

合肥工业大学

操作系统实验报告

实验题目	扫描 FAT12 文件系统管理的软盘
学生姓名	付炎平
学 号	2019217819
专业班级	物联网工程19-2班
指导教师	田卫东
完成日期	2021. 11. 27

合肥工业大学 计算机与信息学院

实验 13 扫描 FAT12 文件系统管理的软盘

1. 实验目的和任务要求

- (1) 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。
- (2) 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。

2. 实验原理

FAT12 文件系统技术细节。

3. 实验内容

3.1 准备实验

按照下面的步骤准备实验：

1. 启动 OS Lab。
2. 新建一个 EOS Kernel 项目。

3.2 阅读控制台命令“sd”相关的源代码，并查看其执行的结果

阅读 ke/sysproc.c 文件中第 1320 行的 ConsoleCmdScanDisk 函数，学习“sd”命令是如何扫描软盘上的 FAT12 文件系统的。在阅读的过程中需要注意下面几点：

- (1) 在开始扫描软盘之前要关闭中断，之后要打开中断，这样可以防止在命令执行的过程中有其它线程修改软盘上的数据。
- (2) 以软盘的盘符“A:”做为 ObpLookupObjectByName 函数的参数，就可以获得 FAT12 文件系统设备对象的指针。
- (3) FAT12 文件系统设备对象的扩展块 (FatDevice->DeviceExtension) 是一个卷控制块 (VCB，在文件 io/driver/fat12.h 的第 115 行定义)，从其中可以获得文件系统的重要参数，并可以扫描 FAT 表。
- (4) FatGetFatEntryValue 函数可以根据第二个参数所指定的簇号，返回簇在 FAT 表中对应项的值，在扫描 FAT 表时通过调用此函数来统计空闲簇的数量 (FreeClusterCount)。

阅读 ConsoleCmdScanDisk 函数如下：

```

1314 //
1315 // 下面是和控制台命令 sd 相关的代码。
1316 //
1317 PRIVATE
1318 VOID
1319 ConsoleCmdScanDisk(
1320     IN HANDLE StdHandle
1321 )
1322 {
1323     /**+
1324     |
1325     | 功能描述：
1326     | 扫描软盘，并输出相关信息。控制台命令“sd”。
1327     |
1328     | 参数：
1329     | StdHandle — 标准输入、输出句柄。
1330     |
1331     | 返回值：
1332     | 无。
1333     |
1334     +**/
1335 {
1336     BOOL IntState;
1337     PDEVICE_OBJECT FatDevice;
1338     PVCB pVcb;
1339     ULONG i, FreeClusterCount, UsedClusterCount;
1340
1341     IntState = KeEnableInterrupts(FALSE); // 关中断
1342
1343     //
1344     // 得到 FAT12 文件系统设备对象，然后得到卷控制块 VCB
1345     //
1346     FatDevice = (PDEVICE_OBJECT)ObpLookupObjectByName(IopDeviceObject);
1347     pVcb = (PVCB)FatDevice->DeviceExtension;
1348
1349     //
1350     // 将卷控制块中缓存的 BIOS Parameter Block (BPB)，以及卷控制块中的
1351     // 将卷控制块中缓存的 BIOS Parameter Block (BPB)，以及卷控制块中的其它重要信息输出
1352     //
1353     fprintf(StdHandle, "***** BIOS Parameter Block (BPB) *****\n");
1354     fprintf(StdHandle, "Bytes Per Sector : %d\n", pVcb->Bpb.BytesPerSector);
1355     fprintf(StdHandle, "Sectors Per Cluster: %d\n", pVcb->Bpb.SectorsPerCluster);
1356     fprintf(StdHandle, "Reserved Sectors : %d\n", pVcb->Bpb.ReservedSectors);
1357     fprintf(StdHandle, "Fats : %d\n", pVcb->Bpb.Fats);
1358     fprintf(StdHandle, "Root Entries : %d\n", pVcb->Bpb.RootEntries);
1359     fprintf(StdHandle, "Sectors : %d\n", pVcb->Bpb.Sectors);
1360     fprintf(StdHandle, "Media : 0x%X\n", pVcb->Bpb.Media);
1361     fprintf(StdHandle, "Sectors Per Fat : %d\n", pVcb->Bpb.SectorsPerFat);
1362     fprintf(StdHandle, "Sectors Per Track : %d\n", pVcb->Bpb.SectorsPerTrack);
1363     fprintf(StdHandle, "Heads : %d\n", pVcb->Bpb.Heads);
1364     fprintf(StdHandle, "Hidden Sectors : %d\n", pVcb->Bpb.HiddenSectors);
1365     fprintf(StdHandle, "Large Sectors : %d\n", pVcb->Bpb.LargeSectors);
1366     fprintf(StdHandle, "***** BIOS Parameter Block (BPB) *****\n\n");
1367
1368     fprintf(StdHandle, "First Sector of Root Directory: %d\n", pVcb->FirstRootDirSector);
1369     fprintf(StdHandle, "Size of Root Directory : %d\n", pVcb->RootDirSize);
1370     fprintf(StdHandle, "First Sector of Data Area : %d\n", pVcb->FirstDataSector);
1371     fprintf(StdHandle, "Number Of Clusters : %d\n\n", pVcb->NumberOfClusters);
1372
1373     //
1374     // 扫描 FAT 表，统计空闲簇的数量，并计算软盘空间的使用情况
1375     //
1376     FreeClusterCount = 0;
1377     for (i = 2; i < pVcb->NumberOfClusters + 2; i++) {
1378         if (0 == FatGetFatEntryValue(pVcb, i))
1379             FreeClusterCount++;
1380     }
1381     UsedClusterCount = pVcb->NumberOfClusters - FreeClusterCount;
1382     fprintf(StdHandle, "Free Cluster Count: %d (%d Byte)\n", FreeClusterCount, FreeClusterCount);
1383     fprintf(StdHandle, "Used Cluster Count: %d (%d Byte)\n", UsedClusterCount, UsedClusterCount);
1384
1385     KeEnableInterrupts(IntState); // 开中断
1386 }
1387 #ifdef _DEBUG

```

按照下面的步骤执行控制台命令“sd”，查看扫描的结果：

1. 按 F7 生成在本实验 3.1 中创建的 EOS Kernel 项目。
 2. 按 F5 启动调试。
 3. 待 EOS 启动完毕，在 EOS 控制台中输入命令“sd”后按回车。
- 观察命令执行的结果，如图 21-1 所示，可以了解 FAT12 文件系统的信息。



```
Bochs for Windows - Display
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: 2272 (1163264 Byte)
Used Cluster Count: 575 (294400 Byte)
>
```

3.3 根据 BPB 中的信息计算出其他信息

3.3.1 要求

修改“sd”命令函数 ConsoleCmdScanDisk 的源代码，在输出 BPB 中保存的信息后，不再通过 pVcb->FirstRootDirSector 等变量的值进行打印输出，而是通过 BPB 中保存的信息重新计算出下列信息，并打印输出：

- (1) 计算并打印输出根目录的起始扇区号，即 pVcb->FirstRootDirSector 的值。
- (2) 计算并打印输出根目录的大小，即 pVcb->RootDirSize 的值。
- (3) 计算并打印输出数据区的起始扇区号，即 pVcb->FirstDataSector 的值。
- (4) 计算并打印输出数据区中簇的数量，即 pVcb->NumberOfClusters 的值。

3.3.2 测试方法

1. ConsoleCmdScanDisk 函数的源代码修改完毕后，按 F7 生成项目。
2. 按 F5 启动调试。
3. 待 EOS 启动完毕，在 EOS 控制台中输入命令“sd”后按回车。输出的内容应该仍然与图 19-1 所示的内容相同。

3.3.3 提示

在 ConsoleCmdScanDisk 函数中，使用了下面的语句打印输出根目录的起始扇区号：fprintf(StdHandle, “First Sector of Root Directroy: %d\n”, pVcb->FirstRootDirSector);

根目录的起始扇区号可以使用保留扇区的数量加上 FAT 表占用扇区的数量来计算获得，而这些信息都可以从 BPB 中获得，所以上面的语句可以修改为：fprintf(StdHandle, “First Sector of Root Directroy: %d\n”, pVcb->Bpb.ReservedSectors +pVcb->Bpb.Fats * pVcb->Bpb.SectorsPerFat);

对于根目录的大小、数据区的起始扇区号、数据区中簇的数量这些信息也可以采用类似的方式计算获得。

修改代码：

```

fprintf(StdHandle, "First Sector of Root Directroy: %d\n",
pVcb->Bpb.ReservedSectors + pVcb->Bpb.Fats * pVcb->Bpb.SectorsPerFat);

fprintf(StdHandle, "Size of Root Directroy          : %d\n",
pVcb->Bpb.RootEntries * 32);

fprintf(StdHandle, "First Sector of Data Area          : %d\n",
pVcb->Bpb.ReservedSectors + pVcb->Bpb.Fats * pVcb->Bpb.SectorsPerFat
+ pVcb->Bpb.RootEntries * 32 / pVcb->Bpb.BytesPerSector);

fprintf(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) / pVcb->Bpb.SectorsPerCluster);

```

如下图所示:

```

usedClusterCount = pVcb->NumberOfClusters - freeClusterCount;
fprintf(StdHandle, "Free Cluster Count: %d (%d Byte)\n", FreeClusterCount, FreeClusterCount*pVcb->Bpb.SectorsPerCluster*pVcb->Bpb.BytesPerSector);
fprintf(StdHandle, "Used Cluster Count: %d (%d Byte)\n", UsedClusterCount, UsedClusterCount*pVcb->Bpb.SectorsPerCluster*pVcb->Bpb.BytesPerSector);

//修改部分
fprintf(StdHandle, "First Sector of Root Directroy: %d\n", pVcb->Bpb.ReservedSectors + pVcb->Bpb.Fats * pVcb->Bpb.SectorsPerFat);
fprintf(StdHandle, "Size of Root Directroy          : %d\n", pVcb->Bpb.RootEntries * 32);
fprintf(StdHandle, "First Sector of Data Area          : %d\n", pVcb->Bpb.ReservedSectors + pVcb->Bpb.Fats * pVcb->Bpb.SectorsPerFat + pVcb->Bpb.RootEntries * 32 / pVcb->Bpb.BytesPerSector);
fprintf(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) / pVcb->Bpb.SectorsPerCluster);

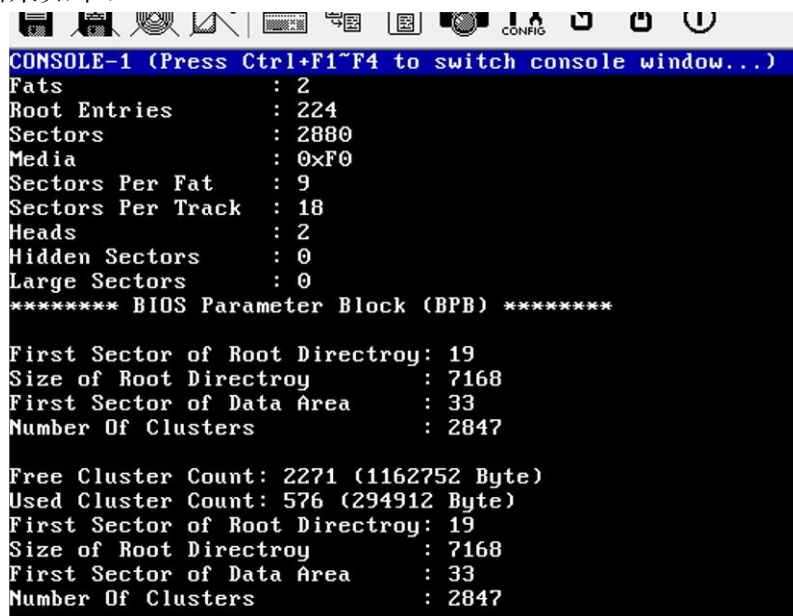
KeEnableInterrupts(IntState); // 开中断

* pVcb->Bpb.SectorsPerFat);

* pVcb->Bpb.SectorsPerFat + pVcb->Bpb.RootEntries * 32 / pVcb->Bpb.BytesPerSector);
ctors + pVcb->Bpb.Fats * pVcb->Bpb.SectorsPerFat + pVcb->Bpb.RootEntries * 32 / pVcb->Bpb.BytesPerSector) / pVcb->Bpb.SectorsPerCluster);

```

测试结果如下:



```

CONSOLE-1 (Press Ctrl+F1~F4 to switch console window...)
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: 2271 (1162752 Byte)
Used Cluster Count: 576 (294912 Byte)
First Sector of Root Directroy: 19
Size of Root Directroy          : 7168
First Sector of Data Area       : 33
Number Of Clusters              : 2847

```

3.4 阅读控制台命令“dir”相关的源代码，并查看其执行的结果

阅读 ke/sysproc.c 文件中第 1226 行的 ConsoleCmdDir 函数, 学习“dir”命令是如何扫描软盘的根目录并输出根目录中的文件信息的。在阅读的过程中需要注意下面几点:

(1) 在开始扫描根目录之前要关闭中断，之后要打开中断，这样可以防止在命令执行的过程中有其它线程修改软盘上的数据。

(2) 以软盘的盘符“A:”做为 ObpLookupObjectByName 函数的参数，就可以获得 FAT12 文件系统设备对象的指针。

(3) FAT12 文件系统设备对象的扩展块 (FatDevice->DeviceExtension) 是一个卷控制块 (VCB, 在文件 io/driver/fat12.h 的第 115 行定义)，从中可以获得文件系统的重要参数，可用于扫描根目录。

(4) 由于根目录的数据在软盘上，所以调用 MmAllocateVirtualMemory 函数分配了一块与根目录大小相同的缓冲区，然后调用 IopReadWriteSector 函数将根目录占用的扇区依次读入了缓冲区。注意在命令执行的最后需要调用 MmFreeVirtualMemory 函数释放缓冲区。

(5) 在扫描缓冲区中的目录项时，跳过了未使用的目录项和已经被删除的目录项，而只输出当前使用的目录项（文件）信息，包括文件名、文件大小和最后改写时间。

研究 ConsoleCmdDir 函数如下：

```
1224 PRIVATE
1225 VOID
1226 ConsoleCmdDir(
1227     IN HANDLE StdHandle
1228 )
1229 /*++
1230
1231     功能描述：
1232     输出软盘根目录中文件的信息。控制台命令“dir”。
1233
1234     参数：
1235     StdHandle -- 标准输入、输出句柄。
1236
1237     返回值：
1238     无。
1239
1240 --*/
1241 {
1242     BOOL IntState;
1243     PDEVICE_OBJECT FatDevice;
1244     PVCB pVcb;
1245     PVOID pBuffer;
1246     SIZE_T BufferSize;
1247     PDIRENT pDirEntry;
1248     CHAR FileName[13];
1249
1250     ULONG i, RootDirSectors;
1251
1252     IntState = KeEnableInterrupts(FALSE); // 关中断
1253
1254     //
1255     // 得到 FAT12 文件系统设备对象，然后得到卷控制块 VCB
1256     //
1257     FatDevice = (PDEVICE_OBJECT)ObpLookupObjectByName(IopDeviceObject1
```



```

258: pVcb = (PVCB)FatDevice->DeviceExtension;
259:
260: //
261: // 分配一块虚拟内存做为缓冲区，然后将整个根目录区从软盘读入缓冲区。
262: //
263: pBuffer = NULL; // 不指定缓冲区的地址。由系统决定缓冲区的地址。
264: BufferSize = pVcb->RootDirSize; // 申请的缓冲区大小与根目录区大小相同。
265: MmAllocateVirtualMemory(&pBuffer, &BufferSize, MEM_RESERVE | MEM_COMMIT, TRUE);
266:
267: RootDirSectors = pVcb->RootDirSize / pVcb->Bpb.BytesPerSector; // 计算根目录区占用的扇区数量
268: for(i=0; i<RootDirSectors; i++) {
269:     // 将根目录区占用的扇区读入缓冲区
270:     IopReadWriteSector( pVcb->DiskDevice,
271:                        pVcb->FirstRootDirSector + i,
272:                        0,
273:                        (PCHAR)pBuffer + pVcb->Bpb.BytesPerSector * i,
274:                        pVcb->Bpb.BytesPerSector,
275:                        TRUE);
276: }
277:
278: //
279: // 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
280: //
281: //
282: fprintf(STD_HANDLE, "Name | Size(Byte) | Last Write Time\n");
283: for(i=0; i<pVcb->Bpb.RootEntries; i++) {
284:     pDirEntry = (PDIRENT)(pBuffer + 32 * i);
285:
286:     //
287:     // 跳过未使用的目录项和被删除的目录项
288:     //
289:     if(0x0 == pDirEntry->Name[0]
290:        || (CHAR)0xE5 == pDirEntry->Name[0])
291:         continue;
292:
293:     FatConvertDirNameToFileName(pDirEntry->Name, FileName);
294:
295:     fprintf(STD_HANDLE, "%s %d %d-%d-%d %d:%d:%d\n",
296:            FileName, pDirEntry->FileSize, 1980 + pDirEntry->LastWriteDate.Year,
297:            pDirEntry->LastWriteDate.Month, pDirEntry->LastWriteDate.Day,
298:            pDirEntry->LastWriteTime.Hour, pDirEntry->LastWriteTime.Minute,
299:            pDirEntry->LastWriteTime.DoubleSeconds);
300: }
301:
302: //
303: // 释放缓冲区
304: //
305: BufferSize = 0; // 缓冲区大小设置为 0，表示释放全部缓冲区
306: MmFreeVirtualMemory(&pBuffer, &BufferSize, MEM_RELEASE, TRUE);
307:
308: KeEnableInterrupts(IntState); // 开中断
309:
310: }

```

测试结果如下：

Bochs for Windows - Display

CONSOLE-1 (Press Ctrl+F1~F4 to switch console window...)

Welcome to EOS shell

```

>dir
Name           | Size(Byte) | Last Write Time
LOADER.BIN     | 1566       | 2020-12-28 18:38:29
HELLO.EXE      | 9276       | 2011-2-12 14:34:0
KERNEL.DLL     | 280756     | 2020-12-28 18:38:29
>_

```

3.5 输出每个文件所占用的磁盘空间的大小

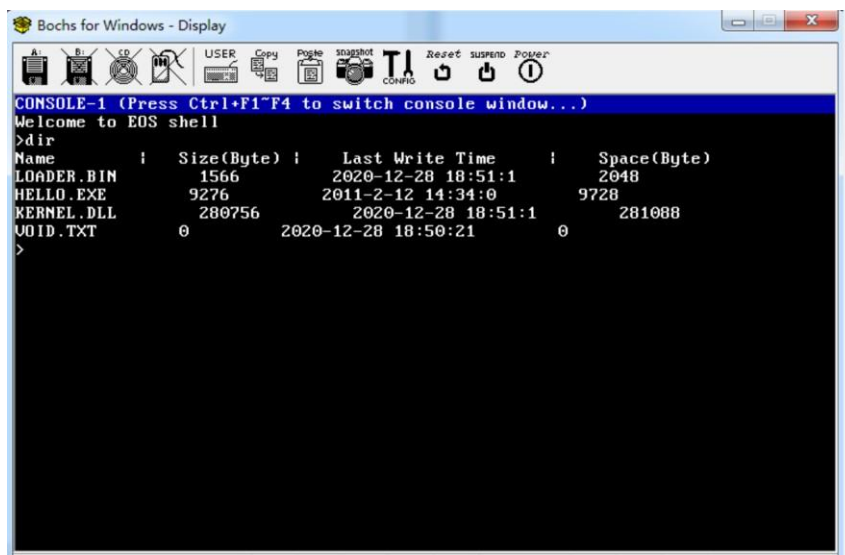
3.5.1 要求

修改“dir”命令函数 ConsoleCmdDir 的源代码，要求在输出每个文件的名称、大小、最后改写时间后，再输出每个文件所占用的磁盘空间（以字节为单位）。

3.5.2 测试方法

1. ConsoleCmdDir 函数的源代码修改完毕后，按 F7 生成项目。
2. 在“项目管理器”窗口中双击 Floppy.img 文件，使用 FloppyImageEditor 工具打开此软盘镜像。
3. 将“学生包”本实验文件夹中的 void.txt 文件（大小为 0）添加到软盘镜像的根目录中（将 void.txt 文件拖动到 FloppyImageEditor 窗口中释放即可）。
4. 点击 FloppyImageEditor 工具栏上的保存按钮，关闭该工具。
5. 按 F5 启动调试。
6. 待 EOS 启动完毕，在 EOS 控制台中输入命令“dir”后按回车。

输出的内容应该与图 21-3 所示的内容相同，或者可以在“项目管理器”窗口中双击 Floppy.img 文件，使用 FloppyImageEditor 查看文件的相关信息，检验输出的结果是否正确。



3.5.3 提示

文件的大小与文件所占用的磁盘空间是两个不同的概念，文件所占用的磁盘空间是簇的整数倍，所以文件所占用的磁盘空间总是大于或等于文件的大小。例如，如果一个簇只包含一个扇区，大小是 1 字节的文件，其占用的磁盘空间至少是一个簇的大小即 512 字节，大小是 600 字节的文件，其占用的磁盘空间至少是两个簇的大小即 1024 字节。但是，不能简单的认为大小是 1 字节的文件就一定只占用一个簇，该文件完全可以占用多个簇。所以，统计文件所占用的磁盘空间的方法应该是，根据文件在 FAT 表中的簇链来进行统计，而不能简单的将文件的大小对齐到簇的大小。其他需要提示的内容有：

- (1) 目录项结构体的定义可以参见文件 io/driver/fat12.h 的第 102 行，

其中的 FirstCluster 域记录了文件的起始簇号。

```
102 typedef struct _DIRENT {
103     CHAR Name[11];           // 文件名 8 字节, 扩展名 3 字节
104     UCHAR Attributes;       // 文件属性
105     UCHAR Reserved[10];     // 保留未用
106     FAT_TIME LastWriteTime;  // 文件最后修改时间
107     FAT_DATE LastWriteDate;  // 文件最后修改日期
108     USHORT FirstCluster;    // 文件的第一个簇号
109     ULONG FileSize;         // 文件大小
110 } DIRENT, *PDIRENT;
```

(2) 获得文件的起始簇号后, 可以循环调用 FatGetFatEntryValue 函数 (在文件 io/driver/fat12.c 的第 307 行定义), 遍历文件占用的簇所组成的簇链, 当 FAT 表项的值为 0xFF8 时, 表示簇链结束。

(3) 对于大小为 0 的文件, 并不代表文件不会占用簇, 只有当文件的起始簇号为 0 时, 才代表文件没有占用簇。

```
USHORT
FatGetFatEntryValue(
    IN PVCB Vcb,
    IN USHORT Index
)
/*++
    功能描述:
        读FAT中指定项的值 (仅读取加载到内存中的FAT缓冲区)。

    参数:
        Vcb -- 卷控制块指针。
        Index -- 指定项的索引。

    返回值:
        FAT项的值。
--*/
{
    USHORT Value;

    ASSERT(2 <= Index && Index < 0xFF0);

    //
    // 将包含FAT项的连续两个字节读入一个16位整形变量。
    //
    CopyUchar2(&Value, (PCHAR)Vcb->Fat + Index * 3 / 2)

    //
    // 根据索引的奇/偶取整形变量的高/低12位的值。
    //
    return (Index & 0x1) != 0 ? Value >> 4 : Value & 0xFFFF;
}
```

4. 实验的思考与问题分析

1. 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: 因为一个 FAT 占一个扇区, 该表的所有指针数加起来有可能大于总簇的数量, 所以在统计时不用 FAT 的表项的数量, 而是使用簇的数量。簇的数量从 2

开始计数是由于前面两个簇是固定的簇，表示固定的大小，不可使用。

2. 在 ConsoleCmdScanDisk 函数中扫描 FAT 表时，统计了空闲簇的数量，然后使用簇的总数减去空闲簇的数量做为占用簇的数量，这种做法正确吗？是否还有其他类型的簇没有考虑到呢？

答：不准确，可能还有坏簇，保留簇等簇的类型存在，没有被考虑到。

5. 总结和感想体会

(1) 通过使用 dir, sd 命令，我对于文件目录的查询方式、FAT12 文件系统在软盘上的扫描方式都有了深刻的体会。在扫描前需要关中断，扫描完成后再进行开中断操作。

(2) 在修改代码的过程中，我对于 BPB 的了解更加深刻，并且学会了对于磁盘数据的计算。

(3) 在对于 dir 命令的使用过程中，我练习了按照簇进行大小计算的方法，对于文件夹大小有了进一步的了解。

(4) 在调试 FAT12 文件系统的扫描功能的过程中，我对于 FAT12 文件系统管理软盘的方式了解更加深刻，也得到了实际操作的练习，明白了 FAT12 文件系统是如何具体实现的。