Packaging a new toolchain

Angus Lees <gus@debian.org>
Rust!

- New programming language
- In the C/C++ end of town
- Built on LLVM
- Very close to Rust 1.0 release
- Rust compiler is written in Rust
  - Circular build dependency!
Challenges

- Bootstrapping the first package
- Bootstrapping new architectures
- x86_64-linux-gnu != x86_64-unknown-linux-gnu
- Rust is still a fast moving language
  - Need very narrow version of rustc to build next rustc
Compiler stages

stage0
- Pre-existing compiler used to build stage1

stage1
- Minimal compiler (no libraries, etc) used to build stage2

stage2
- Full compiler used to build stage3

stage3
- Full compiler distributed to users (ideally identical to stage2)
Bootstrapping the first package

Rust is usually built using a stage0 binary downloaded during the build process. Yuck.

No way around it - must be built using an existing Rust compiler built “somehow.”
Whither stage0?

- Download during build
- Bundle pre-built stage0 in Debian source package
- Use previous rustc package and iterate as often as required
Bootstrapping: The first package

debian/control

Source: rust
Build-Depends: rustc
Package: rustc

Source: rust
Build-Depends:
  rustc <!stage1> | rustc-bootstrap <!stage1>
Package: rustc
Package: rustc-bootstrap

dpkg-buildpackage -Pstage1
- Build stage1 rustc-bootstrap.deb using non-Debian rustc
- Build regular rustc.deb using rustc-bootstrap
ifneq (,,$(findstring stage1,\$(DEB_BUILD_PROFILE)))
  DEB_MAKE_BUILD_TARGET = rustc-stage1
  DEB_MAKE_CHECK_TARGET = check-stage1-rpass
  DH_OPTIONS += --package=rustc-bootstrap
endif
Boostrapping new architecture

- Assume rustc can cross-compile to target
- `dpkg --add-architecture s390x`  
  `apt-get source --compile \`  
  `--host-architecture s390x \`  
  `--build-profiles stage1`
- Basically these set DEB_BUILD_* and DEB_TARGET_* environment variables before invoking `debian/rules`
More Information

- [https://wiki.debian.org/DebianBootstrap](https://wiki.debian.org/DebianBootstrap)
Rust the language

- Looks like C/C++
- Uses LLVM
- Types from Haskell, interfaces/channels from go, etc
- Strong ownership of data
  - Data lifetime built in to compiler
  - Move semantics by default
- All the cleverness is in the compiler
  - Very minimal runtime
  - The result can easily call to/from C, for example
  - Embedded, kernel modules, etc possible
Rust the language

Punting this part of the talk to Rust by example

See also http://rust-lang.org/ and Rust Programming Language Book