EnterCriticalSection function

Waits for ownership of the specified critical section object. The function returns when the calling thread is granted ownership.

Syntax

C++

```
void WINAPI EnterCriticalSection(
  _Inout_ LPCRITICAL_SECTION lpCriticalSection
);
```

Parameters

lpCriticalSection [in, out]
A pointer to the critical section object.

Return value

This function does not return a value.

This function can raise **EXCEPTION_POSSIBLE_DEADLOCK** if a wait operation on the critical section times out. The timeout interval is specified by the following registry value: **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\CriticalSectionTimeout**. Do not handle a possible deadlock exception; instead, debug the application.

Remarks

The threads of a single process can use a critical section object for mutual-exclusion synchronization. The process is responsible for allocating the memory used by a critical section object, which it can do by declaring a variable of type **CRITICAL_SECTION**. Before using a critical section, some thread of the process must call InitializeCriticalSection or InitializeCriticalSectionAndSpinCount to initialize the object.

To enable mutually exclusive access to a shared resource, each thread calls the **EnterCriticalSection** or TryEnterCriticalSection function to request ownership of the critical section before executing any section of code that accesses the protected resource. The difference is that **TryEnterCriticalSection** returns immediately, regardless of whether it obtained ownership of the critical section, while **EnterCriticalSection** blocks until the thread can take ownership of the critical section. When it has finished executing the protected code, the thread uses the LeaveCriticalSection function to relinquish ownership, enabling another thread to become owner and access the protected resource. There is no guarantee about the order in which waiting threads will acquire ownership of the critical section.

After a thread has ownership of a critical section, it can make additional calls to **EnterCriticalSection** or TryEnterCriticalSection without blocking its execution. This prevents a thread from deadlocking itself while waiting for a critical section that it already owns. The thread enters the critical section each time **EnterCriticalSection** and **TryEnterCriticalSection** succeed. A thread must call LeaveCriticalSection once for each time that it entered the critical section.

Any thread of the process can use the DeleteCriticalSection function to release the system resources that were allocated when the critical section object was initialized. After this function has been called, the critical section object can no longer be used for synchronization.

If a thread terminates while it has ownership of a critical section, the state of the critical section is undefined.

If a critical section is deleted while it is still owned, the state of the threads waiting for ownership of the deleted critical section is undefined.

Windows Phone 8: This API is supported.

Windows Phone 8.1: This API is supported.

Examples

For an example that uses **EnterCriticalSection**, see Using Critical Section Objects.

Requirements

Minimum supported client	Windows XP [desktop apps | Windows Store apps]
Minimum supported server	Windows Server 2003 [desktop apps | Windows Store apps]
Header	WinBase.h on Windows XP, Windows Server 2003, Windows Vista, Windows 7, Windows Server 2008, and Windows Server 2008 R2 (include Windows.h); Synchapi.h on Windows 8 and Windows Server 2012
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Critical Section Objects
DeleteCriticalSection
InitializeCriticalSection
InitializeCriticalSectionAndSpinCount
LeaveCriticalSection
Synchronization Functions
TryEnterCriticalSection

Community Additions

NtTerminateProcess may be called during processs shutdown

If a call to EnterCriticalSection that would block is made during process shutdown (for example, when handling the DLL_PROCESS_DETACH notification in a DLL) then the call does not return, but instead NtTerminateProcess is called.


Roger Orr
11/10/2013