When are global objects constructed and destructed by Visual C++?

devblogs.microsoft.com/oldnewthing/20141017-00

October 17, 2014

Raymond Chen

Today we're going to fill in the following chart:

When does it run?	**Constructor**	**Destructor**
Global object in EXE		
Global object in DLL		

The C++ language specification provides some leeway to implementations on when global static objects are constructed. It can construct the object before `main` begins, or it construct the object on demand according to complicated rules. You can read [basic.start.init] for the gory details.

Let's assume for the sake of discussion that global static objects are constructed before `main` begins.

For global objects in the EXE, constructing them is no big deal because the C runtime startup code linked into the EXE does a bunch of preparation before calling the formal entry point, be it `main` or `wWinMain` or whatever. And part of that preparation is calling constructors for global objects. Since the C runtime startup code is in charge, it can construct the objects right there.

When does it run?	**Constructor**	**Destructor**
Global object in EXE	C runtime startup code	
Global object in DLL		

DLLs are similar: The formal `DllMain` entry point is not the actual entry point to the DLL. Instead, the entry point is a function provided by the C runtime, and that function does work before and after calling the `DllMain` function provided by the application. We saw this earlier when we discussed what happens if you return `FALSE` from `DLL_PROCESS_ATTACH`.

Part of this extra work done by the C runtime library is to construct DLL globals in `DLL_PROCESS_ATTACH` and to destruct them in `DLL_PROCESS_DETACH`. In other words, the code conceptually goes like this:

```
BOOL CALLBACK RealDllMain(
    HINSTANCE hinst, DWORD dwReason, void *pvReserved)
{
  ...
  case DLL_PROCESS_ATTACH:
   Initialize_C_Runtime_Library();
   Construct_DLL_Global_Objects();
   DllMain(hinst, dwReason, pvReserved);
   ...
 case DLL_PROCESS_DETACH:
   DllMain(hinst, dwReason, pvReserved);
   Destruct_DLL_Global_Objects();
   Uninitialize_C_Runtime_Library();
   break;
 ...
}
```

Of course, the actual code is more complicated than this, but that's the basic idea. We can fill in two more cells in our table.

When does it run?	**Constructor**	**Destructor**
Global object in EXE	C runtime startup code	
Global object in DLL	C runtime `DLL_PROCESS_ATTACH` prior to `DllMain`	C runtime `DLL_PROCESS_DETACH` after `DllMain` returns

The last entry in our table is the tricky one: Who triggers the destruction of global objects in the EXE destructed? The C runtime startup code in the EXE is guaranteed to run at process startup, but how does the C runtime cleanup code run?

The answer is that the C runtime library hires a lackey. The hired lackey is the C runtime library DLL (for example, `MSVCR80.DLL`). The C runtime startup code in the EXE registers all the destructors with the C runtime library DLL, and when the C runtime library DLL gets

its `DLL_PROCESS_DETACH`, it calls all the destructors requested by the EXE.

That's the final cell in our table.

When does it run?	**Constructor**	**Destructor**
Global object in EXE	C runtime startup code	C runtime DLL hired lackey
Global object in DLL	C runtime `DLL_PROCESS_ATTACH` prior to `DllMain`	C runtime `DLL_PROCESS_DETACH` after `DllMain` returns

You can now answer this customer question and explain the observed behavior:

> Is it okay to call `LoadLibrary` from within constructors of global C++ objects inside a DLL? Currently we am seeing weird behavior when doing so.

The customer went on to describe what they were observing. Their DLL has global C++ objects which do the following operations in their constructor:

- Check a setting.
- If the setting is enabled, call `LoadLibrary` to load a helper DLL, then call a function in the helper DLL, The result of that function call alters the global behavior of the original DLL.
- The function in the helper DLL creates a thread then waits for the thread to produce a result.
- The helper thread never gets started.

Result: Process hangs.

Raymond Chen

Follow