Standard handles are really meant for single-threaded programs

devblogs.microsoft.com/oldnewthing/20141008-00

October 8, 2014

Raymond Chen

When I discussed the conventions for managing standard handles, Sven2 noted that I implied that you need to call `SetStdHandle` with a new handle if you close the current handle and asked "Wouldn't it make more sense to call it the other way round? I.e., first set the new handle, then close the old one? It would ensure that any other thread that runs in parallel won't get an invalid handle." Yes, that would make more sense, but only by a little bit. If you have one thread changing a standard handle at the same time another thread tries to use it, you still have the race condition, as Cesar noted, where the thread that gets the standard handle gets the handle closed out from under it. So you still have a race condition. All you did was narrow the window a little bit. This is basically a fundamental limitation of the standard handles. They are a shared process-wide resource, and if you're going to be mucking with them from multiple threads, it's your responsibility to apply whatever synchronization you need in order to avoid the problems associated with messing with process-wide resources. (This is similar to the problem with inherited handles and `Create-Process`.) The most common way of ensuring that one thread doesn't change a standard handle while another thread is using it is simple: *Never change a standard handle*. Consider standard handles a setting provided by the parent process. If the parent process says that standard output should go to a particular place, then send it to that place. Don't try to override the decision and send it somewhere else. If you really must change a standard handle, you'd be best off doing so right at the start before you start kicking off background threads. Another model you might try is to treat the initial thread as the "console UI thread" and make that the only thread that can communicate with the standard handles. Background threads can do work, but if they want to write to standard output or read from standard input, they need to ask the main thread to do it. This is probably a good plan anyway, because it avoids messy interleaved output as well as confusing input. (If two threads read from standard input at the same time, it's not clear to the user which thread their input will go to.)

Personally, I would recommend combining both approaches: (1) Never change a standard handle, and (2) Restrict all usage of standard handles to your main thread to avoid problems with interleaving.

Raymond Chen

Follow