Institute of Technology
University of Washington – Tacoma
Instructor: Wes Lloyd

Assignment 2

Process Reporter - Linux Kernel Module

Due Date: Monday November 14th, 2016 @ 11:59 pm, tentative

Objective

The purpose of this assignment is to create a Linux Kernel Module that generates a report describing the running processes on a Linux system. This module will traverse the list of running processes, and introspect information about them. A sample kernel module has been provided on the course web page to assist with getting started.

Key objectives of the assignment include working with the Linux Kernel linked list functions to: (1) iterate through the master process list, and (2) drill down into a process's list of child processes. In addition to working with the Linux kernel process list, a secondary objective of this assignment is to generate and provide report output to the console. Linux kernel modules and kernel routines often provide a "file" based interface to interact with users. A number of proc interfaces are found under the "/proc". These /proc interfaces are dynamically generated files produced by kernel code. They can be generated by native kernel code or by kernel modules. As of Linux kernel version 3.10 the kernel API to support creation of a proc file has changed. The default Centos 7 Linux kernel version is >=3.10.x.

It may be easier to divide and conquer the objectives. Tackling generation of the report independently (and probably before) the output part requirements of the module may be easier. Most of the credit for the assignment (75%) is weighted on setting up and producing the report, not the /proc output.

To support development of the report, write a function (or set of functions) to generate the report. **Report generation and process list computation cannot occur in the /proc output routines.** In following with good design and coding practices the report and output routines should be decoupled. To support development, debuggers can be used, or information can be written to the /var/log/messages file on CentOS.

This file can be traced in a separate terminal window using the command:

```
sudo tail -fn 50 /var/log/messages
```

In the kernel module here are some example print statements:

```
printk(KERN_INFO "My string=%s\n",text);
printk(KERN_INFO "My index=%d\n",idx);
printk(KERN_INFO "My pointer=%lu\n", (unsigned long) myptr);
```

The sample kernel module is here:
http://faculty.washington.edu/wlloyd/courses/tcss422/assignments/hello_module.tar.gz

To extract the sample kernel module:
```
tar xzf hello_module.tar.gz
```

To build the sample module:
```
cd hello_module/
make
```

To remove a previously installed the module:
```
sudo rmmod ./helloModule.ko
```

To install a newly built module:
```
sudo insmod ./helloModule.ko
```

This sample kernel module prints messages to the kernel logs.
The "`dmesg`" command provides a command to interface with kernel log messages, but it is simple enough to just trace the output as described above.

***** THE KERNEL MODULE SHOULD BE RENAMED TO "procReport" *****
FAILURE TO RENAME THE MODULE WILL RESULT IN A 10 point deduction.

The kernel module should produce output as below. Output should startwith a line that says "PROCESS REPORT", and follow with one row of text for each running process. Here is a partial output example:

PROCESS REPORT:
Process ID=1 Name=systemd number_of_kids=67 first_child_pid=495 first_child_name=systemd-journal
Process ID=2 Name=kthreadd number_of_kids=86 first_child_pid=3 first_child_name=ksoftirqd/0
Process ID=3 Name=ksoftirqd/0 *No Children
Process ID=5 Name=kworker/0:0H *No Children
Process ID=7 Name=migration/0 *No Children
Process ID=8 Name=rcu_bh *No Children
Process ID=9 Name=rcuob/0 *No Children
Process ID=10 Name=rcuob/1 *No Children
Process ID=11 Name=rcuob/2 *No Children
Process ID=12 Name=rcuob/3 *No Children

Here is a verbose description of the process output row:

"Process ID=" followed by the process ID (integer) of the current process during a traversal of the process list. Then a space, followed by "Name=" and then the name of the process (string).
If the process has children first display "number_of_kids=" followed by the number of children processes started by this process (integer), followed by a space, and then "first_child_pid=" followed by the process ID of the first child created (integer), followed by another space, and then "first_child_name=" followed by the name of the child process (string).
If the process does not have children, simply print "*No Children".

IF SOME FUNCTIONALITY IS MISSING IN YOUR KERNEL MODULE, PLEASE FOLLOW THE OUTPUT FORMAT AND USE PLACEHOLDERS. For example, use a placeholder like: "number_of_kids=XX"

To support development of your kernel module output, it may be helpful to begin by writing code that produces the report, and then print this report to the system log files with `printk`.

Here are some references describing how to create the proc file kernel module interface:

http://www.crashcourse.ca/introduction-linux-kernel-programming/lesson-11-adding-proc-files-your-modules-part-1
http://stackoverflow.com/questions/8516021/proc-create-example-for-kernel-module/

Kernel modules should have a name in the /proc directory.
Please name your module: "**`proc_report`**".

*****Failure to follow the naming convention will result in a loss of 10 points.*****

Grading

This assignment will be scored out of 90* points. (90/90)=100%
* *If necessary the total points scored from may be lowered, while the total available points remains 100.*

Rubric:
100 possible points: (Currently 10 extra credit points are available)

Report Toal:	60 points
10 points	Output of the PID of each running process
10 points	Output of the program name of each running process
10 points	Output of the number of children processes started by every process
	>>> 5 points for the count,
	>>> 5 points for reporting when there are No Children
10 points	Output of the PID of the first child process for every process having children
10 points	Output of the program name of the first child for every process having children
10 points	The ability to generate and provide output of the report multiple times:
	>>> 5 points – reloading your kernel module
	>>> 5 points – crashing the machine

Output Total:	25 points
25 points	Report output uses the Linux /proc
	>>> 10 points - decoupling output routines from report generation

Miscellaneous:	15 points
5 points	Kernel module builds and installs
5 points	Coding style, formatting, and comments
5 points	Following the Output requirements as described above (even without any output)

WARNING!	
10 points	Automatic deduction if your kernel module is not called "procReport"
10 points	Automatic deduction if your /proc directory entry is called something other than "proc_report"

What to Submit

For this assignment, submit a tar gzip archive as a single file upload to Canvas.

Tar archive files can be created by going back one directory from the kernel module code with "`cd ..`", then issue the command "`tar czf hello_module.tar.gz hello_module`". Name the file the same as the directory where the kernel module was developed but with ".tar.gz" appended at the end: `tar czf <module_dir>.tar.gz <module_dir>`.

Please rename modules to something other than hello_module.

To rename a directory in Linux use: "`mv hello_module my_proc_module`".

Pair Programming (optional)

O*ptionally*, this programming assignment can be completed with two person teams.

If choosing to work in pairs, ***only one*** person should submit the team's tar gzip archive to Canvas.

Additionally, ***EACH*** member of a pair programming team must write a one page summary of their contributions and their teammate's contributions to the overall project. This one page write up must be written and submitted INDEPENDENTLY by each team member to capture each person's overall view of the teamwork and outcome of the programming assignment. Contribution summaries should be submitted in confidence to Canvas as a PDF file named: "pair_contributions.pdf". Google Docs and modern versions of MS Word provide the ability to save or export a document in PDF format.

Here is an example write up:

1. Jane Smith

   Jane contributed by writing function_A and function_B in the program. Jane also researched several APIs on the web, and was crucial at determining how algorithm ABC worked. Jane's ideas supported the design of the program output. Jane also discovered and helped code process API routines.

2. John Doe

   John contributed by helping Jane correct a few errors with the implementation of function_A and function_B. Additionally, John wrote most of function_C. John researched how the Linux system Linked List API works for reading and processing from list data structures. John found 3 key routines in the API and deciphered how to use them.

Team members may not share their write ups, but should submit them independently in Canvas as a PDF file. Failure of one or both members to submit the contributions report will result in both members receiving NO GRADE on the assignment… (*considered late until both are submitted*)

<u>Disclaimer regarding pair programming:</u>

The purpose of TCSS 422 is for everyone to gain experience programming in C while working with operating system and parallel coding. Pair programming is provided as an opportunity to harness teamwork to tackle programming challenges. But this does not mean that teams consist of one champion programmer, and a second observer simply watching the champion! The tasks and challenges should be shared as equally as possible.