Reimplementing a GNU Coreutil in Rust

|  |  |  |
| --- | --- | --- |
|  | Project Proposal |  |
| Maike Basmer  h9k1b@ugrad.cs.ubc.ca | John Deppe  k7g8@ugrad.cs.ubc.ca | Talha Khan  x5v0b@ugrad.cs.ubc.ca |

# ABSTRACT

This paper provides a proposal for the CPSC 311 Final Project. It lays out the project type and sketches an outline for accomplishing project milestones.

# ABOUT THE PROJECT

We have chosen to pursue the fourth project type: to create a substantial program in a language that we have not already studied. We plan to employ specific features that distinguish the chosen programming language from another programming language being used for the same purpose.

The programming language we have selected is Rust. Rust is being increasingly considered a viable alternative to C/C++ [9], and so learning it should prove useful beyond just the scope of this project and course. Its main traits are guaranteed memory and thread safety which prevent pitfalls that have been common in C/C++ [11]. Rust has already been used in many major applications in web programming, systems programming, cloud services, and embedded environments amongst others [11].

All members of our group already have some experience in programming with C, and so we intend to study Rust features by comparing with characteristics of the C language. We aim to find the reasoning behind the differences in language design and use those differences to make a practical improvement on existing software. Much of our research will be from existing Rust documentation, but rewriting one of the GNU Coreutils by employing Rust’s trademark features will allow us to directly compare the advantages and disadvantages of Rust’s language design. We plan on analysing “sort” to see how Rust’s memory management and multithreading design could replicate or improve on certain aspects of the existing code.

# PROJECT PLAN

For fulfilling the milestones, much of the initial work we do will involve dealing with resources about Rust that give us both an understanding of the language design as well as some experience programming in the language itself. We plan to meet up twice a week to discuss our insights. The first of the two weeks until the due date for the background report will be used for studying the material. We want to get a good grasp on how exactly we can use Rust’s guaranteed memory safety, threading without data races, and type inference to create a viable project. The last week before the background report deadline will be used to gain an understanding of the “sort” coreutil. Seeing as it is quite a large amount of code relative to the amount of time given for this project, we will have to be effective in focusing on relevant aspects.

For the proof-of-concept, we plan to very specifically outline the exact parts of “sort” that we can implement using Rust. “Sort” has many flags and options available and we need to be careful that our reach does not exceed our grasp with respect to time constraints, i.e. we don't want to pursue elements of “sort” that would require an overt amount of Rust knowledge to implement while producing a final product that isn’t necessarily substantial. 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, we will have code that has been annotated from the proof-of-concept that will supplement the poster presentation as an appealing visual display.

Ultimately, for the 100% milestone, our goal is to actually reimplement the significant parts of 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. Implementing this specific coreutil should also allow us to legitimate benchmark figures as well for a valid comparison between the languages. 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

Fortunately, there is a wide abundance of resources available for pursuing these milestones. Foremost, the Rust community itself as well as the Mozilla research team have published and referenced several documents about Rust that range from an official language reference, to a guide for beginners, to implementations of typical algorithms and a manual specifically for C/C++ programmers.

In order to learn Rust, we will focus on the Rust language reference [1], as well as the book “The Rust Programming Language” [2]. Reviewing examples [3], scanning the blog series “Learning Rust” [4] and watching screencasts by a member of the Rust core team [5] will also help deepen our knowledge. When it comes to writing a program in Rust, the Rust Cookbook [6] will provide the most valuable insight in how to implement basic procedures or data structures.

Additional information on differences between Rust and older systems programming languages will be gathered from resources such as an article found on pointers in C and their equivalents in Rust [7], the manual “Rust for Systems Programmers” [8], and the book “Why Rust? Trustworthy, Concurrent Systems Programming” [9]. We expect that these documents will probably not be studied thoroughly but rather consulted whenever applicable.

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 [10] [11]. Listed under the *References* section are further resources not necessarily specified above that we feel may prove useful in our research.

# REFERENCES

1. The Rust Project Developers. The Rust Reference. Retrieved October 25, 2017 from https://doc.rust-lang.org/stable/reference/
2. The Rust Project Developers. 2011. The Rust Programming Language (2nd. Ed.). Retrieved from https://doc.rust-lang.org/stable/reference/
3. The Rust Project Developers. Rust by Example. Retrieved October 25, 2017 from https://rustbyexample.com
4. Dumindu Madunuwan. 2017. Learning Rust: Rust Programming Language Tutorials. Retrieved from https://medium.com/learning-rust.
5. Nicholas Matsakis. Into\_rust(): Screencasts for Learning Rust!. Retrieved October 25, 2017 from http://intorust.com/
6. Rust Cookbook. Retrieved October 25, 2017 from https://rust-lang-nursery.github.io/rust-cookbook/
7. Github user *hdhoang*. 2016. I need to use pointers – now what? Retrieved from https://github.com/diwic/reffers-rs/blob/master/docs/Pointers.md
8. Github user *nrc*. 2017. Rust For Systems Programmers. Retrieved from https://github.com/nrc/r4cppp
9. Jim Blandy. 2015. *Why Rust? Trustworthy, Concurrent Systems Programming*. (1st. ed.). Sebastopol, CA.
10. Rust. 2016. Videos [YouTube Channel]. Retrieved from https://www.youtube.com/channel/UCaYhcUwRBNscFNUKTjgPFiA/videos
11. Aaron Turon. 2015. Stanford Seminar – The Rust Programming Language. Video. Retrieved from <https://www.youtube.com/watch?v=O5vzLKg7y-k>
12. Rust. Friends of Rust. Retrieved October 25, 2017 from https://www.rust-lang.org/en-US/friends.html