats * pVcb->Bpb.SectorsPerFat);  
//printf(STDHandle, "Size of Root Directroy : %d\n", pVcb->RootDirSize);  
//printf(STDHandle, "Size of Root Directroy : %d\n", pVcb->Bpb.RootEntries*32);  
//printf(STDHandle, "First Sector of Data Area : %d\n", pVcb->FirstDataSector);  
//printf(STDHandle, "First Sector of Data Area : %d\n",  
(pVcb->Bpb.ReservedSectors+pVcb->Bpb.Fats*pVcb->Bpb.SectorsPerFat)+(pVcb->Bpb.RootEntries*32/pVcb->Bpb.BytesPerSector));  
//printf(STDHandle, "Number Of Clusters : %d\n\n", pVcb->NumberOfClusters);  
//printf(STDHandle, "Number Of Clusters : %d\n\n",  
pVcb->Bpb.Sectors-((pVcb->Bpb.ReservedSectors+pVcb->Bpb.Fats * pVcb->Bpb.SectorsPerFat)+(pVcb->Bpb.RootEntries*32/pVcb->Bpb.BytesPerSector)));
```

与原来直接获取信息的输出对比如下:



The screenshot shows a Bochs for Windows - Display window. The title bar is "Bochs for Windows - Display". Below the title bar is a toolbar with icons for A:, B:, CD, USER, Copy, Paste, Snapshot, CONFIG, Reset, SUSPEND, and POWER. The main window is a console window titled "CONSOLE-1 (Press Ctrl+F1~F4 to switch console window...)". The console output shows the BIOS Parameter Block (BPB) information:

```
>sd  
***** BIOS Parameter Block (BPB) *****  
Bytes Per Sector : 512  
Sectors Per Cluster: 1  
Reserved Sectors : 1  
Fats : 2  
Root Entries : 224  
Sectors : 2880  
Media : 0xF0  
Sectors Per Fat : 9  
Sectors Per Track : 18  
Heads : 2  
Hidden Sectors : 0  
Large Sectors : 0  
***** BIOS Parameter Block (BPB) *****  
  
First Sector of Root Directroy: 19  
Size of Root Directroy : 7168  
First Sector of Data Area : 33  
Number Of Clusters : 2847  
  
Free Cluster Count: 2283 (1168896 Byte)  
Used Cluster Count: 564 (288768 Byte)  
>
```

At the bottom of the window, there is a status bar with the text "CTRL + 3rd button enables mouse" and a row of buttons: A:, ID:0-N, NUM, CAPS, SCRL, and several empty buttons.

(直接输出的信息)

```

Bochs for Windows - Display
[Icons: A, B, CD, Mouse, USER, Copy, Paste, Snapshot, CONFIG, Reset, SUSPEND, Power]
CONSOLE-1 (Press Ctrl+F1~F4 to switch console window...)
>sd
***** BIOS Parameter Block (BPB) *****
Bytes Per Sector      : 512
Sectors Per Cluster  : 1
Reserved Sectors      : 1
Fats                  : 2
Root Entries          : 224
Sectors               : 2880
Media                : 0xF0
Sectors Per Fat       : 9
Sectors Per Track     : 18
Heads                 : 2
Hidden Sectors        : 0
Large Sectors         : 0
***** BIOS Parameter Block (BPB) *****

First Sector of Root Directroy: 19
Size of Root Directroy          : 7168
First Sector of Data Area       : 33
Number Of Clusters :2847

Free Cluster Count: 2283 (1168896 Byte)
Used Cluster Count: 564 (288768 Byte)
>
CTRL + 3rd button enables mouse  A:  HD:0-M NUM  CAPS  SCRL

```

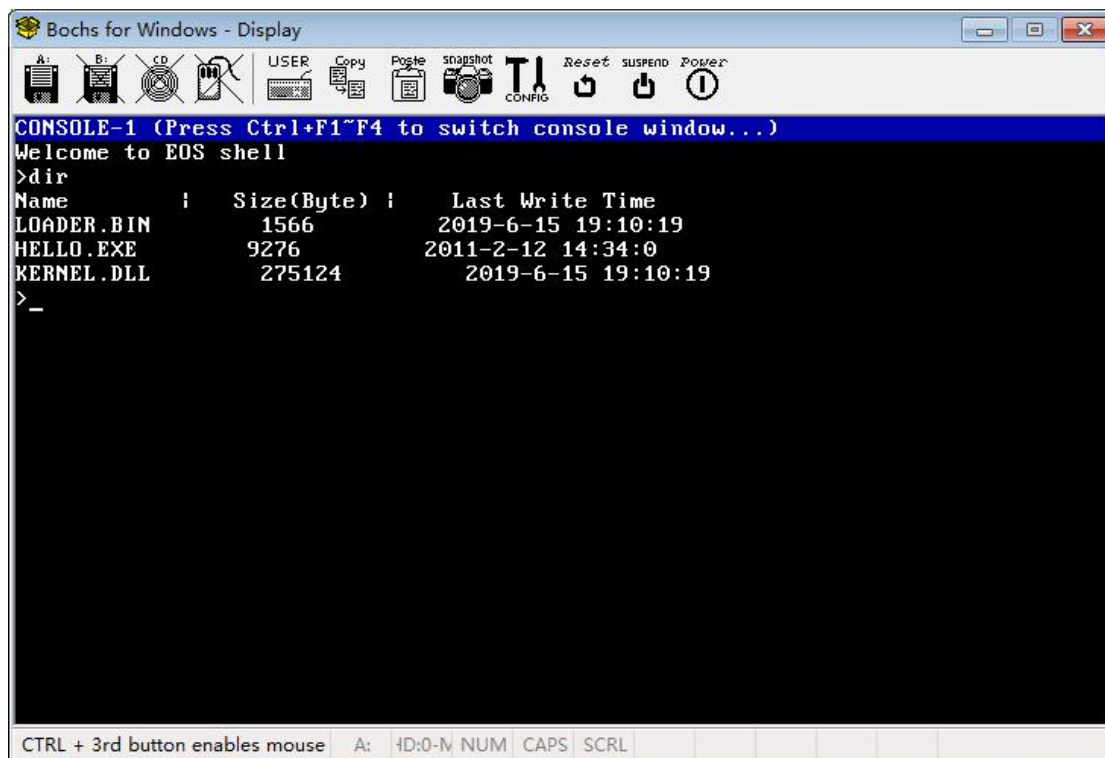
（间接计算输出的信息）

通过对比两次结果一致，从中学习到 FAT12 的主磁盘结构大致如下：



2.4 阅读控制台命令“dir”相关的源代码，并查看其执行的结果

通过执行指导书的步骤 1-3，看到了如下现象：



展示了根目录的相关信息,说明文件系统中的卷控制块 VCB 保存了目录的文件名、文件大小以及修改时间等相关信息。

2.5 输出每个文件所占用的磁盘空间的大小

通过执行指导书的步骤 3.5.1 和 3.5.2, 学习到了文件系统中文件大小和所占磁盘空间大小是不同的, 导致这种现象的原因是文件不是连续存在簇上的, 可能会分散的占用不同的簇, 所占磁盘的大小是以簇为基本单位的, 是簇大小的整倍数, 所以会 \geq 文件大小。具体操作如下:

修改的代码部分:

```
ULONG i, j, SpaceSize, RootDirSectors;
```

```
SpaceSize = 0;
```

```
for(j=pDirEntry->FirstCluster; j!=0&& j!=0xFF8;)
```

修改的代码部分

```
{
```

```
j=FatGetFatEntryValue(pVcb, j);
```

```
SpaceSize+=pVcb->Bpb.BytesPerSector*pVcb->Bpb.SectorsPerCluster;
```

```
}
```

```
fprintf(StdHandle, "%s %d %d-%d-%d %d:%d:%d %d\n",
```

```
FileName, pDirEntry->FileSize, 1980 + pDirEntry->LastWriteDate.Year,
```

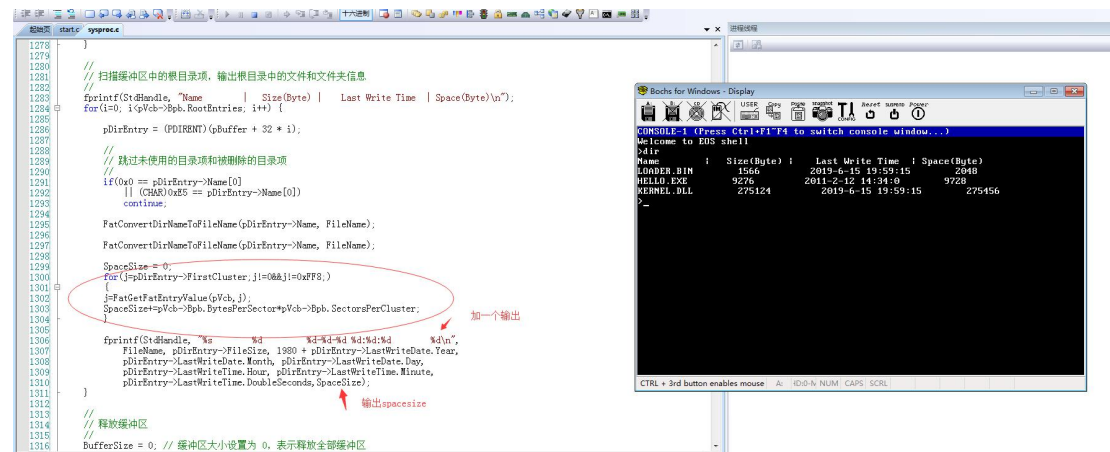
```
pDirEntry->LastWriteDate.Month, pDirEntry->LastWriteDate.Day,
```

```
pDirEntry->LastWriteTime.Hour, pDirEntry->LastWriteTime.Minute,
```

```
pDirEntry->LastWriteTime.DoubleSeconds, SpaceSize);
```

添加的输出
出列

运行结果如下：



The screenshot shows a C++ IDE with a file named 'spacproc.cpp'. The code is a function that scans a FAT file system. It iterates through root entries, skipping unused ones, and then iterates through clusters. A red circle highlights the cluster iteration loop, with a red arrow pointing to it and the text '加一个输出' (Add an output). Another red arrow points to the 'SpaceSize' variable, with the text '输出spaceSize' (Output spaceSize). The console window shows the output of the program, displaying a table of files and their sizes.

```
1278 }
1279 //
1280 // 扫描缓冲区中的根目录项，输出根目录中的文件和文件夹信息
1281 //
1282 //
1283 fprintf(STDHandle, "Name | Size(Byte) | Last Write Time | Space(Byte)\n");
1284 for (i=0; i<pVcb->Bpb.RootEntries; i++) {
1285     pDirEntry = (PDIRENT)(pBuffer + 32 * i);
1286     //
1287     // 跳过未使用的目录项和被删除的目录项
1288     //
1289     if (0x0 == pDirEntry->Name[0]
1290         || (CHAR)0xFF == pDirEntry->Name[0])
1291         continue;
1292     FatConvertDirNameToFileName(pDirEntry->Name, FileName);
1293     FatConvertDirNameToFileName(pDirEntry->Name, FileName);
1294     SpaceSize = 0;
1295     for (j=pDirEntry->FirstCluster; j!=0x0; j=j*2+1) {
1296         //FatGetFatEntryValue(pVcb, j);
1297         SpaceSize += pVcb->Bpb.BytesPerSector * pVcb->Bpb.SectorsPerCluster;
1298     }
1299     fprintf(STDHandle, "%s %d %d %d %d\n",
1300             FileName, pDirEntry->FileSize, 1980 + pDirEntry->LastWriteDate.Year,
1301             pDirEntry->LastWriteDate.Month, pDirEntry->LastWriteDate.Day,
1302             pDirEntry->LastWriteTime.Hour, pDirEntry->LastWriteTime.Minute,
1303             pDirEntry->LastWriteTime.DoubleSeconds, SpaceSize);
1304     // 输出spaceSize
1305 }
1306 // 释放缓冲区
1307 //
1308 BufferSize = 0; // 缓冲区大小设置为 0，表示释放全部缓冲区
1309 }
```

Console output:

Name	Size(Byte)	Last Write Time	Space(Byte)
LOADER.BIN	1566	2019-6-15 19:59:15	2048
HELLO.EXE	9276	2011-2-12 14:34:0	9728
KERNEL.DLL	275124	2019-6-15 19:59:15	275456

三、思考练习

（思考一）在 ConsoleCmdScanDisk 函数中扫描 FAT 表时，为什么不使用 FAT 表项的数量进行计数，而是使用簇的数量进行计数呢？而且为什么簇的数量要从 2 开始计数呢？

答：FAT 是用于将数据区的磁盘空间分配给文件，被划分成紧密排列的若干个表项，每个表项对应数据区的一个簇，表项的序号也是与簇号意义对应的；本来序号为 0 和 1 的 FAT 表项应该对应于簇 0 和 1，但由于这两个表项被设置成了固定值，簇 0 和簇 1 就没有存在的意义了，这样数据区就起始于簇 2。

时间：2020.6.26

签名：

