第5章 Windows的设备管理

5.1 实验一：获取磁盘基本信息

5.1.1 实验目的

(1) 了解磁盘的物理组织。

(2) 熟悉 Windows 系统如何查看磁盘相关参数。

(3) 掌握 Windows 系统提供的有关对磁盘操作 API 函数。

5.1.2 实验准备知识：相关数据结构及 API 函数介绍

1. 相关系统数据结构说明

磁盘基本物理结构原型：

```
typedef struct_DISK_GEOMETRY {
                              LARGE_INTEGER  Cylinders;
                              MEDIA_TYPE  MediaType;
                              DWORD  TracksPerCylinder;
                              DWORD  SectorsPerTrack;
                              DWORD  BytesPerSector;
                             } DISK_GEOMETRY;
```

成员说明：

(1) Cylinders：磁盘的柱面数。

(2) MediaType：介质类型，如 3.5 英寸(1 英寸＝2.54 厘米)、1.44MB 软盘。

(3) TracksPerCylinder：每个柱面的磁道数。

(4) SectorsPerTrack：每个磁道的扇区数。

(5) BytesPerSector：每个扇区的字节数。

2. 相关 API 函数介绍

(1) 文件创建。函数 CreateFile()用于打开磁盘驱动器并返回一个文件句柄，这里驱动器被当作文件来处理。有关文件操作函数的详细说明参见 4.1.2 节。

原型：

```
HANDLE CreateFile(
                  LPCTSTR lpFileName,                    //指向文件名的指针
                  DWORD dwDesiredAccess,                 //读/写访问模式
                  DWORD dwShareMode,                     //共享模式
```

```
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,  //指向安全属性的指针
    DWORD dwCreationDisposition,                 //文件存在标志
    DWORD dwFlagsAndAttributes,                  //文件属性
    HANDLE hTemplateFile                         //指向访问模板文件的句柄
);
```

(2) 获取磁盘的基本信息。函数 DeviceIoControl()用于获取磁盘的基本信息。

原型:

```
BOOL DeviceIoControl(
    HANDLE hDevice,                              //设备句柄
    DWORD dwIoControlCode,                       //操作控制代码
    LPVOID lpInBuffer,                           //输入数据缓冲区
    DWORD nInBufferSize,                         //输入数据缓冲区大小
    LPVOID lpOutBuffer,                          //输出数据缓冲区
    DWORD nOutBufferSize,                        //输出数据缓冲区大小
    LPDWORD lpBytesReturned,                     //可获取的字节计数
    LPOVERLAPPED lpOverlapped                    //指向 OVERLAPPED 结构的指针
);
```

参数说明如下。

① hDevice:目标设备的句柄,由 CreateFile()函数获得。

② dwIoControlCode:指定操作的控制信息,用该值可以辨别将要执行的操作,以及对哪类设备进行操作。该参数取值如表 5-1 所示。

表 5-1 dwIoControlCode 的值

值	描 述
IOCTL_DISK_GET_DRIVE_GEOMETRY	得到磁盘物理结构信息
IOCTL_DISK_GET_PARTITION_INFO	得到磁盘分区信息
FSCTL_QUERY_FAT_BPB	返回 FAT16 或 FAT12 卷的前 36 字节
FSCTL_GET_COMPRESSION	获取文件或目录的压缩信息

③ lpInBuffer:指向一个缓冲区,该缓冲区存放指定操作所输入的数据。

④ nInBufferSize:由 lpInBuffer 所指缓冲区的大小。

⑤ lpOutBuffer:指向一个缓冲区,该缓冲区存放指定操作所输出的数据。

⑥ nOutBufferSize:由 lpOutBuffer 所指缓冲区的大小。

⑦ lpBytesReturned:实际输出结果所占字节数。

⑧ lpOverlapped:指向 OVERLAPPED 结构的指针。

返回值:

如果函数调用成功,则返回值为非 0 值。如果函数调用失败,则返回值为 0。若要得到更多的错误信息,可调用函数 GetLastError()。

5.1.3 实验内容

编写一个函数,根据给出的驱动器号读取磁盘基本信息,包括磁盘的大小、该磁盘包括

多少个扇区、该磁盘有多少个柱面，以及每个柱面的磁道数、每个磁道的扇区数、每个扇区包含的字节数。

5.1.4 实验要求

了解 MSDN Library Visual Studio 6.0 中提供的磁盘主要数据结构 DISK_GEOMETRY 中每个成员的含义，深入理解操作系统将设备当作文件处理的特性，理解函数 CreateFile() 及 DeviceloControl()中每个参数的实际意义并能在本实验中正确使用。

5.1.5 实验指导

在 Microsoft Visual C++ 6.0 环境下选择 Win32 Console Application 选项建立一个控制台工程文件，由于有关设备及文件操作的函数均是 Microsoft Windows 操作系统的系统调用，因此在图 5-1 中选中 An application that supports MFC 单选按钮。

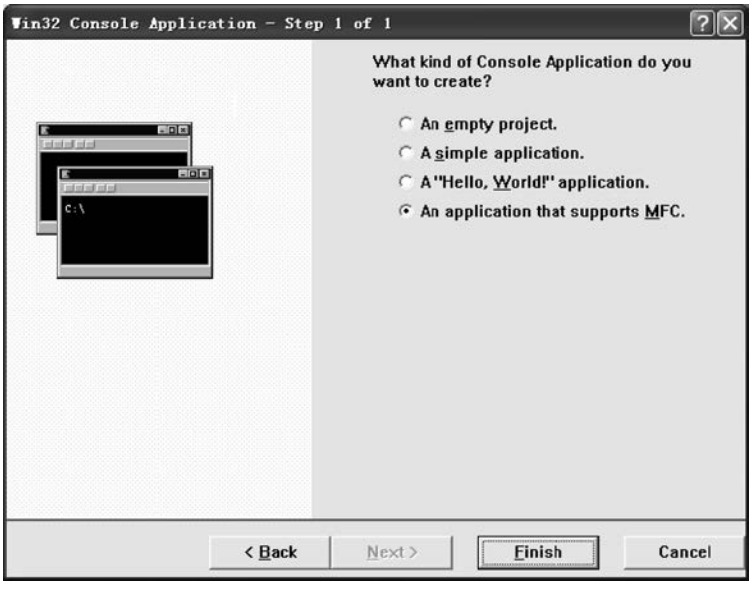

图 5-1 建立一个 MFC 支持的应用程序

本实验使用的主要数据结构 DISK_GEOMETRY 是由系统提供的，其声明在“#include "winioctl.h""中，因此要将其加入到实验程序的头文件说明中，否则程序编译时系统将无法识别 DISK_GEOMETRY 结构。

5.1.6 实验总结

本实验程序的运行结果如图 5-2 所示。

从实验结果可以看出，对给定的磁盘驱动器中的软盘 A，本实验能正确识别出它每个扇区有 512 字节，每个磁道有 18 个扇区，每个柱面有 2 个磁道，共有 80 个柱面，该磁盘共有 2880 个磁道，磁盘的大小为 1.41MB。应当注意，磁盘上有一部分空间是存储磁盘的物理信息的，这部分空间系统是不能够直接存取的，因此没有编入逻辑扇区，也就是说逻辑扇区比磁盘的实际扇区数要小，因此计算出的磁盘大小是磁盘可用空间的大小，比磁盘的物理大小要小。

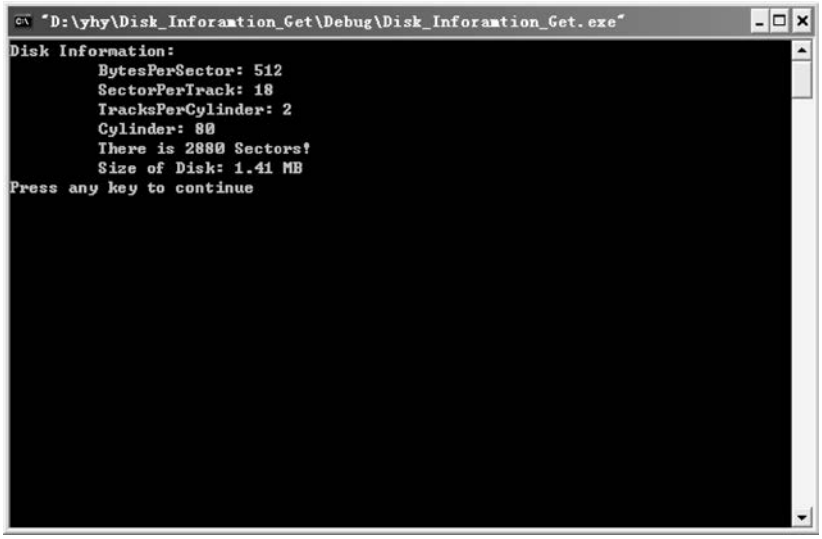

图 5-2　软盘的基本物理组成信息

5.1.7　源程序

```
//Disk_Inforamtion_Get.cpp: Defines the entry point for the console application

#include "stdafx.h"
#include "Disk_Inforamtion_Get.h"
#include "winioctl.h"

#ifdef_DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

DISK_GEOMETRY disk_info;

HANDLE GetDiskInformation(char drivername);

////////////////////////////////////////////////////////////////////////////
//The one and only application object
CWinApp theApp;
using namespace std;

int_tmain(int argc, TCHAR * argv[], TCHAR * envp[])
{
    int nRetCode = 0;
    HANDLE Handle;
    Handle = GetDiskInformation('A');
    return nRetCode;
}
```

```
HANDLE GetDiskInformation(char drivername)
{
    char device[ ] = "\\\\.\\: ";
    device[4] = drivername;
    HANDLE FloopyDisk;
    DWORD ReturnSize;
    DWORD Sector;
    double DiskSize;
    FloopyDisk = CreateFile(device,
                            GENERIC_READ|GENERIC_WRITE,
                            FILE_SHARE_READ|FILE_SHARE_WRITE,
                            NULL,
                            OPEN_EXISTING,
                            FILE_FLAG_RANDOM_ACCESS|FILE_FLAG_NO_BUFFERING,
                            NULL);
    if (FloopyDisk == INVALID_HANDLE_VALUE)
        printf("INVALID_HANDLE_VALUE!\n");
    if (GetLastError() == ERROR_ALREADY_EXISTS)
        printf("Can not Open Disk! %d\n",GetLastError());
    if (!DeviceIoControl(FloopyDisk,
        IOCTL_DISK_GET_DRIVE_GEOMETRY,
        NULL,
        0,
        &disk_info,
        50,
        &ReturnSize,
        (LPOVERLAPPED)NULL))
        printf("Open Disk Error! %d\n",GetLastError());
    printf("Disk Information: \n");
    printf("\t BytesPerSector: %d\n",disk_info.BytesPerSector);
    printf("\t SectorPerTrack: %d\n",disk_info.SectorsPerTrack);
    printf("\t TracksPerCylinder: %d\n",disk_info.TracksPerCylinder);
    printf("\t Cylinder: %d\n",disk_info.Cylinders);
    Sector = disk_info.Cylinders.QuadPart *
        disk_info.TracksPerCylinder *
        disk_info.SectorsPerTrack;
    printf("\t There is %d Sectors!\n",Sector);
    DiskSize = Sector * disk_info.BytesPerSector;
    printf("\t Size of Disk: %4.2f MB\n",(DiskSize)/(1024 * 1024));
    return FloopyDisk;
}
```

5.2 实验二：读/写磁盘指定位置信息

5.2.1 实验目的

(1) 了解磁盘的物理组织。

(2) 掌握 Windows 系统提供的有关对磁盘操作 API 函数。

(3) 根据输入的扇区号读/写指定扇区。

5.2.2 实验准备知识：相关 API 函数介绍

1. 设置读/写操作的位置

函数 SetFilePointer()用于移动一个打开文件中的读/写指针，这里磁盘设备被当作文件处理，因此用于移动文件读/写指针在磁盘上的位置。

原型：

```
DWORD SetFilePointer(
                    HANDLE hFile,                //文件句柄
                    LONG lpDistanceToMove,       //文件指针要移动的偏移量的低 32 位
                    PLONG lpDistanceToMoveHigh,  //文件指针要移动的偏移量的高 32 位
                    DWORD dwMoveMethod           //移动起点
                   );
```

参数说明如下。

(1) hFile：打开的文件句柄，创建的文件必须具有 GENERIC_ READ 或 GENERIC_ WRITE 的存取权限。

(2) lpDistanceToMove：指针要移动的偏移量的低 32 位，用于指定移动文件指针的字节大小。如果参数 lpDistanceToMoveHigh 不为空，那么 lpDistanceToMoveHigh 和 lpDistanceToMove 两个参数形成一个 64 位的值来指定移动的位置。如果参数 lpDistanceToMoveHigh 为空，且 lpDistanceToMove 是一个 32 位带符号值，那么当 lpDistanceToMove 为正值时，文件指针向前移动，否则向后移动。

(3) lpDistanceToMoveHigh：指针要移动的偏移量的高 32 位。如果不需要该参数，可将其设置为空。当该参数不为空时，该参数为文件指针的高 32 位的 DWORD 类型值。

(4) dwMoveMethod：文件指针移动的初始位置，其值如表 5-2 所示。

表 5-2 dwMoveMethod 的值

值	描　述
FILE_BEGIN	开始点为 0 或为文件的开始位置
FILE_CURRENT	开始点为文件指针的当前位置
FILE_END	开始点为文件的结尾位置

返回值：

如果函数调用成功，而且参数 lpDistanceToMoveHigh 为空，那么返回值为文件指针的低 32 位 DWORD 类型值。如果参数 lpDistanceToMoveHigh 不为空，那么返回值为文件指针的低 32 位 DWORD 类型值，并且高 32 位 DWORD 类型值输出到一个 long 类型的参数中。

如果函数调用失败，而且参数 lpDistanceToMoveHigh 为空，那么返回值为 0xFFFFFFFF，若要得到错误信息，请调用函数 GetLastError()。如果函数调用失败，而且参数 lpDistanceToMoveHigh 不为空，那么返回值为 0xFFFFFFFF，但由于 0xFFFFFFFF 不是一个有效的低 32 位 DWORD 类型值，必须通过调用函数 GetLastError()才能判断是

否有错误发生。若发生错误，则函数 GetLastError()返回错误值，否则返回 NO_ERROR。

如果新的文件指针位置是一个负值，则表明函数调用失败，文件指针将不移动，通过调用函数 GetLastError()返回的值是 ERROR_NEGATIVE_SEEK。

2. 读文件

读取磁盘指定区域的内容，函数详细说明参见 4.1.2 节。

原型：

```
BOOL ReadFile(
              HANDLE hFile,                          //要读的文件的句柄
              LPVOID lpBuffer,                       //指向文件缓冲区的指针
              DWORD nNumberOfBytesToRead,            //从文件中要读取的字节数
              LPDWORD lpNumberOfBytesRead,           //指向从文件中要读取的字节数的指针
              LPOVERLAPPED lpOverlapped              //指向 OVERLAPPED 结构的指针
             );
```

3. 写文件

该函数将数据写入磁盘指定区域，函数详细说明参见 4.1.2 节。

原型：

```
BOOL WriteFile(
               HANDLE hFile,                         //要读的文件的句柄
               LPVOID lpBuffer,                      //指向文件缓冲区的指针
               DWORD nNumberOfBytesToWrite,          //从文件中要读取的字节数
               LPDWORD lpNumberOfBytesWritten,       //指向从文件中要读取的字节数的指针
               LPOVERLAPPED lpOverlapped             //指向 OVERLAPPED 结构的指针
              );
```

5.2.3 实验内容

在本章实验一的基础上，继续完成该实验。编写两个函数，分别完成如下功能。

(1) 对给定的扇区号读取该扇区的内容。

(2) 将用户输入的数据写入指定的扇区。

5.2.4 实验要求

深入理解操作系统将设备当作文件处理的特性，理解函数 SetFilePointer()、ReadFile()及 WriteFile()中每个参数的实际意义并能在本实验中正确使用。

5.2.5 实验指导

在主程序中让用户选择：R、W 或 Q，若用户选择 R 选项，则调用函数 BOOL SectorRead(HANDLE Handle)，完成读给定扇区信息的功能；若用户选择 W 选项，则调用函数 BOOL SectorWrite(HANDLE Handle)完成对给定扇区号写入信息的功能；若用户选择 Q 选项，则程序退出。

5.2.6 实验总结

用户读/写扇区的情况如图 5-3 所示。

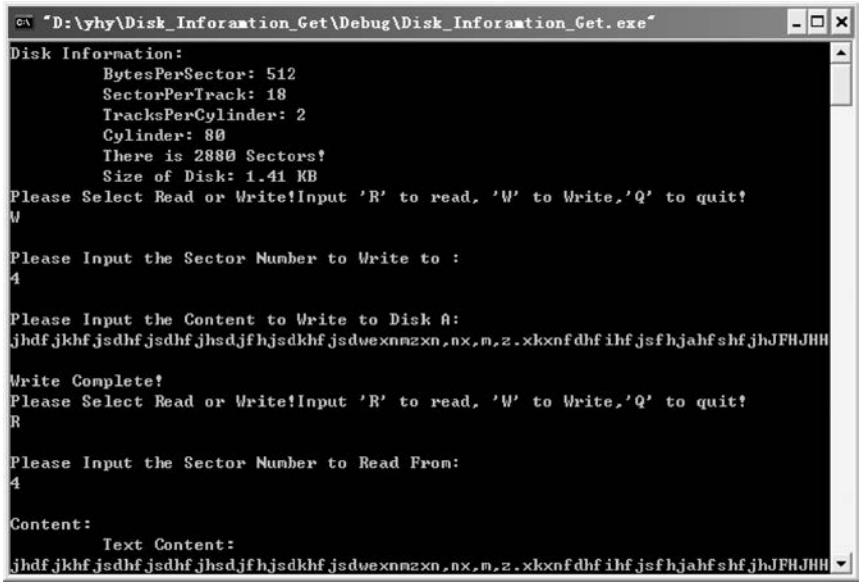

图 5-3　用户读/写扇区的情况

在上面的实验中,应用程序首先显示软盘的信息。

```
Disk Information:
    BytesPerSector: 512
    SectorPerTrack: 18
    TracksPerCylinder: 2
    Cylinder: 80
    There is 2880 Sectors!
    Size of Disk: 1.41 KB
```

然后提示用户进行选择"Please Select Read or Write! Input 'R' to read, 'W' to Write,'Q' to quit!"当用户输入 W 表示要写软盘后,应用程序提示用户"Please Input the Sector Number to Write to:"输入要写的磁道号,当用户输入 4 表示要写第 4 道后,应用程序提示用户"Please Input the Content to Write to Disk A::"输入要写入第 4 道的内容,当用户输入要写的内容后,应用程序提示"Write Complete!"表示写操作完成。

接着,应用程序继续提示用户进行选择"Please Select Read or Write! Input 'R' to read, 'W' to Write,'Q' to quit!"当用户输入 R 表示要读软盘后,应用程序提示用户"Please Input the Sector Number to Read From:"输入要读的磁道号,当用户输入 4 表示要读第 4 道的内容后,应用程序显示"Content:"并分别以字符形式和十六进制形式显示软盘上第 4 道的内容。

5.2.7　源程序

```
//Disk_Inforamtion_Read and Write.cpp: Defines the entry point for the console application

#include "stdafx.h"
#include "Disk_Inforamtion_Get.h"
```

```
#include "winioctl.h"

#ifdef_DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

DISK_GEOMETRY disk_info;

HANDLE GetDiskInformation(char drivername);
BOOL SectorRead(HANDLE Handle);
BOOL SectorWrite(HANDLE Handle);

/////////////////////////////////////////////////////////////////////////////
//The one and only application object
CWinApp theApp;
using namespace std;

int_tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
    int nRetCode = 0;
    HANDLE Handle;
    char Choice;
    Handle = GetDiskInformation('A');

    while(TRUE)
    {
        printf("Please Select Read or Write! Input 'R' to read, 'W' to Write,'Q' to quit!\n");
        Choice = getchar();
        printf("\n");
        switch (Choice)
        {
            case 'W':
                {
                    if (!SectorWrite(Handle))  printf("Write Sector Fail!\n");
                    getchar();
                    break;
                }
            case 'R':
                {
                    if (!SectorRead(Handle))  printf("Read Sector Fail!\n");
                    getchar();
                    break;
                }
            case 'Q':
                {
                    exit(0);
                    break;
                }
            default:
```

```
                    {
                        printf("Input Error!,Try again please!\n");
                        getchar();
                    }
            }

        }
        return nRetCode;
    }

    HANDLE GetDiskInformation(char drivername)
    {
        char device[] = "\\\\.\\: ";
        device[4] = drivername;
        HANDLE FloopyDisk;
        DWORD ReturnSize;
        DWORD Sector;
        double DiskSize;
        FloopyDisk = CreateFile(device,
                                GENERIC_READ|GENERIC_WRITE,
                                FILE_SHARE_READ|FILE_SHARE_WRITE,
                                NULL,
                                OPEN_EXISTING,
                                FILE_FLAG_RANDOM_ACCESS|FILE_FLAG_NO_BUFFERING,
                                NULL);
        if (FloopyDisk == INVALID_HANDLE_VALUE)
            printf("INVALID_HANDLE_VALUE!\n");
        if (GetLastError() == ERROR_ALREADY_EXISTS)
            printf("Can not Open Disk! %d\n",GetLastError());
        if (!DeviceIoControl(FloopyDisk,
                             IOCTL_DISK_GET_DRIVE_GEOMETRY,
                             NULL,
                             0,
                             &disk_info,
                             50,
                             &ReturnSize,
                             (LPOVERLAPPED)NULL))
        printf("Open Disk Error! %d\n",GetLastError());
        printf("Disk Information: \n");
        printf("\t BytesPerSector: %d\n",disk_info.BytesPerSector);
        printf("\t SectorPerTrack: %d\n",disk_info.SectorsPerTrack);
        printf("\t TracksPerCylinder: %d\n",disk_info.TracksPerCylinder);
        printf("\t Cylinder: %d\n",disk_info.Cylinders);
        Sector = disk_info.Cylinders.QuadPart *
            disk_info.TracksPerCylinder *
            disk_info.SectorsPerTrack;
        printf("\t There is %d Sectors!\n",Sector);
        DiskSize = Sector * disk_info.BytesPerSector;
        printf("\t Size of Disk: %4.2f KB\n",(DiskSize)/(1024 * 1024));
        return FloopyDisk;
    }
```

```
BOOL SectorRead(HANDLE Handle)
{
    char ReadBuffer[1024 * 16];
    DWORD SectorNumber;
    DWORD BytestoRead;
    DWORD Sector;
    DWORD rc;
    int i;
    if (Handle == NULL)
    {
        printf("There is No disk!\n");
        return FALSE;
    }
    printf("Please Input the Sector Number to Read From: \n");
    scanf(" % d",&SectorNumber);
    printf("\n");
    Sector = disk_info.Cylinders.QuadPart *
        disk_info.TracksPerCylinder *
        disk_info.SectorsPerTrack;
    if (SectorNumber > Sector) printf("There is not this Sector!\n");
    printf("Content: \n");
    BytestoRead = SectorNumber * (disk_info.BytesPerSector);
    rc = SetFilePointer(Handle,BytestoRead,NULL,FILE_BEGIN);
    if (!ReadFile(Handle,ReadBuffer,BytestoRead,&BytestoRead,NULL))
    {
        printf("Read File Error:  % d\n", GetLastError());
        return FALSE;
    }
    printf("\t Text Content: \n");
    for (i = 0; i < 512; i ++ )
    {
        printf(" % c",ReadBuffer[i]);
    }
    printf("\n");
    printf("\t Hex Text Content: \n");
    for (i = 0; i < 512; i ++ )
    {
        printf(" % x",ReadBuffer[i]);
        printf("");
    }
    printf("\n");
    return TRUE;
}

BOOL SectorWrite(HANDLE Handle)
{
    char WriteBuffer[1024];
    DWORD SectorNumber,SecterMove;
    DWORD BytestoWrite;
    DWORD Sector;
```

```
    DWORD rc;

    if (Handle == NULL)
    {
        printf("There is No disk!\n");
        return FALSE;
    }
    printf("Please Input the Sector Number to Write to: \n");
    scanf("%d",&SectorNumber);
    printf("\n");
    Sector = disk_info.Cylinders.QuadPart *
        disk_info.TracksPerCylinder *
        disk_info.SectorsPerTrack;
    if (SectorNumber > Sector) printf("There is not this Sector!\n");
    printf("Please Input the Content to Write to Disk A: \n");
    scanf("%s",&WriteBuffer);
    SecterMove = SectorNumber * (disk_info.BytesPerSector);
    rc = SetFilePointer(Handle,SecterMove,NULL,FILE_BEGIN);
    if (!WriteFile(Handle,WriteBuffer,512,&BytestoWrite,NULL))
    {
        printf("Read File Error: %d\n", GetLastError());
        return FALSE;
    }
    printf("Write Complete!\n");
    return TRUE;
}
```

5.2.8 实验展望

在上述实验的基础上,读者可以尝试实现下述功能:读取软盘上的文件目录,并查看指定文件信息。