How does the C runtime know whether to use the static-linking or dynamic-linking version of the header file?

devblogs.microsoft.com/oldnewthing/20110321-00

March 21, 2011

Raymond Chen

In response to a description of what happens when you get `dllimport` wrong, nksingh asks, "This seems like a problem for the CRT. As far as I know, VC gives you the option of statically or dynamically linking the CRT. But it seems like the headers will have to make a choice to support one thing better than the other. Conditional compilation would work, but then people would have to remember to include a #define somewhere. Is this dllimport vs. static linking thing something the compiler could figure out on its own if you're doing Link-time codegen?"

Let's start from the beginning.

Yes, this would be a problem for the CRT since it wouldn't know whether to declare the functions as normal static functions or as `dllimport`-style functions, and the headers have to make a choice which way to go.

And if you look at the headers, you can see that it is indeed done via conditional compilation.

```
...
_CRTIMP int __cdecl fflush(FILE * _File);
...
```

This magic `_CRTIMP` symbol is defined in `crtdefs.h` like so:

```
/* Define _CRTIMP */
#ifndef _CRTIMP
#ifdef _DLL
#define _CRTIMP __declspec(dllimport)
#else  /* _DLL */
#define _CRTIMP
#endif  /* _DLL */
#endif  /* _CRTIMP */
```

Conditional compilation decides whether `_CRTIMP` expands to `__declspec(dllimport)` or to nothing at all, depending on whether the `_DLL` symbol is defined.

And yet nobody bothers writing `#define _DLL` before they `#include <stdio.h>`. There must be something else going on.

In fact, we can run some experiments to see what's going on.

```
#ifdef _DLL
#error "_DLL is defined"
#else
#error "_DLL is not defined"
#endif
```

Save this as `dummy.c` and run a few tests.

```
C:\tests> cl /MT dummy.c
dummy.c
dummy.c(4) : fatal error C1189: #error :  "_DLL is not defined"
C:\tests> cl /MD dummy.c
dummy.c
dummy.c(2) : fatal error C1189: #error :  "_DLL is defined"
```

Well how's about that. The compiler uses the `/MT` and `/MD` flag to decide whether or not to define the preprocessor symbol `_DLL`, which is the secret signal it passes to the `crtdef.h` header file to control the conditional compilation.

The compiler has to use this technique instead of deferring the decision to link-time code generation because it cannot assume that everybody has enabled link-time code generation. (Indeed, we explicitly did not in our sample command lines.)

If link-time code generation were enabled, then is this something that could be deferred until that point?

In principle yes, because link-time code generation in theory could just make the .obj file a copy of the source file (and all the header files) and do all the actual compiling at link time. This is a sort of extreme way of doing it, but I guess it could've been done that way.

On the other hand, it also means that the compiler folks would have to come up with a new nonstandard extension that means "This function might be a normal static function or it might be a `dllimport` function. I haven't decided yet; I'll tell you later."

Seeing as how the CRT already has to solve the problem in the case where there is no link-time code generation, it doesn't seem worth the effort to add a feature to link-time-code generation that you don't actually need. It would be a feature for which the only client is the C runtime library itself, for which the C runtime library already requires a separate solution when link-time code generation is disabled, and for which that separate solution *still works when link-time code generation is enabled*.

No engineering purpose is served by writing code just for the sake of writing code.

Raymond Chen

Follow