

Re: Writing device drivers in Delphi.

Source:

<http://coding.derkeiler.com/Archive/Delphi/borland.public.delphi.non-technical/2005-11/msg02552.html>

- *From:* "Martin Harvey" <martin@xx>
 - *Date:* Mon, 28 Nov 2005 22:35:31 -0000
-

"Mat Ballard" <mat@xxxxxxxxxxxxxx> wrote in message
[news:438a404d\\$1@xx](mailto:news:438a404d$1@xx)

> Martin Harvey wrote:

>

>> If I get really bored, I'm well inclined to embark on the worlds largest
>> hack to let you write device drivers in Delphi.

>

> you are bored !

Yep!

>> As I see it, there's only one really difficult problem: Producing an
>> executable of the right format, with appropriate entry point and fixup
>> information.

>

> simple , innit ?

Well, I probably can't say too much about what I'm doing at the moment, but
considering that I'm scrolling through reams of assembler originally
compiled at Microsoft, then

> and in terms of difficulty, where would you see translating the DDK header
> files
> ? and all those pesky macros ?

No harder than translating any other header file ... it's fairly simple,
just takes time and effort.

> of course, you'd also have to replace the Delphi
> memory manager with one that was aware of paged versus non-paged pool. and
> all
> that assembly in system.pas – very tricky !

Not at all. Hang on, let's just have a quick look thru right here and now,
and I'll tell you what I'd do to the various bits. My suggestions here may
not be entirely right (I only have 5 minutes), but things aren't *that*
different. Anyways, large amounts of system.pas could simply be omitted or

Re: Writing device drivers in Delphi.

not used as having no well defined meaning. As a driver, the default entry point won't be invoked anyway. You might want to perform some limited initialization of RTTI tables and memory managers etc in DriverEntry instead, but 95% of the stuff in system.pas assumes a process context, which doesn't really have much meaning in a driver.

Interface:

- Variant stuff. Ditch it.
- Class declarations: ensure RTTI is placed in appropriate non-paged initialized data section in driver.
- Interfaces: hmm, need to check how the compiler does interface dispatching, might disallow to start with.
- Memory manager and heap status: Replace and write a suitable stub layer that redirects calls to alloc / free non-paged pool. May also wish to redirect some stuff to Mm functions.
- Packages: forget it, remove object code and disallow anything that tries to do fixups.
- Global variables: ooo, there are quite a lot here:
- Exception handling mechanisms: 99% of it can be stripped out, might be possible to get the compiler to insert some appropriate code that could be munged into `__try` and `__catch(EXCEPTION_EXECUTE_HANDLER)`.
- Most of the other globals don't really have any well defined meaning in kernel mode. You can leave 'em in the object if you want, but don't expect them to do anything useful. 8087 stuff not useful, because you'll wrap all your stuff in calls to `KeSaveFloatingPointState` and `KeRestoreFloatingPointState`.
- `BeginThread` and `EndThread` functions, you'd probably want to stub onto `PsCreateSystemThread`, and `PsExitSystemThread`, but you'd probably prefer to use the originals, or to use the `WORK_ITEM` routines in the DDK.
- Variant stuff: forget it.
- Standard string stuff ... with interlocked ops, you could probably leave it in, but you'd be strongly advised to use `UNICODE_STRING` and `ANSI_STRING` structures in the DDK.

- `GetMem`, `FreeMem`, `Realloc` mem asm ... replace with pascal code stubbing appropriate pool functions.
- "Error" function: `KeBugcheckEx` works ;-)
- String assembler: probably OK in place, but could replace with plain pascal if required.
- Random number stuff: fine as it is.
- Directory functions – no point in having them.
- floating point stuff: fine if wrapped appropriately.
- objsetup: no real problem here, assuming allocation is done fine.

Looking thru, most of the problems are not to do with the code, which is valid code, the problems are with global data. However, most things, like VMT's and the like will be fine, provided that whatever munging you do to the executable image, you also do to the VMT's to fix them up appropriately.

Re: Writing device drivers in Delphi.

Having said that ... there **is** rather a lot of assembler to look through...
;-)

>and of course no floating point or
> exceptions, and i'm not sure about variants.

Floating point is fine, you just use the KeSave and KeRestore functions to save and restore floating point state. As for variants and such like, the point of all these things is that you're not going to want them in the kernel: **really**. You wont want just about everything – not even the native delphi string handling ... because it just doesn't fit in the kernel model of doing things.

> i would strongly advise you against trying this. if you are still
> determined,
> download the DDK

I already have it, and been writing drivers professionally for 18 months now.

> and work through the sample code there – the comments alone
> ("this should work but the compiler barfs") will give you second thoughts.

God, everyone thinks driver writing is so hard ... well, **yes** it is, and for many reasons which took me a great deal of time to figure out, but it's not like the kernel magically runs a different instruction set or something....

For your information, the current product I'm working on is for very high speed networking and involves running a TCP/IP stack in parallel wth the microsoft stack. The catch is that the stack runs in **both** user and kernel mode on the same socket simultaneously. Oh yeah, and most of the stack cross-compile between windows, linux, and solaris. Compared with that, writing a driver in delphi **is relatively easy**, because the problems are essentially static: How do you generate the same or equivalent machine code output.

MH.

.

• *Follow-Ups:*

◆ *Re: Writing device drivers in Delphi.*

◇ *From:* Ryan McGinty

• *References:*

◆ *Writing device drivers in Delphi.*

◇ *From:* Martin Harvey

Re: Writing device drivers in Delphi.

◆ **Re: Writing device drivers in Delphi.**

◇ From: Mat Ballard

- Prev by Date: **Re: Delphi 2005 vs Visual Studio 2005**
- Next by Date: **Re: Microsoft "Cider" and "Sparkle"**
- Previous by thread: **Re: Writing device drivers in Delphi.**
- Next by thread: **Re: Writing device drivers in Delphi.**
- Index(es):
 - ◆ **Date**
 - ◆ **Thread**