A process inherits its environment from its parent, and the consequences of this simple statement

devblogs.microsoft.com/oldnewthing/20150915-00

September 15, 2015

Raymond Chen

A customer reported that they changed some system-level enviornment variables (include `PATH`). They also have a batch file that, among other things, runs a program that requires that the `PATH` be set a certain way. The batch file is set to run automatically every five minutes, and the customer observed that the batch file is not picking up the changes to the system-wide environment variables.

The customer developed two theories as to why this was happening.

Theory number one was that batch files inherit the environment from its parent process (unless the parent process passes an explicit environment to `CreateProcess`). "Since we changed a global environment variable and the parent process is continually running, it does not pick up the new environment variables. Then it passes this outdated set of environment variables to the batch file."

Theory number two is that changes to global environment variables will eventually become applied to running processes, but the effect is not immediate; you have to wait a while for the change to propagate. "We started a new command prompt, and the new command prompt did not have the new environment variables, even though the new values did appear in the System control panel. After setting the environment variable a few more times, eventually the new values started showing up in the command prompt."

The first theory is correct. Processes inherit their initial environment from their parents. Of course, once the process is running, it is free to change its environment variables by calling `SetEnvironmentVariable`, and those modified environment variables are passed to any child processes launched after the new variable is set. (A parent process can also pass a custom environment to the child process.)

Now, a child process inherits its initial environment from its parent, but it only gets a snapshot of that environment. If the parent subsequently modifies its environment, the child environment is not updated.

When the customer said that they “changed some global environment variables”, that was a rather vague statement, because as we saw above, there is technically no such thing as a “global environment variable”. Every environment variable is local to a process. What the customer actually did was make a chance to the template environment that is used to get the ball rolling.

When you log on, the system constructs an initial environment from the template specified in the Environment Variables control panel. That initial environment is then given to Explorer, where it becomes Explorer’s initial environment, and from there, it becomes inherited by anything that gets launched by Explorer, and then anything that gets launched by anything that gets launched by Explorer, and so on. If you can trace your lineage back to Explorer, then your initial environment was based on the copy in Explorer.

Now, each process along the way may have edited its environment before spawning a child process, in which case those edited values are inherited by the children.

In order for the changes to the environment template to take effect in a process, that process needs to support the “Throw away all your environment variables and create a new set of environment variables based on the current template” message. That message is the `WM_SETTINGCHANGED` message, where the `lParam` points to the string “Environment”.

The only program in common use that actually responds to that message is Explorer.

This means that when you change the template for the initial environment from the Environment Variables control panel, the only program that reacts immediately to those changes is Explorer. Any other programs that are already running will continue to operate with whatever environment variables they had when they started (or subsequently edited).

If you run a program from Explorer, it will get the updated environment because Explorer updated its environment in response to the message.

If you run a program from Task Manager or a command prompt or anything else, then it will not get the updated environment because Task Manager, the command prompt, and pretty much anything else does not update its environment in response to the message.

If you want to regenerate your environment in response to the `WM_SETTINGCHANGED` message with an `lParam` of *Environment*, you can use the `CreateEnvironmentBlock` function. You can enumerate the contents of this environment block and copy it to your current environment.

But personally, I wouldn’t bother, because pretty much nobody else bothers either.

Raymond Chen

Follow