CS342 - Spring 2019
Project #4
In-Memory File System Structures

Assigned: **April 26, 2019.**
Due date: **May 18, 2019, 23:55.** Document Version: 1.0

Objectives

- Practice file systems.
- Practice accessing in-memory file system structures.
- Practice **kernel module** programming.
- Touch to the Linux kernel.

You can do this project in groups of two students each. You will use C and Linux.

In this project, you will develop a Linux module that will access and use some in-memory structures related to files and file systems. But first you will need to learn how to write a Linux kernel module. Then you will write the desired module. You can do the project in a Linux system installed on bare hardware or on a virtual machine. You will use 64-bit Linux. You are recommended to use Ubuntu 18.04.

Learning Kernel Module Programming

Step 1: Learning how to build a kernel. (If you wish you can skip this step 1 if you succeed doing step 2 without doing this step 1). First learn how to compile (build) and run a new Linux kernel, so that you can get prepared to write a kernel module. Learn from Internet how to build and run a new kernel. Download source code of Linux kernel, build it (this may take a while one hour or so the first time you do it) and run it. You can do it on a virtual machine if you wish. If you are doing it directly on your machine, make sure you backup all your data first, so that if you mess up the file system and partitions on the disk, you can recover your data. Note that this part will be quite time consuming, but you will do it only once (until you get your new kernel running).

Step 2: Learning how to write a module and develop a simple Hello World module. In this step you will learn how to develop and run (load/insert) a new kernel module. Compiling a module and loading/running it is very easy and fast (just a few seconds) after you have the right development environment set up. There is documentation available on the web about Linux kernel module programming. Search for Linux kernel module programming. Below are two good references to start with. They can be reached from the course website. Read this documentation. Do some

simple exercises. Write a hello world program. You can write and test other simple modules as well.

1. The Linux Kernel Module Programming Guide,
   http://www.tldp.org/LDP/lkmpg/2.6/html/index.html
2. Linux Device Drivers, Third Edition,
   http://lwn.net/Kernel/LDD3/,
   (especially the Chapter 2: Building and Running Modules).

Develop a Kernel Module to In-Memory File Structures

Implement a Linux kernel module to get and print some file system and file related information from some in-memory kernel structures related to processes, files, file systems and block I/O.

Your module will retrieve the required information from the related kernel data structures: the PCB of the process, open file table, file objects, and block cache. Your module will be a kernel code that can be loaded (inserted) and unloaded (removed) while the system (kernel) is up and running, without rebooting the system. It will be loaded with the insmod command. The module will take one argument, a process identifier (`processpid` - an integer value), while being loaded using insmod. When loaded, a kernel module becomes part of the running kernel and runs in kernel mode and space with kernel privileges. The name of the parameter will be —`processid`, both inside your module code and while inserting the module at command line.

Below are the things that your module will do.

1. It will first find the PCB (task structure) of the process whose pid is specified at the command line while inserting the module. For that, your module can traverse the process list (PCB list). Depending on the kernel version, there may be a `current` variable in Linux kernel that is pointing to the PCB (of type tast_struct *) of the currently running (scheduled) process. Starting from current, you can traverse the list of PCBs until you find the PCB of the process with the given pid. The PCBs of processes are linked together (double linked list) in Linux kernel. There can be some other ways to traverse the process list; you can learn from Internet.

2. After finding the PCB, your module will print information about the file descriptors and open files of the process. This information can be found by following the `fs` and `files` fields (of type `fs_struct` and `files_struct`) of the PCB of the process. The fs_struct structure stores some file system information and files_struct structure stores open file information.

3. Access the open file information of the process and for each descriptor used, print the following information: descriptor number (index into the table/array),

current file position pointer, user's id, process access mode, name of the file (from dentry - directory entry - object), inode number of the file (from inode object of the file which is accessible through the dentry object), file length (from inode object) in bytes, number of blocks allocated to the file.

4. By accessing the necessary structures, print the following information as well: name of the current directory of the process (follow the fs field of the task structure), blocks that are cached for the process in the page cache (buffer cache).

5. Access the buffer cache and print information about some 100 blocks (buffers) there. For each such buffer print the following information: the storage device the block is in, the block number, use count.

6. Some more information that you will print may be requested later (after the project is assigned). We will do the necessary announcements.

7. Write an application program that will work with files: create, write, read files. Using your module inspect what is happening in the kernel while your application is running and using the files. Put your observations into a report.

The printing will be done by using the printk() kernel function. You can not use printf in kernel (since standard C library is not linked with kernel). The output will go to a kernel log file (in /var/log/) that can be examined later by using commands like dmesg, more, cat, tail, etc.

Submission

Put your report.pdf file, your program files, a Makefile, and a README.txt file into a directory named with your ID (for a group, a single file will be uploaded using the ID of one of the students). Then tar and gzip the directory. For example a student with ID 21404312 will create a directory named 21404312 and will put the files there. Then he will tar the directory (package the directory) as follows:

```
tar cvf 21404312.tar 21404312
```

Then he will gzip the tar file as follows:

```
gzip 21404312.tar
```

In this way he will obtain a file called 21404312.tar.gz. Then he will upload this file in Moodle.

Late submission will not be accepted (no exception). A late submission will get 0 automatically (you will not be able to argue it). Make sure you make a submission one day before the deadline. You can then overwrite it.

References

[1] The Linux Kernel Module Programming Guide,
http://www.tldp.org/LDP/lkmpg/2.6/html/index.html

[2] Linux Device Drivers, Third Edition,
http://lwn.net/Kernel/LDD3/,
(especially the Chapter 2: Building and Running Modules).

Tips and Clarifications

- We will insert your module in our virtual machines and see if it is working and doing the desired things. We can also call you for a demo. You may need to bring your computer (laptop or desktop).