

FIXME: Add logo above in footer and header *FIXME*: Add guard page

Embecosm Rust-GCC - Fixed time mission

FIXME: Fix the title

Thanks

I'd like to thanks to Open Source Security and Embecosm for funding us to work on this project. Jeremy Bennett, for his management and the experience he brought to our development, and Brad Spengler, for being kind and benevolent.

I would also like to thank the entire Embecosm team for being so nice and caring towards me.

Particularly, I would like to thank Philip Herron for being my mentor, manager and especially friend when working on this project.

I would also like to thank my girlfriend for being so supportive and helpful during my stay in Germany.

Introduction

Currently, not many systems programming native language. But Rust is coming. However, only one implementation! Contribute to a second implementation (no details as that is for the Subject part)

In our current programming ecosystem, not many programming languages are usable to target small embedded architectures as well as large multithreaded applications. These languages, where speed of execution is a major focus, are mostly comprised of C and C++.

These two languages can be categorized as “systems-oriented”, and “native”, meaning that they are able to target even the lowest level of programming and are compiled directly to native instructions for the CPU. Programs compiled using these languages offer very small overhead and are extremely fast, at the cost of increased mental load for the programmer. What these languages are not, however, is “safe”. A recent study conducted by Microsoft (*source*) showed that around 70% of bugs found in their software were “memory issues”, where memory is not handled properly by the programmer: memory leaks, use-after-frees, double-frees, out-of-bounds accesses. . . are all common C/C++ programming mistakes that in turn can lead to vulnerabilities, exploitable by attackers. These numbers are not due to Microsoft’s lack of talent: The Google Chrome project reported the exact same number two years ago (*source*).

This issue has lead numerous companies to invest in programming language research, with the hopes of creating a safe, fast, systems-oriented native programming language, the most notable being Rust.

Rust is still quite a young programming language, being only around 15 years old, but offers a competitive alternative to C and C++. It focuses on safety as well as speed of execution, achieving speeds similar (or faster in some cases!) to programs written in C or C++. Furthermore, it also targets the embedded market, providing more expressivity than C.

However, a stark difference with C/C++ and brake to Rust’s adoption is the lack of specification or standard. Some companies do not consider the language stable or mature enough to earn a place in their technological stack. Furthermore, only one implementation of the language currently exists, the official `rustc` compiler. This compiler is written in Rust and thus faces bootstrapping issues. It also uses LLVM as its compiling framework, making it available on a lot of hardware architectures but not all.

In an effort to improve the reach of the language, be it in terms of the amount of people using it or for more niche architectures to use Rust programs, an alternative compiler is being developed. This compiler, `gccrs`, aims to integrate the Rust language among the GNU Compiler Collection project. The GNU Compiler Collection (GCC) contains multiple compilers for multiple languages, such as C (`gcc`), C++ (`g++`), Fortran (`gfortran`) or Ada (`gnat`). It is a pillar of the free software movement, making it more engaged than “regular” open source projects, and has been developed for more than 30 years, enabling it to target a multitude of architectures.

The GCC project is an old, complex codebase written in C++11, of around 19 million lines of code, making any changes to it extremely complex but also extremely interesting.

The goal of this internship was, overall, to contribute to the state of the compiler. More specifically, some complex Rust concepts such as macros, privacy restrictions or const generics were not handled yet, and needed to be worked on to achieve a valid Rust implementation as soon as possible.

FIXME: Add more? Yes! Talk about goal of compiling standard Rust library

Subject

The project originally started in 2014. However, the Rust language was not yet stable and saw constant changes, which made it extremely difficult to catch up to `rustc`. Due to this the project was put on hold for a time.

In 2019, Philip Herron started work on `gccrs` again and quickly obtained funding from Open Source Security to work on the project full-time. Philip has a lot of experience dealing with complex financial projects as well as compilers, with numerous contributions to `gcc` and alternative frontends already developed.

I joined the project one and a half year ago, in February 2021. I've always been very interested in compilers and Rust, and had already contributed to `rustc` as well as an alternative compiling backend (`rustc-codegen-craneflight`). `gccrs` seemed like an interesting project with very interesting goals, a lot of which are also very important to me. Later in 2021, I applied as a Google Summer of Code student to contribute to `gccrs`, and did so under the mentoring of Philip Herron. My project focused on integrating Rust's build system tool, `cargo`, to use `gccrs` as a compiler (project link). Once the project was finished, I got back to contributing to the compiler itself, focusing on the module system, for which I gave a talk at EPITA during the LSE Winter Days (link).

Soon after, Philip informed me that Brad Spengler from Open Source Security was looking into funding another engineer to work on the project. I was extremely interested in the offer, as I had grown very fond of the project and its community. Furthermore, this was a fantastic opportunity as compiler engineer jobs for open source projects are extremely hard to come across. The compiler still has a lot of interesting areas to work on, and every contribution counts. With dedication, it is extremely easy to make your mark on the project and learn tremendously while doing it. I accepted the very generous offer, and started working as an Embecosm employee, funded by Open Source Security.

While working on compilers does not directly relate to my major, it is still an extremely important subject. Embedded Software Engineers rely on compilers to produce safe, efficient and correct code, able to work on systems with very restricted resources. Rust, due to its focus on performance, is a perfect fit for embedded development as has been proved a number of times already. However, it still suffers from relying on LLVM, which does not support as many architectures as GCC. Working on an effort to add Rust to GCC means increasing the reach of the language, making it available on basically every platform shipping with a GCC compiler from the last 15 years.

Finally, an important part of embedded systems is systems programming: Low-level, complex, important pieces of code used to build more software. Compilers are an important part of systems programming, and face a lot of the challenges faced by low-level programmers.

FIXME: Can we add anything more to this? A little more content?

Positioning

FIXME: Say they want to enhance it through training, detection, bugfixes?

Open Source Security aims at enhancing security in core open source components used in the tech industry. Such components include Linux, the most used operating system in servers and datacenters in the world. The company fights vulnerabilities in multiple ways: Training, detection and bugfixing. Potentially vulnerable code can be identified in a number of ways: Static analysis and attentive reviews are the most common among them. One tool used to develop new static analysis passes and improve existing ones is compiler plugins: GCC offers ways for users to plug various pieces of code in its backend, allowing security researchers to search a code's GCC representation, while it is being compiled, for dangerous patterns.

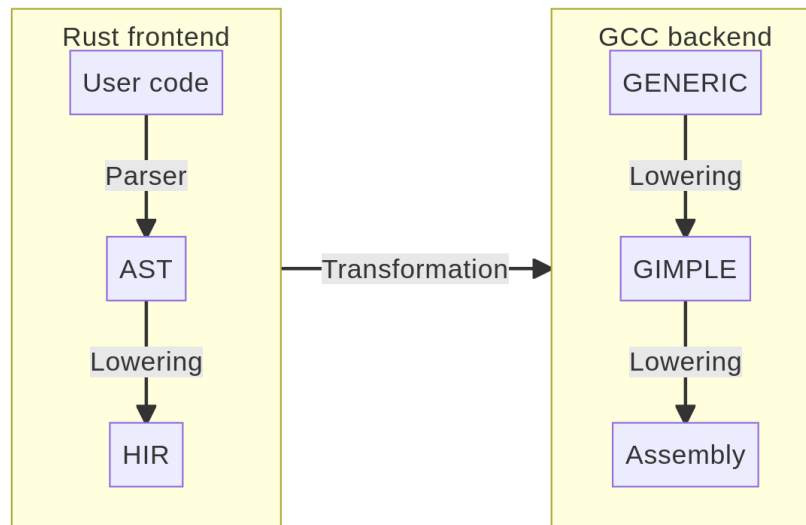


Figure 1: Compiler pipeline. Each major step (AST, HIR, GENERIC...) undergoes various transformations before being lowered

To understand at which level these plugins operate, one needs to understand how compilers typically work. First, the user's code, written in their favorite language, gets parsed and translated into an abstract syntax tree (AST). This tree represents the program in a different way, assigning each programming operation (addition, function call, creation of a variable...) to a type of "node" inside the compiler. This AST is then lowered to an internal, high level intermediate representation (HIR). This process usually involve various passes such as name resolving, disambiguation or macro expansion. The HIR, usually more detailed than the AST, gets through another set of transformations: type-checking, some optimisations, some lints, errors for the user about their mistakes, warnings... Finally, this HIR is lowered to a low-level intermediate representation. Of course, all compilers are different: `rustc` lowers its HIR to MIR (mid-level intermediate representation) before lowering it to LLVM IR (LLVM's intermediate representation), while, GCC compilers usually lower their HIR to GCC's intermediate language, `GENERIC` or `TREE`. This low-level representation is then optimized, analyzed, maybe inlined... and gets finally transformed into assembly language.

GCC plugins operate on `GIMPLE`, a subset of the `GENERIC` representation used by all GCC compilers. Simply put, this means that GCC plugins written to target C programs such as the

Linux kernel can also be used for all languages present in the GCC project. With `gccrs`, they could also be used for Rust programs, enabling even more safety than what the language already offers. Open Source Security, which has already written numerous of these plugins, aims to use them for upcoming Rust drivers in the Linux kernel, as the language is currently being integrated in the operatin system with the Rust-for-Linux project ([link](#)).

Embecosm, on the other hand, is a compiler company: They are tasked with integrating the GCC and LLVM projects for various customer architectures, or even new architectures entirely. The company's engineers have intimate knowledge of these compilers and help guide the `gccrs` project in the right direction by providing management and feedback. Furthermore, the company is extremely interested in the Rust language, providing support, training, and contributing to its existing implementation whenever possible.

Tether your subject and the company's field

3. Open Source Security provides GCC plugins for security purposes, notably for the Linux kernel.
4. Lot of interest around a second implementation of the Rust language which could benefit from said plugins.
5. Embecosm works in compilers
6. However, mostly backend

Present the market and its context

1. There's no market. It's not a market. Stop it.
2. People interested in funding the project maybe?
3. Rust-for-Linux!

Course of the internship

Parts of the internship

1. Development
2. Research
3. Promotion (talks, trade fairs)
4. Hiring (interviews)

Detailed Description

1. Same as above, detailed

Gantt Diagram

1. Add pretty diagram! How? Interlace the PDF?

Engineering approach

One or more structured approache(s)

1. The borrow-checker
2. The macro repetitions implementation
 1. Figuring out an algorithm

2. Issue reporting
3. Still more to do and maintain!

Concrete explanation

1. Show and explain what you did concretely (vs the company, the project, the team, the client...)
 1. Versus David Edelhson

Illustrated analysis

1. Detail 2 to 3 competences in relationship with your major

Added value of the internship

Internship's interest for the company

Internship's interest for the project

1. Qualitative and quantitative
2. Show your results

School's implication - On the school's side?

FIXME: Rework the title

1. Usage of concepts and methodologies used in class
2. Acquisition of new skills

Synthesis and Conclusion

Introspection

1. Between the beginning and the end of the internship

Evolution of your career plans/personal project

1. Confirmation or evolution of your professional project

Vision of the business world