

CreateSemaphoreA function (winbase.h)

12/05/2018 • 4 minutes to read

In this article

Syntax

Parameters

Return value

Remarks


Requirements

See also

Creates or opens a named or unnamed semaphore object.

To specify an access mask for the object, use the CreateSemaphoreEx function.

Syntax

C++	 Copy
<pre>HANDLE CreateSemaphoreA(LPSECURITY_ATTRIBUTES lpSemaphoreAttributes, LONG lInitialCount, LONG lMaximumCount, LPCSTR lpName);</pre>	

Parameters

lpSemaphoreAttributes

A pointer to a SECURITY_ATTRIBUTES structure. If this parameter is **NULL**, the handle cannot be inherited by child processes.

The **lpSecurityDescriptor** member of the structure specifies a security descriptor for the new semaphore. If this parameter is **NULL**, the semaphore gets a default security descriptor. The ACLs in the default security descriptor for a semaphore come from the primary or impersonation token of the creator.

`lInitialCount`

The initial count for the semaphore object. This value must be greater than or equal to zero and less than or equal to *lMaximumCount*. The state of a semaphore is signaled when its count is greater than zero and nonsignaled when it is zero. The count is decreased by one whenever a wait function releases a thread that was waiting for the semaphore. The count is increased by a specified amount by calling the `ReleaseSemaphore` function.

`lMaximumCount`

The maximum count for the semaphore object. This value must be greater than zero.

`lpName`

The name of the semaphore object. The name is limited to **MAX_PATH** characters. Name comparison is case sensitive.

If *lpName* matches the name of an existing named semaphore object, this function requests the **SEMAPHORE_ALL_ACCESS** access right. In this case, the *lInitialCount* and *lMaximumCount* parameters are ignored because they have already been set by the creating process. If the *lpSemaphoreAttributes* parameter is not **NULL**, it determines whether the handle can be inherited, but its security-descriptor member is ignored.

If *lpName* is **NULL**, the semaphore object is created without a name.

If *lpName* matches the name of an existing event, mutex, waitable timer, job, or file-mapping object, the function fails and the `GetLastError` function returns **ERROR_INVALID_HANDLE**. This occurs because these objects share the same namespace.

The name can have a "Global" or "Local" prefix to explicitly create the object in the global or session namespace. The remainder of the name can contain any character except the backslash character (\). For more information, see [Kernel Object Namespaces](#). Fast user switching is implemented using Terminal Services sessions. Kernel object names must follow the guidelines outlined for Terminal Services so that applications can support multiple users.

The object can be created in a private namespace. For more information, see [Object Namespaces](#).

Return value

If the function succeeds, the return value is a handle to the semaphore object. If the named semaphore object existed before the function call, the function returns a handle to the existing object and `GetLastError` returns **ERROR_ALREADY_EXISTS**.

If the function fails, the return value is **NULL**. To get extended error information, call `GetLastError`.

Remarks

The handle returned by `CreateSemaphore` has the **SEMAPHORE_ALL_ACCESS** access right; it can be used in any function that requires a handle to a semaphore object, provided that the caller has been granted access. If a semaphore is created from a service or a thread that is impersonating a different user, you can either apply a security descriptor to the semaphore when you create it, or change the default security descriptor for the creating process by changing its default DACL. For more information, see [Synchronization Object Security and Access Rights](#).

The state of a semaphore object is signaled when its count is greater than zero, and nonsignaled when its count is equal to zero. The *InitialCount* parameter specifies the initial count. The count can never be less than zero or greater than the value specified in the *MaximumCount* parameter.

Any thread of the calling process can specify the semaphore-object handle in a call to one of the wait functions. The single-object wait functions return when the state of the specified object is signaled. The multiple-object wait functions can be instructed to return either when any one or when all of the specified objects are signaled. When a wait function returns, the waiting thread is released to continue its execution. Each time a thread completes a wait for a semaphore object, the count of the semaphore object is decremented by one. When the thread has finished, it calls the `ReleaseSemaphore` function, which increments the count of the semaphore object.

Multiple processes can have handles of the same semaphore object, enabling use of the object for interprocess synchronization. The following object-sharing mechanisms are available:

- A child process created by the `CreateProcess` function can inherit a handle to a semaphore object if the *lpSemaphoreAttributes* parameter of `CreateSemaphore` enabled inheritance.
- A process can specify the semaphore-object handle in a call to the `DuplicateHandle` function to create a duplicate handle that can be used by another process.

- A process can specify the name of a semaphore object in a call to the [OpenSemaphore](/windows/win32/api/synchapi/nf-synchapi-opensemaphorew) or CreateSemaphore function.

Use the CloseHandle function to close the handle. The system closes the handle automatically when the process terminates. The semaphore object is destroyed when its last handle has been closed.

Examples

For an example that uses **CreateSemaphore**, see Using Semaphore Objects.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CloseHandle

CreateProcess

CreateSemaphoreEx

DuplicateHandle

Object Names

OpenSemaphore

ReleaseSemaphore

SECURITY_ATTRIBUTES

Semaphore Objects

Synchronization Functions

Is this page helpful?

 Yes  No

Recommended content

Waiting for Multiple Objects - Win32 apps

The following example uses the CreateEvent function to create two event objects and the CreateThread function to create a thread.

CreateSemaphoreW function (synchapi.h) - Win32 apps

Creates or opens a named or unnamed semaphore object.

SleepConditionVariableCS function (synchapi.h) - Win32 apps

Sleeps on the specified condition variable and releases the specified critical section as an atomic operation.

DeleteCriticalSection function (synchapi.h) - Win32 apps

Releases all resources used by an unowned critical section object.

Show more 