每个线程中访问临界瓷源的那段程序称为临界区（Critical Section）每次只准许一个线程进入临界区，进入后不允许其他线程进入。不论是硬件临界资源，还是软件临界资源，多个线程必须互斥地对它进行访问。

**1. critical\_section 类**

[critical\_section 类 | Microsoft Docs](https://docs.microsoft.com/zh-cn/cpp/parallel/concrt/reference/critical-section-class?view=msvc-160)

**临界区结构对象**

CRITICAL\_SECTION Section

临界区在使用时以 CRITICAL\_SECTION结构对象保护共享资源，如果有多个线程试图同时访问临界区，那么在有个线程进入后、其他试图访问此临界区的线程将被挂起，并一直持续到进入临界区的线程离开。临界区在被释放后，其他线程可以继续抢占，并以此达到用原子方式操作共享资源的目的。

**2. 初始化临界区**

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

void InitializeCriticalSection(

LPCRITICAL\_SECTION lpCriticalSection

);

**3. 进入临界区**

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

void EnterCriticalSection(

LPCRITICAL\_SECTION lpCriticalSection

);

**4. 离开临界区**

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

void LeaveCriticalSection(

LPCRITICAL\_SECTION lpCriticalSection

);

**5. 尝试进入临界区**

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

BOOL TryEnterCriticalSection(

LPCRITICAL\_SECTION lpCriticalSection

);

如果 EnterCriticalsection的代码已经被占用，会使调用线程置于阻塞状态，那么该线程可能很长时间内不能被调度。

而 TryEnterCriticalSection函数决不允许调用线程进入等待状态。它的返回值能够指明调用线程是否能够获得对资源的访问权。TryEntercriticalSection发现该資源已经被另个线程访问，它就返回 FALSE，否则返回TRUE。运用这个函数，线程能够迅速查看它是否可以访问某个共享资源，如果不能访问，那么它可以继续执行某些其他操作，而不必进行等待。

**6. 删除临界区**

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

void DeleteCriticalSection(

LPCRITICAL\_SECTION lpCriticalSection

);

**7. 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;

//1. 临界区结构

CRITICAL\_SECTION Section;

int main()

{

//2.初始化临界区

InitializeCriticalSection(&Section);

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");

//5.删除临界区资源

DeleteCriticalSection(&Section);

return 0;

}

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

{

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

while (tickets>0)

{

//3.进入临界区，禁止其他线程访问

EnterCriticalSection(&Section);

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--);

}

//4.离开临界区

LeaveCriticalSection(&Section);

}

}

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

{

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

while (tickets > 0)

{

//进入临界区，禁止其他线程访问

//EnterCriticalSection(&Section);

//尝试进入临界区，不会阻塞线程

if (TryEnterCriticalSection(&Section))

{

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--);

}

//离开临界区

LeaveCriticalSection(&Section);

}

}

}