actually.fyi

Zig Makes Rust Cross-compilation Just Work

2021-05-22, Max Hollmann (#zig, #rust, #cross-compilation)

> This post is a spiritual successor to Loris Cro's Go cross-compilation.

The encounter

During a recent stage 2 meeting, Jakub Konka wanted to demo his progress on WASM support, but ran into problems with screen-sharing on Linux. After switching to his Mac to run it there, he found out that Wasmtime (a WASM runtime) doesn't ship `aarch64-macos` pre-built binaries.

But Wasmtime is an open-source Rust project, I should be able to clone the repo and build it for Jakub myself. The problem? I don't have an M1 Mac on hand. I will have to cross-compile.

The idea

Rust has built-in cross-compilation support. All you have to do is install the target's toolchain with `rustup`

```
rustup target add aarch64-apple-darwin
```

and build your program with `--target`.

```
cargo build --target aarch64-apple-darwin
```

The problems

The C dependencies

```
warning: cc: error: unrecognized command-line option `-arch`
error: failed to run custom build command for `wasmtime-fiber`
Caused by:
  ...
  running "cc" ...
```

Some Rust crates depend on C code, but the Rust toolchain does not include a C compiler. This is the same problem Loris encountered with CGO and, luckily for us, it has the same solution. Create a shell script which wraps Zig[1]

```
#!/bin/sh
/path-to-zig/zig cc -target aarch64-macos $@
```

and add it to env variables as `CC` when invoking `cargo`

```
CC="/path-to-wrapper/zcc" cargo build --target aarch64-apple-darwin
```

The linker

```
error: linking with `cc` failed: exit code: 1
  |
  = note: "cc" ...
```

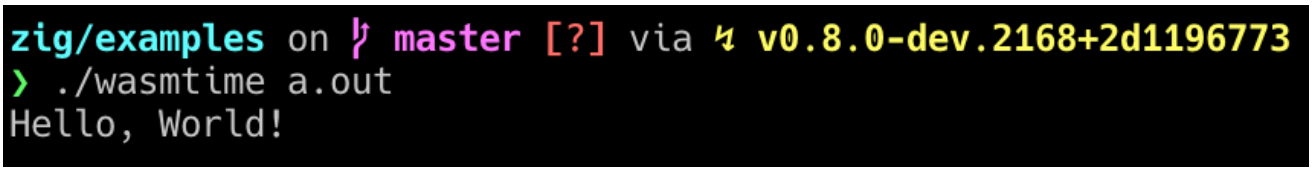
We're now successfully compiling Wasmtime's objects, but we're failing at the last step - linking the final binary.

The Rust toolchain includes the Rust standard library and a compiler to produce object files for the target, but it does not include a linker to link them into the final executable. Furthermore, the `cargo` documentation instructs us to provide it with a target linker in `.cargo/config.toml`. However, we can see that `cargo` is attempting to invoke `cc` like it did before with `wasmtime-fiber`. Can we reuse the same wrapper we used before?

Yes[2].

```
[target.aarch64-apple-darwin]
linker = "/path-to-wrapper/zcc"
```

The result



Success! This is a WASM "Hello, World!" example running on Wasmtime on Jakub's M1, but the Wasmtime binary was compiled inside WSL on my x86_64 PC[3]. Pretty wild.

1. I'm using absolute paths to refer to `zig` because I don't want to add nighlty binaries into `$PATH`. I recommend switching to system-wide install once the 0.8.0 branch gets released. ↩

2. For `aarch64-macos` support we're interested in Zig's self-hosted linker, which is slated for release in 0.8.0. I'm using the latest build from Zig's download section. Built on 2021-05-20. All newer builds should work. ↩

3. This method of cross-compiling Rust is not limited to aarch64 MacOS from x86_64 Linux. If your host and target is supported by Rust and Zig, just change their `target` arguments and install Rust's toolchain. You don't have to install additional `gcc` toolchains for each target. ↩