pintos for Mac: 2017 edition

Levente Kurusa
Email: levex@linux.com

January 19, 2017

1 Installing cross-compilers

The goal of this guide is to fix the pintos sources (and your Mac) so that you can coexist with LabTS and Linux-using friends without much hassle that the previous guides have implied.

1.1 Rationale

macOS runs on the Mach-O executable file format, however Linux runs on ELF format. The reason we are using custom compilers is because on macOS the "gcc" you would use automatically resorts to "clang" (or LLVM) for generating machine code from assembly. Unfortunately, clang can't produce ELF object files on macOS, so hence, you can't get the calling "gcc" to produce ELF files either. We therefore, resort to a custom compiler that is compiling for the operating system "none" (meaning it won't expect anything like a standard library running, in other words, we are compiling for bare-metal), and executable format "elf" for the architecture "i686". (i686 is backwards compatible with i386). NB: These custom compilers cannot produce valid executables for your Mac system.

1.2 Compiling your cross-compilers

See Appendix A if you are curious and/or don't trust my binary files. Otherwise, it's safe to skip.

1.3 Getting your pre-compiled cross-compilers

You can download them from here:

`http://www.doc.ic.ac.uk/~lk1015/i686-elf-gcc6.20-binutils2.27.zip`

Once the download finishes, extract the zip file and place the `com2/bin` folder in your $PATH .

2 Patching up pintos code

NOTE: These patches will *not* break the compilation on Linux/LabTS.

2.1 Rationale

Patch 1 modifies the central `Make.config` file to instruct downstream `Makefile`s to automatically use the custom i686-none-elf compilers if it detects that the running system is Darwin (i.e. macOS or OS X)

Patch 2 silences a warning from `QEMU` that would say some write operations are disabled to disk files that are *detected* as raw images. Naturally, by telling `QEMU` that the images are indeed raw, it will let us write uncontrolled.

Patch 3 will fix the infamous "host CPU timeout" bug that occurs on macOS. According to the Perl documentation, using `alarm` and `setitimer` can cause problems (which is what happens here). To fix this, force using the supplied `setitimer-helper` instead of resorting to Perl.

Patch 4 will remove references to the STREAMS functionality of POSIX in some utils/ code. STREAMS is not implemented by macOS, and since it's not critical, we will just remove code that uses code from `sys/stropts.h`

Patch 5 is not required, but makes a top-level `make check` cleaner. However, it will still fail because it can't find the solutions folder (for obvious reasons).

2.2 Getting the patches

The patches are available at `https://www.doc.ic.ac.uk/~lk1015/pintos-mac/`
Apply each one of them via the command `git am < patchfile.patch`

3 Building setitimer-helper

In order for **Patch 3** to work, we need to build the util `setitimer-helper`. This is really easy to do and can be done via `cd`'ing into `src/utils` and issuing the command `make`
NOTE: You may see some warnings mentioning `fail_io`, but it's safe to ignore them.

4 Enjoy!

Everything should be working as expected now, on your Mac, on LabTS and on Linux machines, but don't forget to add `src/utils` to your `$PATH` in order for the `pintos` script to work.

5 Getting GDB to work

GDB faces the same issue as GCC, so we need a specific GDB to debug pintos:

`http://www.doc.ic.ac.uk/~lk1015/i686-elf-gdb-7.12.zip`

Once you have downloaded the zip, extract the contents somewhere permanent and place `com-gdb/bin` in your `$PATH`. (probably via `~/.bash_profile`).

Once you can access `i686-elf-gdb` from the command line, it is time to fixup Pintos source code:

If you've applied **Patch 6** previously then you can skip this step, you are good to go.

Otherwise, download **Patch 6** from the folder linked above (in section 2.2) and apply it via the same method. You should be able to debug pintos via gdb now.

6 Special thanks

Special thanks to **Thomas Bower**, **Zubair Chowdhury**, **Atanas Gospodinov**, **Norbert Podsadowski**, **Joseph Katsioloudes** and **Daniel Zvara** for their help in testing out this guide.

This guide has been tested on:

- 2016 MacBook Pro w/o Touchbar running macOS Sierra
- 2014 MacBook Pro running macOS Sierra
- 2015 MacBook Pro running macOS Sierra

A Appendix: Compiling cross compilers

This is intended for curious/advanced users, if you just want to get pintos up and running quickly you don't have to do this!
However, if you are going down this road then be prepared that this can take at least an hour to build.

A.1 Requirements

You need to have the following packages installed in order for the following to work.

- existing `gcc` and `g++` compilers
- make
- bison
- flex
- gmp
- mpfr
- mpc
- texinfo

These can usually be found in the package manager of your choice.

A.2 Preparation

Firstly, create a new folder somewhere where all the following steps will take place. Next, let's setup the environment by creating a few new variables:

```
export PREFIX=WHERE_YOU_WANT_YOUR_BINARIES_TO_GO_TO
export TARGET=i686-elf
export PATH="$PREFIX/bin:$PATH"
```

A.3 Binutils

The binutils package contains the assembler and various useful tools that you (at least) need for compiling gcc.

A.3.1 Getting binutils sources

Go to the main GNU FTP server (`ftp://ftp.gnu.org/gnu/binutils/`)and download the latest `binutils-X.XX.tar.gz` file. (This guide was using 2.27). You should then extract it using `tar xzvf binutils-X.XX.tar.gz`

A.3.2 Building binutils

In the folder where you extracted it, issue the following commands:

```
mkdir binutils-build
cd binutils-build

../binutils-X.XX/configure --target=$TARGET --prefix="$PREFIX" --with-sysroot
--disable-nls --disable-werror

make && make install
```

If this succeeded, congratulations you have i686-elf-as in your `$PATH`!

A.4 gcc

We now need to use the previously compiled binutils to create our new cross compilers.

A.4.1 Getting gcc sources

Similarly as for binutils, get the gcc source code from the GNU FTP (`ftp://ftp.gnu.org/gnu/gcc/`). Then, extract it using `tar xzvf gcc-X.Y.Z.tar.gz`. (This guide was using 6.20)

A.4.2 Building gcc

In the folder where you extracted the file, issue the following, similar, commands:

```
mkdir gcc-build
cd gcc-build

../gcc-X.Y.Z/configure --target=$TARGET --prefix="$PREFIX" --disable-nls
--enable-languages=c,c++ --without-headers

make all-gcc && make all-target-libgcc
make install-gcc && make install-target-libgcc
```

If this succeded, then you now have the cross compilers! Congratulations!