# After I made my DLL delay-load another DLL, my DLL has started crashing in its process detach code

devblogs.microsoft.com/oldnewthing/20190718-00

July 18, 2019

Raymond Chen

A customer had a DLL, let's call it `CONTOSO.DLL`, and that DLL linked to another DLL, let's call it `WIDGET.DLL`. To improve DLL load time, they made `WIDGET.DLL` a delay-loaded DLL via the `/DELAYLOAD` linker option. This worked out great, except that sometimes their DLL crashed when shutting down.

When the `WIDGET.DLL` was a static dependency, the loader made a note to ensure that `WIDGET.DLL` was loaded and ready before calling `CONTOSO.DLL`'s initialization function, and made sure that `WIDGET.DLL` remained valid until `CONTOSO.DLL` completed its uninitialization.

Switching the `WIDGET.DLL` to a `/DELAYLOAD` DLL removes this static dependency, and the loader isn't around to help any more.

When the process shuts down, the loader uninitializes the DLLs in an order that tries[1] to preserve the static dependencies, so that a DLL waits until all its dependents are uninitialized before itself uninitializing. However, the loader does not have insight into dynamically-created dependencies, and the DLLs may unload out of order.

What happened is that `CONTOSO.DLL` initialized without `WIDGET.DLL`, and then later somebody needed a widget, so it loaded `WIDGET.DLL` and did some widget stuff, and then cached the widget so it wouldn't have to go through all that nonsense again.

In the `CONTOSO.DLL` module's `DLL_PROCESS_DETACH` code, it checks[2] if there is a cached widget, and if so, destroys it.

WIDGET.DLL was a dynamic dependency, the module loader doesn't take it into account when calculating the order in which modules should be uninitlalized. The loader sees no static dependency between `CONTOSO.DLL` and `WIDGET.DLL`, so the order in which they uninitialize is arbitrary.

And if the arbitrary decision ends up selecting `WIDGET.DLL` to uninitialize first, then you have a crash when `CONTOSO.DLL` tries to call into an already-uninitialized DLL.

Note that this problem occurs only at process shutdown. If `CONTOSO.DLL` unloads via a runtime call to `FreeLibrary`, it will still be able to call into `WIDGET.DLL` because it hasn't yet called `FreeLibrary` on `WIDGET.DLL`. But during process shutdown, the module loader needs to free all the things, and the outstanding `LoadLibrary` won't prevent that from happening.

The solution is to bypass widget cleanup if the `DLL_PROCESS_DETACH` handler realizes that the process is terminating. Just leak the widget. The building is being demolished. You don't need to sweep the floors.

The DLL was able to start without the widget DLL. It should be able to finish without the widget DLL.

[1] I say "tries" because circular dependencies make such an effort impossible to achieve, but the loader does the best it can.

[2] It's important to check for evidence of widgets before trying to clean up widget-related things. Otherwise, you may end up loading a DLL in your `DLL_PROCESS_DETACH` handler, and that's not good.

Raymond Chen

**Follow**