Proposal: Re-implementing a core utility in Rust

Authors:

John Deppe, k7g8@ugrad.cs.ubc.ca

Talha Khan, x5v0b@ugrad.cs.ubc.ca

Ambikeya Singh Sangwan, p6s1b@ugrad.cs.ubc.ca

Maike Basmer, h9k1b@ugrad.cs.ubc.ca

About the project

We have chosen the fourth project type: to create a substantial program in a language that we have not already studied. In doing so, we want to employ the features that distinguish the specific programming language from other programming languages.

The programming language in question will be Rust. Rust seems to be a serious alternative to C/C++; thus we hope it will be useful to learn more about this programming language. Above all, it is promoted for guaranteeing memory and thread safety, thus preventing pitfalls that have been common in C/C++ [1]. Rust has already been used in several applications for web programming, systems programming, cloud services and embedded environments among others [2].

Some of us have already gained experience in programming with C, and so we intend to study Rust features with regard to the characteristics of the C language. Comparing Rust to C, we want to find out how features have been realized in Rust and why those design decisions have been made. Therefore, we plan to rewrite one of the GNU coreutils by employing Rust's distinct features while comparing it to the baseline utility. Apart from that, there is lots of documentation available that enables us to thoroughly study the Rust features.

Project plan

For fulfilling the first and subsequent milestones, we intend to deal with resources about Rust that also enable us to get started with programming in Rust. We plan to meet up twice a week in order to discuss our insights. Thus, the first of the two weeks until the due date for the background report will serve for studying the material. Next, we will study the core utilities written in C and learn how they interact with the operating system. Doing this, we want to figure out specifically which core utilities will exhibit Rust's strengths. Our insights from these study sessions will then be synthesized in the background report during the second week.

For the proof-of-concept, we plan to finally choose a specific core utility, being careful that our reach does not exceed our grasp with respect to operating system concepts and time constraints, i.e. we don't intend to choose a core utility consisting of too many lines of code. Then, we will analyze the source code of the core utility in question and mark the segments of code where Rust's features would be well-applied.

Presenting at the poster session, we plan to pull examples from our background report in order to show the advantages of Rust over older systems programming languages. In addition to that, the marked-up code from the proof-of-concept could supplement this poster presentation as an appealing visual display.

Ultimately, for the 100% milestone, our goal is to actually re-implement the GNU core utility. We will have a solid foundation in terms of what exactly to optimize through the analysis we will have already completed. We are estimating that we will likely have to make sacrifices in functionality, as some edge cases may be beyond the scope of what we have learned in Rust or our grasp of the operating system arcana involved.

Resources

There is a wide abundance of resources available. Foremost, the Rust community itself as well as the Mozilla research team have either published or referenced several documents about Rust that range from a language reference, to a guide for beginners, to implementations of typical algorithms or a manual specifically for C/C++ programmers.

In order to learn and understand Rust, we will focus on the Rust language reference (https://doc.rust-lang.org/stable/reference/) as well as the book “The Rust Programming language” (https://doc.rust-lang.org/book/second-edition/). Reviewing the examples given on rustbyexample.com, scanning the blog series “Learning Rust” (https://medium.com/learning-rust) or watching the screencasts on intorust.com could help us deepen our knowledge. When it comes to writing a program in Rust, the Rust cookbook (https://github.com/rust-lang-nursery/rust-cookbook) might provide us with valuable insights in how to implement basic procedures or data structures.

Additional information on differences between Rust and older systems programming languages could be gathered from other resources such as the article on pointers in C and their equivalents in Rust (https://github.com/diwic/reffers-rs/blob/master/docs/Pointers.md), the manual “Rust for Systems Programmers” (https://github.com/nrc/r4cppp) or the book “Why Rust? Trustworthy, Concurrent Systems Programming” (http://www.oreilly.com/programming/free/files/why-rust.pdf). These documents will not be studied thoroughly but rather consulted whenever applicable. The resources mentioned previously and many more are listed on this website: https://github.com/rust-unofficial/awesome-rust

In addition to that, talks given by members of the Rust community are available online that could provide us with an overview of the language’s main features and specialties:

https://www.youtube.com/channel/UCaYhcUwRBNscFNUKTjgPFiA

https://www.youtube.com/watch?v=O5vzLKg7y-k

References

[1] The Rust Team. 2017. The Rust Programming Language. Retrieved November 1, 2017 from https://www.rust-lang.org/en-US/index.html.

[2] The Rust Team. 2017. Friends of Rust. Retrieved November 1, 2017 from https://www.rust-lang.org/en-US/friends.html.