How do you prevent the linker from discarding a function you want to make available for debugging?

devblogs.microsoft.com/oldnewthing/20150102-00

January 2, 2015

Raymond Chen

We saw some time ago that you can ask the Windows symbolic debugger engine to call a function directly from the debugger. To do this, of course, the function needs to exist.

But what if you want a function for the sole purpose of debugging? It never gets called from the main program, so the linker will declare the code dead and remove it.

One sledgehammer solution is to disable discarding of unused functions. This the global solution to a local problem, since you are now preventing the discard of *any* unused function, even though all you care about is one specific function.

If you are comfortable hard-coding function decorations for specific architectures, you can use the `/INCLUDE` directive.

```
#if defined(_X86_)
#define DecorateCdeclFunctionName(fn) "_" #fn
#elif defined(_AMD64_)
#define DecorateCdeclFunctionName(fn) #fn
#elif defined(_IA64_)
#define DecorateCdeclFunctionName(fn) "." #fn
#elif defined(_ALPHA_)
#define DecorateCdeclFunctionName(fn) #fn
#elif defined(_MIPS_)
#define DecorateCdeclFunctionName(fn) #fn
#elif defined(_PPC_)
#define DecorateCdeclFunctionName(fn) ".." #fn
#else
#error Unknown architecture - don't know how it decorates cdecl.
#endif
#pragma comment(linker, "/include:" DecoratedCdeclFunctionName(TestMe))
EXTERN_C void __cdecl TestMe(int x, int y)
{
    ...
}
```

If you are not comfortable with that (and I don't blame you), you can create a false reference to the debugging function that cannot be optimized out. You do this by passing a pointer to the debugging function to a helper function outside your module that doesn't do anything interesting. Since the helper function is not in your module, the compiler doesn't know that the helper function doesn't do anything, so it cannot optimize out the debugging function.

```
struct ForceFunctionToBeLinked
{
  ForceFunctionToBeLinked(const void *p) { SetLastError(PtrToInt(p)); }
};

ForceFunctionToBeLinked forceTestMe(TestMe);
```

The call to `SetLastError` merely updates the thread's last-error code, but since this is not called at a time where anybody cares about the last-error code, it is has no meaningful effect. The compiler doesn't know that, though, so it has to generate the code, and that forces the function to be linked.

The nice thing about this technique is that the optimizer sees that this class has no data members, so no data gets generated into the module's data segment. The not-nice thing about this technique is that it is kind of opaque.

Raymond Chen

Follow