Lab CSE3211: Assignment 0 - lab01

Marks : 10%

Complete by Next Week

Build Toolchain OS161

To build yourself an OS/161 toolchain, and generally get ready to work on OS/161, you need to do the following. Note that if you're taking a course and your course provides the toolchain installed somewhere for you, you don't need to do any of this. (Unless you're the lucky guy on the course staff who gets to set it up.)

Toolchain components

There are five parts to the toolchain:

- binutils, which provides the assembler and linker and miscellaneous other programs like `nm` and `ar`.
- gcc, the compiler. The current toolchain version uses gcc 4.8. This requires OS/161 1.99.08 or newer; if you are not ready to update OS/161, stick to gcc 4.1 from the previous toolchain version.
- gdb, the debugger.
- System/161, the machine simulator.
- bmake, aka BSD make, the make dialect OS/161's build system is written in.

In general you will need to build and install all of these, even if you already have a compiler; the versions we'll be building and using with OS/161 are configured for cross-compiling to MIPS.

However, if you already have BSD make (it is the system make on most BSD-based operating systems, and comes with the `pkgsrc` package system often also used on MacOS and Solaris) you don't need to install another copy.

Note that you might be able to use a different compiler, such as clang, as long as it has been configured as a MIPS cross-compiler. This has not been tested, however, and you're on your own. (If you try, please report success or failure...)

Preliminaries

- Choose some directories. We'll assume you're going to use `~/os161/toolbuild` to compile, and `~/os161/tools` to run the tools from. If you want to use other directories, such as directories named after the course you're taking, perform the appropriate substitutions below.
- Create some directories:

```
mkdir ~/os161
mkdir ~/os161/toolbuild
mkdir ~/os161/tools
mkdir ~/os161/tools/bin
```

- Add `~/os161/tools/bin` to your `$PATH` and log back in. This is necessary before building because of a long-standing bug in the gcc makefiles.

Prerequisites

There are some prerequisite packages for building the toolchain. You will need the following:

- A C compiler capable of compiling gcc.
- The base "development" library packages, e.g. for libc or glibc, so you can compile things.
- GNU make. Neither BSD make nor vanilla Unix make will do for gcc.
- The GNU math libraries gmp, mpfr, and mpc, required by gcc 4.8. (gcc 4.1 does not require these.)

The best bet is to install these items (if you don't already have them) with your system's package manager. In most package systems where you don't already start out with a compiler, there's an umbrella package to install the things you need to compile; the best bet is to start off by installing this.

*If you are using Linux, your system make (what you get if you just type `make`) is most likely GNU make.

***For library packages, like gmp, mpfr, and mpc, be sure to install the "development" packages as well, if they're separate.**

*If you're using pkgsrc (e.g. on BSD, MacOS, or Solaris) the mpc package is called **"mpcomplex".** There's something else called mpc; installing it instead by accident won't help much.

*If you cannot get a gmake package and need to compile it, you can download it from ftp.gnu.org. This is also true of gmp, mpfr, and mpc.

Installing binutils

You must install binutils first; gcc will not build properly without it.

- Build the binutils distribution archive into a scratch directory.
- Extract the archive.

```
tar -xvzf binutils-2.24+os161-2.1.tar.gz
```

- Touch some of the extracted files to avoid trying to rebuild them and possibly failing:

```
cd binutils-2.24+os161-2.1
find . -name '*.info' | xargs touch
touch intl/plural.c
cd ..
```

- Configure, compile and install binutils. Be sure to use `$HOME` instead of just ~ as shown; ~ is not interpreted except at the beginning of a shell word.

```
   cd binutils-2.24+os161-2.1
   ./configure --nfp --disable-werror --target=mips-harvard-os161
--prefix=$HOME/os161/tools
   make
   make install
   cd ..
```

Installing gcc 4.8

- Configure and build the gcc 4.8 distribution archive into a scratch directory.
- Extract the archive.

```
   tar -xvzf gcc-4.8.3+os161-2.1.tar.gz
```

- Touch some of the extracted files to avoid trying to rebuild them and possibly failing:

```
   cd gcc-4.8.3+os161-2.1
   find . -name '*.info' | xargs touch
   touch intl/plural.c
   cd ..
```

- Create a build directory for gcc, as gcc trips on its own feet if you try to build it directly in its source directory:

```
   mkdir buildgcc
```

- Configure gcc.

```
   cd buildgcc
   ../gcc-4.8.3+os161-2.1/configure \
        --enable-languages=c,lto \
        --nfp --disable-shared --disable-threads \
        --disable-libmudflap --disable-libssp \
        --disable-libstdcxx --disable-nls \
        --target=mips-harvard-os161 \
        --prefix=$HOME/os161/tools
   cd ..
```

  That configure invocation needs to be all one line.

- Once configuration succeeds, compile and install gcc.

```
   cd buildgcc
   gmake
   gmake install
   cd ..
```

*If your system make is GNU make (e.g. on Linux) you can just type 'make' instead of 'gmake' at the appropriate points.

*If you or your package system has installed gmp, mpfr and/or mpc into a directory that the compiler doesn't find by default (such as `/usr/pkg` or `/opt`, pass one or more of these extra arguments to `configure` to help it find them, obviously substituting the right path prefix:

```
        --with-gmp=/usr/pkg
        --with-mpfr=/usr/pkg
        --with-mpc=/usr/pkg
```

On some platforms you may also need to stick the library path in the `LDFLAGS` environment variable before running `configure`:

```
   LDFLAGS=-R/usr/pkg/lib
   export LDFLAGS
```

or

```
   setenv LDFLAGS -R/usr/pkg/lib
```

* I also had trouble with gcc's Chinese translation files failing to build; if you hit this, you can work around it by adding `--without-nls` to the configure arguments.

*I've had a report that gcc 4.8 doesn't build on the latest Ubuntu (16.10) but I haven't had a chance to investigate.

Installing gcc 4.1

(Only install gcc 4.1 if you aren't installing gcc 4.8.)

- Configure the gcc 4.1 distribution archive into a scratch directory.
- Extract the archive.

  ```
     tar -xvzf gcc-4.1.2+os161-2.0.tar.gz
  ```

- Touch some of the extracted files to avoid trying to rebuild them and possibly failing:

  ```
     cd gcc-4.1.2+os161-2.0
     find . -name '*.info' | xargs touch
     cd ..
  ```

  (There is no plural.c in gcc 4.1.)

- Configure, compile and install gcc.

  ```
     cd gcc-4.1.2+os161-2.0
     ./configure --nfp --disable-shared --disable-threads --disable-ibmudflap \
          --disable-libssp --target=mips-harvard-os161
  --prefix=$HOME/os161/tools
     gmake
     gmake install
     cd ..
  ```

  The configure invocation needs to be all one line.

*If your system make is GNU make (e.g. on Linux) you can just type 'make' instead of 'gmake' at the appropriate points.

Installing gdb

- Configure the gdb distribution archive into a scratch directory.
- You may need patch gdb from ‘mips-harvard-os161-gdb’ as:

  patch -p1 -i ${srcdir}/gdb.patch
- where ${srcdir} is the path where mips-harvard-os161-gdb.tar.gz’ extracted
- Then extract the gdb archive.

```
tar -xvzf gdb-7.8+os161-2.1.tar.gz
```

- Touch some of the extracted files to avoid trying to rebuild them and possibly failing:

```
cd gdb-7.8+os161-2.1
find . -name '*.info' | xargs touch
touch intl/plural.c
cd ..
```

- Configure, compile and install gdb.

```
cd gdb-7.8+os161-2.1
./configure --target=mips-harvard-os161 --prefix=$HOME/os161/tools
make
make install
cd ..
```

Installing System/161

- Configure the System/161 distribution archive into a scratch directory.
- Extract the archive.

```
tar -xvzf sys161-1.99.09.tar.gz
```

- Configure, compile and install System/161.

```
cd sys161-1.99.09
./configure --prefix=$HOME/os161/tools mipseb
make
make install
cd ..
```

*If you are using `pkgsrc` for packages (e.g. on BSD, MacOS, or Solaris) you can install the sys161 package (it is in `emulators/sys161`) to get System/161.

Installing bmake

If you already have bmake, you don't need another copy. You may have an adequate bmake if:

- You are working on NetBSD (it's called `make`)
- You are working on FreeBSD, OpenBSD, or Dragonfly (it's called `make` too; but these versions are slightly different. I believe they will work; success/failure reports would be appreciated.)
- You are working on Linux and your Linux distribution includes a "bmake" package. Sufficiently recent Debian does; Ubuntu may have picked it up also. Encourage your Linux distribution maintainer to ship bmake.
- You are using `pkgsrc` for packages (e.g. on MacOS or Solaris); the bmake it installs is the same bmake.

*There is something floating around the Linux world called "pmake" that's an extremely old BSD make. It is known *not* to work. (If your distribution ships this, encourage them to get rid of it and ship bmake instead.)

*You should be able to use any version of bmake shipped within the last ten or more years. However, be advised that the (dysfunctional) build procedure of the standalone bmake distribution changes periodically. The directions here work with the 2010 version posted in the download area but probably not with other arbitrarily chosen versions.

- Build the bmake and "mk" distribution archives into a scratch directory.
- Extract the archives. Important note: `mk.tar.gz` needs to be unpacked *inside* bmake. (Don't blame me! I didn't set it up that way.)

```
   tar -xvzf bmake-20101215.tar.gz
   cd bmake
   tar -xvzf ../mk-20100612.tar.gz
   cd ..
```

- Configure, compile and install bmake. As noted above, if you already have bmake you don't need to install another copy.

```
   cd bmake
   ./configure --prefix=$HOME/os161/tools --with-default-sys-
path=$HOME/os161/tools/share/mk
   sh ./make-bootstrap.sh
   mkdir -p $HOME/os161/tools/bin
   mkdir -p $HOME/os161/tools/share/man/man1
   mkdir -p $HOME/os161/tools/share/mk
   cp bmake $HOME/os161/tools/bin/
   cp bmake.1 $HOME/os161/tools/share/man/man1/
   sh mk/install-mk $HOME/os161/tools/share/mk
   cd ..
```

Note that the build system has some issues. The configure script will tell you that you can just run`./make-bootstrap.sh`, but if you do that it doesn't work; you need to feed it to the shell as shown above. And you have to install by hand because no install mechanism is provided. There is an additional script called `boot-strap`; it works in a different but equally unsatisfactory manner.

Finishing up

Because the tools installed under long GNU names (e.g. `mips-harvard-os161-gcc`) you probably want to create shorter names like `os161-gcc`. Do the following to make a family of symbolic links:

```
cd ~/os161/tools
sh -c 'for i in mips-*; do ln -s $i os161-`echo $i | cut -d- -f4-`; done'
```

Now, type `rehash` or log in again, and you should be ready to go.

*If you have a multiprocessor/multicore machine, you can do parallel makes with bmake. Use the option `-j N`, where N is the maximum number of jobs to run at once. Twice the number of cores you have seems to be a good number nowadays.

Testing Your Installation with DUMBVM kernel

To test your installation, you need a kernel. To do so un-tar os161 kernel as,

t***ar xvf os161-base-2.0.3.tar.gz***

cd os161-base-2.0.3

Configure DUMBVM kernel:

cd kern/conf

./config DUMBVM

Compile the kernel:

cd ../compile/DUMBVM

bmake depend

bmake

Install the kernel:

bmake install

If successful, your new kernel should be installed in $HOME/os161/root directory. Then,

cd ~/os161/root

Now, you need a sys161.conf file to boot the kernel. Therefore, in ~/os161/root directory rename the existing sys161.conf.sample to sys161.conf as

cp sys161.conf.sample sys161.conf

If you don not find the 'sys161.conf' in ~/os161/root directory, the copy it from sys161-2.0.8 to ~/os161/root directory.

Before executing the kernel create disk images in ‘~/os161/root’ as,

disk161 create LHD0.img 5M

disk161 create LHD1.img 5M

It should create 5MB disk images in ‘~/os161/root’.

Executing the kernel. At $HOME/os161/root execute the following command on sys161 machine

sys161 kernel

Press ‘q’ to quit or power-off the kernel. That’s all.