COMS W6156 - Topics in Software Engineering

Spring 2019

Preliminary Project Report

Bruno Martins - bm2787

Introduction

Rust is a new (version 1.0 was released in 2015) systems programming language developed by Mozilla as a fast and safe alternative to the dominant C and C++ languages. In their words: "Rust is a systems programming language that runs blazingly fast, prevents almost all crashes, and eliminates data races". While most of the language features borrows judiciously from other languages, there is one central novel feature that seems to be unique to Rust: the borrow checker. The borrow checker statically evaluates the ownership and the lifetime of all pieces of data in a program to find and prevent entire classes of bugs like data races and use-after-free.

My project is to expand on the work I started for my midterm paper to explore more of Rust's features. I will continue to use the EPICS framework as a source of production-quality code.

Method

My main goal is to reimplement one of EPICS' modules. Initially, I had chosen to reimplement dbStatic (written in C), into Rust. The dbStatic module is responsible for parsing files in a Domain Specific Language and creating corresponding in-memory objects that will be used during the execution of an EPICS IOC (Input/Output Controller) program. This module is relatively self-contained, has a well-defined purpose and a number of test files that can be used to validate the eventual Rust implementation. However, I realized that this module has too many lines of code (6.2 kloc acoording to cloc).

Therefore, I chose to implement a different module, iocsh, to be reimplemented. iocsh is a simple shell that runs inside EPICS IOCs. The responsibilities of this shell are to parse commands given to it, find the parsed commands in a global registry of available commands, and execute them. This also involves parsing the arguments to give to the commands. It has only 1.5 kloc and sizable chunks of them can be replaced by Rust's standard library functions.

C/C++ to Rust transpilation attempts

In order to try to speed up the development, before I started writing any code, I attempted (unsuccessfully) to use a couple of existing C/C++ to Rust transpilers: CRUST and C2Rust.

- **CRUST** was simpler to install and compile, but the program execution "crashed" (panicked, in Rust's parlance) with an "Index out of bounds" error when I tried to use it.
- **C2Rust** is a much more complex and interesting project: it leverages LLVM (and therefore, clang) to apparently do analysis and parsing of the original code. It also uses a tool, called BEAR (Build EAR), that has to be used during a normal compilation of the original EPICS code in order to generate some metadata about the build process itself. Then, C2Rust can use the output of the BEAR program to aid in the transpilation. C2Rust itself took 9 minutes to compile. However, it *also* panicked when I used it, so it is also not ready for prime-time. In this case, it might have been due to EPICS relying on a lot of GCC-dependant code, which might not have made LLVM happy.

In the end, I believe that C/C++ and Rust are too different to have a functioning, general transpiler between them.

Current development

So far I have spent a lot of time trying different approaches to how to reimplement the shell in a way that preserves functionality and is fully interoperable with C. This has been much more difficult than I had anticipated. For example, the registry that iocsh maintains is implemented as a program-global hash table *and* a linked list of structures that describe the available commands, which is not protected by a lock (presumably because the registry is only populated during IOC startup, which is single threaded). Rust ***does not*** like global data like that, *at all*, so I had to fight the compiler for a while until I came up with a working solution. The next difficulty that I am facing is the integration between Rust and C code. I still have to find a way to link (in the linker sense) both Rust-generated and gcc-generated object files together. I believe these problems show the perils of choosing a fairly recent language for this task. However, there's a Rust users forum that seems to be very active. I used it to get some help with the problems I was having.

With that said, I believe once I overcome these two hurdles, the translation process is going to be straightforward. I think that having to shoehorn C-isms into Rust (to keep binary compatibility) is the difficult part, but once the interface between the two is resolved, Rust seems very pleasant to write in. The standard library is rich and there's a growing ecosystem of useful third-party libraries. I am fairly optimistic about being able to deliver the project.

It is worth noting that this project already allowed me to find a bug in EPICS itself (that leads to a SEGFAULT, no less), which I filed on EPICS' Issue Tracker (this bug would have been prevented by Rust, and would have been counted in my midterm paper).

Demonstration

In order to show that the project works, my plan is to design a simple IOC (that would be representative of a real-world IOC) and show that both the unmodified and modified versions of EPICS perform the very same way, with the exact same functionality. This would clearly demonstrate the feasibility of using Rust to gradually replace existing C/C++ code.

Code availability

The resulting work (code and report) will be made available on GitHub.

Why is this work relevant

This work is relevant, in my opinion, because Rust appears to be rising in popularity as a viable C and C++ replacement for systems programs. I am interested in this work because I mostly work with low-level systems and I am always looking for ways of making my programs better.