ECEN 449: Microprocessor System Design
Department of Electrical and Computer Engineering
Texas A&M University

Prof. Sunil P Khatri
(Lab exercise created and tested by Ramu Endluri, He Zhou and Sunil P Khatri)

Laboratory Exercise #9

Linux Kernel: Built-in Modules

Objective

The main objective of lab this week is to familiarize you with built-in kernel drivers, and to add the device drivers that you have created in the lab to the Linux kernel so that they can load during boot. We will also examine other built-in drivers and disable drivers/services that we do not require thereby building a reduced size(lean) kernel image.

Procedure

Device drivers of hardware can be added to the Linux kernel in two ways. Built-in kernel modules - when the kernel is booted up, the kernel automatically inserts this driver in to the kernel; loadable kernel modules(LKM) - modules that can be inserted(loaded) into the kernel after it is booted up and can be removed if not needed.

1. So far you have written device drivers for the 'multiply' and the 'ir_demod' modules. These modules are loaded into the kernel after boot as LKMs. We will now make these modules built-in to Linux kernel.

   (a) Create a lab 9 folder and copy the Linux kernel source code into the lab 9 directory.

(b) Before adding modules into the kernel, you should know how to configure the kernel. Linux supports several architectures and certain features are only supported for particular architectures. These dependencies are listed in kernel configuration files called Kconfig.

(c) In the Linux source directory, run the following command to configure the Linux kernel.

>make menuconfig

Every entry in menuconfig has a configuration file(Kconfig) which contains information on the usage of the entry. This information is used to determine the visibility of an entry.

(d) Exit menuconfig. To notice the difference run the following command in the Linux source directory. (Blackfin is 16-/32-bit embedded processors developed by Analog Devices for multiformat audio, video, voice and image processing applications)

>make ARCH=blackfin menuconfig.

Go to the Device Drivers->Sound Card Support->Advanced Linux Sound Architecture->ALSA for SoC audio support and observe the entries listed. Notice that SoC I2S audio support for the Blackfin chip is listed and can be selected(compile as built-in) or modularized(compile as LKM). Exit menuconfig.

(e) Now run the following command and look under ALSA for SoC audio support.

>make ARCH=arm menuconfig

Notice that SoC I2S audio support for the Blackfin chip is not available for selection. This is because this feature depends on the Blackfin architecture and hence not listed for selection for the ARM architecture.

2. Now that we know how entries are influenced by dependencies, to create an entry for the 'multiply' IP follow the steps below:

(a) Create a directory for 'multiply' called multiplier_driver under the device drivers folder of the Linux source code. Copy the source files for the multiplier kernel module into this folder.

(b) Use a text editor to create a 'Makefile' in the 'multiplier_driver' directory and add the following line to it.

obj-$(CONFIG_MULTIPLIER_DRIVER) += multiplier.o

(c) Use a text editor to create a 'Kconfig' file in the 'multiplier_driver' directory and add the following lines to it.

```
config MULTIPLIER_DRIVER
tristate "multiplier_driver"
depends on ARM
default y if ARM
help
 refer to ECEN449@TAMU
```

(d) In the Kconfig file, every line starts with a key word and can be followed by multiple arguments. 'config' starts a new config entry. The following lines define attributes for this config option. Attributes can be the type of config option, input prompt, dependencies, help text, and default values. A config option can be defined multiple times with the same name, but every definition can have only a single input prompt, and the type must not conflict.

(e) In the above Kconfig entry, the first line defines what configuration option this entry represents. Note that the CONFIG_ prefix is assumed and not written.

(f) The second line states that this option is a tristate, meaning that it can be built into the kernel (Y), built as a module (M), or not built at all (N). The quoted text following the directive provides the name of this option in the various configuration utilities.

(g) The third line specifies the dependency is ARM architecture, as we have developed this driver for an ARM device.

(h) The fourth line specifies the default for this option, which is'y' if the architecture is ARM.

(i) The help directive signifies that the rest of the text, indented as it is, is the help text for this entry

(j) The multiplier driver should be listed under the Device Drivers menu entry. In the Device Drivers directory under the Linux source file, find the Makefile and add the following line to it.

```
# ECEN 449
obj-$(CONFIG_MULTIPLIER_DRIVER) += multiplier_driver/
```

(k) Open the Kconfig file under Device Drivers and add the following lines before the 'endmenu'

source "drivers/multiplier_driver/Kconfig"

(l) Navigate to the Linux source directory and run the following command.

>make ARCH=arm menuconfig

You should be able to see an entry for the multiplier driver under Device Drivers in menuconfig. Select it to be compiled as built-in.

(m) Compile the Linux source and generate uImage(observe the size of uImage).

(n) Use BOOT.bin and devicetree.dtb from lab 5 to boot Linux on the ZYBO board.

(o) You should be able to see the print statements in the init function of your multiplier module during the boot sequence.

(p) Run the user application file for the multiplier and demonstrate your progress to TA. Show the Menuconfig entry, boot message and working of your modules.

(q) Follow the same process to make your 'ir_demod' driver built-in to the Linux kernel, and compile Linux without removing support for the multiplier driver. Observe the uImage size.
NOTE: virtual address pointers, init, and exit functions of 'multiply' and 'ir_demod' IP drivers should have different names to avoid compilation erros. You also need to recreate BOOT.bin and devicetree.dtb so that they include both 'ir_demod' and 'multiply' modules at the same time.

(r) Run the user application file for the ir_demod driver and demonstrate your progress to TA. Show the Menuconfig entry, boot message, and working of your modules.

3. At this point, you should have a Linux kernel with built-in support for the multiplier and ir_demod drivers. Note that the size of the uImage increase as you have added more built-in drivers. We have included device drivers to support our applications 'multiply' and 'ir_demod'. Similarly, we can exclude features which we do not require, which will reduce the kernel size. We should only exclude features that are not critical to the Linux boot, otherwise the Linux kernel may not function properly or even cause a boot failure.

4. The Linux kernel provided to you has built-in support for networking, multimedia, and sound. Disable the support for these features. Exclude the following modules from menuconfig, and recompile the Linux kernel and observe the size of the kernel.

   (a) Networking support

   (b) Device Drivers/Multimedia support

   (c) Device Drivers/Soundcard support

5. Notice that the uImage size is reduced considerably from the initial kernel size.

6. Show your progress to the TA and report your kernel size.

Deliverables

1. [6 points.] Demo the built-in multiplier driver and IR device driver to the TA. Report the kernel size you achieved to the TA.

   Submit a lab report with the following items:

2. [6 points.] Correct format including an Introduction, Procedure, Results, and Conclusion.

3. [4 points.] The output of the the picocom terminal.
4. [4 points.] Answers to the following questions:
   (a) What are the advantage and disadvantages of loadable kernel modules and built-in modules?