Rust is not a good C replacement

March 25, 2019 on Drew DeVault's blog

I have a saying that summarizes my opinion of Rust compared to Go: “Go is the result of C programmers designing a new programming language, and Rust is the result of C++ programmers designing a new programming language”. This isn’t just a metaphor - Go was designed by plan9 alumni, an operating system written in C and the source of inspiration for many of Go’s features, and Rust was designed by the folks at Mozilla - whose flagship product is one of the largest C++ codebases in the world.

The values of good C++ programmers are incompatible with the values of good C programmers[1]. Rust is a decent C++ replacement if you have the same goals as C++, but if you don’t, the design has very similar drawbacks. Both Rust and C++ are what I like to call “kitchen sink” programming languages, with the obvious implication. These languages solve problems by adding more language features. A language like C solves problems by writing more C code.

I did some back of the napkin estimates of the rate at which these languages become more complex, based on the rate at which they add features per year. My approach wasn’t very scientific, but I’m sure the point comes across.

- **C: 0.73 new features per year**, measured by the number of bullet points in the C11 article on Wikipedia which summarizes the changes from C99, adjusted to account for the fact that C18 introduced no new features.
- **Go: 2 new features per year**, measured by the number of new features listed on the Wikipedia summary of new Go versions.
- **C++: 11.3 new features per year**, measured by the number of bullet points in the C++17 article which summarizes the changes from C++14.
- **Rust: 15 new features per year**, measured by the number of headers in the release notes of major Rust versions over the past year, minus things like linters.

This speaks volumes to the stability of these languages, but more importantly it speaks to their complexity. Over time it rapidly becomes difficult for one to keep an up-to-date mental map of Rust and how to solve your problems idiomatically. A Rust program written last year already looks outdated, whereas a C program written ten years ago has pretty good odds of being just fine. Systems programmers don’t want shiny things - we just want things that work. That really cool feature $other_language has? Not interested. It’ll be more trouble than it’s worth.

With the philosophical wish-wash out of the way and the tone set, let me go over some more specific problems when considering Rust as a C replacement.

C is the most portable programming language. Rust actually has a pretty admirable selection of supported targets for a new language (thanks mostly to

LLVM), but it pales in comparison to C, which runs on almost *everything*. A new CPU architecture or operating system can barely be considered to exist until it has a C compiler. And once it does, it unlocks access to a vast repository of software written in C. Many other programming languages, such as Ruby and Python, are implemented in C and you get those for free too.

C has a spec. No spec means there's nothing keeping rustc honest. Any behavior it exhibits could change tomorrow. Some weird thing it does could be a feature *or* a bug. There's no way to know until your code breaks. That they can't slow down to pin down exactly what defines Rust is also indicative of an immature language.

Unable to connect

C has many implementations. C has many competing compilers. They all work together stressing out the spec, fishing out the loosely defined corners, and pinning down exactly what C is. Code that compiles in one and not another is indicative of a bug in one of them, which gives a nice extra layer of testing to each. By having many implementations, we force C to be well defined, and this is good for the language and its long-term stability. Rustc could stand to have some competition as well, maybe it would get faster[2]

C has a consistent & stable ABI. The System-V ABI is supported on a wide variety of systems and has been mostly agreed upon by now. Rust, on the other hand, has no stable internal ABI. You have to compile and link everything all in one go on the same version of the Rust compiler. The only code which can interact with the rest of the ecosystem is unidiomatic Rust, written at some kind of checkpoint between Rust and the outside world. The outside world exists, it speaks System-V, and us systems programmers spend a lot of our time talking to it.

Cargo is mandatory. On a similar line of thought, Rust's compiler flags are not stable. Attempts to integrate it with other build systems have been met with hostility from the Rust & Cargo teams. The outside world exists, and us systems programmers spend a lot of our time integrating things. Rust refuses to play along.

Concurrency is generally a bad thing. Serial programs have X problems, and parallel programs have X^Y problems, where Y is the amount of parallelism you introduce. Parallelism in C is a pain in the ass for sure, and this is one reason I find Go much more suitable to those cases. However, nearly all programs needn't be parallel. A program which uses poll effectively is going to be simpler, reasonably performant, and have orders of magnitude fewer bugs. "Fearless concurrency" allows you to fearlessly employ bad software design 9 times out of 10.

Safety. Yes, Rust is more safe. I don't really care. In light of all of these problems, I'll take my segfaults and buffer overflows. I especially refuse to "rewrite it in Rust" - because no matter what, rewriting an entire program from scratch is *always* going to introduce more bugs than maintaining the C program ever would. I don't care what language you rewrite it in.

C is far from the perfect language - it has many flaws. However, its replacement will be simpler - not more complex. Consider Go, which has had a lot of success in supplanting C for many problems. It does this by specializing on certain classes of programs and addressing them with the simplest solution possible. It hasn't completely replaced C, but it has made a substantial dent in its problem space - more than I can really say for Rust (which has made similar strides for C++, but definitely not for C).

The kitchen sink approach doesn't work. Rust will eventually fail to the "jack of all trades, master of none" problem that C++ has. Wise languages designers start small and stay small. Wise systems programmers extend this philosophy to designing entire systems, and Rust is probably not going to be invited. I understand that many people, particularly those already enamored with Rust, won't agree with much of this article. But now you know why we are still writing C, and hopefully you'll stop bloody bothering us about it.

---

1. Aside: the term "C/C++" infuriates me. They are completely different languages. Idiomatic C++ looks nothing like idiomatic C. ↩

2. Rust does have one competing compiler, but without a spec it's hard to define its level of compatibility and correctness, and it's always playing catch-up. ↩

Have a comment on one of my posts? Start a discussion in my public inbox by sending an email to ~sircmpwn/public-inbox@lists.sr.ht [mailing list etiquette]

Articles from blogs I read

Generated by openring

What's cooking on SourceHut? June 2022

Hello everyone! Let's get straight into the news today. Our user count today is 29,612 users, of which 576 have joined since the last update. Remember to be patient with these new users as they learn the ropes. Welcome! GraphQL Adnan continues to ship GraphQL…

via Blogs on Sourcehut
June 15, 2022

Status update, June 2022

Hi! Yesterday I've finally finished up and merged push notification support for the soju IRC bouncer and the goguma Android client! Highlights & PM notifications should now be delivered much more quickly, and power consumption should go down. Additionally…

via emersion
June 15, 2022

Share your feedback about developing with Go

Help shape the future of Go by sharing your thoughts via the Go Developer Survey

via The Go Blog
June 1, 2022