CMD:

**Uname –r outputs x,y,z-ab-something else**

X is the major number

Y is the minor number: even number means the version is stable

Odd number means it's in experimental version

Z is the revision number

**LKM (Loadable Kernel Modules)**

LKMs are object files that are used to extend the running kernel’s functionalities of the current operating system. It’s binary code that can be inserted and installed in the kernel on the fly without the need to reboot.

Two types of device drivers:

**Character Device:**

-Reads from the device character by character

-Writes to the device character by character

-Operates blocking mode, meaning when a user writes info to the device, they must wait until the device completes execution.

**Block Device Driver:**

-Reads large chunks of info

-Very CPU intensive, takes some time to finish the execution

-They are asynchronous, user doesn’t need to wait reading and writing

Step:

1. Do ‘cat /proc/devices’, and I found major number 230 that’s available.

* **Major number** is used to differentiate the hard disk device driver
* **Minor number** is used to differentiate the different partitions

1. I create a device file by ‘**sudo mknod –m 777 /dev/simple\_char\_device c 230 0’**
2. Touch a file, **simple\_char\_driver.c** under /Desktop/PA/pa2/module
3. Go to the file, **simple\_char\_driver.c**, and **#include <linux/fs.h>** and **#include <asm/uaccess.h>**

* Fs.h: get the functions that are related to device driver coding
* Asm/uaccess.h: enable to get data from userspace

1. Using **register\_chrdev()** function in the init function with 3 params (major number, name of the driver, a pointer to the file operations structure).

//Within the driver, in order to link it with its corresponding /dev file in kernel space, the register\_chrdev function is used

1. Using **unregister\_chrdev()** function in the exit function with 2 params(Major number, the name if the character driver).

**// FREEING MAJOR NUMBER**

1. Initialization functions should be declared static, since they are not meant to be visible outside the specific file; there is no hard rule about this, though, as no function is exported to the rest of the kernel unless explicitly requested. i.e init()
2. The cleanup function has no value to return, so it is declared void, i.e exit()
3. Virtual File System lies in Kernel space just above character driver and low-level I/F. This VFS decodes the file type and transfers the file operations to appropriate device driver. Now for VFS to pass the device file operations onto the driver, it should have been informed about it. That is what we call as registering the file operations by driver with VFS. Basically it involves by using **file\_operations** structure, which holds appropriate functions to be performed during usage.

**STATIC STRUCT FILE\_OPERATIONS MEMORY\_FOPS =**

**{**

**.OWNER = THIS\_MODULE,**

**.OPEN = MEMORY\_OPEN,**

**.RELEASE = MEMORY\_RELEASE,**

**.READ = MEMORY\_READ,**

**.WRITE = MEMORY\_WRITE,**

**}**

**You still need to define .open .release .read .write**

(Similar file operation structure)

under the path: /lib/modules/4.10.0-33-generic/build/include/linux/fs.h line: 3068

1. to identify which device is being opened. The other way to identify the device being opened is to look at the minor number stored in the inode structure. If you register your device with *register\_chrdev*, you must use this technique.

Worked cited:

<http://static.lwn.net/images/pdf/LDD3/ch02.pdf>

<https://smdaudhilbe.wordpress.com/2013/06/11/a-basic-character-driver-to-read-and-write-messages/>

http://www.makelinux.net/ldd3/chp-6-sect-5