Advanced System Programming 4732/5733
In-class Activity for Writing a Character Special Device Driver

Please follow the step by step instructions below. If you have already completed some of the steps, please continue with the step that applies to your case.

1. Install VirtualBox from https://www.virtualbox.org/ and create a virtual machine instance (VM) with Ubuntu. Make sure your username reflects your First Name and/or Last Name. You will need to download the .iso file for an Ubuntu version (latest one is recommended) on to your host machine. You can find the iso files from https://ubuntu.com/download/desktop When you try to run the virtual machine instance for the first time, you will be asked for the .iso file for installing Ubuntu.
2. Install the following software on your VM
    a. make (sudo apt install make)
    b. gcc (sudo apt install gcc)
    c. You will need to have sudo access in your VM. If you have already sudo access, then skip this step.
    If you do not have sudo access then first change to superuser

    ```
    $ su

    $ usermod -aG sudo username

    $ sudo whoami
    ```

    (If you see root displayed then you succeeded to add sudo access. Otherwise, complete the following step and add the following line to the end of /etc/sudoers file using your favorite editor, e.g., nano)

    ```
    username  ALL=(ALL) NOPASSWD:ALL
    ```

    Make sure to get out of su mode with CTRL-D. Now, you can use sudo when needed.
3. Inside your VM download LDD_SOLUTIONS.zip from CANVAS/Files/resources (using a web browser and logging onto CANVAS may be easier than transferring files from the host machine.). When you unzip this file, you'll find lab1_chrdrv.c under the s_02 directory. Alternatively, you can just download lab1_chrdrv.c from CANVAS/Files/lectureNotes.
4. On your VM, create a new directory on your file system. Let APATH denote the full path to this directory. Copy lab1_chrdrv.c under APATH and change its name to tuxdrv.c
5. Create a Makefile under APATH
    a. You can use your favorite editor. We use the pico or nano editor in the examples
    b. pico Makefile
    You just need a single line in your Makefile:

obj-m += tuxdrv.o
This line says that tuxdrv.o will be one of the modules that will be generated in the current directory.

6. Check to have the kernel header files on your system:
   a. ls -l /usr/src/linux-headers-$(uname -r)
      i. If you see some files including a Makefile, it means you already have the linux header files. If not (No such file or directory), get the linux header files:
      ii. $ sudo apt-get install linux-headers-$(uname -r)
      iii. You can execute the above ls -l command to see if this was successful.
7. Now, let's use the Makefile of the kernel to build the module for our driver. Assuming you are under APATH:
   a. make -C /usr/src/linux-headers-$(uname -r) M=$PWD modules
   b. Note that -C tells the make utility to go to that directory and use the Makefile in that directory. With M=$PWD, it tells make to come to the current directory to build the modules target. Remember with obj-m += tuxdrv.o, we just listed our driver as one of the kernel modules to be built.
8. If you can see tuxdrv.ko under APATH your build was successful:
   a. ls -l tuxdrv.ko
9. Let's load our module to the kernel
   a. sudo insmod tuxdrv.ko
10. Check kernel logs to see if the driver could be loaded
    a. dmesg -T
11. Let's try unloading
    a. sudo rmmod tuxdrv
12. When you want to check whether your module is currently loaded, do
    a. ls -l /sys/module/tuxdrv
    b. If you see some files, it is still loaded. If you get No such file or directory, the module is currently not loaded.
13. Now, let's replace the messages printed during loading and unloading, and during Virtual File System (VFS) operations with personal messages (use your name in the message about registering/unregistering). Recompile your driver, load and unload it and check the log messages.
14. Now, let's play with our driver via the VFS Layer. We will first create a node for our hypothetical device tux. (If you get an error that 700 is bigger than max value 511, use 500. But you will also need to change the driver to replace 700 with 500).
    a. sudo mknod /dev/tux0 c 700 0
    b. Check if it gets created
       i. ls –l /dev/tux0
15. Let's write our HelloDriver.c that opens this device file and writes to tux. Compile HelloDriver.c.
16. Make sure that the driver is loaded.
17. Execute HelloDriver.c's executable. To check whether we could write to the device, let's use the dd (data duplicate) command:

   a. sudo dd if=/dev/tux0 bs=14 count=1
      Here bs denotes block size and count denotes to number of blocks to duplicate

18. Congratulations! You just tested your (maybe) first device driver ☺
19. **Here is a challenge question: Could you both write to the device and then read from the device what has been written inside HelloDriver.c? Show how to.**
20. Submit 1) the screenshots of your log messages and the console output that show your personalized message including your name, 2) HelloDriver.c and tuxdrv.c on CANVAS.