Tools Used in 6.828

You'll use two sets of tools in this class: an x86 emulator, QEMU, for running your kernel; and a compiler toolchain, including assembler, linker, C compiler, and debugger, for compiling and testing your kernel. This page has the information you'll need to download and install your own copies. This class assumes familiarity with Unix commands throughout.

We highly recommend using a Debathena machine, such as athena.dialup.mit.edu, to work on the labs. If you use the MIT Athena machines that run Linux, then all the software tools you will need for this course are located in the 6.828 locker: just type '`add -f 6.828`' to get access to them.

If you don't have access to a Debathena machine, we recommend you use a virtual machine with Linux. If you really want to, you can build and install the tools on your own machine. We have instructions below for Linux and MacOS computers.

It should be possible to get this development environment running under windows with the help of Cygwin. Install cygwin, and be sure to install the `flex` and `bison` packages (they are under the development header).

For an overview of useful commands in the tools used in 6.828, see the lab tools guide.

Compiler Toolchain

A "compiler toolchain" is the set of programs, including a C compiler, assemblers, and linkers, that turn code into executable binaries. You'll need a compiler toolchain that generates code for 32-bit Intel architectures ("x86" architectures) in the ELF binary format.

Test Your Compiler Toolchain

Modern Linux and BSD UNIX distributions already provide a toolchain suitable for 6.828. To test your distribution, try the following commands:

```
% objdump -i
```

The second line should say `elf32-i386`.

```
% gcc -m32 -print-libgcc-file-name
```

The command should print something like `/usr/lib/gcc/i486-linux-gnu/`*`version`*`/libgcc.a` or `/usr/lib/gcc/x86_64-linux-gnu/`*`version`*`/32/libgcc.a`

If both these commands succeed, you're all set, and don't need to compile your own toolchain.

If the gcc command fails, you may need to install a development environment. On Ubuntu Linux, try this:

```
% sudo apt-get install -y build-essential gdb
```

On 64-bit machines, you may need to install a 32-bit support library. The symptom is that linking fails with error messages like "`__udivdi3` not found" and "`__muldi3` not found". On Ubuntu Linux, try this to fix the problem:

```
% sudo apt-get install gcc-multilib
```

Using a Virtual Machine

Otherwise, the easiest way to get a compatible toolchain is to install a modern Linux distribution on your computer. With platform virtualization, Linux can cohabitate with your normal computing environment. Installing a Linux virtual machine is a two step process. First, you download the virtualization platform.

- **VirtualBox** (free for Mac, Linux, Windows) — Download page
- VMware Player (free for Linux and Windows, registration required)
- VMware Fusion (Downloadable from IS&T for free).

VirtualBox is a little slower and less flexible, but free!

Once the virtualization platform is installed, download a boot disk image for the Linux distribution of your choice.

- Ubuntu Desktop is what we use.

This will download a file named something like `ubuntu-10.04.1-desktop-i386.iso`. Start up your virtualization platform and create a new (32-bit) virtual machine. Use the downloaded Ubuntu image as a boot disk; the procedure differs among VMs but is pretty simple. Type `objdump -i`, as above, to verify that your toolchain is now set up. You will do your work inside the VM.

Building Your Own Compiler Toolchain

This will take longer to set up, but give slightly better performance than a virtual machine, and lets you work in your own familiar environment (Unix/MacOS). Fast-forward to the end for MacOS instructions.

Linux

You can use your own tool chain by adding the following line to `conf/env.mk`:

```
GCCPREFIX=
```

We assume that you are installing the toolchain into `/usr/local`. You will need a fair amount of disk space to compile the tools (around 1GiB). If you don't have that much space, delete each directory after its `make install` step.

Download the following packages:

- ftp://ftp.gmplib.org/pub/gmp-5.0.2/gmp-5.0.2.tar.bz2
- https://www.mpfr.org/mpfr-3.1.2/mpfr-3.1.2.tar.bz2
- http://www.multiprecision.org/downloads/mpc-0.9.tar.gz
- http://ftpmirror.gnu.org/binutils/binutils-2.21.1.tar.bz2
- http://ftpmirror.gnu.org/gcc/gcc-4.6.4/gcc-core-4.6.4.tar.bz2
- http://ftpmirror.gnu.org/gdb/gdb-7.3.1.tar.bz2

(You may also use newer versions of these packages.) Unpack and build the packages. The `green bold` text shows you how to install into `/usr/local`, which is what we recommend. To install into a different directory, $PFX, click here. If you have problems, see below.

```
tar xjf gmp-5.0.2.tar.bz2
cd gmp-5.0.2
./configure --prefix=/usr/local
make
make install             # This step may require privilege (sudo make install)
cd ..

tar xjf mpfr-3.1.2.tar.bz2
cd mpfr-3.1.2
./configure --prefix=/usr/local
make
make install             # This step may require privilege (sudo make install)
cd ..

tar xzf mpc-0.9.tar.gz
cd mpc-0.9
./configure --prefix=/usr/local
make
make install             # This step may require privilege (sudo make install)
cd ..


tar xjf binutils-2.21.1.tar.bz2
cd binutils-2.21.1
./configure --prefix=/usr/local --target=i386-jos-elf --disable-werror
make
make install             # This step may require privilege (sudo make install)
cd ..

i386-jos-elf-objdump -i
# Should produce output like:
# BFD header file version (GNU Binutils) 2.21.1
# elf32-i386
#  (header little endian, data little endian)
#   i386...


tar xjf gcc-core-4.6.4.tar.bz2
cd gcc-4.6.4
mkdir build              # GCC will not compile correctly unless you build in a separate directory
cd build
../configure --prefix=/usr/local \
    --target=i386-jos-elf --disable-werror \
    --disable-libssp --disable-libmudflap --with-newlib \
    --without-headers --enable-languages=c MAKEINFO=missing
make all-gcc
make install-gcc         # This step may require privilege (sudo make install-gcc)
make all-target-libgcc
make install-target-libgcc     # This step may require privilege (sudo make install-target-libgcc)
cd ../..

i386-jos-elf-gcc -v
# Should produce output like:
# Using built-in specs.
# COLLECT_GCC=i386-jos-elf-gcc
```

```
# COLLECT_LTO_WRAPPER=/usr/local/libexec/gcc/i386-jos-elf/4.6.4/lto-wrapper
# Target: i386-jos-elf


tar xjf gdb-7.3.1.tar.bz2
cd gdb-7.3.1
./configure --prefix=/usr/local --target=i386-jos-elf --program-prefix=i386-jos-elf- \
    --disable-werror
make all
make install             # This step may require privilege (sudo make install)
cd ..
```

Linux troubleshooting

Q. I can't run `make install` because I don't have root permission on this machine.

A. Our instructions assume you are installing into the `/usr/local` directory. However, this may not be allowed in your environment. If you can only install code into your home directory, that's OK. In the instructions above, replace `--prefix=/usr/local` with `--prefix=$HOME` (and click here to update the instructions further). You will also need to change your `PATH` and `LD_LIBRARY_PATH` environment variables, to inform your shell where to find the tools. For example:

```
export PATH=$HOME/bin:$PATH
export LD_LIBRARY_PATH=$HOME/lib:$LD_LIBRARY_PATH
```

Enter these lines in your `~/.bashrc` file so you don't need to type them every time you log in.

Q. My build fails with an inscrutable message about "library not found".

A. You need to set your `LD_LIBRARY_PATH`. The environment variable must include the `PREFIX/lib` directory (for instance, `/usr/local/lib`).

MacOS

First begin by installing developer tools on Mac OSX:
`xcode-select --install`

You can install the qemu dependencies from homebrew, however do not install qemu itself as you will need the 6.828 patched version.

```
brew install $(brew deps qemu)
```

The gettext utility does not add installed binaries to the path, so you will need to run

```
PATH=${PATH}:/usr/local/opt/gettext/bin make install
```

when installing qemu below.

QEMU Emulator

QEMU is a modern and fast PC emulator. QEMU version 2.3.0 is set up on Athena for x86 machines in the 6.828 locker (`add -f 6.828`)

Unfortunately, QEMU's debugging facilities, while powerful, are somewhat immature, so we highly recommend you use our patched version of QEMU instead of the stock version that may come with your distribution. The version installed on Athena is already patched. To build your own patched version of QEMU:

1. Clone the IAP 6.828 QEMU git repository `git clone https://github.com/mit-pdos/6.828-qemu.git qemu`
2. On Linux, you may need to install several libraries. We have successfully built 6.828 QEMU on Debian/Ubuntu 16.04 after installing the following packages: libsdl1.2-dev, libtool-bin, libglib2.0-dev, libz-dev, and libpixman-1-dev.
3. Configure the source code (optional arguments are shown in square brackets; replace PFX with a path of your choice)

1. Linux: `./configure --disable-kvm --disable-werror [--prefix=PFX] [--target-list="i386-softmmu x86_64-softmmu"]`
2. OS X: `./configure --disable-kvm --disable-werror --disable-sdl [--prefix=PFX] [--target-list="i386-softmmu x86_64-softmmu"]` The `prefix` argument specifies where to install QEMU; without it QEMU will install to `/usr/local` by default. The `target-list` argument simply slims down the architectures QEMU will build support for.

4. Run `make && make install`

Questions or comments regarding 6.828? Send e-mail to the TAs at *6828-staff@lists.csail.mit.edu*.