Implementing a Loadable Kernel Module

Summary: In this homework, you will be implement a loadable kernel module that uses Linux data structures to display details about the processes executing in the kernel.

1 Background

The kernel is a program that is the core of a computer's operating system, with complete control over everything in the system. It is the first program loaded on start-up. It performs tasks such as running processes, handling interrupts and all other handling that happens behind the scenes in kernel space whereas everything user does is executed in user-space. This separation prevents user data and kernel data from interfering with each other and causing instability and slowness.

Kernel modules are pieces of code which provide functionality to the kernel. The word "Loadable" is prefixed to kernel module to show that it eases the process of integrating the module with the kernel. Generally, if any functionality is to be provided in the kernel, it has to incorporated by compiling the kernel with the proposed changes. Loadable Kernel Modules (LKM) ease that process by providing the facility to integrate itself within the kernel instead of requiring kernel re-compilation. **For information on creating LKMs, see your textbook (Operating System Concepts 9th edition) page 96**. You will need to use a so-called "makefile" to manage the compilation of your module. **It will not be possible to use NetBeans to develop your code.** The best way to start is with the sample LKM provided on the textbook's website.

In this assignment you will write a LKM for the Linux kernel that displays the certain details of the processes (along with its parent and child) executing in the kernel. As an important step to implementing the program, you should review the section on include files to get an idea of what functionality is available. From there, you may choose to outline your program in pseudocode, and then convert it into actual source code, while reading the documentation for any functions you used. ***Expect that a significant part of your time on this assignment will be spent reading/researching/understanding LKM documentation and reviewing kernel source code files (linux/sched.h, and linux/list.h).*** The source code should be relatively short. For kernel source code, visit www.kernel.org/. Any lines numbers mentioned in this document refer to Linux 4.9.6; please reference that version as well.

This document is separated into five sections: Background, Requirements, Makefiles, Include Files, and Submission. You have almost finished reading the Background section already. In Requirements, we will discuss what is expected of you in this homework. In Include Files, we discuss several header files that include functionality related to file and directory manipulation. Lastly, Submission discusses how your source code should be submitted on BlackBoard.

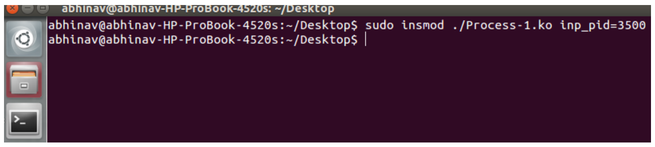
2 Requirements [36 points total]

In this assignment you will write a LKM for the Linux kernel that displays the following details for all the processes whose PID is greater than an integer given by the user as a module parameter: [26 points]

- PROCESS NAME
- PID
- STATE
- PRIORITY
- STATIC-PRIORITY

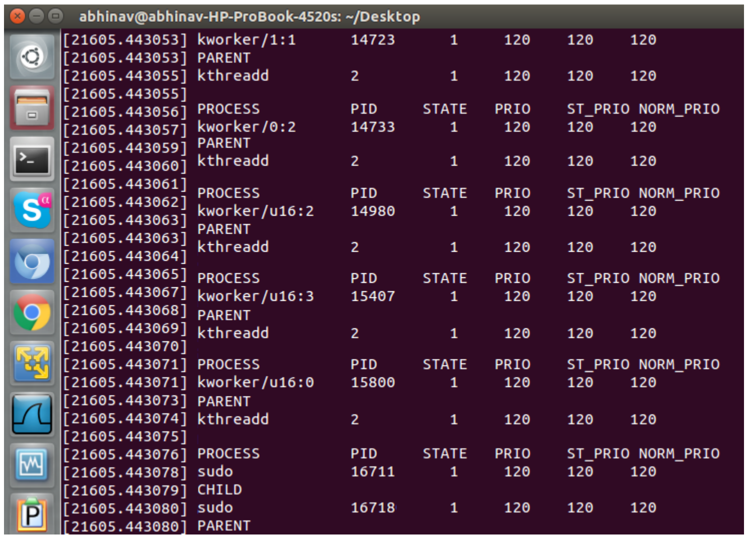
- NORMAL-PRIORITY

As shown below, the PID is given as arguments while inserting the module in the kernel.



```
abhinav@abhinav-HP-ProBook-4520s: ~/Desktop
abhinav@abhinav-HP-ProBook-4520s:~/Desktop$ sudo insmod ./Process-1.ko inp_pid=3500
abhinav@abhinav-HP-ProBook-4520s:~/Desktop$ |
```

The next screenshot shows the required details for a process, its child processes (may have 0 or more), and its parent process. You are expected to keep a similar format (not an exact one) for your output which is readable. **Your programs must compile and run under Xubuntu (or another variant of Ubuntu) 18.04.**



```
abhinav@abhinav-HP-ProBook-4520s: ~/Desktop
[21605.443053] kworker/1:1      14723      1      120     120     120
[21605.443053] PARENT
[21605.443055] kthreadd         2          1      120     120     120
[21605.443055]
[21605.443056] PROCESS          PID        STATE  PRIO    ST_PRIO NORM_PRIO
[21605.443057] kworker/0:2      14733      1      120     120     120
[21605.443059] PARENT
[21605.443060] kthreadd         2          1      120     120     120
[21605.443061]
[21605.443062] PROCESS          PID        STATE  PRIO    ST_PRIO NORM_PRIO
[21605.443063] kworker/u16:2    14980      1      120     120     120
[21605.443063] PARENT
[21605.443064] kthreadd         2          1      120     120     120
[21605.443065] PROCESS          PID        STATE  PRIO    ST_PRIO NORM_PRIO
[21605.443067] kworker/u16:3    15407      1      120     120     120
[21605.443068] PARENT
[21605.443069] kthreadd         2          1      120     120     120
[21605.443070]
[21605.443071] PROCESS          PID        STATE  PRIO    ST_PRIO NORM_PRIO
[21605.443071] kworker/u16:0    15800      1      120     120     120
[21605.443073] PARENT
[21605.443074] kthreadd         2          1      120     120     120
[21605.443075]
[21605.443076] PROCESS          PID        STATE  PRIO    ST_PRIO NORM_PRIO
[21605.443078] sudo             16711      1      120     120     120
[21605.443079] CHILD
[21605.443080] sudo             16718      1      120     120     120
[21605.443080] PARENT
```

2.1 Writeup [10 points]

You are to provide a short write up that illustrates what you found when looking at the Linux source files and documentation. Include the following:

- For your main source file, list each function that you defined and describe its purpose.
- For each header file (except linux/module.h), document each function/macro/struct used:
  - Functions: Include the function's signature, what file/line it is defined, and a description (include parameters and return values). Example:

    printf
    Defined on line 133 in stdio.h.
    Signature:
    int printf(const char *format, ...);
    Parameters:
    The first parameter (format) takes a string, followed by any number of other parameters (see below).
    Description:
    This string will be displayed as standard console output. The string may contain special formatting called control characters. For example, including "%c" indicates a position in the string where a character should appear. The variable containing the string should be passed as the second parameter. The function may take any number of parameters, where the first is always the main string, and everything else is a value that may be subsituted into it. After completion, the function returns the number of displayed (an int).

  - Macro: same as function.
  - Struct: Indicate what file/line it is defined, and document each of the attributes used.

The purpose of the write up is show the “research” you did when creating your program. Do not quote (or paraphrase) any 3rd party documentation - use your own words and understanding. Although there is no mandatory format, please be professional. Your document should be self-contained, well styled (e.g., no comic sans, random font sizes), and readable (e.g., no spelling or grammar errors).

3 Makefiles

A makefile is a file containing a set of directives used with the *make* tool to automate the build process for a file or a set of files. For example, assume we have a codebase that has hundreds of files with lots of dependencies between them. Generally, we would compile each file, generate an object file and link all the object files together to generate an executable. Even if there is a small change in one of the files, we would have to recompile and link them which is quite a cumbersome and inefficient process. Using Makefiles, you can ensure that only the files that have been modified since the last build and those which are dependent on the changed files are the ones which are compiled.

make uses the timestamp on a file to check if the file has been modified and whether it has to be recompiled or not. If you have a "makefile" in your directory which specifies the rules for compiling the files (including the dependencies), all you have to do is run "make". If you want to run only specific rules, you can specify using "make <rule>". A makefile consists of “rules” in the following form:

target: dependencies

    system command(s)

where **target** is usually the name of a file that is generated by a program like executable or object files; a **dependency** (also called prerequisite) is a file that is used as input to create the target; the **system command(s)** (also called recipe) is an action that make carries out. Note the use of meaningful indentation in specifying commands; also note that the indentation must consist of a single <tab> character. For example, if we have following dependency stated in the **Makefile**:

Algorithm 1 Generic makefile for project containing main.c and display.c.

```
edit : main.o display.o
    gcc -o edit main.o display.o

main.o : main.c
    gcc main.c —o main.o

display.o : display.c
    gcc display.c —o display.o
```

The above snapshot specifies that the target **edit** depends on **main.o** and **display.o** for its build, which further depend on their source files **main.c** and **display.c** respectively. For executing above, we need to have all the above files along with **Makefile** in the same directory. When we execute the command **make** in the same directory, the make tool tries to build the target **edit** by figuring out whether any dependencies have been modified since their last build and if so, builds those desired dependencies and produces the target **edit.** For more information on make visit GNU Makefile

For this assignment, you will use a specially set up makefile. See Algorithm 2 below. In this make file, we have two targets, *all* and *clean*. Rather than being dependent on a specific file, they are more actions that the makefile can execute. In this case, *all* is used to build the file, while *clean* is to clean the build outputs.

For us, we would only need to type "make" in the correct folder to use this makefile. The top most line is special, it says "obj-m += simple.o". This say takes the result of compiling the file simple.c (which will be called simple.o), and combine it with the default binaries for creating an module. This produce the .ko file that represents a built LKM that you can load into the operating system.

Algorithm 2 Sample LKM makefile (from Operating System Concepts by Silberschatz, Galvin, Gagne)

```
obj-m += simple.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

4 Include Files

To complete this assignment, you may find the following include files useful:

- *linux/sched.h*: Defines processes and their associated details.
  - Useful types:
    - *struct task_struct* - A structure to contain process details (see line 1475)
  - Useful macros:
    - *for_each_process* - A macro to loop over each currently running process (see line 503). (On more recent kernels, this may be defined in linux/sched/signal.h)
- *linux/list.h:* Implements a circular-linked list in C language which could be used to fetch the child processes.
  - Useful types:
    - *struct list_head*
  - Useful functions:
    - *list_for_each*
    - *list_entry*

Note that you are not limited to using these functions, nor are you required to use all of them.

5 Submission

The submission for this assignment has two parts: a write up, and the source code (optional: include the Makefile which compiles the module). The file(s) should be attached to the homework submission link on BlackBoard.

Writeup: For this assignment, you should provide a write up as discussed in PDF format. Please name your file as "LastNameLKM.pdf" (e.g. "JainLKM.pdf") using the write up submission link.

Source Code: Please name your file(s) as "LastNameLKM.zip" (e.g. "JainLKM.zip"), which should contain a ".c" of the same name (e.g., JainLKM.c), and optionally a makefile. Do NOT include your write up or any binaries.

• If your program fails to compile out-of-the-box, there will be a mandatory deduction 20% from the assignment points.

• If you do not follow the file submission standards (e.g., the submission contains project files, lacks a proper header), there will be a mandatory deduction of 10% from the assignment points.

• If compiling or running your program differs from stated on the assignment and it is not specified as a requirement, please include a readme file with steps to execute the program. If you do not, there will be a mandatory 10% deduction from the assignment points.