2015

KERNEL COMPILATION & MODIFICATION

SEMINAR REPORT

HEMIL MODI-121016

IET, AHMEDABAD UNIVERSITY

WHY COMPILE A KERNEL?

There are many reasons for a user to want to make their own kernel. Many users may want to make a kernel that only contains the code needed to run on their system. For instance, my kernel contains drivers for Bluetooth devices, but my computer lacks Bluetooth hardware. When the system boots up, time and RAM space is wasted on drivers for devices that my system does not have installed. If I wanted to streamline my kernel, I could make my own kernel that does not have Bluetooth drivers. As for another reason, a user may own a device with a special piece of hardware, but the kernel that came with their latest version of Ubuntu lacks the needed driver. Like in my case, I have fingerprint reader hardware installed but my Linux kernel does not support that hardware. So, I could compile my own kernel with the supported driver for fingerprint reader. However, these are two of the most common reasons for users wanting to make their own Linux kernels.

KERNEL SOURCE CODE

After Downloading and extracting a kernel, you will see many files and folder. This are some files in root source code:

- Kbuild - This is a script that sets up some settings for making the kernel. For example, this script sets up a ARCH variable where ARCH is the processor type that a developer wants the kernel to support.
- Kconfig - This script is used when developer configure the kernel which will be discussed in a later article.
- MAINTAINERS - This is a list of the current maintainers, their email addresses, website, and the specific file or part of the kernel that they specialize in developing and fixing. This is useful for when a developer finds a bug in the kernel and they wish to report the bug to the maintainer that can handle the issue.
- Makefile - This script is the main file that is used to compile the kernel. This file passes parameters to the compiler as well as the list of files to compile and any other necessary information.
- README - This text file provides information to developers that want to know how to compile the kernel.
- init - The init folder has code that deals with the startup of the kernel (INITiation). The main.c file is the core of the kernel. This is the main source code file the connects all of the other files.
- include - The include folder contains miscellaneous header files that the kernel uses. The name for the folder comes from the C command "include" that is used to import a header into C code upon compilation.
- kernel - The code in this folder controls the kernel itself. For instance, if a debugger needed to trace an issue, the kernel would use code that originated from source files in this folder to inform the debugger of all of the actions that the kernel performs. There is also code here for keeping track of time. In the kernel folder is a directory titled "power". Some code in this folder provide the abilities for the computer to restart, power-off, and suspend.

CONFIGURING YOUR KENEL

Few of the many ways to configure a kernel:

- `make config` - Plain text interface (most commonly used choice)
- `make menuconfig` - Text-based with colored menus and radiolists. This options allows developers to save their progress. ncurses must be installed (sudo apt-get install libncurses5-dev).
- `make nconfig` - Text-based colored menus - curses (libcdk5-dev) must be installed
- `make xconfig` - QT/X-windows interface – QT is required
- `make gconfig` - Gtk/X-windows interface – GTK is required
- `make defconfig` - This option creates a config file that uses default settings based on the current system's architecture.

COMPILE AND COMPILE KERNEL

To compile the kernel, type "make" in a terminal that is in the same directory as the kernel's source code folders. This will take some time. Once done, the modules must be compiled by typing "make modules". To make the compiling process easier from the beginning, type "make; make modules". This will make the kernel and then the modules instead of the user coming back later to type "make modules". Once the compilation has finished successfully, we can then install the kernel to the local system. In the same terminal, after compilation, type "make install". This will place some files in the /boot/ directory. The modules must also be installed by typing "make modules_install". Both the kernel and modules can be installed using one line - “make install && make modules_install”. Once that process is complete, the user can ensure the kernel was installed by restarting the system and typing "uname -r" in a terminal when the system is back on.

REFERENCES

DevynCJohnson

http://www.linux.org/threads/the-linux-kernel-series-everyarticle.6558/