Windows操作系统

C/C++ 程序实验

首都师范大学 信息工程学院

系统分析与管理实验室

## 实验四 Windows 线程间通信

### 一、背景知识

Windows 提供的线程间通讯类内核对象允许同一进程或跨进程的线程之间互相发送信息，包括文件、文件映射、邮件位和命名管道等，其中最常用的是文件和文件映射。这类对象允许一个线程很容易地向同一进程或其他进程中的另一线程发送信息。

#### 1. 文件对象

文件对象是人们所熟悉的永久存储的传统元素。将一个文件看作是内核对象可使开发人员获得比标准C++ 文件操作更为强大的功能。

内核允许开发人员在系统设备或网络上创建代表永久存储数据块的文件对象。这些文件对象是对永久存储数据的低级访问者；用C++ 运行库或其他方法打开的所有文件最终都要变成对CreateFile() API的调用。

CreateFile() 函数分配一个内核对象来代表一个永久的文件。当在磁盘上创建一个新文件或当打开一个已经存在的文件时，就调用这个API，其参数总结见表4-1。

**表4-1 CreateFile() API的参数**

|  |  |
| --- | --- |
| **参数名** | **使用目的** |
| LPCTSTR lpFileName | 要打开或创建的文件名 |
| DWORD dwDesiredAccess | 所要求的文件访问权；一个包括GENERIC\_READ或GENERIC\_WRITE的屏蔽 |
| DWORD dwShareMode | 指定与其他进程共享的文件类型 (如果有的话) |
| LPSECURITY\_ATTRIBUTES lpSecurityAttributes | 当被文件系统支持时与备份文件对象有关的安全性 |
| DWORD dwCreationDisposition | 在文件系统的级别上所采取的操作的类型。如新文件的创建或打开一个已有的文件 |
| DWORD dwFlagsAndAttributes | 文件系统的属性，如只读、隐藏等。还可以是文件对象的属性，如可缓存写入等 |
| HANDLE hTemplateFile | 指向另一文件对象的句柄，常用于为新创建的文件提供属性 |

创建调用比创建事件、互斥体或信号量要复杂。首先必须在lpFilename中指定对象名，并且要指向文件系统中所访问的位置。接着必须用dwDesiredAccess参数提供所需的访问级别。

由创建函数要求的共享模式参数dwShareMode可以指定当另一进程企图同时访问数据时会发生什么。与所有其他第一级内核对象一样，可以利用lpSecurityAttributes参数指定所创建对象的安全性。接着，要通过dwCreationDisposition参数告诉创建函数，如果数据在指定的永久存储介质中存在或不存在时的行为。

可以使用dwFlagsAndAttributes参数来指定文件的属性 (如只读) ，并确定对数据所执行的读写操作的行为。最后一个参数hTemplateFile可指定另一个文件对象作为模板，以便为新创建的文件复制属性或扩展属性。

Windows 系统中包括许多文件对象的工具函数，表4-2中总结了处理文件对象时需要使用的API。

**表4-2 文件对象API**

|  |  |
| --- | --- |
| **API名称** | **功能描述** |
| CreateFile() | 创建文件内核对象，用于代表文件系统中新的或已经存在的大量数据 |
| ReadFile() | 从文件系统中的由文件对象句柄引用的文件发送数据。读操作开始于当前文件的指针位置，每读取一个字节，该位置增加 |
| WriteFile() | 从文件系统中的由文件对象句柄引用的文件发送数据。写操作开始于当前文件的指针位置，每写入一个字节，该位置增加 |
| SetFilePointer() | 将文件中的当前文件指针位置移动一个相对或绝对距离 |
| SetEndOfFile() | 将文件的终止记号移动到当前文件指针的位置 |
| LockFile() | 防止其他进程访问传递的文件内的一个区域 |
| GetFileType() | 决定传递的句柄是否引用磁盘文件、控制台或命名的管道 |
| GetFileSizeEx() | 提取64位的文件容量 |
| GetFileTime() | 提取文件创建、最后访问和最近修改的时间 |
| GetFileInformationByHandle() | 用传递来的文件中的详细信息填充BY\_HANDLE\_FILE\_ INFORMATION数据结构 |

通常可以使用ReadFile() 和WriteFile() API在永久存储和应用程序间通过文件对象来移动数据。因为创建调用将对象的大多数复杂性封装起来了，这两个函数只是简单地利用指向要交换数据的文件对象的句柄 (即指向内存内的数据缓存区的指针) ，然后计数移动数据的字节数。除此之外，这两个函数还执行重叠式的输入和输出，由于不会“堵塞”主线程，可用来传送大量的数据。

CreateFile() 方法除了可访问标准的永久文件外，还可访问控制台输入和输出，以及从命名的管道来的数据。

GetFileType() API指明要处理的关键文件句柄的结构。除此之外，内核还提供了GetFileInformationByHandle() 和GetFileSize() 、GetFileTime() API用于获得关键数据的详细情况。其他用于在文件中改变数据的工具函数包括LockFile() 、SetFilePointer() 和SetEndOfFile() API。

除了这些基于句柄的API之外，内核还提供了大量的工具，用于按文件名对文件直接操作。文件对象用完之后，应该用CloseHandle() API加以清除。

#### 2. 文件映射对象

比使用ReadFile() 和WriteFile() API通过文件对象来读取和写入数据更为简单的是，Windows 还提供了一种在文件中处理数据的方法，名为内存映射文件，也称为文件映射。文件映射对象是在虚拟内存中分配的永久或临时文件对象区域 (如果可能的话，可大到整个文件) ，可将其看作是二进制的数据块。使用这类对象，可获得直接在内存中访问文件内容的能力。

文件映射对象提供了强大的扫描文件中数据的能力，而不必移动文件指针。对于多线程的读写操作来说，这一点特别有用，因为每个线程都可能想要把读取指针移动到不同的位置去——为了防止这种情况，就需要使用某种线程同步机制保护文件。

在CreateFileMapping() API中，一个新的文件映射对象需要有一个永久的文件对象 (由CreateFile() 所创建) 。该函数使用标准的安全性和命名参数，还有用于允许操作 (如只读) 的保护标志以及映射的最大容量。随后可根据来自OpenFileMapping() API的其他线程或进程使用该映射——这与事件和互斥体的打开进程是非常类似的。

内存映射文件对象的另一个强大的应用是可请求系统创建一个运行映射的临时文件。该临时文件提供一个临时的区域，用于线程或进程互相发送大量数据，而不必创建或保护磁盘上的文件。利用向创建函数中发送INVALID\_HANDLE\_VALUE来代替真正的文件句柄，就可创建这一临时的内存映射文件；指令内核使用系统页式文件来建立支持映射的最大容量的临时数据区。

为了利用文件映射对象，进程必须将对文件的查看映射到它的内存空间中。也就是说，应该将文件映射对象想象为进程的第一步，在这一步中，当查看实际上允许访问的数据时，附加有共享数据的安全性和命名方式。为了获得指向内存区域的指针需要调用MapViewOfFile() API，此调用使用文件映射对象的句柄作为其主要参数。此外还有所需的访问等级 (如读-写) 和开始查看时文件内的偏移和要查看的容量。该函数返回一个指向进程内的内存的指针，此指针可有多种编程方面的应用 (但不能超过访问权限) 。

当结束文件映射查看时，必须用接受到的指针调用UnmapViewOfFlie() API，然后再根据映射对象调用CloseHandle() API，从而将其清除。

### 二、实验目的

在本实验中，通过对文件和文件映射对象的了解，来加深对Windows 线程同步的理解。

1) 回顾系统进程、线程的有关概念，加深对Windows 线程间通讯的理解；

2) 了解文件和文件映射对象；

3) 通过分析实验程序，了解线程如何通过文件对象发送数据；

4) 了解在进程中如何使用文件对象；

5) 通过分析实验程序，了解线程如何通过文件映射对象发送数据；

6) 了解在进程中如何使用文件映射对象。

### 三、工具/准备工作

在开始本实验之前，请回顾教科书的相关内容。

您需要做以下准备：

1) 一台运行Windows 操作系统的计算机。

2) 计算机中需安装Visual Studio Code。

### 四、实验内容

#### 1. 文件对象

清单4-1中的代码展示了线程如何通过文件对象在永久存储介质上互相发送数据。程序只是激活并启动了一个线程接着一个线程的创建进程。每个线程从指定的文件中读取数据，数据的增加是以创建时发送给它的数量进行的，然后将新数值写回文件。

**步骤1**：登录进入Windows 。

**步骤2**：在“开始”菜单中单击“程序”-“Microsoft Visual Studio Code”。

**步骤3：**新建项目名为“4-1”，并且新建项“4-1.cpp”。

**清单4-1 演示线程通过文件对象发送数据**

// fileobj项目

# include <windows.h>

# include <iostream>

// 要使用的文件名

static LPCTSTR g\_szFileName = "w2kdg.Fileobj.file.data.txt" ;

// 在数据文件中读取当前数据的简单线程时将传递来的该数据增加，并写回数据文件中

static DWORD WINAPI ThreadProc (LPVOID lpParam)

{

// 将参数翻译为长整数

LONG nAdd = reinterpret\_cast <LONG> (lpParam) ;

// 建立完全的指定文件名 (包括路径信息)

TCHAR szFullName [MAX\_PATH] ;

:: GetTempPath(MAX\_PATH, szFullName) ; // 取得路径

:: strcat(szFullName, g\_szFileName) ;

// 打开文件对象

HANDLE hFile = :: CreateFile(

szFullName, // 文件的完全名称

GENERIC\_READ | GENERIC\_WRITE, // 具有所有的访问权

FILE\_SHARE\_READ, // 允许其他线程读取

NULL, // 缺省的安全性

OPEN\_ALWAYS, // 创建或打开文件

FILE\_ATTRIBUTE\_NORMAL, // 普通文件

NULL) ; // 无模板文件

if (hFile != INVALID\_HANDLE\_VALUE)

{

// 读取当前数据

LONG nValue(0) ;

DWORD dwXfer(0) ;

:: ReadFile(

hFile, // 要读取的文件

reinterpret\_cast <LPVOID> (&nValue) , // 缓冲区

sizeof(nValue) , // 缓冲区容量

&dwXfer, // 读取的字节数

NULL) ; // 无重叠I/O

if (dwXfer == sizeof(nValue) )

{

// 显示当前数据

std :: cout << "read: " << nValue << std :: endl;

}

// 增加数值

nValue += nAdd;

// 写回永久存储介质

:: SetFilePointer(hFile, 0, NULL, FILE\_BEGIN) ;

:: WriteFile(

hFile, // 要写入的文件

reinterpret\_cast <LPCVOID> (&nValue) , // 数据

sizeof(nValue), // 缓冲区容量

&dwXfer, // 写入的字节数

NULL) ; // 无重叠I/O

if (dwXfer == sizeof(nValue) )

{

std :: cout << "write: "<< nValue << std :: endl;

}

// 清除文件

:: CloseHandle(hFile) ;

hFile = INVALID\_HANDLE\_VALUE;

}

return(0) ;

}

void main()

{

// 创建100个线程从文件中进行读写

for (int nTotal = 100; nTotal > 0; --nTotal)

{

// 启动线程

HANDLE hThread = :: CreateThread(

NULL, // 缺省的安全性

0, // 缺省的堆栈

ThreadProc, // 线程函数

reinterpret\_cast <LPVOID> (1) , // 增量

0, // 无特殊的创建标志

NULL) ; // 忽略线程id

// 等待线程完成

:: WaitForSingleObject(hThread, INFINITE) ;

:: Sleep(500) ; // 放慢显示速度，方便观察

// 释放指向线程的句柄

:: CloseHandle(hThread) ;

hThread = INVALID\_HANDLE\_VALUE;

}

system("pause");

}

**步骤4：**在代码宏定义里添加 #define \_CRT\_SECURE\_NO\_WARNINGS

**步骤5：**按“F5”开始调试，注意路径里不要含有中文。

**步骤6：**按暂停按钮可暂停程序的执行，按终止按钮可终止程序的执行。



运行结果 (如果运行不成功，则可能的原因是什么？) ：

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

阅读和分析程序4-1，请回答问题：

1) 清单4-1中启动了多少个单独的读写线程？

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2) 使用了哪个系统API函数来创建线程例程？

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3) 文件的读和写操作分别使用了哪个API函数？

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

每次运行进程时，都可看到清单4-1中的每个线程从前面的线程中读取数据并将数据增加，文件中的数值连续增加。这个示例是很简单的通讯机制。可将这一示例用作编写自己的文件读/写代码的模板。

请注意程序中写入之前文件指针的重置。重置文件指针是必要的，因为该指针在读取结束时将处于前四个字节之后，同一指针还要用于向文件写入数据。如果函数向该处写入新数值，则下次进程运行时，只能读到原来的数值。那么：

4) 在程序中，重置文件指针使用了哪一个函数？

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5) 从步骤6的输出结果，对照分析4-1程序，可以看出程序运行的流程吗？请予以描述：

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

选作1: 再次执行该程序,结果同初次执行有何变化，如何操作可以让运行结果同第一次执行的一样？

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#### 2. 文件映射对象

清单4-2的程序展示了一个在线程间使用的由页式文件支持的文件映射对象，从中可以看出利用内存映射文件比使用驻留在磁盘上的文件对象更为简单。其中的进程还使用了互斥体，以便公平地访问文件映射对象，然后，当每个线程都释放时，程序将文件的视图映射到文件上并增加数据的值。

**步骤7**：新建项目名为“4-2”，并且新建项“4-2.cpp”。

**清单4-2 演示使用映射文件的内存交换数据的线程**

// mappings项目

# include <windows.h>

# include <iostream>

// 仲裁访问的互斥体

static HANDLE g\_hMutexMapping = INVALID\_HANDLE\_VALUE;

// 增加共享内存中的数值的简单线程

static DWORD WINAPI ThreadProc(LPVOID IpParam)

{

// 将参数看作句柄

HANDLE hMapping = reinterpret\_cast <HANDLE> (IpParam) ;

// 等待对文件的访问

:: WaitForSingleObject(g\_hMutexMapping, INFINITE) ;

// 映射视图

LPVOID pFile = :: MapViewOfFile(

hMapping, // 保存文件的对象

FILE\_MAP\_ALL\_ACCESS, // 获得读写权限

0, // 在文件的开头处 (高32位) 开始

0, // ... (低32位)

0) ; // 映射整个文件

if (pFile != NULL)

{

// 将数据看作长整数

LONG \* pnData = reinterpret\_cast <LONG \*> (pFile) ;

// 改动数据

++(\* pnData) ;

// 显示新数值

std :: cout << "thread: " << :: GetCurrentThreadId()

<< "value: "<< (\* pnData) << std :: endl;

// 释放文件视图

:: UnmapViewOfFile(pFile) ;

pFile = NULL;

}

// 释放对文件的访问权

:: ReleaseMutex(g\_hMutexMapping) ;

return(0) ;

}

// 创建共享数据空间

HANDLE MakeSharedFile()

{

// 创建文件映射对象

HANDLE hMapping = :: CreateFileMapping(

INVALID\_HANDLE\_VALUE, // 使用页式文件临时文件

NULL, // 缺省的安全性

PAGE\_READWRITE, // 可读写权

0, // 最大容量 (高32位)

sizeof(LONG) , // ... (低32位)

NULL) ; // 匿名的

if (hMapping != INVALID\_HANDLE\_VALUE)

{

// 在文件映射上创建视图

LPVOID pData = :: MapViewOfFile(

hMapping, // 保存文件的对象

FILE\_MAP\_ALL\_ACCESS, // 获得读写权

0, // 在文件的开头处(高32位)开始

0, // ... (低32位)

0 ) ; // 映射整个文件

if (pData != NULL)

{

:: ZeroMemory(pData, sizeof(LONG) ) ;

}

// 关闭文件视图

:: UnmapViewOfFile(pData) ;

}

return (hMapping) ;

}

void main()

{

// 创建数据文件

HANDLE hMapping = :: MakeSharedFile() ;

// 创建仲裁的互斥体

g\_hMutexMapping = :: CreateMutex(NULL, FALSE, NULL) ;

// 根据文件创建100个线程来读写

for (int nTotal = 100; nTotal > 0 ; --nTotal)

{

// 启动线程

HANDLE hThread = :: CreateThread(

NULL, // 缺省的安全性

0, // 缺省堆栈

ThreadProc, // 线程函数

reinterpret\_cast <LPVOID> (hMapping) , // 增量

0, // 无特殊的创建标志

NULL) ; // 忽略线程id

// 等待最后的线程释放

if (nTotal == 1)

{

std :: cout << "all threads created, waiting..."<< std :: endl;

:: WaitForSingleObject(hThread, INFINITE) ;

}

:: Sleep(500) ; // 放慢显示速度，方便观察

// 释放指向线程的句柄

:: CloseHandle(hThread) ;

hThread = INVALID\_HANDLE\_VALUE;

}

// 关闭对象

:: CloseHandle(hMapping) ;

hMapping = INVALID\_HANDLE\_VALUE;

:: CloseHandle(g\_hMutexMapping) ;

g\_hMutexMapping = INVALID\_HANDLE\_VALUE;

system("pause");

}

**步骤8：**按“F5”开始调试，注意路径里不要含有中文。

**步骤9：**按暂停按钮可暂停程序的执行，按终止按钮可终止程序的执行。



操作能否正常进行？如果不行，则可能的原因是什么？

操作能够正常进行，如果不行，则可能的原因是文件格式错误，代码中含有中文字符，编译器环境变量错误。\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

运行结果 (如果运行不成功，则可能的原因是什么？) ：

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

阅读和分析程序4-2，请回答：

1) 程序中用来创建一个文件映射对象的系统API函数是哪个？

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2) 在文件映射上创建和关闭文件视图分别使用了哪一个系统函数？

a. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3) 对照清单4-2，分析程序运行并填空：

运行时，清单4-2所示程序首先通过 ( ) 函数创建一个小型的文件映射对象 ( ) ，接着，使用系统API函数 ( ) 再创建一个保护其应用的互斥体 ( ) 。然后，应用程序创建100个线程，每个都允许进行同样的进程，即：通过互斥体获得访问权，这个操作是由语句：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

实现的。再通过函数 ( ) 操作将视图映射到文件，将高32位看作有符号整数，将该数值增加 (即命令：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ) ，再将新数值显示在控制台上。每个线程清除文件的视图并在退出之前释放互斥体释放互斥体的语句是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。当线程完成时，应用程序关闭并退出。

选作2：将清单4-2中的语句 :: Sleep(500) ; 删除 (例如在语句前面加上“//”) 后，重新编译运行，结果有变化吗？为什么？

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_