Term Project 1: PROCRANK kernel module

Due Date: 5/29 (Wed.), 11:59 PM
Submission: i-Campus

1. Project Summary

In this assignment, you will implement a Linux kernel module, called procrank, to get the memory usage information of all processes via /proc.
You will study the following skills and knowledge throughout this assignment.

(1) How to build Linux kernel image
(2) How to extend the functionalities of Linux kernel with loadable kernel modules
(3) How to get kernel information at user space via /proc
(4) Important data structures for virtual memory management in Linux kernel

2. How to build Linux kernel image

- Since you will modify kernel, your system may be crashed frequently during development. So, you'd better to use a virtual machine environment such as VMware at your computer.
- Setup Ubuntu 16.04 or 18.04 machine at VMware.
- Download **5.0.9** kernel from https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.0.9.tar.gz
- Untar kernel: `$tar -zxvf linux-5.0.9.tar.gz`
- Download the package for kernel build
  `$ sudo apt install build-essential libncurses5-dev bison flex libelf-dev libssl-dev`
- Build kernel (root or sudo command)
  `$ cd linux-5.0.9`
  `$ make menuconfig` // select exit!
  `$ make -j8`
  `$ make modules_install`
  `$ make install`
- reboot and check your kernel version
  `$ uname -a`

3. How to make your modules

- The Linux kernel module is an object file that contains code to extend the running kernel. The kernel module can be loaded or unloaded dynamically while kernel is running.
- You can study the basic of linux kernel module at The Linux Kernel Module Programming Guide
- The skeleton code for procrank (procrank.c) is provided for this project, so you can modify it to make your own module.
  - The init/exit functions
    - ✓ procrank_init() will be called at module loading
    - ✓ procrank_exit() will be called at module unloading

- Modify Makefile
  - **KDIR** must be your kernel source directory

```
obj-m := procrank.o
KDIR := /home/eslab/linux-5.0.9      # Your kernel path

default:
    make -C $(KDIR) M=$(shell pwd) modules

clean:
    make -C $(KDIR) M=$(PWD) clean
```

- Build the kernel module (in module directory)

  `$ make`
  - If the build is successful, you can find a kernel object file "procrank.ko".
- Test your modules
  - Load your module: `$ insmod procrank.ko`
  - Unload your module: `$ rmmod procrank`
  - You can check the generated message via dmesg command

```
root@eslab-System-Product-Name:~# dmesg | tail -n 2
[ 6909.117810] init procrank ID:2019123456
[ 6911.117240] exit procrank ID:2019123456
```

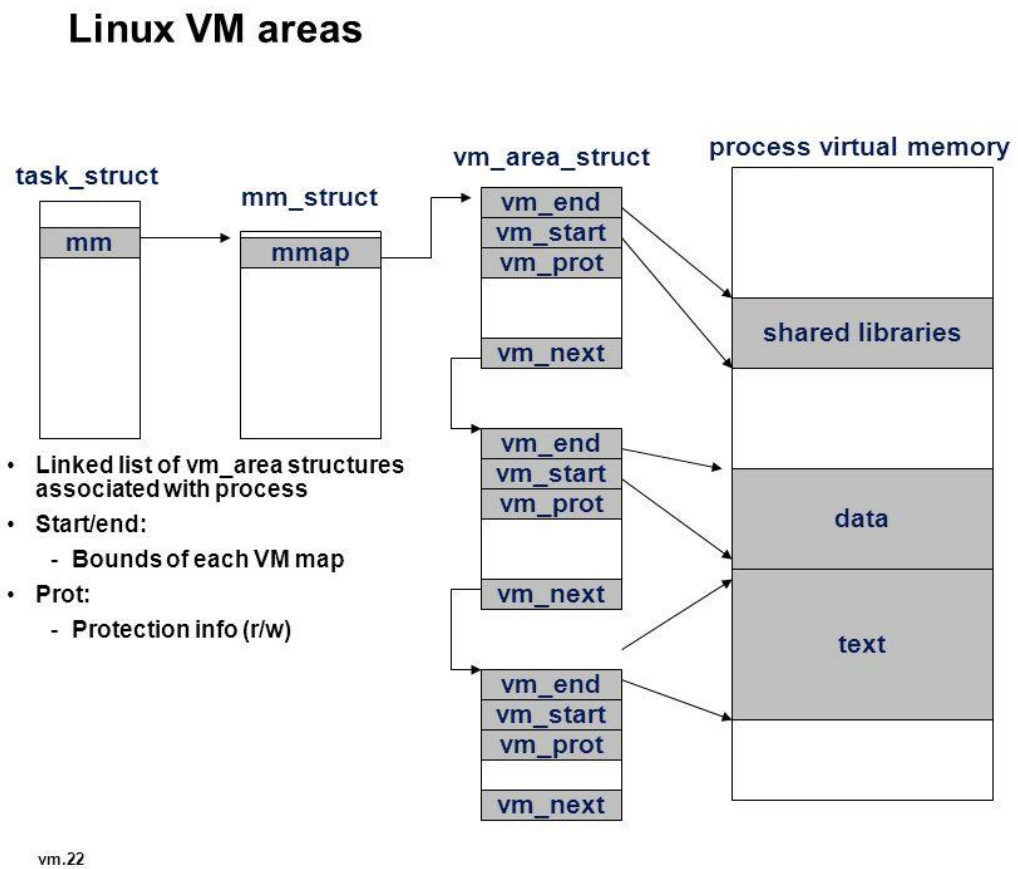
4. What is /proc filesystem

- proc file system
  - A pseudo-filesystem which provides an interface to kernel data structures.
  - For more information, you can refer to https://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html
  - When you use "cat" command to read a proc file, the registered show() function is called, which must be passed to proc_create_single().
    - example) `$ cat /proc/meminfo`
- To create proc file and register a show function at module loading, add the call of proc_create_single() at Init function
  - `proc_create_single(const char *name, umode_t mode, struct proc_dir_entry *parent, int(*show) (struct seq_file*, void *))`
    - name: proc name
    - parent: parent directory, (NULL: in /proc)
    - show: show function
  - When you input "insmod procrank.ko" at shell, "/proc/procrank" will be created.
- To remove proc file at module unloading, add the call of remove_proc_entry() at Exit function
  - `remove_proc_entry(const char *name, struct proc_dir_entry *parent)`
  - When you input "rmmod procrank" at shell, "/proc/procrank" will be removed.

```
root@eslab-System-Product-Name:~# ls /proc/procrank
ls: cannot access '/proc/procrank': No such file or directory
root@eslab-System-Product-Name:~# insmod linux-5.0.9/hw/procrank.ko
root@eslab-System-Product-Name:~# ls /proc/procrank
/proc/procrank
root@eslab-System-Product-Name:~# rmmod procrank
root@eslab-System-Product-Name:~# ls /proc/procrank
ls: cannot access '/proc/procrank': No such file or directory
```

5. Study kernel structures

- To understand process memory address space, you need to see the following structures.
  - task_struct: include/linux/sched.h
  - mm_struct: include/linux/mm_types.h
  - vm_area_struct: include/linux/mm_types.h
  - You can see a basic concept of the linux kernel in Understanding the linux kernel (Chapter 9: Process's address space)
  - The following figure shows the overall virtual memory management structures.



6. Study the definition of VSS, RSS, USS, PSS

- Useful link: http://www.2net.co.uk/tutorial/procrank
- From the website, you can find the definition of memory usage metric
- VSS (Virtual Set Size): the total virtual memory size for a process
- RSS (Resident Set Size): the total memory actually held in RAM for a process (physical page count)
- USS (Unique Set Size): the set of pages that are unique to a process. This is the amount of memory that would be freed if the application was terminated right now
- PSS (Proportional Set Size): the amount of memory shared with other processes, accounted in a way that the amount is divided evenly between the processes that share it. This is memory that would not be released if the process was terminated, but is indicative of the amount that this process is contributing
- You can refer to "procrank" source code
    - https://github.com/csimmonds/procrank_linux
    - `pm_map_usage_flags()` in libpagemap/pm_map.c

7. Make your own procrank

7.1 Analyze the core data structures related to the projects
Your report should describe the relationship among the following kernel data structures and the memory management le in Linux kernel.
: task_struct, vm_area_struct, struct mm_struct

7.2 Implement your own loadable kernel module with the provided skeleton code for procrank (procrank.c)

- procrank should print PID, process name, VSS, PSS, RSS, USS for each process.
    - The skeleton code can already print the PIDs of all processes.
- You can use the kernel API "seq_printf()" to print message
    - #include <linux/seq_file.h>
    - `seq_printf(struct seq_file *m, const char *f, …)`
- You can traverse all processes with the kernel macro "for_each_process"
    - #include/linux/sched/signal.h
    - The name of process can be found in task_struct
    - See an example code: "`show_smaps_rollup()`" // include/task_mmu.c

```
/// in for_each_process
struct vm_area_struct *vma;
for (vma = p->mm->mmap; vma; vma = vma->vm_next) {
        // YOUR CODE
}
```

- Get VSS, PSS, RSS, and USS by traversing vm_struct_area in a process
    - PSS and RSS can be obtained directly from vm_struct_area
    - See an example code: "`show_smap()`" // include/task_mmu.c

```
// vm_area_struct *vma
struct mem_size_stats mss;
memset(&mss, 0 sizeof(mss));
smap_gather_stats(vma, &mss);
// Now, you can get RSS (mss->resident), PSS (mss->pss)
```

- VSS can be calculated from the elements of vm_start and vm_end in vm_area_struct
- USS can be calculated from PSS and RSS.

8. Requirements

- /prc/procrank must print the following information for all processes. One line per one process.

PID	Process_NAME	VSS	RSS	USS	PSS

- Sort by PSS in descending order
- Experiment scenario 1 (VSS and RSS)
  - Test with two programs mem (source code is mem.c in Homework Chap21) and mem2 (mem2.c modified based on mem.c)
  - mem2 allocates memory as mem.c does. However, it does not access all allocated memory pages.
  - Run each program wihile allocating 1MB or 5MB.
    - four test cases of `$./mem 1 $./mem 5 $./mem2 1 $./mem2 5`
  - Check the VSS and RSS while running mem and mem2 and discuss the result.
- Experiment scenario 2 (USS and PSS)
  - Run mem and mem2 simultaneously
    - Two test cases of `$./mem 1&;./mem2 1& $./mem 5&;./mem2 5&`
  - Check the USS and PSS while running two programs and discuss the result.
- Experiment scenario 3 (Shared library)
  - Copy a libc file to your working directory
    - $export LD_LIBRARY_PATH={your working directory}
    - $cp /lib/x86_64-linux-gnu/libc.so.6 {your working directory}
    - You can check the library link through ldd command

```
libc.so.6 => /home/who/testdir/libc.so.6 (0x00007f514f846000)
```

  - Run mem and mem2 while allocating 1MB or 5MB memory.
  - Check the USS and PSS and discuss the result.

9. Evaluation Policy (100 points)

9.1. Implementation (50 points)

- Test 1: $ insmod procrank.ko $ ls /proc/procrank $ rmmod procrank $ ls /proc/procrank
  - We will check the creation and deletion of the file of "procrank" in the /proc filesystem (5 points)

- Test 2: `$ insmod procrank.ko` `$ cat /proc/procrank`
    - Print PID & NAME of all processes (10 points)
    - Print the exact value of VSS, RSS, PSS, and USS (30 points)
    - Sorted by PSS (5 points)

9.2. Report (50 points)

- Kernel data structure study (10 points)
- Detailed explanation of your implementations (10 points)
- Experiment 1 (VSS and RSS) (10 points)
- Experiment 2 (USS and PSS) (10 points)
- Experiment 3 (Shared Library) (10 points)

10. Submission Guidelines

- The final output should be a compressed zip file named [student_id]_proj1.zip, which includes [student_id]_report.pdf and procrank.c
    - At the first line in procrank.c, write your name and student ID
        - // Gildong Hong, 2019123456
    - You don't need to submit Makefile
- Write your development environment in the report
- Explain the key parts of your code in the report.
  Don't attach the whole source codes in the report. Just key parts.
  (you will submit your codes too.)
- Submit at i-Campus.

11. Notice

- Your term project must be your original work.
- If you have any questions, describe your problem along with screenshots and your environment, and send it to oso0931@gmail.com, or visit 400309.
    - Your environment includes OS (kernel version), installation setup (VM, desktop).
- In the case of cheating, you will **FAIL** this course.
    - If two similar source codes or reports are found, both students will fail this course.
- Delay penalty is -5 points per day.