Synchronization and Overlapped Input and Output

You can perform either synchronous or asynchronous (also called overlapped) I/O operations on files, named pipes, and serial communications devices. The **WriteFile**, **ReadFile**, **DeviceIoControl**, **WaitCommEvent**, **ConnectNamedPipe**, and **TransactNamedPipe** functions can be performed either synchronously or asynchronously. The **ReadFileEx** and **WriteFileEx** functions can be performed only asynchronously.

When a function is executed synchronously, it does not return until the operation has been completed. This means that the execution of the calling thread can be blocked for an indefinite period while it waits for a time-consuming operation to finish. Functions called for overlapped operation can return immediately, even though the operation has not been completed. This enables a time-consuming I/O operation to be executed in the background while the calling thread is free to perform other tasks. For example, a single thread can perform simultaneous I/O operations on different handles, or even simultaneous read and write operations on the same handle.

To synchronize its execution with the completion of the overlapped operation, the calling thread uses the **GetOverlappedResult** function, the **GetOverlappedResultEx** function, or one of the **wait functions** to determine when the overlapped operation has been completed. You can also use the **HasOverlappedIoCompleted** macro to poll for completion.

To cancel all pending asynchronous I/O operations, use the **CancelIoEx** function and provide an **OVERLAPPED** structure that specifies the request to cancel. Use the **CancelIo** function to cancel pending asynchronous I/O operations issued by the calling thread for the specified file handle.

Overlapped operations require a file, named pipe, or communications device that was created with the **FILE_FLAG_OVERLAPPED** flag. When a thread calls a function (such as the **ReadFile** function) to perform an overlapped operation, the calling thread must specify a pointer to an **OVERLAPPED** structure. (If this pointer is **NULL**, the function return value may incorrectly indicate that the operation completed.) All of the members of the **OVERLAPPED** structure must be initialized to zero unless an event will be used to signal completion of an I/O operation. If an event is used, the **hEvent** member of the **OVERLAPPED** structure specifies a handle to the allocated event object. The system sets the state of the event object to nonsignaled when a call to the I/O function returns before the operation has been completed. The system sets the state of the event object to signaled when the operation has been completed. An event is needed only if there will be more than one outstanding I/O operation at the same time. If an event is not used, each completed I/O operation will signal the file, named pipe, or communications device.

When a function is called to perform an overlapped operation, the operation might be completed before the function returns. When this happens, the results are handled as if the operation had been performed synchronously. If the operation was not completed, however, the function's return value is **FALSE**, and the **GetLastError** function returns **ERROR_IO_PENDING**.

A thread can manage overlapped operations by either of two methods:

- Use the GetOverlappedResult or GetOverlappedResultEx function to wait for the overlapped operation to be completed. If **GetOverlappedResultEx** is used, the calling thread can specify a timeout for the overlapped operation or perform an alertable wait.
- Specify a handle to the OVERLAPPED structure's manual-reset event object in one of the wait functions and then, after the wait function returns, call GetOverlappedResult or GetOverlappedResultEx. The function returns the results of the completed overlapped operation, and for functions in which such information is appropriate, it reports the actual number of bytes that were transferred.

When performing multiple simultaneous overlapped operations on a single thread, the calling thread must specify an OVERLAPPED structure for each operation. Each **OVERLAPPED** structure must specify a handle to a different manual-reset event object. To wait for any one of the overlapped operations to be completed, the thread specifies all the manual-reset event handles as wait criteria in one of the multiple-object wait functions. The return value of the multiple-object wait function indicates which manual-reset event object was signaled, so the thread can determine which overlapped operation caused the wait operation to be completed.

It is safer to use a separate event object for each overlapped operation, rather than specify no event object or reuse the same event object for multiple operations. If no event object is specified in the OVERLAPPED structure, the system signals the state of the file, named pipe, or communications device when the overlapped operation has been completed. Thus, you can specify these handles as synchronization objects in a wait function, though their use for this purpose can be difficult to manage because, when performing simultaneous overlapped operations on the same file, named pipe, or communications device, there is no way to know which operation caused the object's state to be signaled.

A thread should not reuse an event with the assumption that the event will be signaled only by that thread's overlapped operation. An event is signaled on the same thread as the overlapped operation that is completing. Using the same event on multiple threads can lead to a race condition in which the event is signaled correctly for the thread whose operation completes first and prematurely for other threads using that event. Then, when the next overlapped operation completes, the event is signaled again for all threads using that event, and so on until all overlapped operations are complete.

For examples that illustrate the use of overlapped operations, completion routines, and the GetOverlappedResult function, see Using Pipes.

Windows Vista, Windows Server 2003, and Windows XP:

Be careful when reusing OVERLAPPED structures. If **OVERLAPPED** structures are reused on multiple threads and GetOverlappedResult is called with the *bWait* parameter set to **TRUE**, the calling thread must ensure that the associated event is signaled before reusing the structure. This can be accomplished by using the WaitForSingleObject function after calling **GetOverlappedResult** to force the thread to wait until the operation completes. Note that the event object must be a manual-reset event object. If an autoreset event object is used, calling **GetOverlappedResult** with the *bWait* parameter set to **TRUE** causes the function to be blocked indefinitely. This behavior changed starting with Windows 7 and Windows Server 2008 R2 for applications that specify Windows 7 as the supported operating system in the application manifest. For more information see Application Manifests.

Related topics

I/O Concepts