Rust-Torino 20190418

Cross building with rust

Initial checklist

- Is the target supported by rustup?
- Is your OS providing the compilers and the libraries you need?
- Does it need other compilers beside the standard ones?
  (e.g. `cc-rs` or `nasm-rs` are in the dependency tree)

We assume here that all those are true for today.

Preparation

- You **NEED** to install the **linker** and **libc** for your target
- Make sure **it works** by compiling a `hello.c` with the companion *C* compiler.
- Make sure the `rust` target and the `gcc` / `clang` target do match.
- Be sure to edit `.cargo/config` to have the `linker` name matching the one provided by your system.

Optionally configure qemu

- `qemu-user` would let us run our programs
- Make sure to point qemu to the right runtime path by setting `QEMU_LD_PREFIX` or passing `-L`.
- Make sure you set the `LD_LIBRARY_PATH` in case `libgcc_s.so` is provided by the cross compiler only.
- Add it as `runner` in our `.cargo/config`.

Setup

Our testbed is easy: `aarch64-unknown-linux-gnu` on `Gentoo` (using rustup to manage the rust parts).

- Install the needed components:

```
# crossdev -S aarch64
# emerge qemu # USE=static-user QEMU_USER_TARGETS=aarch64
# rustup target add aarch64-unknown-linux-gnu
```

- prepare the .cargo/config

```
[target.aarch64-unknown-linux-gnu]
linker = "aarch64-unknown-linux-gnu-gcc"
runner = "qemu-aarch64 -L /usr/aarch64-unknown-linux-gnu/'
```

Test

Let's try to make a really simple `hello`.

```
# cargo new hello
# cd hello
# export P=/usr/lib/gcc/aarch64-unknown-linux-gnu/8.2.0
# LD_LIBRARY_PATH=$P cargo run --target=aarch64-unknown-l
    Finished dev [unoptimized + debuginfo] target(s) in 0
     Running `qemu-aarch64 -L /usr/aarch64-unknown-linux-
Hello, world!
```

> **NOTE:** We set the `LD_LIBRARY_PATH` to `/usr/lib/gcc/aarch64-unknown-linux-gnu/8.2.0` since that's the cross compiler we have.

Pitfalls

- `cc-rs` reads the env var `CC`, set it accordingly

```
CC=aarch64-unknown-linux-gnu-gcc cargo build --target aar
```

- `build.rs` is correctly built for the `HOST`, make sure you do not have faulty assumptions there.
- The default `features` might be wrong for your target.
    - `--no-default-features` is the best workaround.
    - Per-target features does not seem to work.

Summary

- cross compiling in rust is relatively *simple* as long the target is provided by `rustup` and the OS provides the toolchain you need.
- `crossdev` in Gentoo and crosstool-ng can make your life much easier.
- `qemu-user` as runner makes everything simpler for running the tests.