mobile haskell user guide Documentation

Release latest

May 12, 2018

Contents

1 Introduction **3**
1.1 Docker Image for Raspberry Pi 3
1.2 Installation 3

2 Troubleshooting **5**
2.1 Segfaults on Raspberry Pi 5

Table of Contents

- *Mobile Haskell User Guide*
  - *Introduction*
    - *Docker Image for Raspberry Pi*
    - *Installation*
  - *Troubleshooting*
    - *Segfaults on Raspberry Pi*

CHAPTER 1

Introduction

This is the accompanying User Guide for building mobile and embedded Haskell application using the The Glorious Glasgow Haskell Compilation System as a cross compiler.

1.1 Docker Image for Raspberry Pi

If your target is a Raspberry Pi image, you can use a pre-defined docker image instead of installing the toolchain on your local machine. You'll still need to create a `sysroot` folder yourself, with the headers and libraries (see Making a Raspbian Cross Compilation SDK for more details).

With the `sysroot` ready, you can run the following command:

```
$ docker run -it -v /path/to/sysroot:/rpi/sysroot tritlo/ghc-to-rpi
```

To launch a docker container where you can cross-compile to your Raspberry Pi by running:

```
$ arm-linux-gnueabihf-ghc
```

For easy access, you can add `-v $(pwd):/code` to the docker launch command to have the current directory mounted to the `/code` directory in the container.

1.2 Installation

Pre-built binary distributions that target iOS (arm64, x86_64), Android (armv7, arm64, x86_64) as well as Raspberry Pi (armv6) for macOS Sierra and linux (deb8) can be dowloaded from http://hackage.mobilehaskell.org. Other architectures may be added at a later date but will for now be built from source. Using hadrian as the build system is highly recommended.

TODO: Document building from source.

The cross compiler use the LLVM code generator. As such an LLVM installation is required and needs to be in `PATH`. The LLVM toolchain provided by Xcode is insufficient as it does not provide the `opt` tool.

LLVM can be downloaded from http://releases.llvm.org/download.html#5.0.0. Extracting and adding the `bin` folder to `PATH` should be sufficient:

```
$ export PATH=/path/to/llvm/bin:$PATH
```

The pre built GHCs are relocatable, as such it is enough to simply extract them and add them to the `PATH`:

```
$ export PATH=/path/to/ghc-x86_64-apple-ios/bin:$PATH
```

To provide a unified interface over the cross compilers the general scheme of the tool chain is `$target-tool`, e.g. for `ghc` targeting `aarch64-apple-ios`, the tools are:

This unified interface is provided via the toolchain-wrappers for the pre-built cross compilers. After downloading the toolchain-wrappers, running the `bootstrap.sh` script and adjusting the `linux-android-toolchain.config` and `raspberrypi-toolchain.config` files to match the local Android NDK and Raspberry Pi SDK (TODO: building a raspberry pi SDK), the toolchain should be usable:

```
$ git clone https://github.com/zw3rk/toolchain-wrapper.git
$ (cd toolchain-wrapper && ./bootstrap)
$ export PATH=/path/to/toolchain-wrapper:$PATH
```

CHAPTER 2

Troubleshooting

This section details a few known bugs.

2.1 Segfaults on Raspberry Pi

Anything that uses the *time* package directly or indirectly may unexpectedly segfault. It appears that the `__tzfile_read` function from `glibc` segfaults when called via `tzset` from GHC. The exact reasons why this happens are unknown right now. A suitable workaround is to set the TZ environment variable to a value that's not a zonefile when launching the application. E.g. `$ TZ="UTC0" ./Application`.