

# Developing Linux drivers in C++

Mark Veltzer

[mark.veltzer@gmail.com](mailto:mark.veltzer@gmail.com)

CTO, Hinbit

Senior Instructor, Interbit

*In fact, in Linux we did try C++ once already, back in 1992. It sucks. Trust me - writing kernel code in C++ is a BLOODY STUPID IDEA.*

*The fact is, C++ compilers are not trustworthy. They were even worse in 1992, but some fundamental facts haven't changed: 1) the whole C++ exception handling thing is fundamentally broken. It's especially broken for kernels. 2) any compiler or language that likes to hide things like memory allocations behind your back just isn't a good choice for a kernel. 3) you can write object-oriented code (useful for filesystems etc) in C, without the crap that is C++.*

*Linus*

# Motivation...

- Better code to noise ratio
- Easier, type safe object oriented
- One driver, many platforms
- Auto completion for faster coding?, well...
- Because it can be done
- I've always wanted to do template based multiple inheritance in kernel space...
- This actually works. No, seriously! I mean it, stop laughing!

# Problems...

- You stand alone in maintaining this
- No upstream for you
- Being looked at funny
- You will be maintaining your own C++ framework (probably...)
- C++ has a bad reputation, arguably well deserved
- Good luck trying to find new kernel developers to maintain this

# Reminder...

- Linux drivers are written in C
- Object oriented features via pointer hiding and structures
- Verbose error paths, construction and destruction
- Drivers are Linux specific (well, sort of...)
- High noise to code ratio as a result
- Frameworks are collections of C functions to be implemented

# Technical issues

- Module compilation uses the kernel makefile system which does not compile C++
- Kernel headers make C++ compilers roll over and die...
- Many flags passed to the compiler that should be matched by the C++ compiler, some of which are not suitable for a C++ compiler...
- Only some C compilers are supported by kernel developers
- C++ compiler is broken?!? Well, maybe...
- C++ is tuned to user space needs...
- C++ needs a little bit of runtime support

# Two roads diverged in a yellow wood

- Fix all kernel headers (and code) so that kernel headers could be included by a C++ compiler (→ lots of work, would probably not be accepted upstream)
- Create a buffer layer (→ We are here)

Lets go... (technical stuff follows)

# Creating the buffer layer...

- Wrap kernel service C functions with struct less equivalents, void\* equivalents
- Not a whole lot of code
- Quite easy to maintain
- (see example)

# Makefiles, makefiles

- Adding makefile rules for C++ parts of a the module
- Relinking after the kernel makefile has done it's bit
- Passing the right flags
- (see example)

# What are the right flags, anyway?

- Well, there's only one way to find out...
- Maintaining the flags is an issue
- Watch out when the compiler changes, ouch!
- You could crash and burn anyway...
- (see example)

# Adding a C++ framework

- There is much to do here
- Frameworks will probably be C++ objects to derive from
- OO wrappers for spin locks, lists, scatter/gather lists and more
- (see example)

# C++ runtime support?

- Memory allocation is easy
- Exception handling is harder
- Some funky symbols have to be defined
- (see example)

# Conclusions...

- Possible, for unique purposes
- Especially good for industrial drivers that don't want to go upstream
- Especially good if maintaining a single driver for a different platforms is your goal → need to repeat this trick for yet another platform (ouch...)
- Exceptions are still a problem but can be fixed

# Next, please...

- Join me next year for “Developing Linux kernel modules in Erlang, a pipe dream?”
- *git clone git@github.com:veltzer/kcpp.git*
- Thank you