Why are there all these processes lingering near death, and what is keeping them alive?

devblogs.microsoft.com/oldnewthing/20150918-00

September 18, 2015

Raymond Chen

A customer (via their customer liaison) asked for assistance debugging a failure that occurs when their system remains up for several weeks.

> The customer seems to have a complicated system where they create and kill processes, and I am seeing hundreds of processes in the following state.
>
> ```
> PROCESS fffffa80082a7960
>     SessionId: 0  Cid: 1490    Peb: 7efdf000  ParentCid: 2614
>     DirBase: 1b3fd0000  ObjectTable: 00000000  HandleCount:   0.
>     Image: contoso.exe
>     VadRoot 0000000000000000
>     DeviceMap fffff8a000008ca0
>     Token                             fffff8a00dbf9060
>     ElapsedTime                       1 Day 01:25:38.983
>     UserTime                          00:00:01.903
>     KernelTime                        00:00:00.265
>     QuotaPoolUsage[PagedPool]         0
>     QuotaPoolUsage[NonPagedPool]      0
>     Working Set Sizes (now,min,max)  (5, 50, 345) (20KB, 200KB, 1380KB)
>     PeakWorkingSetSize                17981
>     VirtualSize                       222 Mb
>     PeakVirtualSize                   246 Mb
>     PageFaultCount                    29532
>     MemoryPriority                    BACKGROUND
>     BasePriority                      8
>     CommitCharge                      0
>
> No active threads
>         THREAD fffffa800a358b50  Cid 1490.2704  TERMINATED
> ```
>
> This got me curious: Why are there so many of these near-death processes, and what could be keeping them alive?

I'll let you puzzle on this for a little bit. But you already know the answer.

(Waiting.)

(Waiting.)

Okay.

First thing you should observe is that this process is not actually alive. It has already exited. "No active threads." The one thread that is still associated with the process has terminated.

Why would you have a terminated thread hanging around inside a terminated process?

Because there is still an outstanding handle to the thread.

Even though the thread has exited, the thread object can't go away until all handles to it have been closed.

Now, when you create a process with the `CreateProcess` function, you get a `PROCESS_INFORMATION` structure back which contains four pieces of information:

1. A handle to the created process.
2. A handle to the initial thread in the process.
3. The ID of the created process.
4. The ID of the initial thread in the process.

Of those things, you are probably interested in the process handle, because that's the thing you can wait on to learn when the process has exited. And you probably ignore the thread handle.

Oops.

You need to close the thread handle, or the thread cannot go away. It may have stopped executing, but the fact that you have a handle to it means that you can still do things like check if the thread has exited (yes, already!), ask for the exit code, ask for the thread ID, ask how much CPU the thread consumed during its lifetime, and so on, and all those statistics are kept in the thread object. And since thread and process IDs need to remain unique as long as there is still a handle to the object, the object needs to hang around so that it "occupies space" so that no other thread can grab its ID.

The customer called this process *near death*, but the more conventional term for it is *zombie*. In fact, zombie isn't a good term either, because this process and this thread are well and truly dead, never to walk again.

A better name would be *corpse*. The process and thread are dead. They're just lying there, rotting away in memory, waiting for all references to be released so they can disappear entirely.

Since the customer liaison said that the customer has “a complicated system where they create and kill processes”, it’s entirely possible that somewhere in the complicated system, somebody loses track of the thread handle, causing it to leak. It also calls into question whether they need this complicated system at all. Maybe their complicated system exists to work around some other problem, and we should be trying to solve that other problem.

Just for completeness, another possibility for the thread lying dead in the process is that some kernel driver has taken a reference to the thread and has gotten stuck.

We left the customer liaison with that information. We didn’t hear back, so either our guess about thread handles was correct, or the customer decided we weren’t being helpful enough and decided to stop talking to us.

Raymond Chen

Follow