|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 数学与信息科学学院 | | | | | | |  |
|  |  | | | | | | |  |
|  |  | | | | | | |  |
|  | 实验报告 | | | | | | |  |
|  |  | | | | | | |  |
|  |  | | | | | | |  |
|  | 课程名称： | | 计算机操作系统 | | | | |  |
|  | 姓 名： | | 任希恒 | | | | |  |
|  | 学 号： | | 541910010217 | | | | |  |
|  | 专业班级： | | 信息与计算科学专业19-02班 | | | | |  |
|  | 指导教师： | | 时海亮 | | | | |  |
|  |  | |  | | | | |  |
|  |  | |  | | | | |  |
|  |  | 2021-2022 | | 学年第 | 2 | 学期 |  |  |

实验5 文件读/写

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验日期： | 2022年05月20日 | 实验类型： | 设计型 | 实验成绩： |  |

# 一、实验目的

通过本实验了解Windows系统文件读写的相关API，掌握无缓冲方式实现文件读写相关参数的设置；了解Windows系统文件高速缓存的概念，掌握采用缓冲方式实现文件读写相关参数的设置；了解Windows系统异步文件读写的概念，掌握采用异步方式实现文件读写相关参数的设置。

# 二、实验内容

1．采用无缓冲方式实现文件读写；

2．采用缓冲方式实现文件读写；

3．采用异步方式实现文件读写。

# 三、实验代码及结果

1.代码如下：

#include<windows.h>

#include<iostream.h>

void FileRW\_NoBuffer(const char\*source,const char \* destination);

void main()

{

cout<<"Now read a txt file to another file!"<<endl;

FileRW\_NoBuffer("source.txt","nobuffer.txt");

}

void FileRW\_NoBuffer(const char \* source,const char \* destination)

{

HANDLE hSource;

HANDLE hDest;

DWORD dwRead;

DWORD dwWrite;

char buf[1024];

hSource = CreateFile(source,GENERIC\_READ,0,NULL,OPEN\_EXISTING,FILE\_FLAG\_NO\_BUFFERING,NULL);

if(INVALID\_HANDLE\_VALUE == hSource)

{

cout<<"Could not open the source file!"<<endl;

return;

}

hDest = CreateFile(destination,GENERIC\_WRITE,0,NULL,CREATE\_ALWAYS,FILE\_ATTRIBUTE\_NORMAL,NULL);

if(INVALID\_HANDLE\_VALUE == hDest)

{

cout<<"Could not create the destination file!"<<endl;

return;

}

if(!ReadFile(hSource,buf,1024,&dwRead,NULL))

{

cout<<"Read source file error!"<<endl;

return;

}

else

cout<<"Read file success!"<<endl;

if(dwRead == 1024)

{

cout<<"你的文件可能被截断，请增加缓冲区大小！"<<endl;

}

if(!WriteFile(hDest,buf,dwRead,&dwWrite,NULL))

{

cout<<"Read source file error!"<<endl;

return;

}

else

cout<<"Write file success!"<<endl;

CloseHandle(hSource);

CloseHandle(hDest);

}

运行结果如图1所示

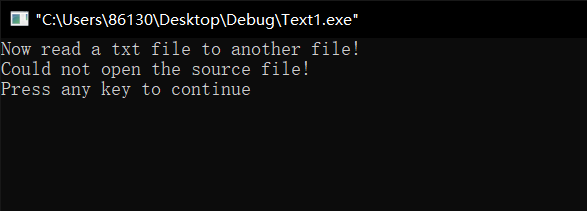


图1

2.代码如下：

#include<windows.h>

#include<iostream.h>

void FileRW\_Buffer(const char\*source,const char \* destination);

void main()

{

cout<<"Now read a txt file to another file!"<<endl;

FileRW\_Buffer("source.txt","buffer.txt");

}

void FileRW\_Buffer(const char \* source,const char \* destination)

{

HANDLE hSource;

HANDLE hDest;

DWORD dwRead;

DWORD dwWrite;

char buf[1024];

hSource = CreateFile(source,GENERIC\_READ,0,NULL,OPEN\_EXISTING,FILE\_FLAG\_SEQUENTIAL\_SCAN,NULL);

if(INVALID\_HANDLE\_VALUE == hSource)

{

cout<<"Could not open the source file!"<<endl;

return;

}

hDest = CreateFile(destination,GENERIC\_WRITE,0,NULL,CREATE\_ALWAYS,FILE\_ATTRIBUTE\_NORMAL,NULL);

if(INVALID\_HANDLE\_VALUE == hDest)

{

cout<<"Could not create the destination file!"<<endl;

return;

}

if(!ReadFile(hSource,buf,1024,&dwRead,NULL))

{

cout<<"Read source file error!"<<endl;

return;

}

else

cout<<"Read file success!"<<endl;

if(dwRead == 1024)

{

cout<<"你的文件可能被截断，请增加缓冲区大小！"<<endl;

}

if(!WriteFile(hDest,buf,dwRead,&dwWrite,NULL))

{

cout<<"Read source file error!"<<endl;

return;

}

else

cout<<"Write file success!"<<endl;

CloseHandle(hSource);

CloseHandle(hDest);

}

运行结果如图2所示；

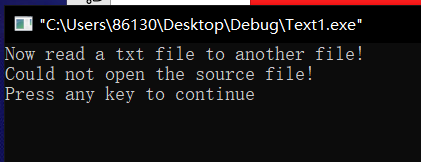


图2

3.代码如下：

#include<windows.h>

#include<iostream.h>

void FileRW\_Asyn(const char\*source,const char \* destination);

void main()

{

cout<<"Now read a txt file to another file!"<<endl;

FileRW\_Asyn("source.txt","Asyn.txt");

}

void FileRW\_Asyn(const char \* source,const char \* destination)

{

HANDLE hSource;

HANDLE hDest;

DWORD dwRead;

DWORD dwWrite;

char buf[1024];

hSource = CreateFile(source,GENERIC\_READ,0,NULL,OPEN\_EXISTING,FILE\_FLAG\_NO\_BUFFERING | FILE\_FLAG\_OVERLAPPED,NULL);//指定异步方式进行文件读写

if(INVALID\_HANDLE\_VALUE == hSource)

{

cout<<"Could not open the source file!"<<endl;

return;

}

hDest = CreateFile(destination,GENERIC\_WRITE,0,NULL,CREATE\_ALWAYS,FILE\_FLAG\_NO\_BUFFERING | FILE\_FLAG\_OVERLAPPED,NULL);

if(INVALID\_HANDLE\_VALUE == hDest)

{

cout<<"Could not create the destination file!"<<endl;

return;

}

OVERLAPPED ovlap;

ovlap.Offset = -1024;

ZeroMemory(&ovlap,sizeof(OVERLAPPED));

if(!ReadFile(hSource,buf,1024,NULL,&ovlap))

{

if(ERROR\_IO\_PENDING != GetLastError())

{

cout<<"Read source file error!"<<endl;

return;

}

}

cout<<"Read file success!"<<endl;

dwRead = 1024;

if(!WriteFile(hDest,buf,dwRead,&dwWrite,&ovlap))

{

cout<<"Read source file error!"<<endl;

return;

}

else

cout<<"Write file success!"<<endl;

CloseHandle(hSource);

CloseHandle(hDest);

}

运行结果如图3所示

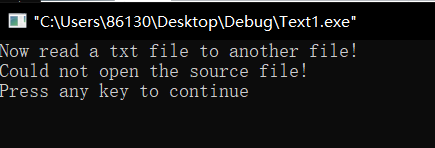


图3

# 四、实验总结

函数CreateFile( ) 用于创建一个新文件，如果文件已经存在，则得到该文件的句柄。该函数的参数dwFalgsAndAttributes决定了文件的传输方式，对于普通的文件传输，可将参数设置为FILE\_ATTRIBUTE\_NORMAL；而若设置为FILE\_FLAG\_NO\_BUFFERING，表示不使用高速缓存进行文件传输；若同时使用标志FILE\_FLAG\_NO\_BUFFERING和FILE\_FLAG\_OVERLAPPED，可对文件进行异步传输；若设置为FILE\_FLAG\_SEQUENTIAL\_SCAN，表示使用高速缓存进行文件的传输。文件创建成功，该函数返回文件句柄，否则返回INVALID\_HANDLE\_VALUE，可调用函数GetLastError( )查询失败的原因。函数Readfile( ) 从文件指针指示的位置开始读取文件中的数据。如果文件不是用FILE\_FLAG\_OVERLAPPED属性创建的，则文件指针移动到实际读出字节数所处的位置；如果文件是用FILE\_FLAG\_OVERLAPPED属性创建的，则文件指针由应用程序来调整其位置。