Basics of writing a Linux kernel module

1. **Lecture**

Writing a Linux Kernel Module.pdf

2. **Laboratory**

2.1. Introduction

A loadable kernel module (LKM) is a mechanism for adding code to, or removing code from, the Linux kernel at runtime.
They are ideal for device drivers, enabling the kernel to communicate with the hardware without it having to know how the hardware works. The alternative to LKMs would be to build the code for each and every driver into the Linux kernel.
A kernel module is not an application — for a start there is no main() function.

A character device typically transfers data to and from a user application — they behave like pipes or serial ports, instantly reading or writing the byte data in a character-by-character stream. They provide the framework for many typical drivers, such as those that are required for interfacing to serial communications, video capture, and audio devices.

The main alternative to a character device is a block device.
Block devices behave in a similar fashion to regular files, allowing a buffered array of cached data to be viewed or manipulated with operations such as reads, writes, and seeks. Both device types can be accessed through device files that are attached to the file system tree.

Device drivers have an associated major and minor number.
The major number is used by the kernel to identify the correct device driver when the device is accessed.
The role of the minor number is device dependent, and is handled internally within the driver.
You can see the major/minor number pair for each device if you perform a listing in the /dev directory.

2.2. How to

2.2.1. Compile the Linux kernel

```
$ source ~/work/scripts/setenv.sh
$ cd ~/work/nss-linux
$ make tn_imx8_defconfig # this needs to be done only once when you first compile the kernel
$ make -j4
```

2.2.2. Boot the board

Console #1

```
$ mincom -D /dev/ttyUSB0
```

Console #2

```
$ sudo ~/work/scripts/uuu ~/work/scripts/uuu_script
```

And then hit `Reset` button on the board.

2.2.3. How to change rootfs with new kernel modules

```
$ source ~/work/scripts/setenv.sh
~/work/scripts/module_install.sh -k ~/work/nss-linux -r ~/work/images/rootfs.ext2 -m tmp
```

2.3. *Exercises*

2.3.1. Linux Kernel Module

1. Starting from the skeleton found at *drivers/nss/lab01/ex1/hello.c* , write a Linux kernel module.
   This must print a "Hello world" and "Goodbye" message in its initialization and cleanup function, respectively.
   Also, add some module information. Use MODULE_DESCRIPTION, MODULE_AUTHOR, MODULE_LICENSE, MODULE_VERSION macros.

2. Update the Makefile from *drivers/nss/lab01/Makefile* to compile the hello.c and obtain the module.
   Enable CONFIG_NSS_DRIVERS_LAB01_EX01. For this you will need to run:

   ```
   $ source ~/work/scripts/setenv.sh
   $ make menuconfig

   # Select: Device Drivers -> NXP summer school drivers -> NXP summer school lab 01 drivers -> Exercise 01
   (M)
   ```

3. Test the kernel module
   a. copy kernel module on target - see How to change rootfs with new kernel modules;
   b. insert module : Use *insmod /lib/modules/6.1.55-version/kernel/drivers/nss/lab01/ex1/hello.ko* OR *modprobe hello* commands
   c. check if the modules is loaded: use *lsmod* command
   d. show information about the module: use *modinfo hello* command
   e. remove the module, use *rmmod hello* command
   f. check the log, use *dmesg* command

4. Starting from the skeleton found at *drivers/nss/lab01/ex4/hello_name.c*, add a parameter to Hello <name> Linux kernel module and test it (see 3. above).
   Enable CONFIG_NSS_DRIVERS_LAB01_EX04.
   Compile and test .

   ```
   $ modprobe hello_name name="Jane"
   ```

   Hints

   a. Conventional argc and argv[] does not work with kernel modules
   b. Steps needed to add a parameter:
      i. declare the variables that will take the values of the command line arguments as global
      ii. module_param() macro is used to set the mechanism up - takes 3 arguments: the name of the variable, its type and permissions for the corresponding file in sysfs (see https://www.gnu.org/software/libc/manual/html_node/Permission-Bits.html)
      iii. MODULE_PARM_DESC() is used to document the argument that the module can take - takes 2 parameters: a variable name and a free form string describing that variable.
   c. "World" will be the default value of the parameter
   d. Check the __init and __exit functions

Rather than using the lsmod command, you can also find out information about the kernel module that is loaded, using command "cat /proc/modules | grep hello"

The LKM also has an entry under /sys/module, which provides you with direct access to the custom parameter state.

2.3.2. Character Device Driver

5. Starting from the skeleton found at *drivers/nss/lab01/ex5/imx8mq_chardev.c,* write a char device driver:

   a. Add file operation callbacks. Implement TODO1.

   b. Implement the registration and deregistration of the device with the name imx8mq_chardev, respectively in the init and exit module functions.
   Implement TODO2 (see https://elixir.bootlin.com/linux/latest/source/fs/char_dev.c#L225) and TODO3 (see https://elixir.bootlin.com/linux/latest/source/fs/char_dev.c#L302).
   Enable CONFIG_NSS_DRIVERS_LAB01_EX05.
   Compile.

   c. Update rootfs (see How to change rootfs with new kernel modules) and boot the board (see Boot the board)
   Load the module (imx8mq_chardev) into the kernel.
   Check character devices in /proc/devices - "cat /proc/devices | less".
   Unload the kernel module.
   Check log with:
   $ dmesg | tail
   or
   $ cat /var/log/syslog | tail

   We got a crash!? Debugging...
   Check error message.
   Hint: see https://elixir.bootlin.com/linux/latest/source/drivers/base/class.c#L288

   d. Implement the open and release functions in the driver.
   Display a message in the open and release functions.
   Implement TODO4 and TODO5.

   e. Implement the read function in the driver. Use the copy_to_user() function to copy information from kernel space to user space.
   Implement TODO6

   f. Implement the write function in the driver. Use the copy_from_user() function to copy information from user space to kernel space. Implement TODO7

   g. Test
   Load the module (imx8mq_chardev) into the kernel.
   Read from kernel space to userspace: $ cat /dev/imx8mq_chardev
   Write to kernel space from user space: $ echo "NXP" > /dev/imx8mq_chardev
   Unload the kernel module.

   h. Can you spot an error in the code? Try fix it!
   Hint:
   Look closer on write() callback!
   Try to write to kernel space from user space: $ echo "NXP Summer School" > /dev/imx8mq_chardev

   i. We can then use the testimx8mqchar, from *drivers/nss/lab01/ex5/test/* program to test that the LKM is working correctly.
   First build the application using the Makefile from *drivers/nss/lab01/ex5/test/.*

   ```
   $ cd ~/work/nss-linux/drivers/nss/lab01/ex5/test
   $ make
   ```

   Update rootfs (see How to change rootfs with new kernel modules) and boot the board (see Boot the board)

   Run the test, on target:
   $ sudo ./test
   Starting device test code example...
   Type in a short string to send to the kernel module:
   NXP
   Writing message to the device [NXP].
   Press ENTER to read back from the device...

   Reading from the device...
   The received message is: [NXP Summer School]
   End of the program

Hints

1. Similar to the code in the first part of this lab, there is an init() function and an exit() function.
   However, there are additional file_operations functions that are required for the character device:
   a. dev_open(): Called each time the device is opened from user space.
   b. dev_read(): Called when data is sent from the device to user space.
   c. dev_write(): Called when data is sent from user space to the device.
   d. dev_release(): Called when the device is closed in user space.
2. The *testimx8mqchar* is a short program that requests a string from the user, and writes it to the /dev/imx8mq_chardev. After a subsequent key press (ENTER) it then reads the response from the device and displays it in the terminal window.
3. A Makefile is provided.

Solutions:
hello_solution.c
hello_name_solution.c
imx8mq_chardev_solution.c