基于 Arm 的 Linux 内核编译指导

psmax2008 发表于 2006-5-18 15:30:00

This guide will give you step by step instructions about compiling a kernel for ARM machines on the target platform. If you find something missing or unclear, please send a mail to the linux-arm list.

Here are some initial notes to help you understand the terminology and conventions used in this document:

"bash$" and "bash#" are shell prompts, not commands to be typed.
"host" means the machine you are building the ARM kernel on.
"target" means the machine you are building the ARM kernel for.
Decide where to build your kernel
Firstly, you need to decide where you are going to build your ARM Linux kernel. A good place to build the kernel is in your home directory, which we will refer to as $HOME thoughout this document. If you wish to use some other location, replace $HOME as appropriate.
However, please note that building a kernel in /usr/src/linux is highly discouraged.

Deciding on a kernel version
Firstly, you need to decide which version of the Linux kernel you wish to compile. Most people will want the latest stable kernel release. To help you identify a particular release, a system of version numbers is used by the kernel developers.
For any kernel version x.y.z,

x - This is the major revision number
y - This is the minor revision number, where:
Even numbers indicate "stable" kernel releases
Odd numbers indicate "development" or "beta" kernel releases which may be less stable.
z - This is the patch level of the kernel
This version number represents the main line kernel version.

At the time of writing this document, the latest stable kernel on ARM is 2.6.6.

Under the ARM kernel tree, you will find a suffix to the kernel version number: -rmkN, or -vrsN where 'N' is the patch release number. For instance, 2.4.26-vrs1. This indicates the version of the main ARM kernel patch which should be applied. (note: kernels later than 2.6.0-test2 do not require a -rmk or -vrs patch to be applied since ARM architecture support is already merged.)

Other maintainers, such as Nicolas Pitre, may produce additional patches, and these will add an

additional suffix to denote their version. Nicolas Pitre's patches add a -np suffix, eg 2.4.21-rmk2-np1.

This means that 2.4.21-rmk2-np1 is based upon 2.4.21-rmk2, which in turn is based upon the 2.4.21 kernel. Therefore, to get the kernel source for 2.4.21-rmk2-np1, you need the 2.4.21 main line kernel source, the 2.4.21-rmk2 patch and the 2.4.21-rmk2-np1 patch.

Therefore, if you need extra patches from other maintainers, you have to be careful to choose an appropriate kernel version. You will need to locate the maintainer patches first, and then work your way up through the version number towards the mainline kernel version.

Downloading the maintainer-specific patch.
In some circumstances, you will need to patch the kernel with a maintainer specific patch. These patches add extra features or other device drivers which may be specific to various machines. However, as a general rule, maintainers forward upstream parts of their patches into the -rmk or -vrs trees as and when they are happy with the change.
Please refer to the machine list for information concerning extra patches.

Downloading an ARM patch.
(You only need this step if you are using a kernel prior to 2.6.0-test2. There are no -rmk or -vrs patches for later kernels.)
You may need to download a kernel patch, which contains all the ARM specific updates for a particular kernel version. These can be found in ftp://ftp.arm.linux.org.uk/pub/armlinux/source/kernel-patches/. The kernel releases are separated out into directories corresponding to the major and minor version numbers of the kernel.

The individual files are named patch-x.y.z-rmkN.gz or patch-x.y.z-vrsN.gz, where 'x', 'y', 'z' and 'N' are the version numbers mentioned above. You should select and download the latest patch for the kernel into the $HOME directory. This is the one which will have either the most features, or the most bug fixes in. You will need the version of the patch later when downloading the main kernel source.

Note: Some files may be named (eg) pre-patch-x.y.z-rmkN.gz. These are alpha or beta patches, which are probably unstable. You should not use these unless you are sure that you know what you are doing, and you don't mind the target system being unstable. However, they are useful when new ideas need to be tested out.

Note2: Some kernels are based on the Alan Cox series of kernels. These have names similar to patch-x.y.z-acm-rmkN.gz where x.y.z is Linus' version number and m is Alan's version number. In this case, you will need to obtain Alan Cox's corresponding patch from the kernel.org servers, in the directory /pub/linux/kernel/people/alan/linux-2.4/.

Downloading the main line kernel source

A patch file on its own usually does not contain any compilable code. It is a machine-readable description of changes to make to a set of text files (in this case, the kernel source.) You need to obtain the main kernel source tree.

The kernel source can be found on one of the kernel.org FTP sites. There are many sites scattered around the world, and are named according to a unified naming scheme. All sites start with 'ftp.' and end in '.kernel.org'. In the middle is placed a country identifier. For example:

ftp.uk.kernel.org
ftp.us.kernel.org
ftp.de.kernel.org

and so forth. You can find out more information on these sites by looking at the main http://www.kernel.org/ site.

Once you have selected a site, you need to find the kernel sources. They will be stored in the subdirectories of /pub/linux/kernel. Each kernel release is accompanied by several files:

linux-x.y.z.tar.gz
linux-x.y.z.tar.bz2
patch-x.y.z.gz
patch-x.y.z.bz2

You will want to download the linux-x.y.z.tar.gz file, again into your $HOME directory. Again, you should look for a version which matches the version of the patch you obtained above. These files are large (about 14MB or more), so if you are on a slow connection, be prepared for it to take some time.

Unpacking the ARM kernel source

Unpack the tar archive you downloaded above using:

```
bash$ cd $HOME
bash$ tar zxvf linux-x.y.z.tar.gz
```

This will create a directory called linux or linux-x.y.z in your home directory. Change into the newly created directory and apply the patch files, eg:

```
bash$ cd linux-2.4.26
bash$ zcat ../patch-2.4.26-vrs1.gz | patch -p1
```

The patches are heirarchial, so you need to apply them in the correct order. The patch files with more extensions depend on the ones with less extensions, so you need to apply, for example, the -rmk patch before the -rmk-np patch.

The kernel source tree is now ready to be configured.

Configuration of the kernel build environment

Normally, the kernel build system will build the kernel for the native machine architecture. This is not appropriate when cross compiling, so you will need to change two lines in the top level kernel Makefile. Examine the top level Makefile in an editor and find the definitions for ARCH and CROSS_COMPILE. On 2.4.x kernels, they will look like this:

```
ARCH := $(shell uname -m | sed -e s/i.86/i386/ -e s/sun4u/sparc64/ -e s/arm.*/arm/ -e s/sa110/arm/)

[...]
CROSS_COMPILE	=
```

and on 2.6.x kernels:

```
ARCH		?= $(SUBARCH)
CROSS_COMPILE	?=
```

Edit these two lines to read:

```
ARCH		?= arm
CROSS_COMPILE	?= /usr/local/bin/arm-linux-
```

replacing /usr/local/bin/arm-linux- with the path to your ARM Linux toolchain.

This completes the configuration of the top level kernel makefile. The next step is to configure the kernel build to select the drivers that your platform requires.

You may like to read linux/README and linux/Documentation/arm/README before proceeding. Both these files provide further useful information which may be specific to your kernel version.

Configuration of the kernel sources

There are a range of 'make' targets which allow a set of defaults to be selected for the particular machine you are compiling the source for. The process is much simpler for 2.6 kernels.

Configuration of 2.4 kernels

For 2.4 kernels use <machinename>_config format, for example:

a5k_config
ebsa110_config
netwinder_config
rpc_config
assabet_config

You should select one of these as the "basic" configuration as follows, and run make oldconfig immediately afterwards:

```
bash$ make netwinder_config
bash$ make oldconfig
```

The oldconfig step will prompt you for any new configuration options which may have been added since the default machine configuration file was submitted. It is normally safe to say 'N' to these new options.
Note: If you want to change the configuration via make xxx_config, please remove the file linux/.config immediately prior to executing this command.

Configuration of 2.6 kernels

For 2.6 kernels, the process is similar. Use <machinename>_defconfig to select the machine, eg:

```
bash$ make netwinder_defconfig
```

In this case, there is no need to run a separate oldconfig step.

Compiling the kernel source

If you are only installing the kernel source tree for other programs, then you have finished. If you want to compile up a new kernel, type the following commands:

```
bash$ make clean
bash$ make dep
bash$ make zImage
bash$ make modules
```

The final two commands will actually compile the kernel and the kernel modules.
Note: With 2.6 kernels, the make dep stage is not necessary.

Installing the kernel

After the kernel has successfully compiled, you should have the kernel image, arch/arm/boot/zImage. What you do next depends if you are cross compiling or not.
If you are cross compiling, goto the section "Installing a cross compiled kernel".

If you are building natively (ie, for the target on the target), continue.

Installing a native kernel

Since you are about to upgrade system files, you need to become 'root'. To do this, type:

```
bash$ su
Password:
bash#
```

It is highly advisable to keep a backup of your current kernel and modules. What you need to do is machine dependent. Note that it is a good idea to always keep a known good previous version of the kernel and modules in case you need to back down to a previous version.
The following is given as an example (for a 2.4.3-rmk1 kernel):

```
bash# cd /lib/modules
```

```
bash# mv 2.4.3-rmk1 2.4.3-rmk1.old
bash# cd /boot
bash# mv vmlinuz vmlinuz.bak
bash# mv System.map System.map.bak
bash#
```

Now, install the new kernel modules:

```
bash# cd $HOME/linux
bash# make modules_install
bash#
```

This will copy the modules into the /lib/modules/x.y.z directory. Next, install the kernel image (normally into /boot):

```
bash# cd /boot
bash# cat $HOME/linux/arch/arm/boot/zImage >vmlinuz
bash# cp $HOME/linux/System.map .
bash#
```

Note that the command to copy the new kernel image is cat and is not the usual cp. Unix traditionally will not allocate space on the filesystem to sections of files containing zero data, but instead creates "holes" in the file. Some kernel loaders do not understand files with holes in, and therefore using cat in this way ensures that this does not happen.

Running loadmap

Loadmap is part of the Linux loader on Acorn machines, as well as EBSA285 machines using EBSA285BIOS with an IDE disk. For other machines, please refer to the documentation for your machine.

Edit the loader configuration file /etc/boot.conf so that you can boot either the vmlinuz.bak or vmlinuz images. If you place the vmlinuz kernel first, then this will be the default kernel which the kernel loader will use.

More information can be found by typing man boot.conf.

Run the boot loader map utility:

```
bash# loadmap -v
bash#
```

to update the maps.

You have finished, and are now ready to reboot your machine and try out your new kernel! If you experience problems, please go to the "Problems" step below.

Installing a cross compiled kernel

Kernel modules are installed into the /lib/modules/x.y.z directory on the target system, though this will normally be a different directory on the host system. Where this directory is depends on your setup, but we will call it $TARGETDIR.

Install the modules into $TARGETDIR as follows:

```
bash$ make modules_install INSTALL_MOD_PATH=$TARGETDIR
bash$
```

This will place the modules into the $TARGETDIR/lib/modules/x.y.z directory on the host, which can then be placed into an suitable filesystem, or transferred to the target machine. Please also note that you must not install these kernel modules into the hosts root filesystem (eg by omitting INSTALL_MOD_PATH or giving $TARGETDIR as "/"), since they are incompatible with your host kernel and therefore may leave you with an unbootable host system.

The kernel will be available in $HOME/linux/arch/arm/boot/zImage and the kernel symbol information in $HOME/linux/System.map. Exactly how do install this is outside the scope of this document.

It is important that you keep the System.map file safe - it contains the symbolic information for this kernel, which will be required if you need to debug or report a problem.

Reporting Problems

Please follow the REPORTING-BUGS guide in the kernel source tree. However, please use the linux-arm-kernel mailing list to report problems, rather than the linux-kernel mailing list