Reading Plan – Linux Device Drivers

Important Notes

- All students to meticulously follow the reading plan for this module.
- Students have to issue **Linux Device Drivers, Third Edition, O'Reilly, Rubini, Alessandro** text book from library. Most of the topics are from this text book.
- The suggested **reading plan and topics should be completed before each lecture session**. The lecture will only be a discussion with the students and there will be scope for interactions based on the student's reading, understanding and doubts.
- Failing to read the text book may result in failure to understand the concepts and therefore, students will find it difficult to perform well in this module.
- Students are advised to follow the reading plan as prescribed below.
- Students may create internal groups and discuss as well, to create a set of questions for lecture time discussion and interactions.
- Students should read and re-read the suggested material before coming to class.
- Students should plan the reading in advance because there is a lot of programming in this module. The programming will consume time because of many new concepts involved.
- Happy Learning.

Day/Date	Topic	Reading Material
Days 1 & 2	Introduction to Linux Embedded Systems	--
Day 3	Kernel Compilation and Installation Process	Annexure 1 in this document
Day 4 & 5	Module Programming	Pg 1-36, Introduction & Module Programming
Day 6, 7 & 8	Character Drivers	Pg 42 – 68, Character Drivers
Day 9	Advanced Character Drivers	Pg 135 – 140, IOCTL
Day 10 & 11	Wait Qs, Timing in the kernel, Deferred Executions	Wait Qs – Pg 147 (Blocking I/O) to Pg 162 Timing – Ch 7, Pg 183 to 211
Day 12 &13	Kernel Synchronization Techniques	Pg – 107 to 120 - Synchronization
Day 14 & 15	Dealing with Hardware and Interrupts	Pg – 235 to 281 – Hardware and Interrupts

Annexure 1 - Cross-compiler and kernel environment for rpi4

I. Steps to flash Raspbian OS onto sd card:

On Host(ubuntu) :

1. Open terminal

```
$ sudo apt install rpi-imager
        or
$ snap install rpi-imager

$ rpi-imager
```

Choose OS : Raspberry Pi OS (other) => raspberry pi OS LITE 32-bit
Choose storage : choose your sd card
Click on write and then click on yes - This will take some time.

2. After completing flashing image plug out sd card and insert sd card again.

```
$ cd /media/<user-name>/boot
$ touch ssh
$ touch wpa_supplicant.conf
$ vim wpa_supplicant.conf
```

Write the following code in wpa_supplicant.conf file and save it.

```
country=IN
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
network={
        ssid="PrasadMob"
        psk="prasad@08"
        key_mgmt=WPA-PSK

}
```

3. Plug out the SD card and insert into your raspberry pi board.
4. Board will start booting and access it.

II. Steps for cross-compiling kernel :

On Host (ubuntu) :

1. Install Required dependencies :

```
$ sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev
```

2. Install 32-bit toolchain

```
$ sudo apt install crossbuild-essential-armhf
```

3. Download/clone kernel source

~ $ mkdir rpi
$ cd rpi
$ git clone --depth=1 --branch rpi-5.15.y https://github.com/raspberrypi/linux
$ cd linux

4. Apply the config file of rpi4 :

Check config file for your board(rpi4) using below command

$ ls arch/arm/configs

Default config file for rpi4 is bcm2711_defconfig

Now apply config file using below command

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2711_defconfig

5. Build kernel image and kernel modules for rpi4 :

$ make -j8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules

Result of above command :

$ ls arch/arm/boot

zImage

6. Plug in your sd card to your HOST PC(ubuntu)

$ cp arch/arm/boot/zImage /media/<user_name>/boot

7. Install modules onto rootfs partition of SDcard

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=<path-to-sdcard rootfs partition> modules_install

Example In my pc :

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=/media/embedded/rootfs modules_install

Modules gets installed in rootfs/lib/modules path

8. Configuring config.txt to boot our new kernel

$ cd /media/<user-name>/boot

Open config.txt:

$ vim config.txt

Add below line at the end of the file and save file :

kernel=zImage

9. If “ssh” and “wpa_supplicant.conf” files are not in your boot partition then follow steps of 2 of flashing raspbian OS.
10. Plug out sd card and insert into your raspberry pi board.
11. Board will start booting and access it .

III. Cross compile module for the Raspberry Pi.

1. Write a source code on HOST Machine(Ubuntu)

hello.c

```
#include<linux/module.h>
#include<linux/init.h>

static int  __init hello_init(void)
{
      pr_info("Hello World\n");
      return 0;
}

static void  __exit hello_exit(void)
{
      pr_info("Good Bye\n");

}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("CDAC");
MODULE_DESCRIPTION("A simple hello_world kernel module");
MODULE_INFO(board,"RASPBERRY PI 4");
```

Makefile for Cross Compilation.

```
obj-m := hello.o
KERN_DIR=/lib/modules/5.10.52-v7l+/build/

all:
      Make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C
$(KERN_DIR)          M=$(PWD) modules
```

```
clean:
	make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C
$(KERN_DIR)		M=$(PWD) clean
```

- After compiling the module, Copy the Kernel Object(.ko) file to the RaspberryPi.

- You can copy the contents from host to destination using command,

scp -r <path-to-source> <path-to-destination>

Example-
scp hello.ko /home/pi/<your-folder>

IV. Native compile for the Raspberry Pi

hello.c
- source code will be same - refer earlier hello.c file.

Makefile for the Native Compilation

```
obj-m := hello.o
KERN_DIR=/lib/modules/$(shell uname -r)/build/

all:
	make -C $(KERN_DIR) M=$(PWD) modules

clean:
	make -C $(KERN_DIR) M=$(PWD) clean
```