## 一、文件的异步写入

|  |  |
| --- | --- |
|  | #include <Windows.h>  #include <stdio.h>    int main() {      HANDLE hFile = CreateFile(TEXT("test.txt"), GENERIC\_WRITE | GENERIC\_READ, 0, NULL, OPEN\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);      if (hFile == INVALID\_HANDLE\_VALUE) {          printf("无法打开文件。错误码:%d\n", GetLastError());          return 0;      }        OVERLAPPED ol1 = { 0 };      char buffer[] = "Hello World";      DWORD writeCount = 0;      BOOL ret = WriteFile(hFile, buffer, strlen(buffer), &writeCount, &ol1);      /\*if (!ret) {          CloseHandle(hFile);          printf("文件写入失败!\n");          return -1;      }\*/ //异步IO操作这样子判断是不对的.        CloseHandle(hFile);        return 0;  }  注意:以异步方式写文件, writeCount根本得不到数据,因为可能还没有等到IO完成函数已经返回了.所以上面的判断永远都是FALSE  异步方式下, strlen(buffer)和writeCount不相等,在同步方式下面他们是相等的. |

## 二、文件的异步读取

|  |  |
| --- | --- |
|  | #include <Windows.h>  #include <stdio.h>    int main() {      HANDLE hFile = CreateFile(TEXT("test.txt"), GENERIC\_WRITE | GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);      if (hFile == INVALID\_HANDLE\_VALUE) {          printf("无法打开文件。错误码:%d\n", GetLastError());          return 0;      }      OVERLAPPED ol2 = { 0 };      char readBuffer[255] = { 0 };      DWORD readCount = 0;      ret = ReadFile(hFile, readBuffer, 255, &readCount, &ol2);      /\*if (!ret) {          CloseHandle(hFile);          printf("文件读取失败!\n");          return 0;      }\*/      printf("readBuffer:%s\n", readBuffer);        CloseHandle(hFile);        return 1;  } |

## 三、异步读写操作的判断方法

(1).异步文件写入操作的判断

|  |  |
| --- | --- |
|  | OVERLAPPED ol1 = { 0 };  char buffer[] = "Hello World";  DWORD writeCount = 0;  BOOL ret = WriteFile(hFile, buffer, strlen(buffer), &writeCount, &ol1);  if (!ret) {      DWORD err = GetLastError();      if (ERROR\_IO\_PENDING == err) {          printf("正在进行异步写入操作!\n");      }      else {          printf("文件写入失败!\n");          CloseHandle(hFile);          return 0;      }  } |

(2).异步文件读取操作的判断

|  |  |
| --- | --- |
|  | OVERLAPPED ol2 = { 0 };  char readBuffer[255] = { 0 };  DWORD readCount = 0;  ret = ReadFile(hFile, readBuffer, 255, &readCount, &ol2);  if (!ret) {      DWORD err = GetLastError();      if (ERROR\_IO\_PENDING == err) {          printf("正在进行异步读取操作!\n");      }      else {          printf("文件读取失败!\n");          CloseHandle(hFile);          return 0;      }  }  printf("readBuffer:%s\n", readBuffer); |

## 四、异步IO完成通知的方法

### 1.触发设备内核对象

触发设备内核对象：允许一个线程发出IO请求，另一个线程对结果进行处理，当向一个设备同时发出多个IO请求的时候，此方法无效

(1).等待文件写入完毕

|  |  |
| --- | --- |
|  | OVERLAPPED ol1 = { 0 };  char buffer[] = "Hello World";  DWORD writeCount = 0;  BOOL ret = WriteFile(hFile, buffer, strlen(buffer), &writeCount, &ol1);  if (!ret) {      DWORD err = GetLastError();      if (ERROR\_IO\_PENDING == err) {          printf("正在进行异步写入操作!\n");          WaitForSingleObject(hFile, INFINITE);          printf("异步写入完毕!\n");        }      else {          printf("文件写入失败!\n");          CloseHandle(hFile);          return 0;          }  } |

(2).等待文件读取完毕

|  |  |
| --- | --- |
|  | OVERLAPPED ol2 = { 0 };  char readBuffer[255] = { 0 };  DWORD readCount = 0;  ret = ReadFile(hFile, readBuffer, 255, &readCount, &ol2);  if (!ret) {      DWORD err = GetLastError();      if (ERROR\_IO\_PENDING == err) {          printf("正在进行异步读取操作!\n");          WaitForSingleObject(hFile, INFINITE);          printf("异步读取完毕\n");      }      else {          printf("文件读取失败!\n");          CloseHandle(hFile);          return 0;      }  }  printf("readBuffer:%s\n", readBuffer); |

### 2.触发事件内核对象

这种方法允许我们向一个设备同时发出多个IO请求，它允许一个线程发出IO请求，另一个线程对结果进行处理

示例代码：

|  |  |
| --- | --- |
|  | #include <Windows.h>  #include <stdio.h>    int main() {      HANDLE hFile = CreateFile(TEXT("test.txt"), GENERIC\_WRITE | GENERIC\_READ, 0, NULL, OPEN\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);      if (hFile == INVALID\_HANDLE\_VALUE) {          printf("无法打开文件。错误码:%d\n", GetLastError());          return 0;      }        OVERLAPPED ol1 = { 0 };      ol1.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);      OVERLAPPED ol2 = { 0 };      ol2.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);      char buffer[] = "Hello World";      char readBuffer[256] = { 0 };      DWORD writeCount = 0;      DWORD readCount = 0;      BOOL ret = WriteFile(hFile, buffer, strlen(buffer), &writeCount, &ol1);      ret = ReadFile(hFile, readBuffer, strlen(readBuffer), &readCount, &ol2);      if (!ret) {          DWORD err = GetLastError();          if (ERROR\_IO\_PENDING == err) {              HANDLE handle[2];              handle[0] = ol1.hEvent;              handle[1] = ol2.hEvent;              DWORD objnum = WaitForMultipleObjects(2, handle, TRUE, INFINITE);              switch (objnum) {              case WAIT\_OBJECT\_0: {                  printf("文件写入操作完毕\n");              }              case WAIT\_OBJECT\_0 + 1: {                  printf("文件读取操作完毕：%s\n", readBuffer);              }              default:                  break;              }              CloseHandle(hFile);              return 0;          }          else {              printf("File failed\n");          }      }        CloseHandle(ol1.hEvent);      CloseHandle(ol2.hEvent);      CloseHandle(hFile);        return 1;  } |

### 3.使用可提醒IO

这种方法允许我们向一个设备发出多个IO请求，发出IO请求的线程必须对结果进行处理

示例代码：

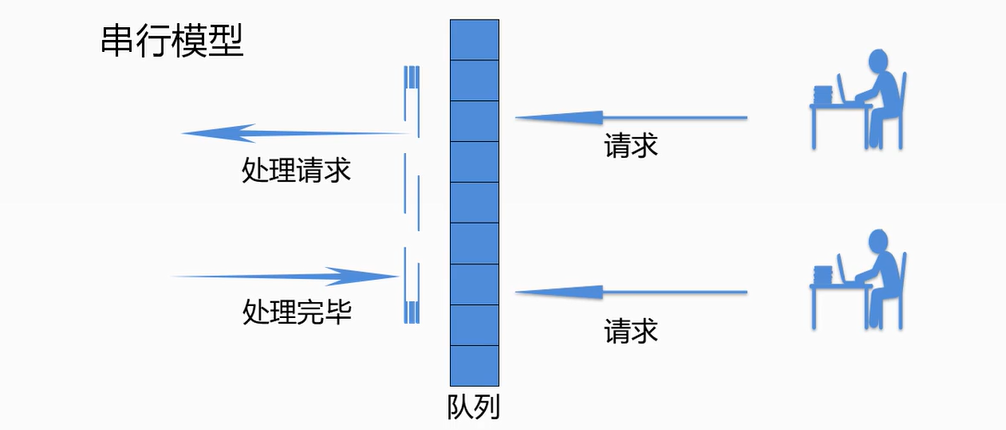
|  |  |
| --- | --- |
|  | #include <Windows.h>  #include <stdio.h>    char readBuffer[256] = { 0 };  char writeBuffer[256] = "1234567890";    VOID WriteFunc(DWORD dwErrorCode, DWORD dwNumberOfBytesTransfered, LPOVERLAPPED lpOverlapped) {      printf("文件写入结束\n");  }    VOID ReadFunc(DWORD dwErrorCode, DWORD dwNumberOfBytesTransfered, LPOVERLAPPED lpOverlapped) {      printf("%s\n", readBuffer);  }    int main() {      HANDLE hFile = CreateFile(TEXT("test.txt"), GENERIC\_WRITE | GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);      if (hFile == INVALID\_HANDLE\_VALUE) {          printf("无法打开文件。错误码:%d\n", GetLastError());          return 0;      }      OVERLAPPED ol2 = { 0 };        BOOL ret = WriteFileEx(hFile, writeBuffer, strlen(writeBuffer), &ol2, (LPOVERLAPPED\_COMPLETION\_ROUTINE)WriteFunc);      SleepEx(1000, true);      if (!ret) {          DWORD err = GetLastError();          if (ERROR\_IO\_PENDING == err) {              printf("文件写入中！！！\n");          }          else {              printf("文件写入失败！！！\n");              CloseHandle(hFile);              return 0;          }      }        ret = ReadFileEx(hFile, readBuffer, sizeof(readBuffer), &ol2, (LPOVERLAPPED\_COMPLETION\_ROUTINE)ReadFunc);      //SleepEx(1000, true);      WaitForSingleObjectEx(hFile, INFINITE, true);      if (!ret) {          DWORD err = GetLastError();          if (ERROR\_IO\_PENDING == err) {              printf("文件读取中！！！\n");          } else {              printf("文件读取失败！！！\n");              CloseHandle(hFile);              return 0;          }      }        CloseHandle(hFile);        return 1;  } |

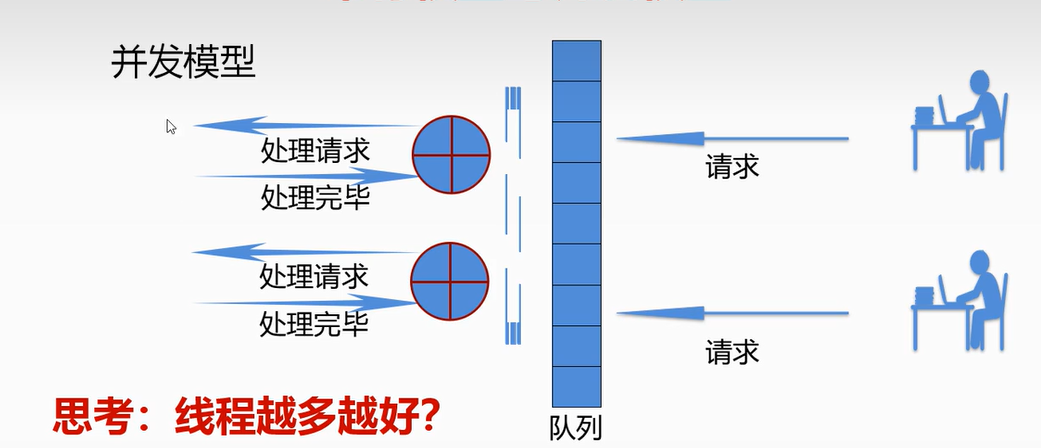
### 4.使用IO完成端口

这种方法允许我们向一个设备同时发出多个IO请求。它允许一个线程发出IO请求，另一个线程对结果进行处理，推荐使用，伸缩性和灵活性都很好，IO完成端口的初衷就是与线程池配合使用  
示例代码：

|  |  |
| --- | --- |
|  | #include <Windows.h>  #include <stdio.h>    int main() {      char userName[256] = { 0 };      HANDLE hFile = CreateFile(TEXT("test.txt"), GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED, NULL);      if (INVALID\_HANDLE\_VALUE == hFile) {          printf("文件创建失败，错误码：%d\n", GetLastError());            return 1;      }      HANDLE hCicp = CreateIoCompletionPort(INVALID\_HANDLE\_VALUE, NULL, 0, 0);      if (NULL == hCicp) {          printf("创建CreateIoCompletionPort失败，错误码：%d\n", GetLastError());          CloseHandle(hFile);            return 1;      }      ULONG\_PTR CK\_READ = 0;      CreateIoCompletionPort(hFile, hCicp, CK\_READ, 0);        OVERLAPPED ol = { 0 };      ReadFile(hFile, userName, 256, NULL, &ol);        DWORD transferedByte = 0;      void\* lpContext = NULL;      OVERLAPPED\* pOl = NULL;      while (GetQueuedCompletionStatus(hCicp, &transferedByte, (LPDWORD)&lpContext, &pOl, INFINITE)) {          printf("%s\n", userName);      }        CloseHandle(hFile);      CloseHandle(hCicp);        return 0;  } |

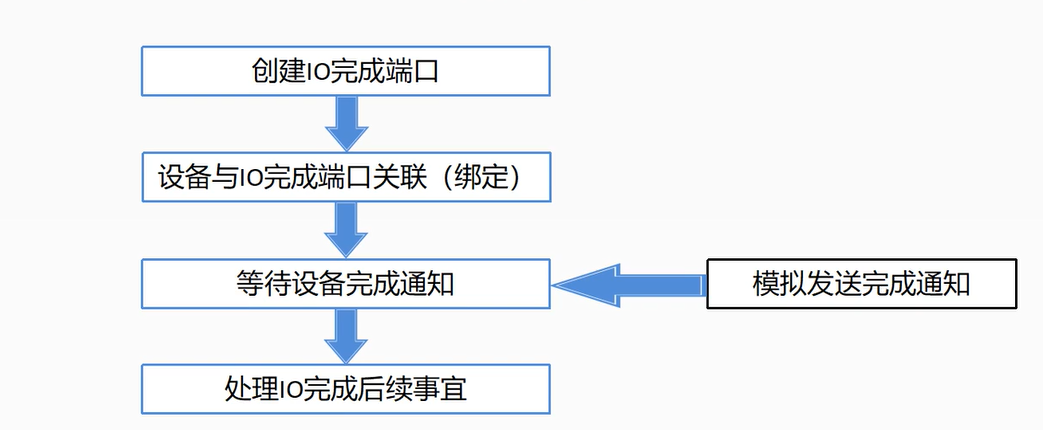
## 五、串行模型与并发模型

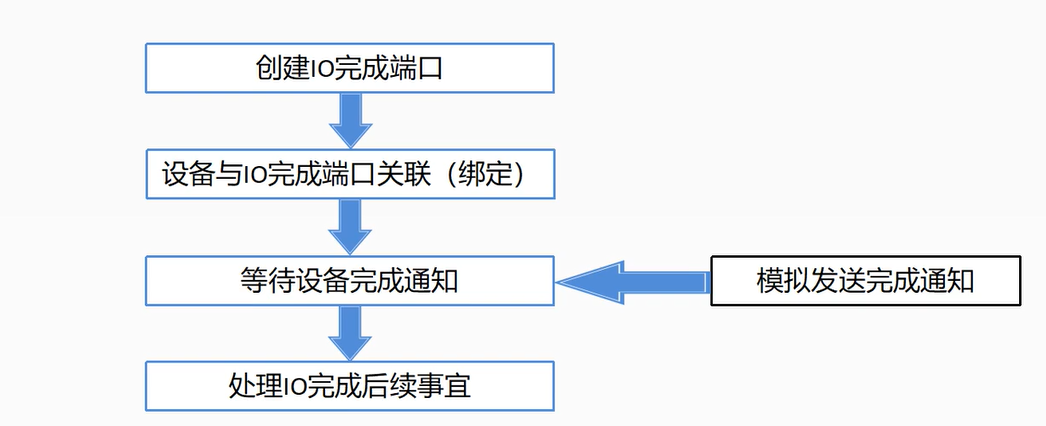




CPU是通过时间片轮转的方式执行线程，比如说，A线程执行一段时间，然后切换到B线程，B线程执行一段时间，再切换到C线程，C线程执行一段时间，然后再切换到A线程；当CPU执行的线程过多时，可以发现，CPU处理排队线程的请求远远小于线程之间来回切换的时间，这个时候CPU貌似十分繁忙(忙着切换线程，没有去处理线程请求)

### 1.IO完成端口





#### (1).CreateIoCompletionPort函数

CreateIoCompletionPort 函数是 Windows API 中的一个函数，用于创建输入/输出完成端口（I/O Completion Port），这是一种高效的异步 I/O 操作管理机制，常用于多线程的异步 I/O 编程。

函数原型：

|  |  |
| --- | --- |
|  | HANDLE CreateIoCompletionPort(  HANDLE FileHandle,  HANDLE ExistingCompletionPort,  ULONG\_PTR CompletionKey,  DWORD NumberOfConcurrentThreads  ); |

参数解释：

* ExistingCompletionPort：可选参数，如果要将文件句柄关联到现有的 I/O 完成端口，则传递现有的 I/O 完成端口句柄，否则传递 NULL。
* FileHandle：要与 I/O 完成端口关联的文件句柄。
* CompletionKey：关联的完成键，是一个用户定义的值，通常用于标识关联的文件句柄。
* NumberOfConcurrentThreads：指定可以同时执行 I/O 操作的线程数目。

返回值：

如果函数调用成功，将返回新创建的 I/O 完成端口的句柄（HANDLE）。如果函数失败，则返回 NULL，您可以通过调用 GetLastError 获取错误信息。

#### (2).GetQueuedCompletionStatuse函数

GetQueuedCompletionStatus 函数是 Windows API 中用于从 I/O 完成端口 (I/O Completion Port) 中获取已完成的 I/O 操作的函数。它通常用于与异步 I/O 操作相关的多线程编程，以便检查和处理已完成的 I/O 操作结果。

函数原型：

|  |  |
| --- | --- |
|  | BOOL GetQueuedCompletionStatus(    HANDLE       CompletionPort,    LPDWORD      lpNumberOfBytesTransferred,    PULONG\_PTR   lpCompletionKey,    LPOVERLAPPED \*lpOverlapped,    DWORD        dwMilliseconds  ); |

 参数解释：

* CompletionPort：要从中获取已完成 I/O 操作的 I/O 完成端口句柄。
* lpNumberOfBytesTransferred：用于接收已传输字节数的指针。如果不关心传输的字节数，可以传递 NULL。
* lpCompletionKey：用于接收已完成 I/O 操作关联的完成键的指针。完成键通常用于标识 I/O 操作的类型或源。
* lpOverlapped：用于接收指向 OVERLAPPED 结构的指针，该结构与已完成的 I/O 操作相关联。如果不关心此参数，可以传递 NULL。
* dwMilliseconds：等待时间，以毫秒为单位。如果没有已完成的 I/O 操作，函数将阻塞等待指定的时间。如果传递零，函数将立即返回，如果传递 INFINITE，函数将无限期地等待。

返回值：

如果函数调用成功并且获取了已完成的 I/O 操作，它将返回 TRUE。如果函数调用失败或等待超时，它将返回 FALSE。您可以通过调用 GetLastError 获取错误信息。

#### (3).PostQueuedCompletionStatus函数

PostQueuedCompletionStatus 函数是 Windows API 中用于将已完成的 I/O 操作结果（或其他自定义完成状态）添加到 I/O 完成端口 (I/O Completion Port) 队列的函数。它通常用于多线程编程中，用于将异步操作的结果通知给使用 I/O 完成端口的线程。

函数原型：

|  |  |
| --- | --- |
|  | BOOL PostQueuedCompletionStatus(    HANDLE       CompletionPort,    DWORD        dwNumberOfBytesTransferred,    ULONG\_PTR    dwCompletionKey,    LPOVERLAPPED lpOverlapped  ); |

参数解释：

* CompletionPort：要将完成状态添加到的 I/O 完成端口句柄。
* dwNumberOfBytesTransferred：指定已传输的字节数。通常用于通知 I/O 完成端口关于 I/O 操作的结果。
* dwCompletionKey：指定完成状态的关键字，通常用于标识 I/O 操作的类型或源。
* lpOverlapped：指向 OVERLAPPED 结构的指针，通常用于关联已完成的 I/O 操作。可以为 NULL。

返回值：

如果函数调用成功，将返回 TRUE。如果函数调用失败，将返回 FALSE。您可以通过调用 GetLastError 获取错误信息。

示例代码：

|  |  |
| --- | --- |
|  | #include <Windows.h>  #include <stdio.h>  #include <process.h>    HANDLE hCicp = NULL;  unsigned int ThreadFunc(void\* arg) {      getchar();      ULONG\_PTR key = 10;      OVERLAPPED ol = { 0 };      PostQueuedCompletionStatus(hCicp, 4, 10, &ol);        return 0;  }    int main() {      char userName[256] = { 0 };      HANDLE hFile = CreateFile(TEXT("test.txt"), GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED, NULL);      if (INVALID\_HANDLE\_VALUE == hFile) {          printf("文件创建失败，错误码：%d\n", GetLastError());            return 1;      }      hCicp = CreateIoCompletionPort(INVALID\_HANDLE\_VALUE, NULL, 0, 0);      if (NULL == hCicp) {          printf("创建CreateIoCompletionPort失败，错误码：%d\n", GetLastError());          CloseHandle(hFile);            return 1;      }      ULONG\_PTR CK\_READ = 0;      CreateIoCompletionPort(hFile, hCicp, CK\_READ, 0);      unsigned int uiThreadID = 0;      HANDLE hThread =  (HANDLE)\_beginthreadex(NULL, 0, (\_beginthreadex\_proc\_type)ThreadFunc, NULL, 0, &uiThreadID);        OVERLAPPED ol = { 0 };      ReadFile(hFile, userName, 256, NULL, &ol);        DWORD transferedByte = 0;      void\* lpContext = NULL;      OVERLAPPED\* pOl = NULL;      while (GetQueuedCompletionStatus(hCicp, &transferedByte, (LPDWORD)&lpContext, &pOl, INFINITE)) {          if (lpContext != NULL && 10 == (unsigned long)lpContext) {              printf("IO端口完成退出！\n");              break;          }            printf("%s\n", userName);      }      CloseHandle(hThread);      CloseHandle(hFile);      CloseHandle(hCicp);        return 0;  } |

 参考资料：

例子:异步读: Opening a File for Reading or Writing

<https://msdn.microsoft.com/en-us/library/windows/desktop/bb540534(v=vs.85).aspx>

第六章 Overlapped I/O，在你身后变戏法(1)

<https://blog.csdn.net/yylooker2008/article/details/5668756>

# 完整代码

## overlappedIodemo.cpp

|  |
| --- |
| // overlappedIodemo.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。  //  #include<Windows.h>  #include <iostream>  void OverlappedWrite(LPCTSTR szFileName)  {  HANDLE hFile;  hFile = CreateFile(szFileName, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  CREATE\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);//异步方式这两个标记不能少  /\* hFile = CreateFile(szFileName, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);\*/ //同步方式  if (INVALID\_HANDLE\_VALUE == hFile)  {  wprintf\_s(L"Create File Failed:%d\n", GetLastError());  return;  }  OVERLAPPED ovl = { 0 };  DWORD dwWritten = 0;  WCHAR buf[] = L"Hello Overlapped File Demo";  DWORD lenToRead = lstrlen(L"Hello Overlapped File Demo") \* sizeof(WCHAR);  WriteFile(hFile, buf, lenToRead, &dwWritten, &ovl);  //if (dwWritten!=lenToRead)  //{  // CloseHandle(hFile);  // //wprintf\_s(L"Write File Failed!\n");  // wprintf\_s(L"lenToRead=%d,dwWritten=%d\n", lenToRead, dwWritten);  // return ;  //}  //注意:以异步方式写文件, writeCount根本得不到数据,因为可能还没有等到IO完成函数已经返回了.  //所以上面的判断永远都是FALSE  CloseHandle(hFile);  }  void OverlappedWrite2(LPCTSTR szFileName)  {  HANDLE hFile;  hFile = CreateFile(szFileName, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  CREATE\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);//异步方式这两个标记不能少  /\* hFile = CreateFile(szFileName, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);\*/ //同步方式  if (INVALID\_HANDLE\_VALUE == hFile)  {  wprintf\_s(L"Create File Failed:%d\n", GetLastError());  return;  }  OVERLAPPED ovl = { 0 };  DWORD dwWritten = 0;  BOOL ret;  WCHAR buf[] = L"Hello Overlapped File Demo Method2";  DWORD lenToRead = lstrlen(L"Hello Overlapped File Demo Method2") \* sizeof(WCHAR);  ret = WriteFile(hFile, buf, lenToRead, &dwWritten, &ovl);  if (!ret)  {  DWORD err = GetLastError();  if (ERROR\_IO\_PENDING == err) {  printf("正在进行异步写入操作!\n");  }  else {  printf("文件写入失败!\n");  CloseHandle(hFile);  return ;  }  }  CloseHandle(hFile);  }  void OverlappedWrite3(LPCTSTR szFileName)  {  HANDLE hFile;  hFile = CreateFile(szFileName, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  CREATE\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);//异步方式这两个标记不能少  /\* hFile = CreateFile(szFileName, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);\*/ //同步方式  if (INVALID\_HANDLE\_VALUE == hFile)  {  wprintf\_s(L"Create File Failed:%d\n", GetLastError());  return;  }  OVERLAPPED ovl = { 0 };  DWORD dwWritten = 0;  BOOL ret;  WCHAR buf[] = L"Hello Overlapped File Demo Method3";  DWORD lenToRead = lstrlen(L"Hello Overlapped File Demo Method3") \* sizeof(WCHAR);  ret = WriteFile(hFile, buf, lenToRead, &dwWritten, &ovl);  if (!ret)  {  DWORD err = GetLastError();  if (ERROR\_IO\_PENDING == err) {  printf("正在进行异步写入操作!\n");  WaitForSingleObject(hFile, INFINITY);  printf("异步写入操作完成!\n");  }  else {  printf("文件写入失败!\n");  CloseHandle(hFile);  return;  }  }  CloseHandle(hFile);  }  void OverlappedRead(LPCTSTR szFileName)  {  HANDLE hFile;  hFile = CreateFile(szFileName, GENERIC\_READ, FILE\_SHARE\_READ, NULL,  OPEN\_EXISTING, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);  WCHAR buf[255];  memset(buf, 0, 255);  OVERLAPPED ovl = { 0 };  DWORD dwRead;  ReadFile(hFile, buf, 255, &dwRead, &ovl);  wprintf\_s(L"Content:%s\n", buf);  CloseHandle(hFile);  }  void OverlappedRead2(LPCTSTR szFileName)  {  HANDLE hFile;  hFile = CreateFile(szFileName, GENERIC\_READ, FILE\_SHARE\_READ, NULL,  OPEN\_EXISTING, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);  WCHAR buf[255];  memset(buf, 0, 255);  OVERLAPPED ovl = { 0 };  DWORD dwRead;  BOOL ret;  ret = ReadFile(hFile, buf, 255, &dwRead, &ovl);  if (!ret)  {  DWORD err = GetLastError();  if (ERROR\_IO\_PENDING == err) {  printf("正在进行异步读取操作!\n");  }  else {  printf("文件读取失败!\n");  CloseHandle(hFile);  return;  }  }  wprintf\_s(L"Content:%s\n", buf);  CloseHandle(hFile);  }  void OverlappedRead3(LPCTSTR szFileName)  {  HANDLE hFile;  hFile = CreateFile(szFileName, GENERIC\_READ, FILE\_SHARE\_READ, NULL,  OPEN\_EXISTING, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);  WCHAR buf[255];  memset(buf, 0, 255);  OVERLAPPED ovl = { 0 };  DWORD dwRead;  BOOL ret;  ret = ReadFile(hFile, buf, 255, &dwRead, &ovl);  if (!ret)  {  DWORD err = GetLastError();  if (ERROR\_IO\_PENDING == err) {  printf("正在进行异步读取操作!\n");  WaitForSingleObject(hFile, INFINITY);  printf("异步读取操作完成!\n");  }  else {  printf("文件读取失败!\n");  CloseHandle(hFile);  return;  }  }  wprintf\_s(L"Content:%s\n", buf);  CloseHandle(hFile);  }  void WaitForMultipleObjectsFileOp(LPCTSTR szFileName)  {  HANDLE hFile;  OVERLAPPED ovlw = { 0 };  ovlw.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);  DWORD dwWritten = 0;  WCHAR bufRead[255];  memset(bufRead, 0, 255);  OVERLAPPED ovlr = { 0 };  ovlr.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);  DWORD dwRead=0;  BOOL ret;  WCHAR buf[] = L"Hello Overlapped File Read Write";  hFile = CreateFile(szFileName, GENERIC\_READ| GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,  OPEN\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);  ret = WriteFile(hFile, buf, lstrlen(buf) \* sizeof(WCHAR), &dwWritten, &ovlw);  ret = ReadFile(hFile, bufRead, 255, &dwRead, &ovlr);  if (!ret)  {  DWORD err = GetLastError();  if (err == ERROR\_IO\_PENDING)  {  HANDLE handle[2];  handle[0] = ovlw.hEvent;  handle[1] = ovlr.hEvent;  DWORD objs = WaitForMultipleObjects(2, handle, TRUE, INFINITY);  switch (objs)  {  case WAIT\_OBJECT\_0:  printf("异步写入操作完成!\n");  break;  case WAIT\_OBJECT\_0+1:  printf("异步读取操作完成,内容:%s\n",bufRead);  break;  default:  break;  }  CloseHandle(hFile);  }  else  {  printf("文件操失败!\n");  }  }  CloseHandle(hFile);  CloseHandle(ovlw.hEvent);  CloseHandle(ovlr.hEvent);  }  int main()  {  //OverlappedWrite(L"hello.txt");  //OverlappedRead(L"hello.txt");  //OverlappedWrite2(L"hello.txt");  //OverlappedRead2(L"hello.txt");  //OverlappedWrite3(L"demo.txt");  //OverlappedRead3(L"demo.txt");  WaitForMultipleObjectsFileOp(L"demo.txt");  system("pause");  return 0;  }  // 运行程序: Ctrl + F5 或调试 >“开始执行(不调试)”菜单  // 调试程序: F5 或调试 >“开始调试”菜单  // 入门使用技巧:  // 1. 使用解决方案资源管理器窗口添加/管理文件  // 2. 使用团队资源管理器窗口连接到源代码管理  // 3. 使用输出窗口查看生成输出和其他消息  // 4. 使用错误列表窗口查看错误  // 5. 转到“项目”>“添加新项”以创建新的代码文件，或转到“项目”>“添加现有项”以将现有代码文件添加到项目  // 6. 将来，若要再次打开此项目，请转到“文件”>“打开”>“项目”并选择 .sln 文件 |

## overlappedandnotifydemo.cpp

|  |
| --- |
| // overlappedandnotifydemo.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。  //  #include<Windows.h>  #include <iostream>  #include<locale.h>  WCHAR readBuffer[256] = { 0 };  WCHAR writeBuffer[256] = L"Overlapped write with notification";  VOID WriteFunc(DWORD dwErrorCode, DWORD dwNumberOfBytesTransfered, LPOVERLAPPED lpOverlapped) {  wprintf\_s(L"文件写入结束\n");  }  VOID ReadFunc(DWORD dwErrorCode, DWORD dwNumberOfBytesTransfered, LPOVERLAPPED lpOverlapped) {  wprintf\_s(L"%s\n", readBuffer);  }  void WriteWithNotify(LPCTSTR szFileName)  {  HANDLE hFile;  hFile = CreateFile(szFileName, GENERIC\_WRITE, FILE\_SHARE\_READ, 0,  CREATE\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);  if (INVALID\_HANDLE\_VALUE == hFile)  {  wprintf\_s(L"Create File Failed:%d", GetLastError());  return;  }  OVERLAPPED ovl = { 0 };  BOOL ret = WriteFileEx(hFile, writeBuffer, lstrlen(writeBuffer) \* sizeof(WCHAR), &ovl, (LPOVERLAPPED\_COMPLETION\_ROUTINE)WriteFunc);  SleepEx(1000, true);  if (!ret)  {  DWORD err = GetLastError();  if (ERROR\_IO\_PENDING == err)  {  wprintf\_s(L"文件写入中！！！\n");  }  else  {  wprintf\_s(L"文件写入失败！！！\n");  CloseHandle(hFile);  }  }    CloseHandle(hFile);  }  void ReadWithNotify(LPCTSTR szFileName)  {  HANDLE hFile;  hFile = CreateFile(szFileName, GENERIC\_READ, FILE\_SHARE\_READ, 0,  OPEN\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);  if (INVALID\_HANDLE\_VALUE == hFile)  {  wprintf\_s(L"Create File Failed:%d", GetLastError());  return;  }  OVERLAPPED ovl = { 0 };  BOOL ret = ReadFileEx(hFile, readBuffer, sizeof(readBuffer), &ovl, (LPOVERLAPPED\_COMPLETION\_ROUTINE)ReadFunc);  WaitForSingleObjectEx(hFile, INFINITE, true);  if (!ret)  {  DWORD err = GetLastError();  if (ERROR\_IO\_PENDING == err)  {  wprintf\_s(L"文件读取中！！！\n");  }  else  {  wprintf\_s(L"文件读取失败！！！\n");  CloseHandle(hFile);  }  }  wprintf\_s(L"%s\n", readBuffer);  CloseHandle(hFile);  }  int main()  {  setlocale(LC\_ALL, "");  //WriteWithNotify(L"Test.txt");  ReadWithNotify(L"Test.txt");  system("pause");  return 0;  }  // 运行程序: Ctrl + F5 或调试 >“开始执行(不调试)”菜单  // 调试程序: F5 或调试 >“开始调试”菜单  // 入门使用技巧:  // 1. 使用解决方案资源管理器窗口添加/管理文件  // 2. 使用团队资源管理器窗口连接到源代码管理  // 3. 使用输出窗口查看生成输出和其他消息  // 4. 使用错误列表窗口查看错误  // 5. 转到“项目”>“添加新项”以创建新的代码文件，或转到“项目”>“添加现有项”以将现有代码文件添加到项目  // 6. 将来，若要再次打开此项目，请转到“文件”>“打开”>“项目”并选择 .sln 文件 |

## overlapwithiocompletionport.cpp

|  |
| --- |
| // overlapwithiocompletionport.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。  //  #include<Windows.h>  #include <iostream>  #include<locale.h>  void OverlapReadWithIOCompletionPort(LPCTSTR fileName)  {  WCHAR content[255];  memset(content, 0, 255);  OVERLAPPED ovl = { 0 };  HANDLE hFile, hCicp;  hFile = CreateFile(fileName, GENERIC\_READ, 0, NULL,  OPEN\_EXISTING, FILE\_FLAG\_SEQUENTIAL\_SCAN | FILE\_FLAG\_OVERLAPPED, NULL);    if (INVALID\_HANDLE\_VALUE==hFile)  {  wprintf\_s(L"Open File Failed:%d", GetLastError());  return;  }  hCicp = CreateIoCompletionPort(INVALID\_HANDLE\_VALUE, NULL, 0, 0);  if (NULL == hCicp) {  printf("创建CreateIoCompletionPort失败，错误码：%d\n", GetLastError());  CloseHandle(hFile);  return ;  }  ULONG\_PTR CK\_READ = 0;  DWORD dwRead = 0;  CreateIoCompletionPort(hFile, hCicp, CK\_READ, 0);  ReadFile(hFile, content, 255, &dwRead, &ovl);  DWORD transferedByte = 0;  void\* lpContext = NULL;  OVERLAPPED\* pOl = NULL;  while (GetQueuedCompletionStatus(hCicp, &transferedByte, (LPDWORD)&lpContext, &pOl, INFINITE))  {  printf("%s\n",content);//这里使用wprintf函数会有乱码  }  CloseHandle(hFile);  CloseHandle(hCicp);  }  int main()  {  setlocale(LC\_ALL, "");  OverlapReadWithIOCompletionPort(L"demo.txt");  system("pause");  return 0;  }  // 运行程序: Ctrl + F5 或调试 >“开始执行(不调试)”菜单  // 调试程序: F5 或调试 >“开始调试”菜单  // 入门使用技巧:  // 1. 使用解决方案资源管理器窗口添加/管理文件  // 2. 使用团队资源管理器窗口连接到源代码管理  // 3. 使用输出窗口查看生成输出和其他消息  // 4. 使用错误列表窗口查看错误  // 5. 转到“项目”>“添加新项”以创建新的代码文件，或转到“项目”>“添加现有项”以将现有代码文件添加到项目  // 6. 将来，若要再次打开此项目，请转到“文件”>“打开”>“项目”并选择 .sln 文件 |

## Iocompletionportdemo2.cpp

|  |
| --- |
| #include <Windows.h>  #include <stdio.h>  #include <process.h>  HANDLE hCicp = NULL;  unsigned int ThreadFunc(void\* arg) {  printf("press any key to continue...\n");  getchar();  ULONG\_PTR key = 10;  OVERLAPPED ol = { 0 };  PostQueuedCompletionStatus(hCicp, 4, 10, &ol);  return 0;  }  int main() {  char userName[256] = { 0 };  HANDLE hFile = CreateFile(TEXT("test.txt"), GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED, NULL);  if (INVALID\_HANDLE\_VALUE == hFile) {  printf("文件创建失败，错误码：%d\n", GetLastError());  return 1;  }  hCicp = CreateIoCompletionPort(INVALID\_HANDLE\_VALUE, NULL, 0, 0);  if (NULL == hCicp) {  printf("创建CreateIoCompletionPort失败，错误码：%d\n", GetLastError());  CloseHandle(hFile);  return 1;  }  ULONG\_PTR CK\_READ = 0;  CreateIoCompletionPort(hFile, hCicp, CK\_READ, 0);  unsigned int uiThreadID = 0;  HANDLE hThread = (HANDLE)\_beginthreadex(NULL, 0, (\_beginthreadex\_proc\_type)ThreadFunc, NULL, 0, &uiThreadID);  OVERLAPPED ol = { 0 };  ReadFile(hFile, userName, 256, NULL, &ol);  DWORD transferedByte = 0;  void\* lpContext = NULL;  OVERLAPPED\* pOl = NULL;  while (GetQueuedCompletionStatus(hCicp, &transferedByte, (LPDWORD)&lpContext, &pOl, INFINITE)) {  if (lpContext != NULL && 10 == (unsigned long)lpContext) {  printf("IO端口完成退出！\n");  break;  }  printf("%s\n", userName);  }  CloseHandle(hThread);  CloseHandle(hFile);  CloseHandle(hCicp);  return 0;  } |