Closing the race window between creating a suspended process and putting it in a job

devblogs.microsoft.com/oldnewthing/20181116-00

November 16, 2018

Raymond Chen

A customer had a test harness that spawns a very large number of processes. To make sure everything gets cleaned up if the test harness closes unexpectedly, they start the process suspended, then place the processes in a job object before resuming the process. The job object is marked as `JOB_ OBJECT_ LIMIT_ KILL_ ON_ JOB_ CLOSE` so that when the last handle to the job is closed, the processes in the job are terminated.

This was successful for the most part, except for the small race window when the process has been created suspended but has not been added to a job. A poorly-timed cancellation of the test harness would leave the zombie process behind: Suspended, but with nobody around to terminate it.

The customer was looking for ideas to close this last remaining race window.

My suggestion was to put the test harness in the job, too! This entail splitting the test harness into two processes. The first process is what the test infrastructure launches. It doesn't actually run any tests, but rather creates the environment for tests to run. Put all the test harness code in the second process.

The first process creates the kill-on-close job, creates the second process (not suspended), passing it a handle to itself and a handle to an event. The first process puts the second process in the job and then signals the event, to tell the second process, "You're in a job now. You can do your thing."

The second process, when it starts up, waits for either of the passed-in handles to be signaled. If the handle to the first process is signaled, then it means that the test was canceled, and the second process should kill itself. If the event handle is signaled, then it means that the second process is now safely in a job and can start launching tests. Those tests will immediately belong to the job object, since they were created by a process already in the job. There is no "suspended process" window.

So now we have these points at which the test can be stopped:

Before the first process creates the second process, terminating the first process terminates the test.

After the first process creates the second process, but before it puts the second process into the job, terminating the first process will cause its own process handle to become signaled, and the second process responds by terminating itself. No tests have started yet.

After the second process is placed in the job, terminating the first process will cause the job handle to become closed, at which point everything in the job (the second process plus all of the tests) will be terminated.

If it’s essential that the tests run in a separate job from the test harness, the test harness can create a second job for the tests themselves. It creates the tests suspended, then moves them into the second job. The tests always belong to *some* job, so they will get terminated eventually.

Raymond Chen

Follow