Modernising the `libc` Crate

Google Summer of Code Program 2025 Project Proposal

Abdul Muiz
EMAIL
UNIVERSITY
CITY, COUNTRY
PHONE NUMBER
TIMEZONE

Project Abstract: The **`libc`** crate has been a core part of Rust since its early days, making breaking changes increasingly difficult over time. Redundant and legacy code have complicated maintenance. This project aims to release a new major version of **`libc`**, introducing breaking changes and improving the testing infrastructure to ensure better long-term maintainability.

Project Scope: Medium (approximately 175 hours)

Project Mentor: Trevor Gross (GitHub, Zulip)

Project Description:

The goal of the project is to release **`libc v1.0`**, going through all of the issues as highlighted in 1.0 Milestone. Most importantly, the testing architecture of **`libc`** must also be overhauled. Currently, it makes use of **`garando_syntax`** to parse the bindings written in Rust and generate a Rust version of the test, and also generating a C version of the test (based on the **`libc`** crate itself) to compare the padding, size of the types, etc.

However, **`garando_syntax`** is outdated as it was based on an older version of the Rust syntax parser. This results in CI errors whenever it encounters new syntax (such as for example **`..=`** inclusive range syntax). We can therefore either rewrite **`ctest2`** such that it uses a syntax parser that better supports new syntax (such as **`syn`**) or we look into existing solutions, such as **`bindgen`**.

Project Deliverables

- **Automatic Testing Infrastructure -** Rewrite the existing infrastructure to support newer syntax, replacing **`garando_syntax`** with **`syn`**. Document the new library as well as tailor it more specifically to libc (or make it generic as needed).

- **Improved CI System** - Right now the CI system is definitely a bottleneck, requiring spurious jobs to be restarted as well as contributors being unable to sometimes test locally. Additionally, currently symbols are extracted automatically by the testing infrastructure, but due to how the **`ctest2`** crate works, prevents these symbols from being reused, causing the need of a **`semver`** file, causing duplication. While a complete rewrite would not be feasible, certain quality of life changes would help, especially given the fact that there are a large number of backed up issues because of this.

- **All issues under 1.0 Milestone taken care of** - Especially breaking changes, like removing Eq, Hash traits, removing placeholder constants, as well as no_std. Most of these issues are trivial in terms of complexity. Others require talking to the community and need a decision to be taken.

- **Libc 1.0 released -** This is the ultimate goal, and moreover, it is released in such a way that future contribution is extremely smooth for both the contributor (being able to run tests locally before performing a PR for example, which is something I was unable to do) as well as future GSOC aspirants.

Proposed Schedule

March 24 - May 8 (Pre-Acceptance Period)

Try to contribute towards helping reach the 1.0 Milestone as much as possible by completing smaller issues, as this will help me get to know the libc codebase well as well as figure out the main pain points for people who work on the crate as well as work with it.

May 8 - June 1 (Community Bonding Period)

- Going through the issues listed in the milestone, in order to figure out exactly what needs to be done and in what order. Creating a list of issues in order of priority, as well as the requirements and complexity of each issue.

- Go through the `ctest2` codebase and create a proper plan for rewriting it, figuring out the tooling required, major requirements, etc.

- Communicating with the mentor as well as others who have been discussing this new release of `libc` to understand requirements.

June 2 - July 14 (First Half)

Week 1-3

- Rewrite the visitor that parses and extracts symbols from the `libc` crate to use `syn`.
- Write the code required to convert these Rust specific symbols to C symbols to be used for the testing code.
- Write the boilerplate code required for both the Rust and C side of the testing code. Currently it is embedded within the other code, although abstracting it out would be better design as well as make it easier to modify.

Week 4-5

- Write the code to generate the tests using the extracted and converted C symbols.

Week 6

- `ctest2` rewrite completed and integrated into `libc`.
- For this integration, the CI scripts will be modified. The bulk of its improvement however will be in weeks 9-10.

July 14 - August 25 (Second Half)

Week 7-8

- Go through and complete all issues in the milestone that are marked as high and medium priority.

Week 9-10

- Improve the CI system, removing the need for duplication of symbols in the `semver` folder.
- Add targets that have been missing, according to the milestone.
- Make the CI system more robust when it comes to different versions of the linux kernel. (Maybe consider attribute macros to signify constants to skip for example)

Week 11

- Complete as many smaller issues leftover as possible.

August 25 - September 1 (Final Week)

- Spruce up documentation for the `ctest2` rewrite as well as all other code that was written.

- Update documents related to adding new bindings to conform to the new system.

Future Improvements

Further fixing of issues that have not been marked for the 1.0 Milestone. There are around 23 of these of varying difficulty.

Continued Involvement

I would like to continue helping the Rust Project on a volunteer basis even after the end of GSoC. I will be especially interested in improving support for Tier 3 microcontroller targets (although I do realise that these targets are mainly supported by the community and not the Rust Project itself).

Other Commitments

I intend to work around 3 hours a day for 5 days a week, which is 15 hours a week across 12 weeks, approximately 175 hours.

REDACTED INFORMATION ABOUT EXAMS

REDACTED INFORMATION ABOUT EXTRA CURRICULAR COMPETITIONS

This is why I propose prioritizing the testing infrastructure first. It's the main reason this project was proposed for GSoC and currently the biggest pain point for `libc`. Regardless of my other commitments, I'm committed to ensuring this part is completed. That said, I fully intend to give this project my all and work toward fixing as many issues as possible.

Major Challenges foreseen

- Rewriting the `ctest2` in a maintainable and platform agnostic way requires a large amount of planning as well as discussion with a mentor.
- Not every platform has the same structs and functions in the libc crate (for example due to kernel version differences), which currently is hacked around in the build scripts.
- Fixing spurious errors on the CI jobs, if they can even be fixed.

- Decisions about structure and how to perform breakage, which are numerous and must be discussed with mentor. Thus planning the rewrite is probably the most important phase.

References

- rust-bindgen (considered to minimise redundant code)
- syn (intended to replace `gerando_syntax`)

Relevant Background Experience

- I have previously made a PR for the `libc` project and intend to continue to do so.

Personal

REDACTED

Experience

Free Software Experience/Contributions:

- Volunteer Software Developer for Code For Pakistan, a Civic-Tech non-profit organisation. Worked on Watchtower, a performance-monitoring tool for government websites, focusing on backend data scraping. (Backend Repo)
- Multiple projects open sourced on GitHub, mainly for learning purposes, including a toy task scheduler written in Rust (currently WIP).

Language Skill Set

- Python (Fluent)
- Rust (Intermediate)
- C/C++ (Intermediate)

Reference Links and Web URLs:

- GitHub
- LinkedIn

(Proposal Template lovingly taken from RTEMS)