

## Re: writing drivers using C++

---

*Source:* <http://linux.derkeiler.com/Newsgroups/comp.os.linux.development.system/2007-09/msg00042.html>

---

- *From:* Wolfgang Draxinger <[wdraxinger@xxxxxxxxxxxxxxxxxxx](mailto:wdraxinger@xxxxxxxxxxxxxxxxxxx)>
  - *Date:* Sat, 08 Sep 2007 18:25:37 +0200
- 

Ulrich Eckhardt wrote:

I beg to differ, exceptions and RTTI require some code.

No, both can be done at compile time, though the implementation may greatly differ from how it's done usually. Been there, done that, programming  $\mu$ C with tight RAM constraints, but loads of program ROM. However I've done that with pure C and a lot of helper macros then.

For example exceptions can be done using a global backtrace table instead of using a stack unwind approach (in multithreaded environments you'd use thread local storage for those). RTTI can be implemented in a similar way: Primitive types are dealt with at compile time. Class instances carry a pointer (similar to a vtbl) with them, that directs to a static classinfo structure. All RTTI stuff can be done at compile time, but of course it's easier to implement with a runtime.

I'd say that is a bad reason. Firstly, C integrates pretty well with C++, so you can always use some C code, just like you can step to assembly today. I'd much rather say that C++ is a very sharp tool, and most programmers can't use it safely.

You can of course dive deep into the internals of C++ with the C subset and assembler of course, but the resulting code will be tied to a certain compiler or runtime implementation. And then don't forget that you can't access identifiers by their mangled names from within C++ code, unless you make use of some probably unavailable `#pragma`.

Just have a look on how to access COM/ActiveX modules from pure C, it's an utter mess. Microsoft developed those concepts with only their own C++ compilers in mind and it took other compiler vendors some iterations to adapt their's to the new "needs". The

## Re: writing drivers using C++

problem is, that C++ doesn't define a strict ABI (yet), which in the early days of C++ has lead to a lot of different ways how polymorphism, RTTI, exception handling and the lot were implemented (at compile time BTW). Today it's done the same way by most compilers, though IMHO the way it's done is not really the best one.

Well, binary objects are a pain in the ass either way, whether it's Linux that changes its internal APIs or whether it's the C++ compiler matters little. I'd say that this could be fixed with tools though, but nobody's writing them. :/

They're not, as soon as you got a strict ABI. Any compiled program is a binary object of some sort, and somehow even the oldest ELF binaries happily run on even the newest \*nix-boxes as long as the accompanying libraries are installed – or you've got it even purely static in an a.out.

And at least on a programming language level something like an ABI should be the least problematic thing to tie down – I still wonder why it hasn't been done for C++ in the first place. Maybe because C never needed a pinned down ABI, as CPU architectures implicitly brought something like an ABI with them and things like exception handling or RTTI never were done on a language level, but on a third party library/runtime level.

And one thing's for sure: My favor of C over C++ comes not from an ignorance of C++, but from being fed up with it. I took my first steps in the OOP programming world with C++, and nearly all my programs for the next 10 years since then were written in C++. But there more I worked with it, the more I figured it's flaws, from a technical purist's point of view. I can't pin down my feelings about it, but to me C++ is something like Lego technic, while C feels like crafting a machine from bare materials in a workshop – it may take longer, be harder, but you've got far more possibilities to express your ideas, while C++ gets you faster to a result. But this is a highly subjective, even emotional view I got. Other people probably differ, and it's good that way. Still I think C is one of the finest programming languages in existence.

Wolfgang Draxinger

—

E-Mail address works, Jabber: hexarith@xxxxxxxxxx, ICQ: 134682867

.