Why do people take a lock around CreateProcess calls?

devblogs.microsoft.com/oldnewthing/20200306-00

March 6, 2020

Raymond Chen

If you look around, you often see people take a lock around their calls to the `CreateProcess` function. Why do they do that? Isn't the `CreateProcess` function thread safe?

Yes, the `CreateProcess` function is thread safe. But thinking about thread safety is the right thing.

The issue is with inheritable handles.

When you ask for handles to be inherited, the `CreateProcess` inherits *all* the handles in the process that are marked as inheritable. If you have multiple threads creating processes, you run into trouble if each thread wants a different set of handles to be inherited. The two threads each create their respective inheritable handles, and as a result, the handles get inherited into *both* processes.

Prior to Windows Vista, the standard workaround was to use a mutex so that only one thread at a time can go through the steps of

1. Creating inheritable handles.
2. Calling `CreateProcess` with `bInheritHandles = true`.
3. Closing the inheritable handles created in step 1.

Windows Vista introduced the `PROC_THREAD_ATTRIBUTE_LIST`, which I discussed some time ago. This addresses the concurrency problem by allowing each call to `CreateProcess` to specify a custom list of handles to be inherited. That way, you can have two threads calling `CreateProcess` at the same time without interfering with each other's inherited handles. You don't need a mutex any more.

There's still a problem with this, though: It requires everybody to be playing the same game.

In order for a handle to be inherited, you not only have to put it in the `PROC_THREAD_ATTRIBUTE_LIST`, but you also must make the handle inheritable. This means that if another thread is not on board with the `PROC_THREAD_ATTRIBUTE_LIST` trick

and does a straight `CreateProcess` with `bInheritHandles = true`, it will inadvertently inherit your handles.

A colleague of mine came up with a sneaky trick for addressing this new problem: Create a dummy parent process and put the inheritable handles in there.

Here's the basic idea:

Preparation

> Create a process suspended. This process will never run. It is just a container for handles.

Call this process the "helper" process. This process will end up helping us, despite the process not actually doing anything!

Process creation: Do this each time you need to create a process with specific inherited handles.

- Create all the handles as non-inheritable. This ensures they don't accidentally get inherited if another thread (not written by you) calls `CreateProcess`.
- Use `DuplicateHandle` to duplicate the handles you want to inherit into the helper process, with `bInheritHandles = true`.
- Add those handles to a `PROC_THREAD_ATTRIBUTE_LIST`.
- Add the handle of the helper process as a `PROC_THREAD_ATTRIBUTE_PARENT_PROCESS`, so that it acts as the nominal parent process. Specifically, it is the source of inherited handles.
- Call `CreateProcess` with this attribute list.
- Use `DuplicateHandle` with `DUPLICATE_CLOSE_SOURCE` to close the handles you injected into the helper process.

Cleanup

> Terminate the helper process.

This technique works because the handles are never marked as inheritable in the main process. Therefore, they can never be accidentally inherited. The only place the handles are marked inheritable is in the helper process. Since the helper process is always suspended, there's no way that anybody in the helper process can call `CreateProcess`. The only way somebody can accidentally inherit the handles is if they accidentally get a handle to your helper process, which would be quite an accident.

You probably want to put the helper process in a "terminate on close" job, so that the cleanup occurs automatically when your process terminates. That way, you don't leak helper processes if your main process crashes before it can clean up properly.

Raymond Chen

Follow