The case of the redirected standard handles that won't close even though the child process has exited (and a smidge of Microspeak: reduction)

devblogs.microsoft.com/oldnewthing/20131018-00

October 18, 2013

Raymond Chen

A customer had a supervisor process whose job is to launch two threads. Each thread in turn launches a child process, let's call them A and B, each with redirected standard handles. They spins up separate threads to read from the child processes' stdout in order to avoid deadlocks. What they've found is that even though child process A has exited, the threads responsible for monitoring the output of child process A will get stuck in the `ReadFile` until child process B also exits. The customer further reported that if they added a brief `Sleep` call between creating the thread that launches child process A and creating the thread that launches child process B, then the problem goes away. The customer attached a small sample program which demonstrated the same issue and asked for advice on how to diagnose and fix the problem. First of all, it was great of the customer to include a small sample program that demonstrates the problem. This is an important step in troubleshooting, which goes by the Microspeak term *reduction*.

> **re·duc·tion**. *n*. The process of simplifying a bug to the smallest scenario that still reproduces it.

For source code, this usually takes the form of a small sample program. For Web pages, this means removing irrelevant styles, script, and HTML. In both cases, the reduction can be substantial. (You'd be surprised how big Web pages are nowadays.) Reduction is so important that our defect tracking database has a special field: *Reduced by*. I took a look at the sample program and didn't see anything obviously wrong with it. One of my colleagues, however, was able to use his psychic powers to determine the problem without even reading the code!

> I'll bet $10 that you're launching processes in parallel, specifying `TRUE` for `bInherit-Handles` but *not* using a `PROC_THREAD_ATTRIBUTE_HANDLE_LIST`. You create your pipe handles inheritable and give them to your children. The problem is that if thread 1 is in the middle of setting up these inheritable handles for child process A, and thread 2 calls `Create-Process` for child process B, then child process B will accidentally inherit the handles intended for child process A. As a result, child process B unwittingly holds open the pipe handles you gave to child process A. Reads from a pipe will not return EOF until all writers have closed the handle, so the visible effect is that the monitoring thread for child process A will not complete its read until child process B also exits.
>
> Another possibility is that the child processes are launching their own child processes which are inadvertently inheriting the pipe handles.

(Turns out the first guess was right on the money.) The solution is to use the technique we discussed a few years ago: Use the `PROC_THREAD_ATTRIBUTE_HANDLE_LIST` to control explicitly which handles are inherited by specific child processes.

If the client application must run on versions of Windows prior to Windows Vista, then they can use the workaround described in the linked article: Manually serialize the calls which set up and then launch the child processes, so that handle inheritance management for a child process don't start until the previous one has completed.

Raymond Chen

Follow