# CreateThread Function

Creates a thread to execute within the virtual address space of the calling process.

To create a thread that runs in the virtual address space of another process, use the **CreateRemoteThread** [ http://msdn.microsoft.com/en-us/library/ms682437(VS.85).aspx ] function.

## Syntax

C++

```
HANDLE WINAPI CreateThread(
  __in_opt   LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in       SIZE_T dwStackSize,
  __in       LPTHREAD_START_ROUTINE lpStartAddress,
  __in_opt   LPVOID lpParameter,
  __in       DWORD dwCreationFlags,
  __out_opt  LPDWORD lpThreadId
);
```

## Parameters

*lpThreadAttributes* [in, optional]
A pointer to a **SECURITY_ATTRIBUTES** [ http://msdn.microsoft.com/en-us/library/aa379560(VS.85).aspx ] structure that determines whether the returned handle can be inherited by child processes. If *lpThreadAttributes* is NULL, the handle cannot be inherited.

The **lpSecurityDescriptor** member of the structure specifies a security descriptor for the new thread. If *lpThreadAttributes* is NULL, the thread gets a default security descriptor. The ACLs in the default security descriptor for a thread come from the primary token of the creator.

> **Windows XP/2000:** The ACLs in the default security descriptor for a thread come from the primary or impersonation token of the creator. This behavior changed with Windows XP with SP2 and Windows Server 2003. For more information, see Remarks.

*dwStackSize* [in]
The initial size of the stack, in bytes. The system rounds this value to the nearest page. If this parameter is zero, the new thread uses the default size for the executable. For more information, see Thread Stack Size [ http://msdn.microsoft.com/en-us/library/ms686774(VS.85).aspx ] .

*lpStartAddress* [in]
A pointer to the application-defined function to be executed by the thread. This pointer represents the starting address of the thread. For more information on the thread function, see **ThreadProc** [ http://msdn.microsoft.com/en-us/library/ms686736(VS.85).aspx ] .

*lpParameter* [in, optional]
A pointer to a variable to be passed to the thread.

*dwCreationFlags* [in]
The flags that control the creation of the thread.

| Value | Meaning |
|---|---|
| 0 | The thread runs immediately after creation. |
| CREATE_SUSPENDED<br>0x00000004 | The thread is created in a suspended state, and does not run until the **ResumeThread** [ http://msdn.microsoft.com/en-us/library/ms685086(VS.85).aspx ] function is called. |
| STACK_SIZE_PARAM_IS_A_RESERVATION<br>0x00010000 | The *dwStackSize* parameter specifies the initial reserve size of the stack. If this flag is not specified, *dwStackSize* specifies the commit size.<br><br>**Windows 2000:** The STACK_SIZE_PARAM_IS_A_RESERVATION flag is not supported. |

*lpThreadId* [out, optional]
A pointer to a variable that receives the thread identifier. If this parameter is NULL, the thread identifier is not returned.

## Return Value

If the function succeeds, the return value is a handle to the new thread.

If the function fails, the return value is NULL. To get extended error information, call **GetLastError** [ http://msdn.microsoft.com/en-us/library/ms679360(VS.85).aspx ] .

Note that **CreateThread** may succeed even if *lpStartAddress* points to data, code, or is not accessible. If the start address is invalid when the thread runs, an exception occurs, and the thread terminates. Thread termination due to a invalid start address is handled as an error exit for the thread's process. This behavior is similar to the asynchronous nature of **CreateProcess** [ http://msdn.microsoft.com/en-us/library/ms682425(VS.85).aspx ] , where the process is created even if it refers to invalid or missing dynamic-link libraries (DLLs).

## Remarks

The number of threads a process can create is limited by the available virtual memory. By default, every thread has one megabyte of stack space. Therefore, you can create at most 2,048 threads. If you reduce the default stack size, you can create more threads. However, your application will have better performance if you create one thread per processor and build queues of requests for which the application maintains the context information. A thread would process all requests in a queue before processing requests in the next queue.

The new thread handle is created with the THREAD_ALL_ACCESS access right. If a security descriptor is not provided when the thread is created, a default security descriptor is constructed for the new thread using the primary token of the process that is creating the thread. When a caller attempts to access the thread with the **OpenThread** [ http://msdn.microsoft.com/en-us/library/ms684335(VS.85).aspx ] function, the effective token of the caller is evaluated against this security descriptor to grant or deny access.

> **Windows XP/2000:** If a security descriptor is not provided when the thread is created, a default security descriptor is constructed using the effective token of the thread. If the thread is impersonating another user, the thread's effective token is the impersonation token and the default security descriptor allows access only to the impersonation token's TokenDefaultDacl owner or members. If the thread is not impersonating another user, the thread's effective token is its primary token. This behavior changed starting with Windows XP with SP2 and Windows Server 2003. For more information, see Thread Security and Access Rights [ http://msdn.microsoft.com/en-us/library/ms686769(VS.85).aspx ] .

The newly created thread has full access rights to itself when calling the **GetCurrentThread** [ http://msdn.microsoft.com/en-us/library/ms683182(VS.85).aspx ] function.

> **Windows Server 2003 and Windows XP/2000:** The thread's access rights to itself are computed by evaluating the primary token of the process in which the thread was created against the default security descriptor constructed for the thread. If the thread is created in a remote process, the primary token of the remote process is used. As a result, the newly created thread may have reduced access rights to itself when calling **GetCurrentThread**. Some access rights including THREAD_SET_THREAD_TOKEN and THREAD_GET_CONTEXT may not be present, leading to unexpected failures. For this reason, creating a thread while impersonating another user is not recommended.

The thread execution begins at the function specified by the *lpStartAddress* parameter. If this function returns, the **DWORD** return value is used to terminate the thread in an implicit call to the **ExitThread** [ http://msdn.microsoft.com/en-us/library/ms682659(VS.85).aspx ] function. Use the **GetExitCodeThread** [ http://msdn.microsoft.com/en-us/library/ms683190(VS.85).aspx ] function to get the thread's return value.

The thread is created with a thread priority of THREAD_PRIORITY_NORMAL. Use the **GetThreadPriority** [ http://msdn.microsoft.com/en-us/library/ms683235(VS.85).aspx ] and **SetThreadPriority** [ http://msdn.microsoft.com/en-us/library/ms686277(VS.85).aspx ] functions to get and set the priority value of a thread.

When a thread terminates, the thread object attains a signaled state, satisfying any threads that were waiting on the object.

The thread object remains in the system until the thread has terminated and all handles to it have been closed through a call to **CloseHandle** [ http://msdn.microsoft.com/en-us/library/ms724211(VS.85).aspx ] .

The **ExitProcess** [ http://msdn.microsoft.com/en-us/library/ms682658(VS.85).aspx ] , **ExitThread** [ http://msdn.microsoft.com/en-us/library/ms682659(VS.85).aspx ] , **CreateThread**, **CreateRemoteThread** [ http://msdn.microsoft.com/en-us/library/ms682437(VS.85).aspx ] functions, and a process that is starting (as the result of a call by **CreateProcess**) are serialized between each other within a process. Only one of these events can happen in an address space at a time. This means that the following restrictions hold:

- During process startup and DLL initialization routines, new threads can be created, but they do not begin execution until DLL initialization is done for the process.
- Only one thread in a process can be in a DLL initialization or detach routine at a time.
- **ExitProcess** does not complete until there are no threads in their DLL initialization

or detach routines.

A thread in an executable that calls the C run-time library (CRT) should use the **_beginthreadex** [ http://go.microsoft.com/fwlink/?LinkId=125829 ] and **_endthreadex** [ http://go.microsoft.com/fwlink/?LinkId=125830 ] functions for thread management rather than **CreateThread** and **ExitThread**; this requires the use of the multi-threaded version of the CRT. If a thread created using **CreateThread** calls the CRT, the CRT may terminate the process in low-memory conditions.

## Examples

For an example, see Creating Threads [ http://msdn.microsoft.com/en-us/library/ms682516(VS.85).aspx ] .

## Requirements

| | |
|---|---|
| **Minimum supported client** | Windows 2000 Professional |
| **Minimum supported server** | Windows 2000 Server |
| **Header** | Winbase.h (include Windows.h) |
| **Library** | Kernel32.lib |
| **DLL** | Kernel32.dll |

## See Also

**CloseHandle** [ http://msdn.microsoft.com/en-us/library/ms724211(VS.85).aspx ]
**CreateProcess** [ http://msdn.microsoft.com/en-us/library/ms682425(VS.85).aspx ]
**CreateRemoteThread** [ http://msdn.microsoft.com/en-us/library/ms682437(VS.85).aspx ]
**ExitProcess** [ http://msdn.microsoft.com/en-us/library/ms682658(VS.85).aspx ]
**ExitThread** [ http://msdn.microsoft.com/en-us/library/ms682659(VS.85).aspx ]
**GetExitCodeThread** [ http://msdn.microsoft.com/en-us/library/ms683190(VS.85).aspx ]
**GetThreadPriority** [ http://msdn.microsoft.com/en-us/library/ms683235(VS.85).aspx ]
Process and Thread Functions [ http://msdn.microsoft.com/en-us/library/ms684847(VS.85).aspx ]
**ResumeThread** [ http://msdn.microsoft.com/en-us/library/ms685086(VS.85).aspx ]
**SetThreadPriority** [ http://msdn.microsoft.com/en-us/library/ms686277(VS.85).aspx ]
SECURITY_ATTRIBUTES [ http://msdn.microsoft.com/en-us/library/aa379560(VS.85).aspx ]
**SuspendThread** [ http://msdn.microsoft.com/en-us/library/ms686345(VS.85).aspx ]
**ThreadProc** [ http://msdn.microsoft.com/en-us/library/ms686736(VS.85).aspx ]
Threads [ http://msdn.microsoft.com/en-us/library/ms684254(VS.85).aspx ]

**Tags:**