Improving User Errors

Abstract:

This project aims to improve user errors in gccrs by integrating rustc error codes into the GCC Rust front-end and making gccrs to emit error codes similar to rustc. This project requires the research of rustc error codes and their various guarantees, and we will then work on emitting these codes throughout the frontend code of gccrs. The project will also focus on fleshing out the code responsible for emitting errors to allow more functionality, and to start looking at a better user experience for gccrs by emitting more errors in more places with more hints for users on how to fix the code.

Motivation & Outcomes:

The goals of this project are to:

1. Allow gccrs to emit the same error codes as rustc, in order to help bridge the gap between the two test suites and enable the rustc test suite to eventually be run on gccrs.
2. Research rustc error codes and their various guarantees in order to emit them throughout the frontend code of gccrs.
3. Optimizing and testing (fleshing out) the code responsible for emitting error codes, to allow more functionality, such as emitting hints, notes, or suggestions on how to fix the code.
4. Improve the user experience of gccrs by emitting more errors, in more places, with more hints to the users about ways to fix the code.

Overall, the project aims to improve the usability of gccrs and make it more accessible to users & developers, while also bringing it closer to parity with rustc.

Milestones:

The milestones for this project:

1. Researching Rustc error codes:

- Gain a thorough understanding of the current implementation of error handling in gccrs.
- Research rustc error codes and their various guarantees.
- Identify areas where rustc error codes can be integrated.

2. Implementing error code emission:

- Collaborating with mentors to work on previously identified areas, where rustc error codes can be integrated into gccrs, and determining the appropriate error codes to use in each case.
- Updating the code responsible for emitting errors in the gccrs frontend to include rustc error codes, allowing for better interoperability with rustc compiler.
- Emitting rustc error codes throughout the codebase and ensuring that they are being used correctly according to their various guarantees.

3. Fleshing out the error emission code:

- Building on the basic error code emission, we will focus on expanding and improving the error emission code in gccrs, by refactoring, optimizing, testing and debugging error emitting code.
- We will work on adding support for additional error categories beyond the ones already implemented.
- We have to provide more detailed and informative hints to users on how to fix their code.

4. Integration with Rustc test suite:

- In this milestone, we will start integrating the changes made in previous milestones with the rustc test suite, in order to bridge the gap between our two test suites and enable us to eventually run the rustc one.
- This will need to modify the test cases to use the new error codes and verify that the tests pass with the new error messages.

Timeline:

- May 4 - May 28:
    - Community Bonding Period.
- Milestone 1: Researching rustc error codes (4 weeks):
    - May 29 - June 26.
- Milestone 2: Implementing error code emission (3 weeks):
    - June 27 - July 18.
- Milestone 3: Fleshing out the error emission code (2 weeks):
    - July 19 - August 2.
- Milestone 4: Integration with Rustc test suite (2 weeks):
    - August 3 - August 17.
- Buffer days for testing and finalizing the project:
    - August 18 - August 20.
- Final week: GSoC contributors submit their final work product and their final mentor evaluation:
    - August 21 - August 28.
- Mentors submit final GSoC contributor evaluations:
    - August 29 - September 4.
- Initial results of Google Summer of Code 2023 announced:
    - September 5.

Note: We have the option to merge milestones 2 and 3 if necessary, as we can flesh out the error emitting code during implementation. Also, be aware that the progress of the project may be subject to change due to unforeseen circumstances or adjustments needed to accommodate my university schedule during the spring and fall semesters. I will keep my mentors informed of any changes and work diligently to ensure that the project is delivered within the

timeline provided by GSoC. In the event, that we are unable to complete the project by the September 5 deadline, I am willing to discuss an extension with my mentors and work towards delivering the project as soon as possible within the revised timeline

My university schedule (for only spring 2023) is:

- April 8 (Sat) - April 11 (Tue): Second Sessional Exam.
- April 12 (Wed) - April 14: Normal Classes.
- April 17 - April 21: Normal Classes. (courses project evaluation)
- April 24 - April 28: Normal Classes.
- May 1 - May 5: Quiz 3.
- May 8 - May 12: Last week of classes.
- May 15 - May 19: Makeup classes.
- May 22 - June 10 (Sat): Final Exams.

Current Progress:

Currently, I have successfully reproduced several rustc error codes in gccrs by passing the error code to the *"rust_error_at"* function. The error codes that I have been able to reproduce are E0069 (mismatched types), E0117 (violation of rust orphan rules), E0124 (duplicate field in struct), E0133 (unsafe code used outside of an unsafe block), and E0093 (unrecognized intrinsic function). You can view the implementation of this approach and a sample video in this commit. Moreover, there are some flaws in this implementation e.g., If I change "***return***" to "***return()***" in the code mentioned in E0069, the error code in rust1 changes from E0069 to E0308. This change was observed on godbolt.org. Despite the emitting point of both errors being the same in gccrs, gccrs emitted E0069 instead of E0308. During my GSoC journey, I will resolve these issues, in my milestone 3 within the guidance and support of my mentors.

My Contributions to RUST-GCC/gccrs:

I have been a member of RUST-GCC/gccrs since the start of this year (2023) and I have made two contributions so far. In my first contribution, I fixed errors in the documentation and refactored the README.md file, which was merged into gccrs via pull request #1751. In my second contribution, I made changes to the file rust-buffered-queue.h by moving it to util/rust-buffered-queue.h, shown in pull

request #1816. Additionally, my approach to fix issue-1702 is shown here. A full list of my pull requests to RUST-GCC/gccrs can be found here.

About the Contributor:

Name: Muhammad Mahad.
Email: mahadpy@gmail.com (personal)
l216195@lhr.nu.edu.pk (university)
LinkedIn: mmahad.
GitHub: MahadMuhammad
University: FAST-NUCES, Lahore, Pakistan.

I am currently a second-year student pursuing a bachelor's degree in computer science at FAST NUCES, Lahore. Throughout my academic career, I have gained proficiency in C/C++ and Linux system programming. I am also an active contributor to the open-source community, and I am currently helping my university fellows in their assembly language course by creating and maintaining a repository titled *"Learn Assembly the Hard Way"* on GitHub.
In my free time, I enjoy exploring the art of calligraphy in both English and Arabic, which allows me to express my creativity and sharpen my attention to detail.