# Project 3: s2fs (Super Simple File System)

- **Handed out:** Wednesday, October 16, 2019
- **Due dates**
  - Parts 1 and 2: Monday, November 25, 2019
  - Parts 3 and 4: Monday, December 9, 2019

## Introduction

The goal of this project is to implement a simple virtual file system that can be used to access the information stored in `task_struct` of each process.

## Recommended Background Reading

- Process Management Chapter3 Linux Kernel Development, Robert Love
- File System Implementation
- Creating Linux virtual filesystems
- lwnfs code

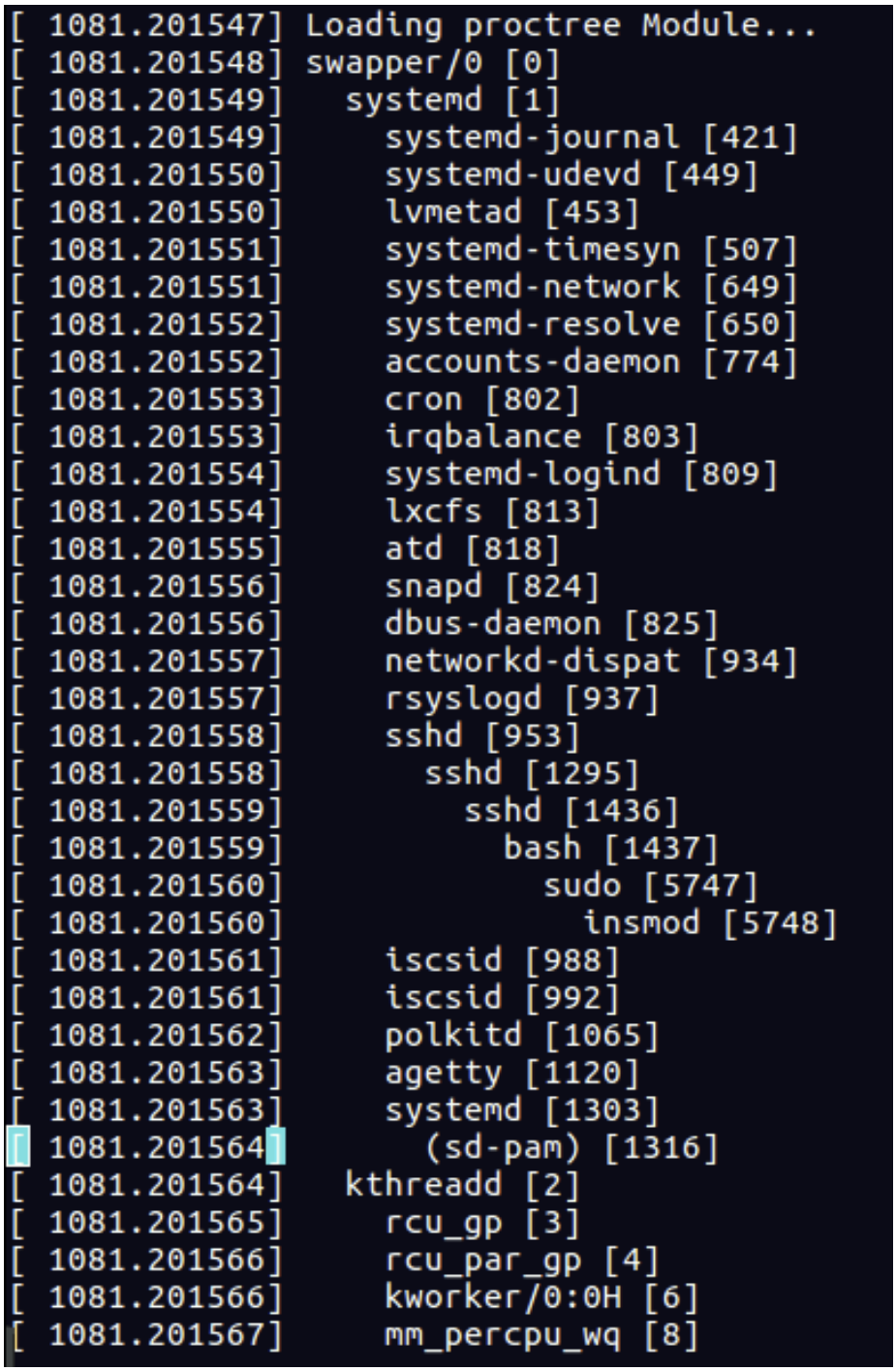
## Part 1. Print the Process Tree of Linux

### Part 1.1. Implement a kernel module

[5 points] Linux kernel stores the information of a process in task_struct structure. A process can create zero or more processes called child process. Children of same process are called sibilings. task_struct stores this information using children and sibling pointer in task struct. The children and sibling pointer together form a tree call the process tree of the kernel. In part 1, you have to iterate recursively from the root of the process tree to print the process tree in a hierarchical fashion (children of same parent should be at same indent level). Please submit you code as a tarball named s2fs_part_1.tar.gz and it should include a makefile to compile and load the kernel module.

## Part 1.2. Test your code

[5 points] To test you code, load the kernel module and check the dmesg output. Compare your dmesg output with the output of pstree command. Upload a screen shot of the dmesg output.


[ 1081.201547] Loading proctree Module...
[ 1081.201548] swapper/0 [0]
[ 1081.201549]   systemd [1]
[ 1081.201549]     systemd-journal [421]
[ 1081.201550]     systemd-udevd [449]
[ 1081.201550]     lvmetad [453]
[ 1081.201551]     systemd-timesyn [507]
[ 1081.201551]     systemd-network [649]
[ 1081.201552]     systemd-resolve [650]
[ 1081.201552]     accounts-daemon [774]
[ 1081.201553]     cron [802]
[ 1081.201553]     irqbalance [803]
[ 1081.201554]     systemd-logind [809]
[ 1081.201554]     lxcfs [813]
[ 1081.201555]     atd [818]
[ 1081.201556]     snapd [824]
[ 1081.201556]     dbus-daemon [825]
[ 1081.201557]     networkd-dispat [934]
[ 1081.201557]     rsyslogd [937]
[ 1081.201558]     sshd [953]
[ 1081.201558]       sshd [1295]
[ 1081.201559]         sshd [1436]
[ 1081.201559]           bash [1437]
[ 1081.201560]             sudo [5747]
[ 1081.201560]               insmod [5748]
[ 1081.201561]     iscsid [988]
[ 1081.201561]     iscsid [992]
[ 1081.201562]     polkitd [1065]
[ 1081.201563]     agetty [1120]
[ 1081.201563]     systemd [1303]
[ 1081.201564]       (sd-pam) [1316]
[ 1081.201564]   kthreadd [2]
[ 1081.201565]     rcu_gp [3]
[ 1081.201566]     rcu_par_gp [4]
[ 1081.201566]     kworker/0:0H [6]
[ 1081.201567]     mm_percpu_wq [8]

# Part 2: Mount and Unmount a Pseudo File system

A pseudo filesystem is not backed by disk but resides in memory and is usually used to provide information regarding the kernel to the user (e.g., proc file system). In Part 2, you will have to create a minimal mountable prototype of a pseduo filesystem which we will name `s2fs`. For this part, `s2fs` should mount and unmount on a mount point safely. Implementation of file operation is not required. Please read the following article to understand how to write a simple VFS using libfs.

- Creating Linux virtual filesystems

The article is for Linux 2.6, so some of the API might have been renamed/removed but it is a good starting point. Code that compiles for linux 4.18. Understanding the full source code is highly recommended.

- lwnfs code

## Part 2.1: Minimal Mountable Filesystem Prototype

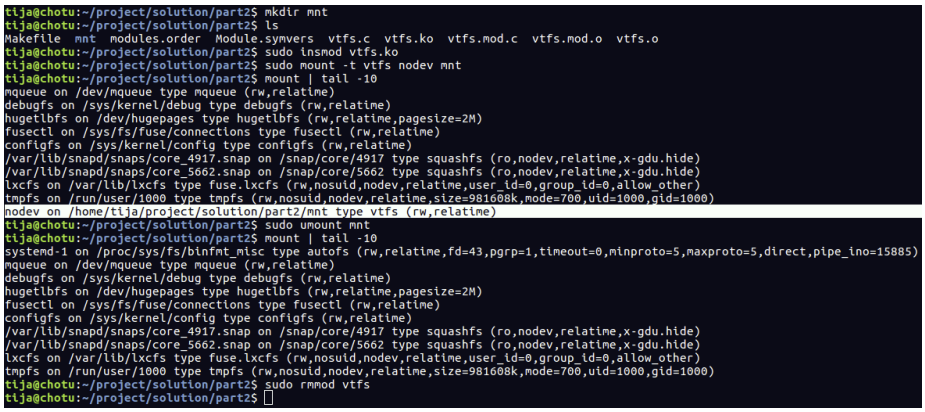
[10 points] Following is a guide to make a minimal muntable filesystem prototype - Writing a File System in Linux Kernel

Please note the above tutorial is for disk backed filesystem. Use the code in Part 2.1 and the above guide to write a filesystem prototype. You do not need to implement file operations. The file system should mount and unmount safely. Submit you code as a tarball named `s2fs_part2.tar.gz`. Include a makefile to compile, load and mount the kernel module. The filesystem will be mounted to directory `mnt` in the current folder.

## Part 2.2: Test Your Code

[10 points] To check whether you filesystem mounts successfully use the following commands.

```
$ mkdir mnt
$ sudo insmod s2fs.ko # Inserting the filesystem kernel module
$ sudo mount -t s2fs nodev mnt # Mount at dir mnt with option nodev
$ mount # Print out all mounted file systems
$ sudo umount ./mnt # Unmount the file system
$ sudo rmmod s2fs.ko # Remove the kernel module
```




Add the screenshot of the output of mount command. If mount is successful, the command will show s2fs as a mounted file system.

# Part 3: Implement File Operation

Now we will add inode and file operations to `s2fs`. In this part, on mounting, `s2fs` will create a directory named foo and inside the directory it will create a file named `bar`. You can use code from Part 2.1.

## Part 3.1: Function to create Inode with given mode

[5 points] Write a function: `static struct inode *s2fs_make_inode(struct super_block *sb, int mode);`. The function accepts two input, the superblock of the filesystem and the mode which decides if the inode is for a directory or file and the permission bits. Make sure you set the `i_ino` field of inode to `get_next_ino()`. Post a file name and line number of your implementation.

## Part 3.2: Function to create directory

[5 points] First write a function to create a directory. The function prototype will be `static struct dentry *s2fs_create_dir(struct super_block *sb, struct dentry *parent, const char *dir_name);` Post a file name and line number of your implementation.

## Part 3.3: Functions to perform file operations

[5 points] To handle a file, the filesystem needs to know how to open, read and write a file. Write three functions, `s2fs_open`, `s2fs_read_file`, and `s2fs_write_file`. In this project we will not use the open and the write fuction so it will just return 0. The read function should return "Hello World!" string to the user. Create a `s2fs_fops` of type `file_operations` and assign

`.read`, `.write` and `.open` to the functions you have written. Post a file name and line number where you define `s2fs_fops`.

### Part 3.4: Fuction to create file

[5 points] Write a function: `static struct dentry *s2fs_create_file(struct super_block *sb, struct dentry *dir, const char *file_name)` It should create a file with name stored in `file_name` inside directory pointed by `dir` dentry. Post a file name and line number of your implementation.

### Part 3.5: Putting it all together

[5 points] After mounting add code which will create a directory named `foo` in the root folder and inside `foo` create a file named `bar`. They can be created using the `s2fs_create_dir` and `s2fs_create_file` functions. Submit a tarball with your code and makefile named `s2fs_part3.tar.gz`.

### Part 3.6: Testing the code

[10 points] First mount the filesystem. The change directory to `foo`. Then use `cat bar` to check if reading the file outputs `Hello World!`

```
$ sudo insmod s2fs.ko
$ sudo mount -t s2fs nodev mnt
$ cd mnt/foo
$ cat bar
$ cd ../..
$ sudo umount ./mnt
$ sudo rmmod s2fs.ko
```

Add the screenshot of the above commands to your submission.

# Part 4: Use s2fs to get task information

In part 4, we will use part 1 and part 3 to display information related to task.

## Part 4.1: Creating files/directory corresponding to each PID

### Part 4.1.1: Create files/directory on mounting

[5 points] Modify the part 3 code to create directories and files at mount time in a recursive manner as explained below:

- If a task has children
  - Create a folder with name as the pid of the task
  - Inside the created folder create a file with the pid of the task
  - Recursively follow the rule for all children of the task
- If a task has no children create a file in it's parent task's folder

For example, if 0 has children 1, 2, 3 and 1 has children 4. Then the directory structure will look like as follows.

```
0
|--0
|--1
|  |--1
|  |--4
|--2
|--3
```

Post a file name and line number of your implementation.

### Part 4.1.2: Testing Code

[5 points] Mount the file system and run `tree` command on the mount folder to see the directory tree. Add screen the screenshot of the same in your submission.

## Part 4.2: Reading a file with name X should give task info of PID X

For part 4.2 you have to modify the `s2fs_read_file` and `s2fs_open` function, so that when you read a file with name X it will output the task info of task with PID X.

### Part 4.2.1: Get Task Info

[5 points] Write a function `int get_task_info(int pid, char* data)` which takes input a pid and a buffer and returns number of bytes written to buffer and

the task information in the `data` buffer. If the task does not exists, it should return -1. The information to be returned are:

- Task Name
- Task State
- Process Id
- CPU ID
- Thead Group ID (TGID)
- Parent's PID (PPID)
- Start Time
- Dynamic Priority
- Static Priority
- Normal Priority
- Real-time Priority
- Memory Map Base
- Virtual Memory Space
- Virtual Memory Usage
- No. of Virtual Memory Address
- Total Pages Mapped

Post a file name and line number of your implementation.

### Part 4.2.2: Modify the inode creation file to store the PID

[5 points] When creating file for a task, store the PID in `inode->i_private`. Post a file name and line number where you assign `inode->i_private`.

### Part 4.2.3: Modify the file open and read function

[5 points] Modify the `s2fs_open` function to get the pid from the file inode and store it in `filp->private_data`. Modify the `s2fs_read_file` function to call `get_task_info` and output the task information instead of `Hello World`. If the task does not exists, output `Task no longer exists.` Create a tarball named `s2fs_part4.tar.gz` of the code and makefile.

### Part 4.2.4: Test your code

[10 points] Add screenshot of following commands.

```
$ mkdir mnt
$ sudo insmod s2fs.ko # Inserting the filesystem kernel module
$ sudo mount -t s2fs nodev mnt # Mount at dir mnt with option nodev
$ mount # List all mounted file system
$ cat mnt/0/1/1 # Print out task information of the systemd process
$ sudo umount ./mnt # Unmount the file system
$ sudo rmmod s2fs.ko # Remove the kernel module
$ dmesg
```