操作系统

实验报告

实验名称：设备管理--磁盘I/O

院系：计算机学院·网络空间安全学院

班级：

学号：

姓名：

指导老师：匡林爱

日期：2020年10月11日

1. **实验目的和要求：**

通过本上机实验了解Windows系统中如何直接使用磁盘的读写功能；所编应用程序能够响应用户指定的读写磁盘扇区的请求，也能提供查看磁盘相关参数的功能。

程序使用了Windows 操作系统提供的API来实现所要求的功能。

通过本实验，用户可以利用Win32 API进行底层的磁盘操作，了解Windows 操作系统设备管理的一些知识。

注意！！**由于本实验是对磁盘进行底层I/O，读写操作并不经由操作系统的文件系统控制的，即写盘操作时并不考虑目标扇区是否已经被某个文件使用，所以写操作会破坏磁盘中的数据，甚至可能使该磁盘无法使用。因此强力建议做本实验时使用一张格式化的空白U盘或软盘（若不是空白盘但盘中数据不要了，则也可以）来进行写操作测试。对于C盘等，程序中严禁对它们进行写操作，但可以对它们执行读数据(包括读磁盘基本信息)的操作。读盘操作是安全的。**

1. **实验内容：**

除主函数外，程序主要有如下几个函数组成：

* interwindow( ) 完成功能选择(文本窗口下的菜单)。
* opendisk( ) 打开磁盘并记录磁盘参数。
* physicalDisk( ) 显示磁盘想过数据。
* sectorDump( ) 读取指定的扇区域的内容。
* sectorWrite( ) 将键入的数据写到指定扇区中去。
* sectorRead( ) 进行实际的读盘操作。
* changeDrive( ) 改变操作的驱动器。
* partitionInfo( ) 显示磁盘分区信息
* scandrivetype( ) 扫描驱动器，寻找可移动磁盘

程序总体流程如图8-1所示。

### 数据结构和程序中用到的API函数介绍

**1．数据结构**

程序中定义了一个disk结构：

typedef struct disk {

HANDLE hDisk; //磁盘句柄

CHAR driveletter; //驱动器名(盘符)

UINT driveType; //驱动器类型

BOOL writeFlag; //可写标志

DISK\_GEOMETRY theSupportedGeometry;

} \*Disk ; //Disk是disk结构的指针

**结构的主要成员的说明：**

hDisk：当前打开的磁盘的句柄。调用CreateFile函数获得句柄，此后使用句柄来操作磁盘。

theSupportedGeometry：是DISK\_GEOMETRY结构型成员。DISK\_GEOMETRY是系统定义的用于记录磁盘参数的结构，该结构描述磁盘的物理(几何尺寸)参数，在对磁盘操作时需要这些参数，该结构的具体内容可参看MSDN或“OS实验涉及的API函数中使用的几个数据结构介绍.doc”。

（完成相应操作后返回）

调用功能选择函数interwindows( )

若目前尚未打开磁盘，则调用opendisk( )函数，

并从函数返回值获得该磁盘的句柄

choice=?

调用 physicalDisk( )

显示磁盘相关信息

调用 sectorDump ( )

读取指定扇区内容

调用 sectorWrite ( )将键入数据写到指定扇区中

调用 changeDrive ( )

改变操作的驱动器

程序结束

=1

=2

=3

=4

=5

调用sectorRead( )

进行实际的读盘操作

**图8-1 程序的总体框架图**

调用 partitionInfo( )

显示磁盘分区信息

=6

调用scandrivetype( )寻找可移动磁盘，并设为初始磁盘。

**2．程序中用到的API函数介绍**

**(1) CreateFile( )函数**

**功能**：用来创建或打开一个文件、管道、磁盘设备等对象，它返回一个句柄用于以后对这个对象的访问。该函数正确执行返回文件句柄，执行失败，则返回INVALID\_HANDLE\_VALUE(即-1)。

**格式**：

HANDLE CreateFile (

LPCTSTR lpszName ,

DWORD fdwAccess ,

DWORD fdwShareMode ,

LPSECURITY\_ATTRIBUTES lpsa ,

DWORD fdwCreate ,

DWORD fdwAttrsAndFlags ,

HANDLE hTemplateFile

)

**参数说明：**

**lpszName**：指向文件名的指针(可含空格，支持默认路径)；当需要打开硬盘分区或U盘、软盘时，文件名格式为“\\.x”，其中x指磁盘名，例如C盘为“\\.C”。本程序中出示设定为I:盘(U盘)。

**fdwAccess**：创建或打开的文件的读写属性。取值可为：GENERIC\_READ(值0x80000000)即可读，GENERIC\_WRITE(值0x40000000)指可写，它们可通过“或”运算联接；取值0是查询模式，用于查询设备属性而不访问设备。

**fdwShareMode**：共享模式，值可取FILE\_SHARE\_READ(即值1)或FILE\_SHARE\_WRITE(值为2)或它们的“或”：. FILE\_SHARE\_READ|FILE\_SHARE\_WRITE(值为3)。

**lpsa**：一个LPSECURITY\_ATTRIBUTES型指针，指定返回的句柄是否可以被子进程继承以及指定被打开对象的安全问题。NULL表示不可继承。

**fdwCreate**：指定当目标对象存在或不存在时分别以何种方式动作，例如取OPEN\_EXISTING表示存在则打开，不存在则失败返回。对于磁盘对象，应使用OPEN\_EXISTING。其它请参阅MSDN。

**fdwAttrsAndFlags**：设置文件属性和标志。FILE\_ATTRIBUTE\_NORMAL属性是最普通的文件使用方式，也是默认取值。下面对程序中用的标志作简单介绍：

FILE\_FLAG\_RANDOM\_ACCESS：使所打开的文件(磁盘)可随机访问，操作系统将据此优化文件缓存。

FILE\_FLAG\_NO\_BUFFERING：告诉系统，以无中间缓冲无高速缓存的方式打开文件。下述需求情况下需要以此标志打开文件：

* 访问文件的开始偏移位置必须是扇区长度的整数倍；
* 要读写文件的字节数必须是扇区长度的整数倍；
* 读写文件所用的缓冲区的大小必须是扇区大小的整数倍。

扇区大小可调用GetDiskFreeSpace( )函数得到(参见MSDN)。

**hTemplateFile**：临时文件的句柄，该临时文件提供本操作要创建文件的属性和扩展属性。这个参数通常不使用，取NULL即可。另，在Windows 95中，此值只能取NULL。

**(2) DeviceIoControl( )函数** （**注**：这是一个功能强大的函数）

**功能**：直接向相应设备的驱动程序发出指令，以完成在函数参数中所指定的操作。不同的操作由函数的第二个参数dwIoControlCode控制。函数正确执行返回非0值，操作失败返回0.。

**格式**：

BOOL DeviceIoControl(

HANDLE hDevice, // handle to device of interest

DWORD dwIoControlCode, // control code of operation to perform

LPVOID lpInBuffer, // pointer to buffer to supply input data

DWORD nInBufferSize, // size, in bytes, of input buffer

LPVOID lpOutBuffer, // pointer to buffer to receive output data

DWORD nOutBufferSize, // size, in bytes, of output buffer

LPDWORD lpBytesReturned, // pointer to variable to receive byte count

LPOVERLAPPED lpOverlapped // pointer to structure for asynchronous operation

);

**参数说明：**（以下的输入/输出是从函数角度讲的）

hDevice：指向目标设备的句柄，由CreateFile函数取得。

dwIoControlCode：指定对目标设备进行何种操作以及目标设备的种类，具体的可取值请参考MSDN；本程序取IOCTL\_DISK\_GET\_DRIVE\_GEOMETRY表示获取物理磁盘的信息(执行此操作后，在lpOutBuffer所指的输出缓冲区中存有一个DISK\_GEOMETRY结构，用户可从该结构的成员获得磁盘的几何尺寸数据：扇区大小BytesPerSector、每磁道扇区数SectorsPerTrack、每柱面磁道数TracksPerCylinder、柱面数Cylinders、介质类型MediaType等，参看MSDN)；

IOCTL\_DISK\_GET\_PARTITION\_INFO表示获取磁盘分区信息(在本函数的输出缓冲区中，返回一个PARTITION\_INFORMATION结构中，该结构的成员可参看MSDN或从本程序了解)。

lpInBuffer：是指向输入缓冲区的指针，该缓冲区中包含执行本次操作所需要的数据；如果参数dwIoControlCode指定的操作不需要输入数据，则该参数可以取NULL。

nInBufferSize：上述输入缓冲区所占的字节数。

lpOutBuffer：是指向接收该操作的输出数据的缓冲区的指针。

nOutBufferSize：上述区域所占的字节数。

lpBytesReturned：实际返回结果所占字节数。

lpOverlapped：指向OVERLAPPED结构，在本设备带参数FILE\_FLAG\_OVERLAPPED时来实现异步操作的，其含义是对这种目标对象的操作如果需要一定时间，则可以在操作未完成时便继续调用程序的执行，而该动作异步执行。本程序用不到，设为NULL。

**(3) WriteFile( )函数**

**功能**：向文件(或磁盘等对象)中写入数据。

**格式**：

BOOL WriteFile(

HANDLE hFile, // handle to file to write to

LPCVOID lpBuffer, // pointer to data to write to file

DWORD nNumberOfBytesToWrite, // number of bytes to write

LPDWORD lpNumberOfBytesWritten, // pointer to number of bytes written

LPOVERLAPPED lpOverlapped // pointer to structure for overlapped I/O

);

**参数说明：**

hFile：文件句柄。

lpBuffer：写数据缓冲区。

nNumberOfBytesToWrite：要写入的字节数。注意：在打开目标文件的函数CreateFile中，倒数第二个参数若取值为FILE-FLAG\_NO\_BUFFER，则在读写磁盘时读、写字节数必须是磁盘每扇区包含字节数的整数倍，且文件指针初始位置也要对准扇区边缘的。

lpNumberOfBytesWritten：成功写入的字节数，由该参数返回。

lpOverlapped：OVERLAPPED结构体指针，本程序取值为NULL。

**(4) ReadFile( )函数**

**功能**：从文件(或磁盘等对象)中读出数据。用法同WriteFile极其相似。

**格式**：

BOOL ReadFile(

HANDLE hFile, // handle of file to read

LPVOID lpBuffer, // pointer to buffer that receives data

DWORD nNumberOfBytesToRead, // number of bytes to read

LPDWORD lpNumberOfBytesRead, // pointer to number of bytes read

LPOVERLAPPED lpOverlapped // pointer to structure for data

);

**参数说明：**

hFile：文件句柄。

lpBuffer：读数据缓冲区。

nNumberOfBytesToRead：要读入的字节数。注意：在打开目标文件的函数CreateFile中，倒数第二个参数若取值为FILE-FLAG\_NO\_BUFFER，则在读/写磁盘时，读/写字节数必须是磁盘每扇区包含字节数的整数倍，且文件指针初始位置也要对准扇区边缘的。

lpNumberOfBytesread：成功读入的字节数，由该参数返回。

lpOverlapped：OVERLAPPED结构体指针，本程序取值为NULL。

**(5) SetFilePointer( )函数**

**功能**：移动一个打开的文件(或磁盘等对象)中的读写指针。

**格式**：

DWORD SetFilePointer(

HANDLE hFile, // handle of file

LONG lDistanceToMove, // number of bytes to move file pointer

PLONG lpDistanceToMoveHigh, // pointer to high-order DWORD of distance to move

DWORD dwMoveMethod // how to move

);

**参数说明：**

hFile：文件句柄。

lDistanceToMove：要将指针移动的偏移量(字节数)的低32位

lpDistanceToMoveHigh：要将指针移动的偏移量(字节数)的高32位

dwMoveMethod：确定开始移动的初始位置(取值FILE\_BEGIN表示从文件头开始，类似地也可取FILE\_CURRENT，FILE\_END)。对于磁盘而言，初始位置必须是扇区长度的整数倍。

**(6) GetDriveType( )函数**

**功能**：过得驱动器类型。

**格式**：

UINT GetDriveType(

LPCTSTR lpRootPathName // pointer to root path

);

**参数说明：**

lpRootPathName：根目录路径名。例如"C:\"。若用NULL，则函数使用当前目录的根目录名。

**返回值：**

函数返回一个无符号整数，其值含义如下表：

|  |  |  |
| --- | --- | --- |
| 值 | 符号常量 | 含 义 |
| 0 | DRIVE\_UNKNOWN | 未知驱动器类型 |
| 1 | DRIVE\_NO\_ROOT\_DIR | 驱动器无根目录（驱动器不存在） |
| 2 | DRIVE\_REMOVABLE | 可移动驱动器（例如U盘） |
| 3 | DRIVE\_FIXED | 固定盘（硬盘） |
| 4 | DRIVE\_REMOTE | 远程（网络）驱动器 |
| 5 | DRIVE\_CDROM | CD-ROM（光驱） |
| 6 | DRIVE\_RAMDISK | RAM盘（内存中的虚拟盘） |

1. **实验步骤：**

**1．Disk.h头文件**

// \*\*\*\*\*\*\*\*\*\*\* Disk.h \*\*\*\*\*\*\*\*\*\*\*

#include <windows.h>

#include <winioctl.h> // 包含了DISK\_GEOMETRY等定义

#include <stdio.h>

//#include <iostream.h>

//Disk是结构指针

typedef struct disk {

HANDLE hDisk; //磁盘句柄

CHAR driveletter; //驱动器名(盘符)

UINT driveType; //驱动器类型

BOOL writeFlag; //可写标志

DISK\_GEOMETRY theSupportedGeometry;

} \*Disk; //Disk是指向disk结构的指针

// 以下程序中用到的几个函数

// 打开磁盘，并获得相关物理信息，存入返回的disk结构的theSupportedGeometry项中

int opendisk(Disk &theDisk);

char interwindow(Disk theDisk); // 功能选择接口

bool phyysicalDisk(Disk theDisk); // 获得磁盘的物理参数显示出来

// 读取特定的磁盘区域的内容并将它们显示出来(文件和十六进制两种方式)

bool sectorDump(Disk theDisk);

// 从某磁盘扇区中读出指定字节数的数据到指定缓冲区RdBuf

BOOL sectorRead(Disk theDisk,unsigned logSectorNumber,char \*RdBuf);

// 从指定缓冲区WrBuf写指定字节数的数据到某磁盘扇区中

BOOL sectorWrite(Disk theDisk);

bool partitionInfo(Disk theDisk); //显示分区信息

void changeDrive(Disk &theDisk); //改变驱动器

UINT getdrivetype(char drivel); //获取驱动器类型

void scandrivetype(Disk theDisk); //扫描驱动器，寻找可移动磁盘

**2．Disk.cpp文件**

// \*\*\*\*\*\*\*\*\*\*\*\*\* Disk.cpp \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "Disk.h"

// 驱动器类型

char DriveType[][30]={"DRIVE\_UNKNOWN","NO\_THIS\_DRIVE","DRIVE\_REMOVABLE",

"DRIVE\_FIXED","DRIVE\_REMOTE","DRIVE\_CDROM","DRIVE\_RAMDISK"};

void main()

{

disk objdisk;

Disk theDisk=&objdisk;

objdisk.hDisk=NULL; //句柄初始为空

char choice;

bool runFlag=true;

int openFlag;

//扫描驱动器，寻找可移动磁盘作为初始磁盘

scandrivetype(theDisk);

while (runFlag)

{

if (theDisk->hDisk==NULL) //打开磁盘

openFlag=opendisk(theDisk);

if (openFlag==NULL) //若打开失败，显示出错信息

{

printf("\n\topen disk failed\n");

printf("\tdisk %c: may be not existence\n\n",theDisk->driveletter);

break;

}

if (openFlag==-1)

{

printf("\n\tdisk %c: can't be read\n",theDisk->driveletter);

if (theDisk->driveType==5) //若是光驱

printf("\tthere is no CD-ROM in drive %c:",theDisk->driveletter);

}

choice=interwindow(theDisk);

if (choice=='6')

{

printf("\n\n");

runFlag=false; //退出循环

continue;

}

switch(choice)

{

case '1' : if (phyysicalDisk(theDisk) == false)

printf("\n\tcan't read disk information\n");

break;

case '2' : if (partitionInfo(theDisk)==false)

printf("\n\tGet the information of partition Error.\n");

break;

case '3' : if (!theDisk->writeFlag)

printf("\n\tDrive %c can't be written\n",theDisk->driveletter);

else if (sectorWrite(theDisk) == false)

printf("\n\tWrite sector failed\n");

break;

case '4' : sectorDump(theDisk);

break;

case '5' : changeDrive(theDisk); break;

default : printf("\n\tWrong choice\n");

}

printf("\n\n\tPress \"Enter\" key to continue ");

flushall(); //清除缓冲区

getchar(); //等待按键

}

if (theDisk!=NULL) //关闭磁盘句柄

CloseHandle(theDisk->hDisk);

}

// 功能选择窗口

char interwindow(Disk theDisk)

{

system("cls"); //清屏

char choice;

char chyn[][6]={" not "," "};

printf("\n\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* disk I/O test \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\t Current disk is %c:, ",theDisk->driveletter);

printf("It can%sbe written\n\n",chyn[theDisk->writeFlag]);

printf("\n\t Push 1 to get the information of disk\n");

printf("\t Push 2 to get the information of partition\n");

printf("\t Push 3 to write information to a sector\n");

printf("\t Push 4 to read a sector from disk\n");

printf("\t Push 5 to change the drive for I/O\n");

printf("\t Push 6 to exit from the test\n\n");

printf("\t you choice is : ");

scanf("%c",&choice);

flushall(); //清除键盘输入缓冲区

return choice;

}

UINT getdrivetype(char drivel)

{

char rootdir[]=" :\\";

UINT type;

rootdir[0]=drivel; //构成跟目录字符串"x:\"

type=GetDriveType(rootdir);

return type;

}

void scandrivetype(Disk theDisk)

{

char drivel='C';

UINT type;

while ((type=getdrivetype(drivel))!=1)

{

if (type==2) { //找到可移动磁盘

theDisk->driveletter=drivel; //初始盘符是U盘盘符

theDisk->driveType=type;

theDisk->writeFlag=true; //设定对U盘可写

break;

}

drivel++;

}

if (type==1) { //未找到可移动磁盘

theDisk->driveletter='C'; //初始盘符设为'C'

theDisk->driveType=3; //初始值为硬盘

theDisk->writeFlag=false; //不可写

}

}

bool partitionInfo(Disk theDisk) // 获取磁盘分区信息并显示

{ // 光驱中无光盘时能获取部分分区信息

char yesno[][6]={"No","Yes"};

char PartitionType[][30]={"PARTITION\_ENTRY\_UNUSED","PARTITION\_FAT\_12",

"PARTITION\_XENIX\_1","PARTITION\_XENIX\_2","PARTITION\_FAT\_16",

"PARTITION\_EXTENDED","PARTITION\_HUGE","PARTITION\_IFS"};

int v[]={14,15,65,99,192};

char type[][30]={"PARTITION\_XINT13","PARTITION\_XINT13\_EXTENDED",

"PARTITION\_PREP","PARTITION\_UNIX","VALID\_NTFT"};

DWORD ReturnSize;

HANDLE hDisk=theDisk->hDisk;

PARTITION\_INFORMATION partitionInfo;

PARTITION\_INFORMATION \*pp=&partitionInfo;

int flag=DeviceIoControl(hDisk,IOCTL\_DISK\_GET\_PARTITION\_INFO,NULL,0,

pp,50,&ReturnSize,NULL);

if (!flag) //若调用DeviceIoControl函数失败则返回0

return false;

printf("\n\n\tPARTITION INFORMATION (Drive %c:)\n\n",theDisk->driveletter);

printf("\tStartingOffset : %I64d\n",pp->StartingOffset);

printf("\tPartitionLength : %I64d\n",pp->PartitionLength);

printf("\tHiddenSectors : %d\n",pp->HiddenSectors);

printf("\tPartitionNumber : %d ( %c: )\n",pp->PartitionNumber,theDisk->driveletter);

int n=pp->PartitionType;

if (n<8)

printf("\tPartitionType : %s\n",PartitionType[n]);

else

{

for (int i=0,flag=0;i<5;i++)

{

if (n==v[i])

{

flag=1;

printf("\tPartitionType : %s\n",type[i]);

break;

}

}

if (!flag)

printf("\tPartitionType : %d\n",n);

}

printf("\tBootIndicator : %s\n",yesno[pp->BootIndicator]);

printf("\tRecognizedPartition : %s\n",yesno[pp->RecognizedPartition]);

printf("\tRewritePartition : %s\n",yesno[pp->RewritePartition]);

return true;

}

void changeDrive(Disk &theDisk)

{

char drivel;

char yesno[][6]={"false","true"};

UINT type;

printf("\n\tcurrent driveletter is %c\n",theDisk->driveletter);

printf("\tinput new driveletter : ");

scanf("%c",&drivel);

flushall(); //清除键盘输入缓冲区

if (drivel=='\0'||drivel=='\n') //若直接键入回车键

{

printf("\n\n\tDrive does not change.\n");

return;

}

drivel=toupper(drivel); //转换成大写字母

if (theDisk->driveletter!=drivel)//若输入的驱动器符与当前驱动器符不同

{

type=getdrivetype(drivel);//获取新指定驱动器的类型

if (type==1) //新指定的驱动器不存在

{

printf("\n\tdrive %c: may not be existence\n",drivel);

printf("\tDrive does not change.\n");

return;

}

theDisk->driveType=type; //保存驱动器类型

theDisk->driveletter=drivel; //保存盘符

printf("\n\t%c: is %s\n",drivel,DriveType[type]);//显示驱动器类型

if (type==2) //若新指定的驱动器是U盘

theDisk->writeFlag=true; //允许对新指定驱动器执行写操作

else

theDisk->writeFlag=false; //其它类型的驱动器都不允许写盘

printf("\n\n\tNew drive is : %c\n",drivel); //显示新驱动器符

printf("\twriteFlag is : %s\n",yesno[theDisk->writeFlag]);//显示新驱动器能否“写”

theDisk->driveletter=drivel; //新指定驱动器作为当前驱动器

if (theDisk) //关闭原来的磁盘句柄

{

CloseHandle(theDisk->hDisk);

theDisk->hDisk=NULL; //新驱动器尚未打开，其句柄为0

}

}

else //新指定驱动器与当前驱动器向相同

printf("\n\n\tDrive does not change.\n");

}

// 将获得磁盘的物理参数显示出来

bool phyysicalDisk(Disk theDisk)

{

char mediatype[][40]={"Format is unknown","5.25\", 1.2MB, 512 bytes/sector",

"3.5\", 1.44MB, 512 bytes/sector","3.5\", 2.88MB, 512 bytes/sector",

"3.5\", 20.8MB, 512 bytes/sector"," "," "," "," "," "," ",

"RemovableMedia","FixedMedia"};

if (theDisk->hDisk==NULL)

{

printf("there is no disk available!\n");

return false;

}

DWORD ReturnSize;

int flag=DeviceIoControl(theDisk->hDisk,IOCTL\_DISK\_GET\_DRIVE\_GEOMETRY,

NULL,0,&(theDisk->theSupportedGeometry),50,&ReturnSize,NULL);

if (!flag)

return false;

printf("\n\n\tDISK INFORMATION (Drive %c:)\n\n",theDisk->driveletter);

DWORD sectorsize=theDisk->theSupportedGeometry.BytesPerSector;

printf("\tBytesPerSector : %d\n",sectorsize);

printf("\tSectorPerTrack : ");

printf("%d\n",theDisk->theSupportedGeometry.SectorsPerTrack);

printf("\tTrackPerCylinder : ");

printf("%d\n",theDisk->theSupportedGeometry.TracksPerCylinder);

printf("\tCylinders : %d\n",theDisk->theSupportedGeometry.Cylinders);

int mtype=theDisk->theSupportedGeometry.MediaType;

printf("\tMediaType : %s\n\n",mediatype[mtype]);

return true;

}

//打开磁盘，获得句柄存入返回的一个disk结构

int opendisk(Disk &theDisk)

{

char buffer[]="\\\\.\\ :";

buffer[4]=theDisk->driveletter;

DWORD ReturnSize;

// 调用API函数CreateFile( )打开磁盘，返回的磁盘句柄存于hDisk

theDisk->hDisk=CreateFile(

buffer, //根目录路径名，例如 \\.\C

GENERIC\_READ|GENERIC\_WRITE, //可读、可写。此处可用0xc0000000

FILE\_SHARE\_READ|FILE\_SHARE\_WRITE, //读写共享模式。此处可用3

NULL,

OPEN\_EXISTING, //若对象存在，则打开它；否则，本操作失败

FILE\_FLAG\_RANDOM\_ACCESS|FILE\_FLAG\_NO\_BUFFERING,

NULL

); //失败时返回INVALID\_HANDLE\_VALUE(即-1)

if (theDisk->hDisk==INVALID\_HANDLE\_VALUE)//若驱动器不存在

{

theDisk->hDisk=NULL;

return 0;

}

//获取它的物理参数(磁盘打开后一般需执行此操作后才能正确对磁盘读/写)

int flag=DeviceIoControl(theDisk->hDisk,IOCTL\_DISK\_GET\_DRIVE\_GEOMETRY,

NULL,0,&(theDisk->theSupportedGeometry),50,&ReturnSize,NULL);

if (!flag)

return -1;

return 1;

}

// 读取指定的磁盘区域的内容并将它们显示出来(文件和十六进制两种方式)

bool sectorDump(Disk theDisk)

{

if (theDisk->hDisk==NULL)

{

printf("\n\tthere is no disk available!\n");

return false;

}

DWORD sectorsize=theDisk->theSupportedGeometry.BytesPerSector;

char \*RdBuf=new char[sectorsize]; //动态分配输入缓冲区

int j,logSectorNumber;

//从磁盘某扇区中读出内容并显示(文件和十六进制两种方式)

printf("\n\n\tPlease Input the Sector NO to read from : ");

scanf("%d",&logSectorNumber);getchar();

if (!sectorRead(theDisk,logSectorNumber,RdBuf))

{

printf("\n\tError occurred while reading the sector!\n");

delete []RdBuf; //释放输入缓冲区

return false;

}

printf("\nText Content : \n");

for (DWORD i=0;i<sectorsize;i++)

printf("%c",RdBuf[i]);

printf("\n\n Hex Content :\n");

for (i=0,j=0;i<sectorsize;i++)

{

if (j%16==0)

printf("\n %03x : ",j);

printf("%02x ",(BYTE)RdBuf[i]); //BYTE是单字节整数

j++;

}

printf("\n");

delete []RdBuf; //释放输入缓冲区

return true;

}

// 从某磁盘扇区中读出指定字节数的数据到指定缓冲区RdBuf

BOOL sectorRead(Disk theDisk,unsigned logSectorNumber,char \*RdBuf)

{

DWORD BytesRead;

DWORD sectorsize=theDisk->theSupportedGeometry.BytesPerSector;

long sectortomove=logSectorNumber\*(theDisk->theSupportedGeometry.BytesPerSector);

SetFilePointer(theDisk->hDisk,sectortomove,NULL,FILE\_BEGIN);

if (!ReadFile(theDisk->hDisk,RdBuf,sectorsize,&BytesRead,NULL))

return false;

return true;

}

// 将用户输入的数据写到指定的磁盘扇区中

BOOL sectorWrite(Disk theDisk)

{

DWORD BytesWrite;

DWORD sectorsize=theDisk->theSupportedGeometry.BytesPerSector;

int logSectorNumber;

if (theDisk->hDisk==NULL)

{

printf("\n\nthere is no disk available!\n");

return false;

}

char \*WrBuf=new char[sectorsize]; //动态分配输出缓冲区

ZeroMemory(WrBuf,sectorsize); // 输出缓冲区内存空间清零

// 从指定缓冲区WrBuf写指定字节数的数据到某磁盘扇区中

printf("\n\n\tPlease Input the sector NO to Write to : ");

scanf("%d",&logSectorNumber);

getchar();

printf("\n\tPlease input the content to write to disk\n\t");

gets(WrBuf);

long sectortomove=logSectorNumber\*(theDisk->theSupportedGeometry.BytesPerSector);

SetFilePointer(theDisk->hDisk,sectortomove,NULL,FILE\_BEGIN);//读写指针移到指定位置

if (!WriteFile(theDisk->hDisk,WrBuf,sectorsize,&BytesWrite,NULL))

{

delete []WrBuf; //释放输出缓冲区

return false;

}

printf("\n\twrite complete successfully\n");

delete []WrBuf; //释放输出缓冲区

return true;

}

1. **实验源程序**

**1.Disk.h**

**// \*\*\*\*\*\*\*\*\*\*\* Disk.h \*\*\*\*\*\*\*\*\*\*\***

**#include <windows.h>**

**#include <winioctl.h> // 包含了DISK\_GEOMETRY等定义**

**#include <stdio.h>**

**#include <iostream>**

**using namespace std;**

**//Disk是结构指针**

**typedef struct disk {**

**HANDLE hDisk; //磁盘句柄**

**CHAR driveletter; //驱动器名(盘符)**

**UINT driveType; //驱动器类型**

**BOOL writeFlag; //可写标志**

**DISK\_GEOMETRY theSupportedGeometry;**

**} \*Disk; //Disk是指向disk结构的指针**

**// 以下程序中用到的几个函数**

**// 打开磁盘，并获得相关物理信息，存入返回的disk结构的theSupportedGeometry项中**

**int opendisk(Disk &theDisk);**

**char interwindow(Disk theDisk); // 功能选择接口**

**bool phyysicalDisk(Disk theDisk); // 获得磁盘的物理参数显示出来**

**// 读取特定的磁盘区域的内容并将它们显示出来(文件和十六进制两种方式)**

**bool sectorDump(Disk theDisk);**

**// 从某磁盘扇区中读出指定字节数的数据到指定缓冲区RdBuf**

**BOOL sectorRead(Disk theDisk,unsigned logSectorNumber,char \*RdBuf);**

**// 从指定缓冲区WrBuf写指定字节数的数据到某磁盘扇区中**

**BOOL sectorWrite(Disk theDisk);**

**bool partitionInfo(Disk theDisk); //显示分区信息**

**void changeDrive(Disk &theDisk); //改变驱动器**

**UINT getdrivetype(char drivel); //获取驱动器类型**

**void scandrivetype(Disk theDisk); //扫描驱动器，寻找可移动磁盘**

**2.Disk.cpp**

**// \*\*\*\*\*\*\*\*\*\*\*\*\* Disk.cpp \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include "Disk.h"**

**// 驱动器类型**

**char DriveType[][30]={"DRIVE\_UNKNOWN","NO\_THIS\_DRIVE","DRIVE\_REMOVABLE",**

**"DRIVE\_FIXED","DRIVE\_REMOTE","DRIVE\_CDROM","DRIVE\_RAMDISK"};**

**int main()**

**{**

**disk objdisk;**

**Disk theDisk=&objdisk;**

**objdisk.hDisk=NULL; //句柄初始为空**

**char choice;**

**bool runFlag=true;**

**int openFlag;**

**//扫描驱动器，寻找可移动磁盘作为初始磁盘**

**scandrivetype(theDisk);**

**while (runFlag)**

**{**

**if (theDisk->hDisk==NULL) //打开磁盘**

**openFlag=opendisk(theDisk);**

**if (openFlag==NULL) //若打开失败，显示出错信息**

**{**

**printf("\n\topen disk failed\n");**

**printf("\tdisk %c: may be not existence\n\n",theDisk->driveletter);**

**break;**

**}**

**if (openFlag==-1)**

**{**

**printf("\n\tdisk %c: can't be read\n",theDisk->driveletter);**

**if (theDisk->driveType==5) //若是光驱**

**printf("\tthere is no CD-ROM in drive %c:",theDisk->driveletter);**

**}**

**choice=interwindow(theDisk);**

**if (choice=='6')**

**{**

**printf("\n\n");**

**runFlag=false; //退出循环**

**continue;**

**}**

**switch(choice)**

**{**

**case '1' : if (phyysicalDisk(theDisk) == false)**

**printf("\n\tcan't read disk information\n");**

**break;**

**case '2' : if (partitionInfo(theDisk)==false)**

**printf("\n\tGet the information of partition Error.\n");**

**break;**

**case '3' : if (!theDisk->writeFlag)**

**printf("\n\tDrive %c can't be written\n",theDisk->driveletter);**

**else if (sectorWrite(theDisk) == false)**

**printf("\n\tWrite sector failed\n");**

**break;**

**case '4' : sectorDump(theDisk);**

**break;**

**case '5' : changeDrive(theDisk); break;**

**default : printf("\n\tWrong choice\n");**

**}**

**printf("\n\n\tPress \"Enter\" key to continue ");**

**\_flushall(); //清除缓冲区**

**getchar(); //等待按键**

**}**

**if (theDisk!=NULL) //关闭磁盘句柄**

**CloseHandle(theDisk->hDisk);**

**}**

**// 功能选择窗口**

**char interwindow(Disk theDisk)**

**{**

**system("cls"); //清屏**

**char choice;**

**char chyn[][6]={" not "," "};**

**printf("\n\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* disk I/O test \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\n\t Current disk is %c:, ",theDisk->driveletter);**

**printf("It can%sbe written\n\n",chyn[theDisk->writeFlag]);**

**printf("\n\t Push 1 to get the information of disk\n");**

**printf("\t Push 2 to get the information of partition\n");**

**printf("\t Push 3 to write information to a sector\n");**

**printf("\t Push 4 to read a sector from disk\n");**

**printf("\t Push 5 to change the drive for I/O\n");**

**printf("\t Push 6 to exit from the test\n\n");**

**printf("\t you choice is : ");**

**scanf("%c",&choice);**

**\_flushall(); //清除键盘输入缓冲区**

**return choice;**

**}**

**UINT getdrivetype(char drivel)**

**{**

**char rootdir[]=" :\\";**

**UINT type;**

**rootdir[0]=drivel; //构成跟目录字符串"x:\"**

**type=GetDriveType(rootdir);**

**return type;**

**}**

**void scandrivetype(Disk theDisk)**

**{**

**char drivel='C';**

**UINT type;**

**while ((type=getdrivetype(drivel))!=1)**

**{**

**if (type==2) { //找到可移动磁盘**

**theDisk->driveletter=drivel; //初始盘符是U盘盘符**

**theDisk->driveType=type;**

**theDisk->writeFlag=true; //设定对U盘可写**

**break;**

**}**

**drivel++;**

**}**

**if (type==1) { //未找到可移动磁盘**

**theDisk->driveletter='C'; //初始盘符设为'C'**

**theDisk->driveType=3; //初始值为硬盘**

**theDisk->writeFlag=false; //不可写**

**}**

**}**

**bool partitionInfo(Disk theDisk) // 获取磁盘分区信息并显示**

**{ // 光驱中无光盘时能获取部分分区信息**

**char yesno[][6]={"No","Yes"};**

**char PartitionType[][30]={"PARTITION\_ENTRY\_UNUSED","PARTITION\_FAT\_12",**

**"PARTITION\_XENIX\_1","PARTITION\_XENIX\_2","PARTITION\_FAT\_16",**

**"PARTITION\_EXTENDED","PARTITION\_HUGE","PARTITION\_IFS"};**

**int v[]={14,15,65,99,192};**

**char type[][30]={"PARTITION\_XINT13","PARTITION\_XINT13\_EXTENDED",**

**"PARTITION\_PREP","PARTITION\_UNIX","VALID\_NTFT"};**

**DWORD ReturnSize;**

**HANDLE hDisk=theDisk->hDisk;**

**PARTITION\_INFORMATION partitionInfo;**

**PARTITION\_INFORMATION \*pp=&partitionInfo;**

**int flag=DeviceIoControl(hDisk,IOCTL\_DISK\_GET\_PARTITION\_INFO,NULL,0,**

**pp,50,&ReturnSize,NULL);**

**if (!flag) //若调用DeviceIoControl函数失败则返回0**

**return false;**

**printf("\n\n\tPARTITION INFORMATION (Drive %c:)\n\n",theDisk->driveletter);**

**printf("\tStartingOffset : %I64d\n",pp->StartingOffset);**

**printf("\tPartitionLength : %I64d\n",pp->PartitionLength);**

**printf("\tHiddenSectors : %d\n",pp->HiddenSectors);**

**printf("\tPartitionNumber : %d ( %c: )\n",pp->PartitionNumber,theDisk->driveletter);**

**int n=pp->PartitionType;**

**if (n<8)**

**printf("\tPartitionType : %s\n",PartitionType[n]);**

**else**

**{**

**for (int i=0,flag=0;i<5;i++)**

**{**

**if (n==v[i])**

**{**

**flag=1;**

**printf("\tPartitionType : %s\n",type[i]);**

**break;**

**}**

**}**

**if (!flag)**

**printf("\tPartitionType : %d\n",n);**

**}**

**printf("\tBootIndicator : %s\n",yesno[pp->BootIndicator]);**

**printf("\tRecognizedPartition : %s\n",yesno[pp->RecognizedPartition]);**

**printf("\tRewritePartition : %s\n",yesno[pp->RewritePartition]);**

**return true;**

**}**

**void changeDrive(Disk &theDisk)**

**{**

**char drivel;**

**char yesno[][6]={"false","true"};**

**UINT type;**

**printf("\n\tcurrent driveletter is %c\n",theDisk->driveletter);**

**printf("\tinput new driveletter : ");**

**scanf("%c",&drivel);**

**\_flushall(); //清除键盘输入缓冲区**

**if (drivel=='\0'||drivel=='\n') //若直接键入回车键**

**{**

**printf("\n\n\tDrive does not change.\n");**

**return;**

**}**

**drivel=toupper(drivel); //转换成大写字母**

**if (theDisk->driveletter!=drivel)//若输入的驱动器符与当前驱动器符不同**

**{**

**type=getdrivetype(drivel);//获取新指定驱动器的类型**

**if (type==1) //新指定的驱动器不存在**

**{**

**printf("\n\tdrive %c: may not be existence\n",drivel);**

**printf("\tDrive does not change.\n");**

**return;**

**}**

**theDisk->driveType=type; //保存驱动器类型**

**theDisk->driveletter=drivel; //保存盘符**

**printf("\n\t%c: is %s\n",drivel,DriveType[type]);//显示驱动器类型**

**if (type==2) //若新指定的驱动器是U盘**

**theDisk->writeFlag=true; //允许对新指定驱动器执行写操作**

**else**

**theDisk->writeFlag=false; //其它类型的驱动器都不允许写盘**

**printf("\n\n\tNew drive is : %c\n",drivel); //显示新驱动器符**

**printf("\twriteFlag is : %s\n",yesno[theDisk->writeFlag]);//显示新驱动器能否“写”**

**theDisk->driveletter=drivel; //新指定驱动器作为当前驱动器**

**if (theDisk) //关闭原来的磁盘句柄**

**{**

**CloseHandle(theDisk->hDisk);**

**theDisk->hDisk=NULL; //新驱动器尚未打开，其句柄为0**

**}**

**}**

**else //新指定驱动器与当前驱动器向相同**

**printf("\n\n\tDrive does not change.\n");**

**}**

**// 将获得磁盘的物理参数显示出来**

**bool phyysicalDisk(Disk theDisk)**

**{**

**char mediatype[][40]={"Format is unknown","5.25\", 1.2MB, 512 bytes/sector",**

**"3.5\", 1.44MB, 512 bytes/sector","3.5\", 2.88MB, 512 bytes/sector",**

**"3.5\", 20.8MB, 512 bytes/sector"," "," "," "," "," "," ",**

**"RemovableMedia","FixedMedia"};**

**if (theDisk->hDisk==NULL)**

**{**

**printf("there is no disk available!\n");**

**return false;**

**}**

**DWORD ReturnSize;**

**int flag=DeviceIoControl(theDisk->hDisk,IOCTL\_DISK\_GET\_DRIVE\_GEOMETRY,**

**NULL,0,&(theDisk->theSupportedGeometry),50,&ReturnSize,NULL);**

**if (!flag)**

**return false;**

**printf("\n\n\tDISK INFORMATION (Drive %c:)\n\n",theDisk->driveletter);**

**DWORD sectorsize=theDisk->theSupportedGeometry.BytesPerSector;**

**printf("\tBytesPerSector : %d\n",sectorsize);**

**printf("\tSectorPerTrack : ");**

**printf("%d\n",theDisk->theSupportedGeometry.SectorsPerTrack);**

**printf("\tTrackPerCylinder : ");**

**printf("%d\n",theDisk->theSupportedGeometry.TracksPerCylinder);**

**printf("\tCylinders : %d\n",theDisk->theSupportedGeometry.Cylinders);**

**int mtype=theDisk->theSupportedGeometry.MediaType;**

**printf("\tMediaType : %s\n\n",mediatype[mtype]);**

**return true;**

**}**

**//打开磁盘，获得句柄存入返回的一个disk结构**

**int opendisk(Disk &theDisk)**

**{**

**char buffer[]="\\\\.\\ :";**

**buffer[4]=theDisk->driveletter;**

**DWORD ReturnSize;**

**// 调用API函数CreateFile( )打开磁盘，返回的磁盘句柄存于hDisk**

**theDisk->hDisk=CreateFile(**

**buffer, //根目录路径名，例如 \\.\C**

**GENERIC\_READ|GENERIC\_WRITE, //可读、可写。此处可用0xc0000000**

**FILE\_SHARE\_READ|FILE\_SHARE\_WRITE, //读写共享模式。此处可用3**

**NULL,**

**OPEN\_EXISTING, //若对象存在，则打开它；否则，本操作失败**

**FILE\_FLAG\_RANDOM\_ACCESS|FILE\_FLAG\_NO\_BUFFERING,**

**NULL**

**); //失败时返回INVALID\_HANDLE\_VALUE(即-1)**

**if (theDisk->hDisk==INVALID\_HANDLE\_VALUE)//若驱动器不存在**

**{**

**theDisk->hDisk=NULL;**

**return 0;**

**}**

**//获取它的物理参数(磁盘打开后一般需执行此操作后才能正确对磁盘读/写)**

**int flag=DeviceIoControl(theDisk->hDisk,IOCTL\_DISK\_GET\_DRIVE\_GEOMETRY,**

**NULL,0,&(theDisk->theSupportedGeometry),50,&ReturnSize,NULL);**

**if (!flag)**

**return -1;**

**return 1;**

**}**

**// 读取指定的磁盘区域的内容并将它们显示出来(文件和十六进制两种方式)**

**bool sectorDump(Disk theDisk)**

**{**

**if (theDisk->hDisk==NULL)**

**{**

**printf("\n\tthere is no disk available!\n");**

**return false;**

**}**

**DWORD sectorsize=theDisk->theSupportedGeometry.BytesPerSector;**

**char \*RdBuf=new char[sectorsize]; //动态分配输入缓冲区**

**int j,logSectorNumber;**

**//从磁盘某扇区中读出内容并显示(文件和十六进制两种方式)**

**printf("\n\n\tPlease Input the Sector NO to read from : ");**

**scanf("%d",&logSectorNumber);getchar();**

**if (!sectorRead(theDisk,logSectorNumber,RdBuf))**

**{**

**printf("\n\tError occurred while reading the sector!\n");**

**delete []RdBuf; //释放输入缓冲区**

**return false;**

**}**

**printf("\nText Content : \n");**

**for (DWORD i=0;i<sectorsize;i++)**

**printf("%c",RdBuf[i]);**

**printf("\n\n Hex Content :\n");**

**for (int i=0,j=0;i<sectorsize;i++)**

**{**

**if (j%16==0)**

**printf("\n %03x : ",j);**

**printf("%02x ",(BYTE)RdBuf[i]); //BYTE是单字节整数**

**j++;**

**}**

**printf("\n");**

**delete []RdBuf; //释放输入缓冲区**

**return true;**

**}**

**// 从某磁盘扇区中读出指定字节数的数据到指定缓冲区RdBuf**

**BOOL sectorRead(Disk theDisk,unsigned logSectorNumber,char \*RdBuf)**

**{**

**DWORD BytesRead;**

**DWORD sectorsize=theDisk->theSupportedGeometry.BytesPerSector;**

**long sectortomove=logSectorNumber\*(theDisk->theSupportedGeometry.BytesPerSector);**

**SetFilePointer(theDisk->hDisk,sectortomove,NULL,FILE\_BEGIN);**

**if (!ReadFile(theDisk->hDisk,RdBuf,sectorsize,&BytesRead,NULL))**

**return false;**

**return true;**

**}**

**// 将用户输入的数据写到指定的磁盘扇区中**

**BOOL sectorWrite(Disk theDisk)**

**{**

**DWORD BytesWrite;**

**DWORD sectorsize=theDisk->theSupportedGeometry.BytesPerSector;**

**int logSectorNumber;**

**if (theDisk->hDisk==NULL)**

**{**

**printf("\n\nthere is no disk available!\n");**

**return false;**

**}**

**char \*WrBuf=new char[sectorsize]; //动态分配输出缓冲区**

**ZeroMemory(WrBuf,sectorsize); // 输出缓冲区内存空间清零**

**// 从指定缓冲区WrBuf写指定字节数的数据到某磁盘扇区中**

**printf("\n\n\tPlease Input the sector NO to Write to : ");**

**scanf("%d",&logSectorNumber);**

**getchar();**

**printf("\n\tPlease input the content to write to disk\n\t");**

**gets(WrBuf);**

**long sectortomove=logSectorNumber\*(theDisk->theSupportedGeometry.BytesPerSector);**

**SetFilePointer(theDisk->hDisk,sectortomove,NULL,FILE\_BEGIN);//读写指针移到指定位置**

**if (!WriteFile(theDisk->hDisk,WrBuf,sectorsize,&BytesWrite,NULL))**

**{**

**delete []WrBuf; //释放输出缓冲区**

**return false;**

**}**

**printf("\n\twrite complete successfully\n");**

**delete []WrBuf; //释放输出缓冲区**

**return true;**

**}**

1. **实验结果**
2. **实验总结：**

**本次实验主要是要求通过本上机实验了解Windows系统中如何直接使用磁盘的读写功能；所编应用程序能够响应用户指定的读写磁盘扇区的请求，也能提供查看磁盘相关参数的功能。通过调用相关的WIN32 API函数，通过本实验，用户可以利用Win32 API进行底层的磁盘操作，了解Windows 操作系统设备管理的一些知识。通过自己的努力发现问题并解决问题，我相信这对于我以后的代码生涯也是很宝贵的经历。总而言之，我在这次实验过程中学到了很多。**