Why is processor affinity inherited by child processes?

devblogs.microsoft.com/oldnewthing/20050817-10

August 17, 2005

Raymond Chen

Consider why a typical program launches child processes. (Shell programs like Explorer aren't typical.) It's because the task at hand is being broken down into sub-tasks which for whatever reason has been placed into a child process. An Example of this would be, say, a multi-pass compiler/linker, where each pass is implemented as a separate process in a pipeline.

Now consider why you might want to set a process's affinity mask to restrict it to a single processor. One reason is that the program may have bugs that cause it to crash or behave erratically on multi-processor machines. This was common for older programs that were written for uni-processor versions of Windows or when multi-processor machines were still prohibitively expensive. In this case, you would launch the program in a suspended state, by passing the `CREATE_SUSPENDED` flag to the `CreateProcess` function, then set the processor affinity mask for that process to a single processor, then resume the process's main thread.

But what if the problem was in a child process of the process you're launching? Since you don't have control over how the process launches its child, you have no way to sneak in and set the child process's processor affinity mask. That's why the processor affinity mask is inherited: If you set it on the parent process, this covers all the child helper processes that process may launch as part of its execution.

Another reason why you might want to set a process's affinity mask is to restrict its CPU usage. (For example, you might restrict a CPU-intensive application to a single processor of your dual-processor machine.) And again, if the process launches child processes, you want those child processes to be subject to the same restriction as their parent so that the task as a whole remains restricted to a single processor.

That's why processor affinity is inherited by child processes. Because it's nearly always what you want.

Raymond Chen

Follow