实验 5　文件读/写

一、实验目的

通过本实验熟悉 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、实验准备知识

1. 创建文件函数

```
HANDLE CreateNamedPipe(
  LPCTSTR lpName,
  DWORD dwOpenMode,
  DWORD dwPipeMode,
  DWORD nMaxInstances,
  DWORD nOutBufferSize,
  DWORD nInBufferSize,
  DWORD nDefaultTimeOut,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

2. 读取文件函数

```
BOOL ReadFile(
  HANDLE hFile,
  LPVOID lpBuffer,
  DWORD nNumberOfBytesToRead,
  LPDWORD lpNumberOfBytesRead,
  LPOVERLAPPED lpOverlapped
);
```

3. 写入文件函数

```
BOOL WriteFile(
  HANDLE hFile,
  LPCVOID lpBuffer,
  DWORD nNumberOfBytesToWrite,
  LPDWORD lpNumberOfBytesWritten,
  LPOVERLAPPED lpOverlapped
);
```

四、实验步骤

1. 采用无缓冲方式实现文件读写

其主要代码如下所示。

```
#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, // 指定无缓冲方式进行文件读写
      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);
}
```

2. 采用缓冲方式实现文件读写

其主要代码如下所示。

```
#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,  // 指定高速缓冲方式进行文件读写
        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);
}
```

3. 采用异步方式实现文件读写

其主要代码如下所示。

```
#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,  // 指定异步方
式进行文件读写
        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_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);
}
```

五、作业

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。