

Windows 操作系统

C/C++ 程序实验

姓名：_____陈展博_____

学号：_____1221001003_____

班级：_____计科一班_____

院系：_____信工_____

_____2024_____年__10__月__30__日

实验四 Windows 线程间通信

一、背景知识

二、实验目的

三、工具/准备工作

四、实验内容

1. 文件对象

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”。

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?)：

```
F:\Junior1\czb_os\4-1\4-1.exe  ×  +  ∨  
  
write: 81  
read: 81  
write: 82  
read: 82  
write: 83  
read: 83  
write: 84  
read: 84  
write: 85  
read: 85  
write: 86  
read: 86  
write: 87  
read: 87  
write: 88  
read: 88  
write: 89  
read: 89  
write: 90  
read: 90  
write: 91  
read: 91  
write: 92  
read: 92  
write: 93  
read: 93  
write: 94  
read: 94  
write: 95  
read: 95  
write: 96  
read: 96  
write: 97  
read: 97  
write: 98  
read: 98  
write: 99  
read: 99  
write: 100  
请按任意键继续. . .
```

图片 1 运行结果如下

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？

清单 4-1 中启动了 100 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？

使用了 `CreateThread` 的系统 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？

读文件用了 `:: ReadFile` 函数，写文件用了 `:: WriteFile` 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？

使用了 `:: SetFilePointer(hFile, 0, NULL, FILE_BEGIN)` ;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？请简单描述：

该程序在主函数中借助 100 轮 `for` 循环创建 100 个线程从文件中进行读写，调用 `CreateThread` 的 API，并在其参数中调用 `ThreadProc` 线程函数，在线程函数中在 `Temp` 临时文件目录下生成临时文件 `"w2kdg.Fileobj.file.data.txt"`，让所有线程都具有对该临时文件的读写权限，从文件中读取数值 `nValue`，每次加增量 1，再通过重置文件指针，将修改后的 `nValue` 写回临时文件，最后 100 个线程的增量为 100，通过查看临时文件我们可以发现临时文件中有字符 `"d"`，恰好为 ASCII 码 100。

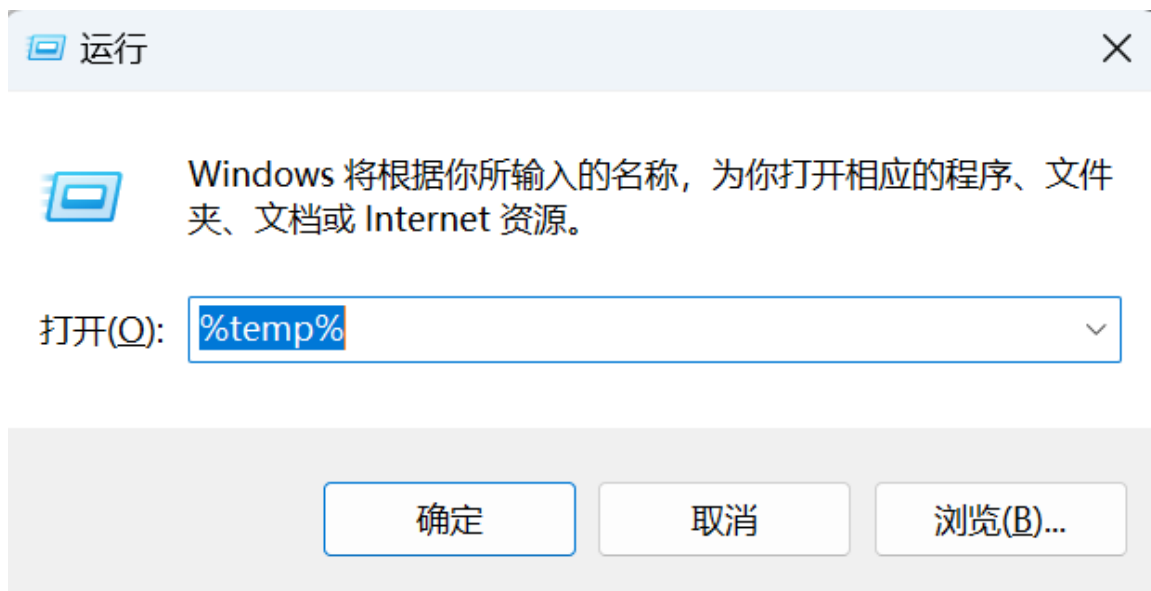


图片 2 临时文件内容

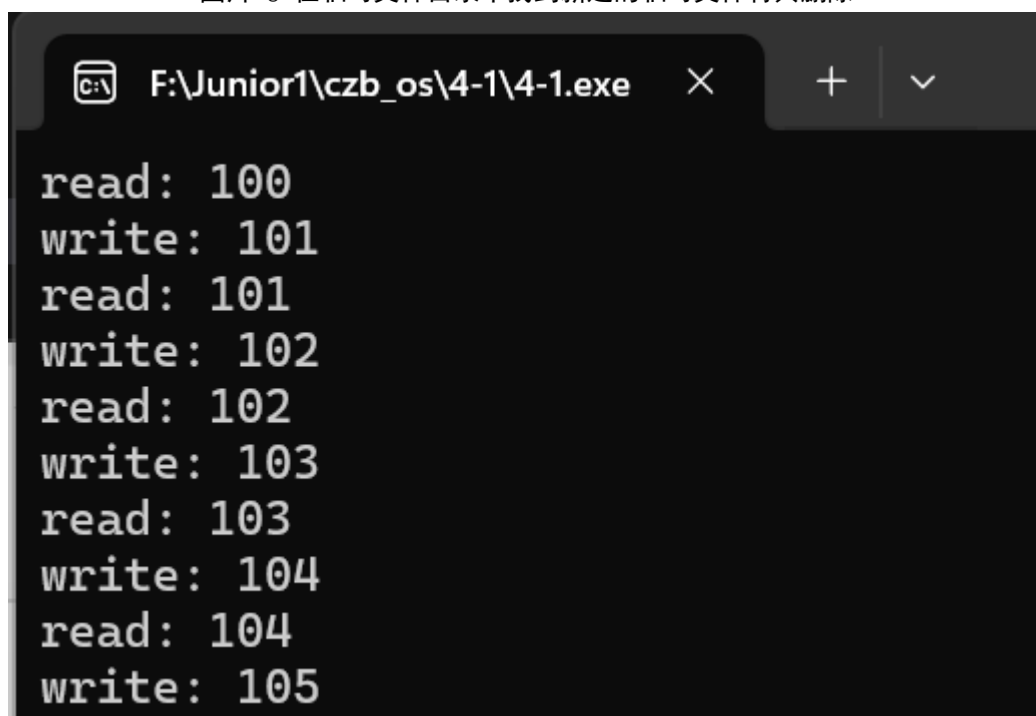
选作 1: 再次执行该程序,结果同初次执行有何变化，如何操作可以让运行结果同第一次执行的

一样？

再次执行该程序,结果会在初次执行的最终结果 `nValue==100` 上继续自增,我们可以通过删除临时文件让他同第一次执行的结果一样。



图片 3 在临时文件目录中找到新建的临时文件将其删除



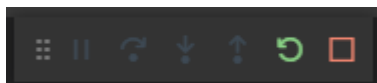
图片 4 再次执行结果

2. 文件映射对象

步骤 7: 新建项目名为“4-2”，并且新建项“4-2.cpp”。

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?)：

```
F:\Junior1\czb_os\4-2\4-2.exe X + v
thread: 12316value: 63
thread: 12044value: 64
thread: 6948value: 65
thread: 18012value: 66
thread: 4580value: 67
thread: 12160value: 68
thread: 6304value: 69
thread: 13016value: 70
thread: 18508value: 71
thread: 1960value: 72
thread: 10296value: 73
thread: 18904value: 74
thread: 17900value: 75
thread: 7960value: 76
thread: 16984value: 77
thread: 15156value: 78
thread: 3524value: 79
thread: 13248value: 80
thread: 21952value: 81
thread: 2004value: 82
thread: 11188value: 83
thread: 2276value: 84
thread: 10396value: 85
thread: 16876value: 86
thread: 3396value: 87
thread: 2676value: 88
thread: 20324value: 89
thread: 5488value: 90
thread: 2236value: 91
thread: 4604value: 92
thread: 18292value: 93
thread: 1572value: 94
thread: 14840value: 95
thread: 17868value: 96
thread: 14224value: 97
thread: 22412value: 98
thread: 8316value: 99
all threads created, waiting...
thread: 968value: 100
请按任意键继续. . . |
```

图片 5 运行结果如下

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？

用来创建一个文件映射对象的系统 API 函数是 `:: CreateFileMapping`。

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. 在文件映射上创建文件视图 `:: MapViewOfFile` 系统函数

b. 在文件映射上关闭文件视图 `:: UnmapViewOfFile` 系统函数

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 `(MakeSharedFile)` 函数创建一个小型的文件映射对象 (`hMapping`)，接着，使用系统 API 函数 `(CreateMutex)` 再创建一个保护其应用的互斥体 (`g_hMutexMapping`)。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：

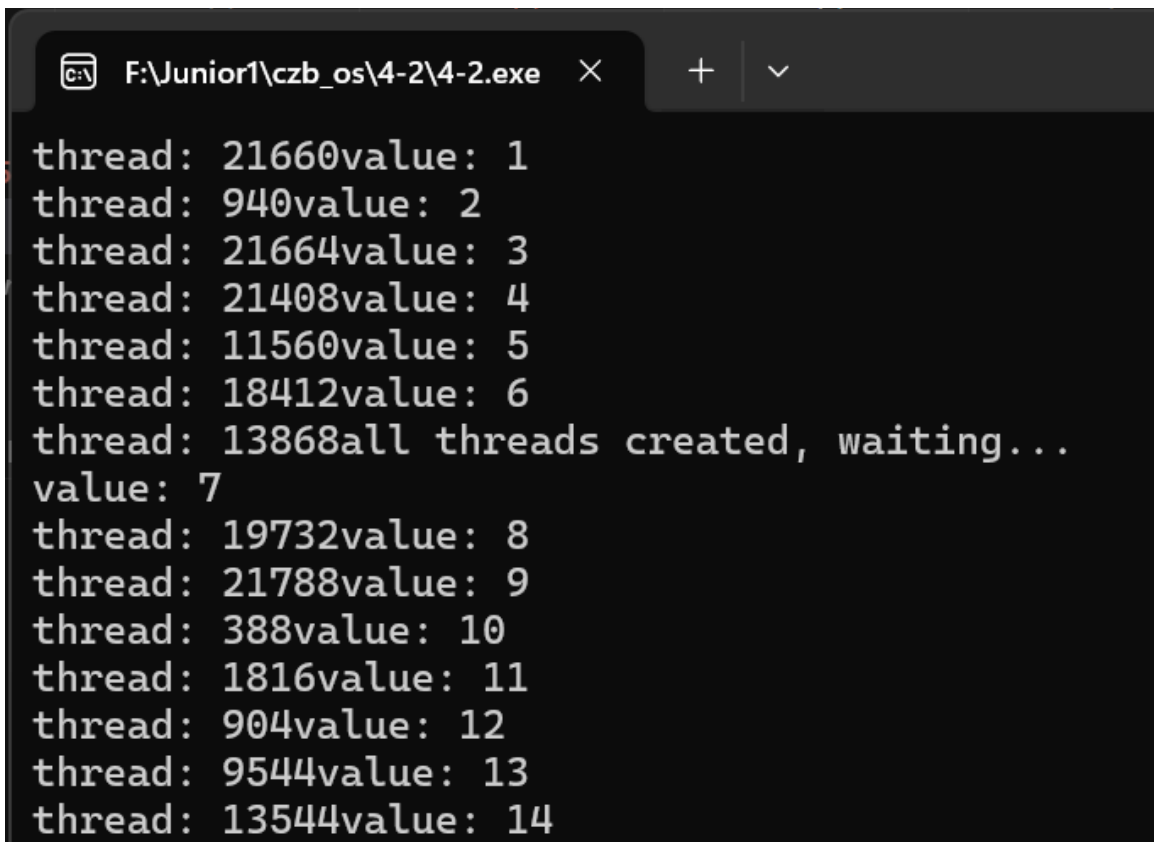
```
:: WaitForSingleObject(g_hMutexMapping, INFINITE);
```

实现的。再通过函数 `(:: MapViewOfFile)` 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：`++(*pnData);`），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

```
:: ReleaseMutex(g_hMutexMapping);
```

。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？



```
thread: 21660value: 1
thread: 940value: 2
thread: 21664value: 3
thread: 21408value: 4
thread: 11560value: 5
thread: 18412value: 6
thread: 13868all threads created, waiting...
value: 7
thread: 19732value: 8
thread: 21788value: 9
thread: 388value: 10
thread: 1816value: 11
thread: 904value: 12
thread: 9544value: 13
thread: 13544value: 14
```

图片 6 如图所示

all threads created, waiting...语句会在所有线程控制台输出结束之前提前输出。

因为在主程序中 `std :: cout << "all threads created, waiting..."<< std :: endl;`语句和新建的线程中的代码并行运行，在所有线程创建完之后，每个线程的线程函数还没有运行完，但主程序中的控制台输出正常输出，就会在 thread 控制台输出完之前输出。