The Linux Kernel

Zilogic Systems

1. Introduction

- The Linux kernel being a complex piece of software has an elaborate build process. The kernel build process involves the following stages.
  1. Configuring the Kernel
  2. Compiling the Kernel and Modules
  3. Installing the Kernel and Modules.
- Each of these stages are invoked using appropriate `make` targets.
- The build process explained does not require root access. It is adviced not to use the root account for building the kernel.

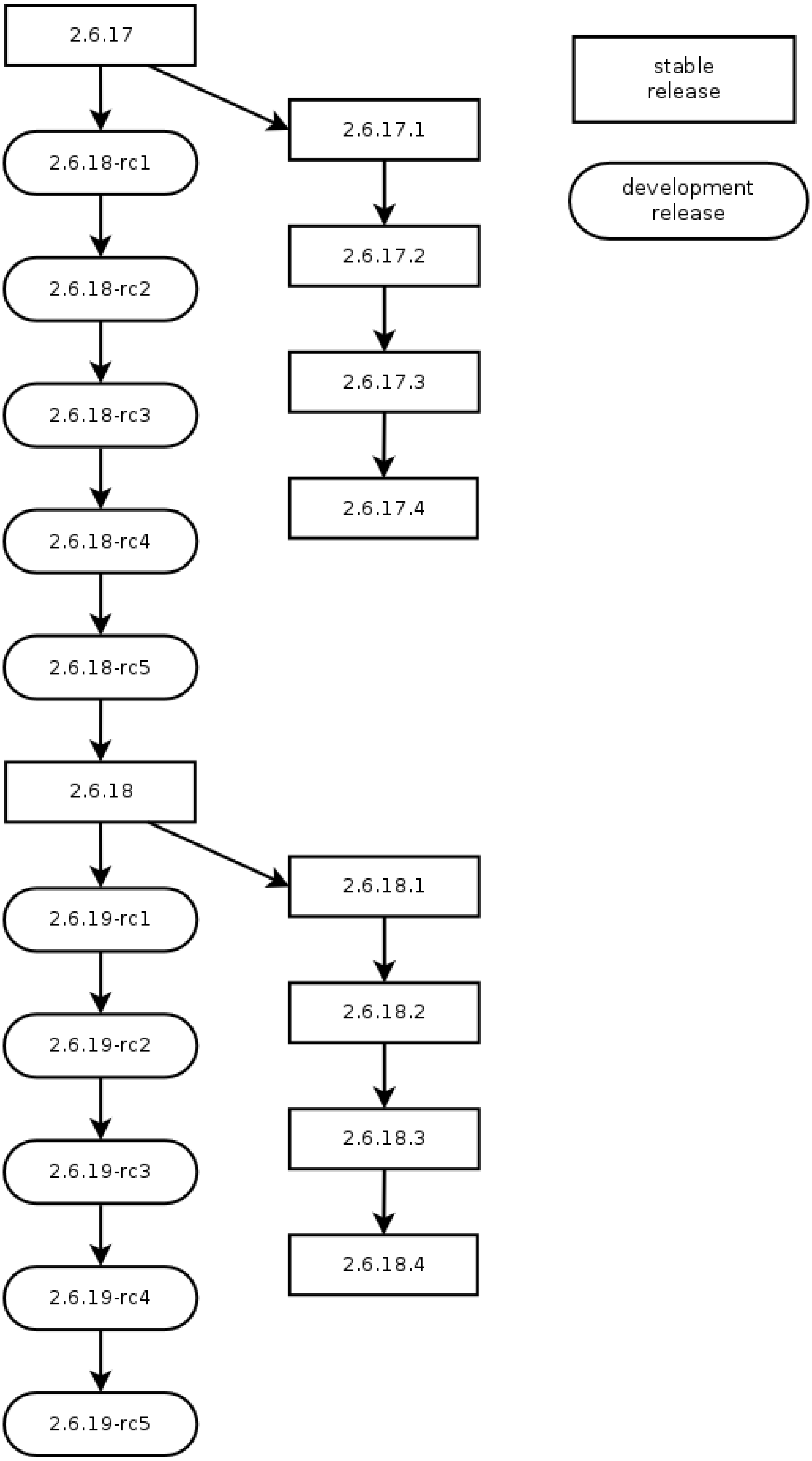
1.1. STEP 0: Obtaining the Kernel Sources

1.1.1. Kernel Release Numbering

- Initially, stable kernels were released once a year, on an average.
- With the kernel moving into maintenance mode, the release became more frequent i.e there was a stable release every 2 or 3 months.
- A new robust release numbering scheme was proposed owing to the frequent releases of the kernel.
- Kernels have a release no. of the format `2.6.x` for 2.6 versions and `3.x` from 3.0 onwards.
- Kernels released as `2.6.x`/`3.x` are all stable kernels. Development kernels were released as `2.6.x-rcN`/`3.x-rcN`. Bug fixes to the stable kernels were released as `2.6.x.a`/`3.x.a`.
- Longterm-support(LTS) Kernels are supported with important bugfixes for a long period of time.

Release	Date
2.6.17	18 June 2006
2.6.18	20 September 2006
2.6.19	29 November 2006
2.6.20	04 February 2007
2.6.21	26 April 2007
2.6.22	08 July 2007

Figure 1. The 2.6 Kernel Release Cycle



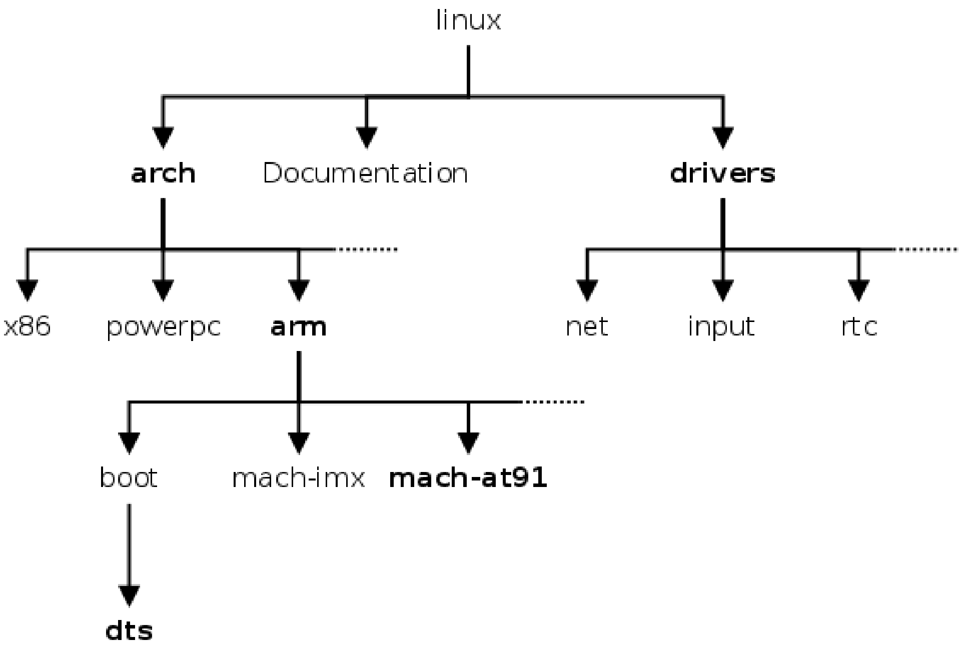
1.1.2. Downloading the Kernel

- The kernel can be obtained from http://www.kernel.org
- Extract the kernel sources using

```
[~]$ tar -j -x -f linux-2.6.18.tar.bz2
```

- From this point onwards we will refer to the kernel dir as `$KERNEL_DIR`

1.1.3. An Overview of the Kernel source



- `arch` directory consists of architecture specific kernel code with each architecture represented by a separate directory.
- `arm` directory consists of code for various ARM microcontrollers wtih similar microcontrollers grouped together in a directory.
- `mach-at91` consists of code for Atmel AT91 series of microcontrollers
- `dts` consists of microcontroller and board specific configurations.
- `drivers` contains drivers for various character and network drivers

1.2. STEP 0.5: Meeting Tool Dependencies

- Before you build and install the kernel you will have to ensure that you meet the minimum dependencies for the kernel.
- The kernel has certain build time dependencies - gcc, make, binutils.
- The kernel also has some runtime dependencies - grub, udev, module-init-tools, e2fsprogs, etc.
- The complete list of tools is available from `$KERNEL_DIR/Documentation/Changes`.
- It is not sufficient to just have these tools installed. It should also be ensured that the minimum version requirements are also met.
- Using older versions than specified could cause problems ranging from kernel boot failure to inconsistent system behaviour.

1.3. STEP 1: Configuring the Kernel

- This step involves configuring the kernel by specifying
  - what features are required
  - what are not required
  - what features are required as modules.

1.3.1. Configuration Targets

- The kernel build system provides multiple targets to configure the kernel.
- As a result of step 1, a file `.config` (pronounced "dot config") is produced. The file contains the values for all configuration options, and forms the input for the compilation stage.
- Creating a kernel configuration

  `config`
  bombards the user with a series of yes/no type questions. Reminds you of objective questions common in entrance exams. :-)

  `xyz_defconfig`
  provides a sane default configuration to start from. There is one default configuration for each machine/board supported by the kernel.

  `oldconfig`
  same as `config` target but answers all questions from previously generated `.config` file. The user is questioned only for configuration options that are not specified in the existing configuration file. This target is useful when upgrading to a newer kernel version, the older kernel's configuration file can be used, and the user is questioned only for newly added configuration options.

- Modifying the kernel configuration

  `menuconfig`
  provides a text based menu interface is provided through which the kernel can be configured.

  `gconfig`, `xconfig`
  same as `menuconfig` but provides a graphical interface instead of a text interface, using the GTK and QT libraries respectively.

1.3.2. Cross Configuration

- When building a kernel for an architecture other than the build system's architecture, the `ARCH` `make` variable should be set to the appropriate architecture.

```
[linux-2.6.18]$ make ARCH=arm menuconfig
```

- This can also be done as

```
[linux-2.6.18]$ export ARCH=arm
[linux-2.6.18]$ make menuconfig
```

1.4. STEP 2: Compiling the Kernel and Modules

- This is perhaps the longest and the most boring stage. If you do not have a high end machine, it is time for a cup of coffee.
- To compile the kernel, use the `uImage` target.
- When cross-compiling the kernel `ARCH` should be set to the architecture and `CROSS_COMPILE` should be set to the cross-compiler's prefix.

```
[linux-2.6.18]$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- uImage
```

- Note that the `mkimage` program is required to create `uImage`. The `mkimage` program is distributed as part of U-boot source tree.

- To build the modules, use the `modules` target.

```
[linux-2.6.18]$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- modules
```

1.4.1. Accelerating Builds

- It is possible to do parallel builds of the kernel, using the `-j` option of make.
- The argument to `-j` specifies the no. of parallel compilations to be performed.

1.5. STEP 3: Installing the Kernel and the Modules

- After build completes the kernel can be obtained from `arch/$ARCH/boot/uImage`.
- Modules are generally installed using

```
[linux-2.6.18]$ make ARCH=arm modules_install
```

- This will copy the modules to the filesystem of the system in which kernel is built, instead of the target’s filesystem.
- To copy the modules to a user specified directory, the `INSTALL_MOD_PATH` variable can be set accordingly.

```
[linux-2.6.18]$ make ARCH=arm INSTALL_MOD_PATH=/my/target/fs modules_install
```

1.6. Cleaning Up

- To clean up the kernel tree and remove the files generated during the build process, the many clean targets are available.

`clean`	Removes files generated as part of the build process.
`mrproper`	`clean` plus remove `.config`
`distclean`	`mrproper` plus patch backup files.

- The targets in the order of greater cleanliness is shown below.

```
distclean > mrproper > clean
```

2. Kernel Recipes

2.1. ARM EABI

The latest ARM ABI is called EABI. If the user space is built for the EABI, the EABI option should be enabled in the kernel.

```
Kernel Features
    [*] Use the ARM EABI to compile the kernel
```

2.2. Boot Time IP Configuration

Enabling configuration of IP address passed through the command line or obtained through DHCP, BOOTP or RARP.

```
Network Support
    Networking Options
        [*] TCP/IP Options
            [*] IP: kernel level autoconfiguration
```

```
                [*] IP: DHCP Support
                [*] IP: BOOTP Support
                [*] IP: RARP Support
```

2.3. NFS Root

Enabling NFS based root file system.

```
File Systems
    [*] Network File Systems
        <*> NFS Client Support
            [*] Root file system on NFS
```

2.4. MTD DataFlash

The DataFlash is accessed through the SPI interface. The SPI driver should be enabled first.

```
Device Drivers
    [*] SPI Support
        <*> Atmel SPI Controller
```

Enabling the MTD driver for DataFlash.

```
Device Drivers
    <*> Memory Technology Device Support
        Self-contained MTD device drivers
            <*> Support for AT45xxx DataFlash
```

Disabling MMC support.

```
Device Drivers
    < > MMC/SD card support
        < > AT91 SD/MMC Card Interface support
```

2.5. JFFS2 Filesystem

Enabling support for JFFS2 filesystem.

```
File Systems
    Miscellaneous filesystems
        <*> Journalling Flash File System v2 (JFFS2) support
```

References

- Chapter 3: "Retrieving the Kernel Source" from the book "Linux Kernel in a Nutshell" by Greg Kroah-Hartman.
- Chapter 4: "Configuring and Building the Kernel" from the book "Linux Kernel in a Nutshell" by Greg Kroah-Hartman.