If you return from the main thread, does the process exit?

devblogs.microsoft.com/oldnewthing/20100827-00

August 27, 2010

Raymond Chen

If instead of calling `ExitProcess` you merely return from the main thread of a process, does the process terminate?

No, but maybe yes.

This is another one of the places where the C runtime behaves differently from raw Win32.

Under raw Win32, a process exits when any thread chooses to exit the process explicitly (usually by calling `ExitProcess`) or when all threads have exited. Exiting the main thread will not result in the process exiting if there are any other threads still active. According to the old-fashioned model of how processes exit, a process was in control of all its threads and could mediate the shutdown of those threads, thereby controlling the shutdown of the process. (Of course, nowadays, with the thread pool, COM worker threads, and other threads doing random background work, the idea of being in control of all the threads in the process is now just a reminder of those simpler days.)

On the other hand, the C runtime library automatically calls `ExitProcess` when you exit the main thread, regardless of whether there are any worker threads still active. This behavior for console programs is mandated by the C language, which says that (5.1.2.2.3) "a return from the initial call to the `main` function is equivalent to calling the `exit` function with the value returned by the `main` function as its argument." The C++ language has an equivalent requirement (3.6.1). Presumably, the C runtime folks carried this behavior to `WinMain` for consistency.

This also means that if you decide to exit your main thread by calling `ExitThread` directly, then you aren't returning from the `main` function. Instead, you've leapt into the Win32 world where the process will not exit until all threads are gone.