C++使用内核对象互斥体（Mutex）来实现线程同步锁。当两个或更多线程需要同时访问一个共享资源时，Mutex可以只向一个线程授予对共享资源的独占访问权。如果一个线程获取了互斥体，则要获取该互斥体的第二个线程将被挂起，直到第一个线程释放该互斥体。

**1. CreateMutex()**

[CreateMutexA function (synchapi.h) - Win32 apps | Microsoft Docs](https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createmutexa)

HANDLE CreateMutexA(

LPSECURITY\_ATTRIBUTES lpMutexAttributes,

BOOL bInitialOwner,

LPCSTR lpName

);

**2. ReleaseMutex()**

[ReleaseMutex function (synchapi.h) - Win32 apps | Microsoft Docs](https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-releasemutex)

BOOL ReleaseMutex(

HANDLE hMutex

);

**3. WaitForSingleobject()**

[WaitForSingleObject function (synchapi.h) - Win32 apps | Microsoft Docs](https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject)

DWORD WaitForSingleObject(

HANDLE hHandle,

DWORD dwMilliseconds

);

该函数需要传递一个内核对象句柄，如果该内核对象处于未通知状态，则该函数导致线程进入阻塞状态；如果该内核对象处于已通知状态，则该函数立即返回 WAIT\_OBJECT()第二个参数指明要等待的时间（毫秒），INFINITE表示无限等待，如果第二个参数为0，那么函数立即返回。如果等待超时，该函数返 WAIT\_TIMEOUT如果该函数失败，返回 WAIT\_FAILED。

该函数需要传递一个内核对象句柄，如果该内核对象处于未通知状态，则该函数导致线程进入阻塞状态；如果该內核对象处于已通知状态，则该函数立即返回 WAIT\_OBJECT()。第二个数指明要等待的时间（毫秒），INFINITE表示无限等待，如果第二个参数为0，那么函数立即返回。如果等待超时，该函数返 WAIT\_TIMEOUT。  
如果该函数失败，返回 WAIT\_FAILED

**4. CloseHandle()**

[CloseHandle function (handleapi.h) - Win32 apps | Microsoft Docs](https://docs.microsoft.com/en-us/windows/win32/api/handleapi/nf-handleapi-closehandle)

BOOL CloseHandle(

HANDLE hObject

);

**5. 示例代码**

#include<process.h>

#include<windows.h>

#include<stdio.h>

void \_\_cdecl SellThread1(void\* param);

void \_\_cdecl SellThread2(void\* param);

//100张票

int tickets = 100;

HANDLE hMutex = INVALID\_HANDLE\_VALUE;

int main()

{

//创建互斥体，此刻为有信号状态

hMutex = CreateMutex(NULL, FALSE, L"售票互斥体");

printf("开始卖票了！\n");

//创建两个售票窗口

uintptr\_t t1 = \_beginthread(SellThread1, 0, "售口窗口A");

uintptr\_t t2 = \_beginthread(SellThread2, 0, "售口窗口B");

//无限等待两个线程全部执行完毕

HANDLE hArr[] = { (HANDLE)t1, (HANDLE)t2 };

WaitForMultipleObjects(2, hArr, true, INFINITE);

printf("卖票结束！\n");

return 0;

}

void \_\_cdecl SellThread1(void\* param)

{

char \*name = (char \*)param;

while (tickets>0)

{

//如果这个互斥体为有信号状态（没有线程拥有它），则线程获取它后继续执行

WaitForSingleObject(hMutex, INFINITE);

if (tickets > 0)

{

Sleep(10);

//CPU恰好执行到这里，这个时候线程时间片到了，并且此时还剩最后一张票

printf("%s卖出第%d张票！\n", name, tickets--);

}

//释放对互斥体的拥有权，它变成有信号状态

ReleaseMutex(hMutex);

}

}

void \_\_cdecl SellThread2(void\* param)

{

char \*name = (char \*)param;

while (tickets > 0)

{

//如果这个互斥体为有信号状态（没有线程拥有它），则线程获取它后继续执行

WaitForSingleObject(hMutex, INFINITE);

if (tickets > 0)

{

Sleep(10);

//CPU恰好执行到这里，这个时候线程时间片到了，并且此时还剩最后一张票

printf("%s卖出第%d张票！\n", name, tickets--);

}

//释放对互斥体的拥有权，它变成有信号状态

ReleaseMutex(hMutex);

}

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**6. Mutex实现一个程序只允许允许一个实例（进程）**

#include<process.h>

#include<windows.h>

#include<stdio.h>

int main()

{

//创建互斥体实现一个程序只允许允许一个实例（进程）

HANDLE hMutex = CreateMutex(NULL, FALSE, L"售票互斥体");

if (GetLastError() == ERROR\_ALREADY\_EXISTS)

{

printf("程序已经运行了，退出！\n");

getchar();

CloseHandle(hMutex);

return 0;

}

printf("第一次运行程序！\n");

getchar();

return 0;

}