Cross Compiling The Linux Kernel

Introduction

I freely admit there are other HOWTOs on this subject, but I wanted to show you a neat way that rides on the back of the great **Buildroot** project, instead of doing all the compiler setup by hand. As a bonus you can target different architectures pretty easily.

For this demonstration I will build a 32-bit Slackware kernel on a 64-bit machine. Yes I know you can potentially do this with Multilib (though I've never tried) and other methods, however this method is quite simple, will allow targeting MIPS, m68k, Microblaze, PowerPC, SPARC and God knows what else, all with the same technique, and you don't even need to be root to do it. Interested? Read on.

About Buildroot

The Buildroot project has been running for some years and allows one to, quite literally, build a root filesystem (duh), however we will completely ignore the rootfs building part of the equation and just use the cross-compiler which it builds first. Since I have a fast machine I didn't even bother working out how to skip the rootfs generation step. You only have to build your cross-compiler once after all.

Slackware Install

My test cross-compiler build machine is Slackware64 14.2 with disk sets A,AP,D,K,L,N. I'm sure you can install a lot less to build Buildroot, in fact I'd expect only A and D to be needed, however the spirit of the Slackware community is that you install everything so the risk is yours.

Cross-compiler

Grab Buildroot latest stable version from https://buildroot.org, unpack and configure it:

```
$ cd /usr/src
$ wget https://buildroot.org/downloads/buildroot-2018.02.2.tar.gz
$ tar xf buildroot-2018.02.2.tar.gz
$ cd buildroot-2018.02.2
$ make menuconfig
```

You will see a configuration system much like the kernel config. For building 32-bit kernels with maximum compatibility I generally select the 486 output option, (586 is the default):

```
Target options -> Target Architecture Variant (i486)
```

To make the compiler we are about to build behave more like the Slackware one we want to use glibc instead of the default uClibc-ng (which is more suited to embedded applications):

```
Toolchain -> C library (glibc)
```

If you don't do this you will need to disable stack protection in your kernel config when we come to compile that and we want to keep a standard Slackware config, because we're true Slackers heart-and-soul right? :).

You can play around with many other options like kernel header versions however for building the kernel itself none of these matter. The only option that may conceivably make a difference is the GCC version, particularly if you are building an old kernel version which doesn't support later versions of GCC. For this demonstration we can leave the defaults though. Save the config and then:

```
$ make
```

While that's running we'll configure the kernel so it's ready to compile.

Kernel preparation

Pull down a kernel config appropriate to building a 32-bit Slackware kernel:

```
$ cd /usr/src/linux
$ wget
https://mirrors.slackware.com/slackware/slackware-14.2/kernels/hugesmp.s/con
fig -O .config
```

You should now have a .config in /usr/src/linux which says 'CONFIG_64BIT is not set' at the top. That will replace your old 64-bit kernel .config that you had before (included from the 'K' disk set). Obviously copy the kernel someplace else if you wanted to keep that!

Kernel compilation

When the Buildroot build is done, you need to include the generated cross-compiler in your path.

```
export PATH=/usr/src/buildroot-2018.02.2/output/host/bin:$PATH
```

The compiler executable has the architecture prefix in it's name to avoid collision with the system GCC, you can now run it and test if it works:

```
$ i486-linux-gcc --version
```

If you are interested, you can find all the other toolchain tools like ld, ar and so on with similar prefixes. Now configure your kernel if you want to:

```
$ cd /usr/src/linux
$ make menuconfig CROSS_COMPILE=i486-linux- ARCH=i386
```

I'm leaving the default options here, just appending a '-buildroot' to the kernel name. Finally make the kernel:

```
$ make bzImage CROSS_COMPILE=i486-linux- ARCH=i386
```

Copy the built kernel to a 32-bit machine and it should boot. If you want to compile/install the modules as well, just make sure you don't forget to use the same CROSS_COMPILE and ARCH variables every time you specify the make commands, everything should use the cross-compiler:

```
$ make modules CROSS_COMPILE=i486-linux- ARCH=i386
$ make modules_install CROSS_COMPILE=i486-linux- ARCH=i386
```

and so on. You will probably get away without these appended for some commands like 'make clean', but it's safest to just include them whenever you do any work on that kernel, they certainly won't hurt.

Sources

- Originally written by User bifferos

howtos, nfs, author bifferos