COMS W6156 - Topics in Software Engineering

Spring 2019

Preliminary Project Proposal

Bruno Martins - bm2787

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 proposal 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.

My main goal is to reimplement one of EPICS' modules, 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. It also presents an opportunity to test Rust's claims on having good binary interoperability with C.

The final reimplementation should be fully integrated with the EPICS build system and it should be capable of being used as a drop-in replacement for the original dbStatic module.

This work will help me evaluate Rust more comprehensively, regarding its ease of use and its safety claims. I will observe if any unknown bug will be revealed by the Rust implementation. I will run a performance comparison between the two implementations, even though I don't expect to see a big difference. I will note any difficulties I will eventually have in porting from C to Rust. If the port turns out to be too easy, I will pick an additional module (maybe iocsh, which has a different purpose but also has a clear purpose and is somewhat self contained) to also be reimplemented in Rust.

The resulting work (code and report) will be made available on GitHub. A successful demonstration will involve showing two versions of the same program, one from the original C code, one with the Rust reimplementation of dbStatic, working side by side.

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.

This is an example of a file written in the DSL that dbStatic expects. While the parser itself might end up being straightforward to write, I believe that the hardest part will be integrating the parser in Rust with the rest of the system.

```
include "menuGlobal.dbd"
include "menuConvert.dbd"
include "menuScan.dbd"
recordtype(arr) {
  include "dbCommon.dbd"
  field(VAL, DBF_NOACCESS) {
    prompt("Value")
    special(SPC_DBADDR)
    pp(TRUE)
    extra("void *val")
  }
  field(NELM, DBF_ULONG) {
    prompt("Number of Elements")
    special(SPC_NOMOD)
    initial("1")
  }
  field(FTVL, DBF_MENU) {
    prompt("Field Type of Value")
    special(SPC_NOMOD)
    menu(menuFtype)
  }
  field(NORD, DBF_ULONG) {
    prompt("Number elements read")
    special(SPC_NOMOD)
  }
  field(OFF, DBF_ULONG) {
    prompt("Offset into array")
  }
  field(BPTR, DBF_NOACCESS) {
    prompt("Buffer Pointer")
    special(SPC_NOMOD)
    extra("void *bptr")
  }
  field(INP, DBF_INLINK) {
    prompt("Input Link")
  }
  field(CLBK, DBF_NOACCESS) {
    prompt("Processing callback")
    special(SPC_NOMOD)
    extra("void (*clbk)(struct arrRecord*)")
  }
}
```