Don't let more than one process try to read from stdin at the same time

devblogs.microsoft.com/oldnewthing/20111202-00

December 2, 2011

Raymond Chen

A customer reported a problem with a program that ran a series of other programs in parallel.

> We have a main program (call it main.exe) that runs a bunch of child processes with stdout and stderr redirected. (We are not redirecting stdin.) We've found that some of the child processes get stuck inside the C runtime startup code on a call to `GetFileType` on the stdin handle. What could be the reason for this? Is there something we can do that doesn't require us to modify the child processes? (They are third party code we do not have control over.)

This is one of those *once you've debugged this problem you never forget it* type of problems. Notice that each of the child processes inherits the same stdin from main.exe, since you aren't redirecting stdin. Since the stdin handle was not opened as overlapped, all I/O to the handle is serialized. The C runtime calls `GetFileType` at startup to determine whether or not to use buffering. When each child process starts up, it calls `GetFileType`, enters its `main`, and goes about its business. Everything is great *until one of them tries to read from stdin*. At that point, everything falls apart. The next child process to start calls `GetFile-Type`, but instead of returning with a result, it waits for the previous I/O request (the read) to complete because the handle is marked synchronous, and synchronous handles permit only one operation at a time. The user, of course, doesn't realize that the first program is waiting for input (the prompt got redirected), so the user sits and waits for the program while the program sits and waits for the user. To solve this problem, you first need to decide what you want to happen to stdin. Right now, you gave stdin to a dozen child processes, and each line of input the user types will be randomly assigned to one of those programs. In this case, the customer's answer is "I don't care about stdin; these programs aren't supposed to be reading from stdin anyway", in which case you can redirect stdin of the child processes to `NUL`.

Bonus chatter: This is also why, when you hit Ctrl+C to exit a console program which launched child processes with `CREATE_NEW_PROCESS_GROUP`, the command prompt that comes back sometimes behaves kind of strangely. Since the child processes were launched in a separate process group, the Ctrl+C killed the main program but left the child processes running. If any of those child processes read from stdin, you get the “Randomly assign input” effect described above because you have two programs racing to read from stdin: the orphaned child process and `cmd.exe`.

Raymond Chen

Follow