New York University
Tandon School of Engineering
Department of Computer Science and Engineering

Introduction to Operating Systems
Fall 2025

Assignment 7
(15 points)

In this assignment, we shall build on what we did in assignment 3 and further extend it. You shall **write a dynamically loadable module that registers itself as a character device**.

- When registering your kernel module as a character device, you need to provide some methods that allow a user-mode application to access your device via `open()`, `read()`, `write()` and `ioctl()` (these are typical file operations). You do so by implementing those methods (naming them as you wish) and providing a "file operations struct" that contains function pointers to them. This takes place during the registration of your character device.The file operations struct contains many function pointers, you should set all of them to NULL except the ones you are implementing, where you provide pointers to the actual functions you implemented.

- In this assignment, **you shall implement** the `open()` and the `read()` functions.

  In your implementation of the `open()` method, you shall record the jiffies value. Yes, you may save it in a static global variable. We assume one process will open the file at a time, read then close it.

  In your implementation of the `read()` method (and again, you may name it as you wish), you shall compute and **provide** the time elapsed between the `open()` and the first `read()` calls. You shall use the jiffies to do so, as we did in assignment 3. You shall also **provide** the process ID of the parent process on the next line. You provide the required information (time elapsed and process ID) as a c-string written to the buffer provided to you by the user-mode application (the second parameter of the read call).

  You get the process ID of the parent of the invoking process using the current (running) process' PCB (`struct task_struct`) to get to the parent process' PCB, and then the process ID of the parent is a field there.
  The kernel global variable `cur` points to the current process' `task_struct` (i.e. `cur` is a pointer).
  Hint: you should `#include <linux/sched.h>` in your module's source file.

- You shall also implement a use-mode program to test your kernel module. Typically, a user-mode application will open your "virtual" device, read from it (till it encounters and end-of-file character, EOF), then close it. Thus, **when testing** your module, you **shall create a simple user-mode**

program that opens your virtual device and reads from it. It records the time before the `open()` and then computes the elapsed time after the `read()` then compares it to the elapsed time provided by the `read()` method of your kernel module (i.e. the driver of the virtual device). It shall print both elapsed times. It also shall print the process ID of the parent provided in the `read()` call and compare it to what it gets from getppid() (i.e. print it both ways to compare).

Some Linux Notes:

- Devices are organized as character, block or network devices.
- Devices have major and minor numbers. The major number specifies the device type whereas the minor number specifies an instance of that type. As such, it is common that a single device driver takes care of an entire major number.
- Most major numbers are already assigned to specific device drivers so you are not supposed to use them.
- Linux uses the concept of virtual file systems, one of those is the devfs (whose "virtual" devices appear on /dev/)
- Generally,
  1) You driver registers itself with **the devfs subsystem** using a major number, and then
  2) Create a node for it in the `/dev` directory (e.g. /dev/labx), using the `mknod` command. You must pass some parameters to the mknod command; the major number your driver used (that's how you link /dev/labx virtual file to your driver, using the major number) and a minor number of your choice (for the instance). You also pass the type as "c"

  User-mode applications may then treat it as a file and file operations such as `open()`, `fprintf()`, `read()`, `write()`, etc. are routed to functions that your module handles.

- Instead of using this complex registration and node creation process, we shall use the `misc_register()`, which handles both steps automatically. It registers us under device major number 10 (amongst with many other devices besides us using the same major number, where each has a different minor number):
  - You need to provide a minor number to `misc_register()`, the device's name ("labx") and a pointer to the file operations.
  - In order to avoid collisions, use the `MISC_DYNAMIC_MINOR` macro as the minor number when you register your device.
- For information regarding misc_register(), see: http://www.linuxjournal.com/article/2920.
- You may find further information at kernel.org or lwn.net
- You should not **directly** use the pointers provided by the `read()` function, but instead you need to use:

```
unsigned long copy_to_user(void __user *to, const void *from,
unsigned long n);
```

Brief (but not full) Summary of what you need to do:

- Review what you did in assignment 3.
- Read the article in http://www.linuxjournal.com/article/2920 to familiarize yourself with how to create and register a misc device.

- Create an "open: function (e.g. name it `misc_open()` or any other name of your choosing) and a "read" function (e.g. named `misc_read()`)
    - Please read the notes pdf document for further information.
- Create a `struct file_operations` containing the function pointers to `misc_open` and `misc_read` you've created earlier, and containing null pointers for the remainder of the struct.
    - Consult with the "Linux Device Driver 3" book, page 50 for further info about the open and read functions
    - Further information is also in "hw8_24F_notes.pdf"
- Create a `struct miscdevice` as per article 2920 above.
- Create your init and exit functions (as you did in assignment 3). In the module init function, you use the `misc_register()` functionality to register your module as a character device, using the structs you've already created.
- Build your kernel module as you did in assignment 3.

What to submit to gradescope:

Please submit the following files individually:

1) Source file(s) with appropriate comments.
   The naming should be similar to "**lab#_$.c**" (**#** is replaced with the assignment number and **$** with the question number within the assignment, e.g. lab4_b.c, for lab 4, question b OR lab5_1a for lab 5, question 1a).
2) A single pdf file (for images + report/answers to questions), named **"lab#.pdf"** (**#** is replaced by the assignment number), containing:
    - Screen shot(**s**) of your terminal window showing the current directory, the command used to compile your program, the command used to run your program and the output of your program.
3) Your Makefile, if any. This is applicable only to kernel modules.

RULES:

- You shall **use kernel version 4.x.x or above**. You shall not use kernel version 3.x.x.
- You may consult with other students about GENERAL concepts or methods but copying code (or code fragments) or algorithms is NOT ALLOWED and is considered cheating (whether copied form other students, the internet or any other source).
- If you are having trouble, please ask your teaching assistant for help.
- You must submit your assignment prior to the deadline.