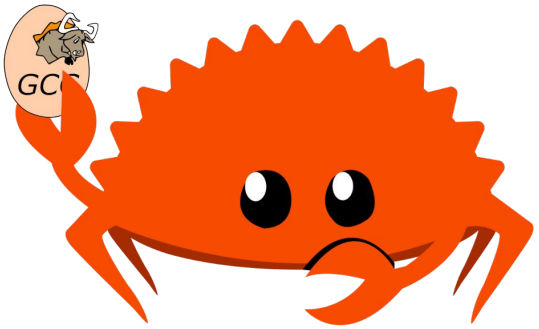
EMBECOSM®

OPEN SOURCE SECURITY

ORACLE

Rust-GCC



Philip Herron
David Faust

Summary

- This is a discussion not a talk
- Quick overview of project
- Versioning
  - Rust editions
  - Stability
    - unstable features
- Making things easier
  - Command line arguments

Overview

- Full implementation of Rust on GNU toolchain
  - Use GNU `as`, `ld`, ...
  - Entirely distinct alternative to `rustc`
  - Brings Rust to new platforms and users
- Rust language front-end for GCC
  - Macro and `#[cfg]` expansion
  - Type checking + inference
  - Checks, lints (privacy, unsafety, unused vars, ...)
  - Lower HIR to GENERIC and pass off to rest of GCC
- GCC plugin support: LTO, CFI
- Reuse `libcore`, `libstd`, `libproc`

Overview - Current Status

- Last steps to compile `libcore` 1.49 - winter
- Const generics
- Intrinsics
- Borrow Checking - `polonius`
- Passing `rustc`'s testsuite

Rust-for-Linux Unstable Features

- https://github.com/Rust-for-Linux/linux/issues/2
- `feature (receiver_trait)`
  - Tracking Issue: None (internal to the compiler).
  - Used by: rust/kernel/sync/arc.rs
  - Status: ?
- `cfg(no_rc)` and `cfg(no_sync)`
  - Related to custom `liballoc`?
- `-Zsymbol-mangling-version=v0`
  - Stabilized in 1.59, will be new default
- and many more...
- Unstablized = hard to guarantee equivalence between gccrs and rustc
  - How important are these unstable features for RfL?
  - What is the path to stabilization? (in an Edition?)

Versioning

- What version of Rustc does Rust for Linux require?
  - Rustc updates can change any or all of the following:
    - dependency updates (eg LLVM)
    - security fixes
    - bug fixes
    - language changes
    - stability updates
  - Can we rely on targeting a Rust Editions
    - 2015, 2018, 2022
  - Targeting a specific Rustc version
    - This seems overly granular and subject to change

Making things easier

- gccrs exposes the classic gcc interface eg:
  - gccrs -g -O2 -c src/lib.rs -o src/lib.o
  - All gcc flags are exposed and available just as you would for C/C++
    - some flags are enabled by default
    - we need to document these
  - Our cargo wrapper could be used to help here
    - map rustc command line arguments gcc arguments

More Discussion Points

- Custom `liballoc`?
- Procedural macros (`libproc`) - used by RfL?
- Bindings and FFI - `libc` crate does not provide bindings for targets not supported by LLVM
  - How does RfL handle these targets (if at all)?
- Compiler integration with `libcore`/`libstd`
  - `for` loops are NOT lang items - rustc calls into libcore


Questions?

github.com/Rust-GCC/gccrs/

gcc-rust.zulipchat.com/

irc.oftc.net #gccrust