

Implementing a Loadable Kernel Module

Summary: In this homework, you will 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 forms the core of a computer's operating-system, with control over everything in the system. It performs tasks such as running processes, allocating resources, interfacing with hardware, and all other tasks that happen behind the scenes in “kernel space”. In contrast, everything the user does is executed in “user space”. This separation prevents user data and kernel data from interfering with each other and causing instability or security issues.

Kernel modules are pieces of code that 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. Typically, if functionality is to be provided in the kernel, it has to be incorporated by compiling the kernel with the proposed changes. Loadable Kernel Modules (LKM) ease that process by providing the facility to integrate with the existing kernel instead of requiring kernel re-compilation. The best way to start is with the sample LKM provided on the textbook's website (also posted on Canvas), and then add your own functionality.

In this assignment, you will write an LKM for the Linux kernel that displays certain details of the processes (along with its parent and children) executing in the kernel. As an important step in implementing the program, you should review the section on include files to get an idea of what functionality is available. From there, start to develop the actual source code, while reading the documentation for any functions you used. Much of the documentation exists as source code with the Linux kernel, meaning that you will need to read the code yourself. ***The assignment source code should be relatively short but uses features you will have to research on your own (such as passing a value to a LKM, or how to manipulate the list of running processes). 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 (such as linux/sched.h, and linux/list.h).*** The website for Linux kernel source is www.kernel.org/. Any line numbers mentioned in this document refer to Linux 5.11. The specific version of the Linux kernel on your machine may differ slightly.

This document is separated into five sections: Background, Running an LKM, Requirements, 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 Running an LKM, a basic demonstration of an LKM is given. 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 Canvas.

1.1 Accessing Kernel Source Code

To find the version of your kernel, use the command “uname -r”. Below the version is shown as 5.11.0-34-generic. The screenshot comes from a VM running an install of Xubuntu 20.04.3 (which can be checked by running “lsb_release -a”). (Your version will most likely be more recent.) The bit of text after 5.11 (the major and minor version information) indicates the patch version. Depending on how up to date your machine is, you may see a more recent kernel version. When you submit your write up, you will be asked to list the version of the kernel you use to find line numbers. (The version is important since line numbers change as updates occur.)

```

Terminal - ruben@ruben-VirtualBox: ~/Desktop
File Edit View Terminal Tabs Help
ruben@ruben-VirtualBox:~/Desktop$ uname -r
5.11.0-34-generic
ruben@ruben-VirtualBox:~/Desktop$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.3 LTS
Release:        20.04
Codename:       focal
ruben@ruben-VirtualBox:~/Desktop$

```

To see source code for different kernels, use: <https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/refs/tags> and find the version closest to the one installed on your VM. Following from the screenshot above where the machine was running “5.11.0-34-generic”, we would use the “v5.11” version below. Also shown in the list is v5.11.1 which would be a newer version (the screenshot’s version ends in .0 not .1).

index : kernel/git/stable/linux.git

Linux kernel stable tree

about summary **refs** log tree commit diff stats

Tag	Download	Author	Age
v5.15-rc3	linux-5.15-rc3.tar.gz	Linus Torvalds	2 days
v5.14.8	linux-5.14.8.tar.gz (sig)	Greg Kroah-Hartman	2 days
v5.10.69	linux-5.10.69.tar.gz (sig)	Greg Kroah-Hartman	2 days
v5.4.149	linux-5.4.149.tar.gz (sig)	Greg Kroah-Hartman	2 days
v4.19.208	linux-4.19.208.tar.gz (sig)	Greg Kroah-Hartman	2 days
v4.14.248	linux-4.14.248.tar.gz (sig)	Greg Kroah-Hartman	2 days
v4.9.284	linux-4.9.284.tar.gz (sig)	Greg Kroah-Hartman	2 days
v4.4.285	linux-4.4.285.tar.gz (sig)	Greg Kroah-Hartman	2 days

...scroll scroll...

v4.9.258	linux-4.9.258.tar.gz (sig)	Greg Kroah-Hartman	7 months
v4.4.258	linux-4.4.258.tar.gz (sig)	Greg Kroah-Hartman	7 months
v5.11.1	linux-5.11.1.tar.gz (sig)	Greg Kroah-Hartman	7 months
v5.10.17	linux-5.10.17.tar.gz (sig)	Greg Kroah-Hartman	7 months
v5.4.99	linux-5.4.99.tar.gz (sig)	Greg Kroah-Hartman	7 months
v5.11	linux-5.11.tar.gz (sig)	Linus Torvalds	7 months
v5.10.16	linux-5.10.16.tar.gz (sig)	Greg Kroah-Hartman	7 months
v5.4.98	linux-5.4.98.tar.gz (sig)	Greg Kroah-Hartman	7 months
v4.19.176	linux-4.19.176.tar.gz (sig)	Greg Kroah-Hartman	7 months

Once the source code is downloaded, extract it to a local folder. You will then be able to navigate within that folder to view different files. Shown below is an example of looking up task_struct from the file linux/sched.h (see the section on Include Files later). (If you are not using the v5.11 files, then most likely the file will look different!)

linux - File Manager

File Edit View Go Help

/home/ruben/Downloads/linux-5.11/include/linux/

Name	Size	Type	Date Modified
scsi.h	20.0 KB	C header	02/14/2021
sched.h	58.1 KB	C header	02/14/2021
sched_clock.h	1.4 KB	C header	02/14/2021
scmi_protocol.h	25.4 KB	C header	02/14/2021

/home/ruben/Downloads/linux-5.11/include/linux/sched.h - Mousepad

```

630 };
637
638 struct wake_q_node {
639     struct wake_q_node *next;
640 };
641
642 struct kmap_ctrl {
643     #ifdef CONFIG_KMAP_LOCAL
644         int idx;
645         pte_t pteval[KM_MAX_IDX];
646     #endif
647 };
648
649 struct task_struct {
650     #ifdef CONFIG_THREAD_INFO_IN_TASK
651         /*
652          * For reasons of header soup (see current_thread_info()), this

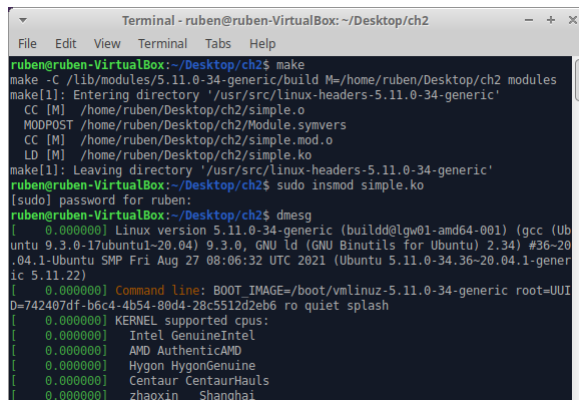
```

2 Running a LKM

As a first step in this assignment, we suggest trying to get a simple module to run. A simple module such as the sample provided by the textbook! The textbook (Operating System Concepts 9e) provides additional information on creating and running LKMs on page 96. Reading that section of the book is strongly recommended. Next, we give a short summary of the basic instructions to run an LKM. Note that we need to use a so-called "makefile" to manage the compilation of modules. **Do not try to use an IDE such as CLion or NetBeans.** Four basic commands are needed for dealing with LKMs: 1) "make" to create the binary files, 2) insmod to insert the LKM into the running kernel, 3) dmesg to view the kernel log file, and 4) rmmod to remove a loaded LKM. See below:

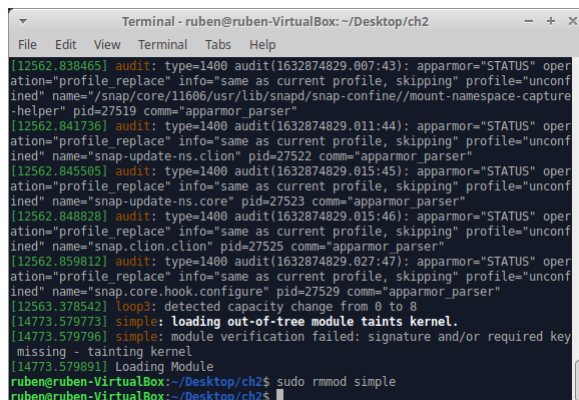
```
make
sudo insmod simple.ko
dmesg
sudo rmmod simple
```

As seen below, running make compiles the LKM into an insertable .ko file. It runs the makefile for the module (see the Makefiles subsection for more information). We then do the insert with insmod which means that it is running. To see the results (printf won't work), we use dmesg to view the log (where the output from commands like printk will go). There will be a lot of information shown when using dmesg...



```
Terminal - ruben@ruben-VirtualBox: ~/Desktop/ch2
ruben@ruben-VirtualBox:~/Desktop/ch2$ make
make -C /lib/modules/5.11.0-34-generic/build M=/home/ruben/Desktop/ch2 modules
make[1]: Entering directory '/usr/src/linux-headers-5.11.0-34-generic'
  CC [M] /home/ruben/Desktop/ch2/simple.o
  MODPOST /home/ruben/Desktop/ch2/Module.symvers
  CC [M] /home/ruben/Desktop/ch2/simple.mod.o
  LD [M] /home/ruben/Desktop/ch2/simple.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.11.0-34-generic'
ruben@ruben-VirtualBox:~/Desktop/ch2$ sudo insmod simple.ko
[sudo] password for ruben:
ruben@ruben-VirtualBox:~/Desktop/ch2$ dmesg
[ 0.000000] Linux version 5.11.0-34-generic (build@lgw01-amd64-001) (gcc (Ubuntu 9.3.0-17ubuntu1-20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #36-20.04.1-Ubuntu SMP Fri Aug 27 08:06:32 UTC 2021 (Ubuntu 5.11.0-34.36-20.04.1-generic 5.11.22)
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.11.0-34-generic root=UUID=742497df-b6c4-4b54-8bd4-28c5512d2eb6 ro quiet splash
[ 0.000000] KERNEL supported cpus:
[ 0.000000] Intel GenuineIntel
[ 0.000000] AMD AuthenticAMD
[ 0.000000] Hygon HygonGenuine
[ 0.000000] Centaur CentaurHauls
[ 0.000000] zhaoxin Shanghai
```

So we typically need to scroll down to find the output from our module. In the screen shot below, the last line ("Loading Module") is the output from the LKM we compiled and inserted. To remove the LKM, we finally issue the rmmod command (we could also reboot).



```
Terminal - ruben@ruben-VirtualBox: ~/Desktop/ch2
[12562.838465] audit: type=1400 audit(1632874829.007:43): apparmor="STATUS" operation="profile replace" info="same as current profile, skipping" profile="unconfined" name="/snap/core/11606/usr/lib/snapd/snap-confine/mount-namespace-capture-helper" pid=27519 comm="apparmor_parser"
[12562.841736] audit: type=1400 audit(1632874829.011:44): apparmor="STATUS" operation="profile replace" info="same as current profile, skipping" profile="unconfined" name="snap-update-ns.clion" pid=27522 comm="apparmor_parser"
[12562.845505] audit: type=1400 audit(1632874829.015:45): apparmor="STATUS" operation="profile replace" info="same as current profile, skipping" profile="unconfined" name="snap-update-ns.core" pid=27523 comm="apparmor_parser"
[12562.848828] audit: type=1400 audit(1632874829.015:46): apparmor="STATUS" operation="profile replace" info="same as current profile, skipping" profile="unconfined" name="snap.clion.clion" pid=27525 comm="apparmor_parser"
[12562.859812] audit: type=1400 audit(1632874829.027:47): apparmor="STATUS" operation="profile replace" info="same as current profile, skipping" profile="unconfined" name="snap.core.hook.configure" pid=27529 comm="apparmor_parser"
[12563.378542] loop3: detected capacity change from 0 to 8
[14773.579773] simple: loading out-of-tree module taints kernel.
[14773.579796] simple: module verification failed: signature and/or required key missing - tainting kernel
[14773.579891] Loading Module
ruben@ruben-VirtualBox:~/Desktop/ch2$ sudo rmmod simple
ruben@ruben-VirtualBox:~/Desktop/ch2$
```

2.1 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 that are compiled.

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 topmost line is special, it says “obj-m += simple.o”. This says take the result of compiling the file simple.c (which will be called simple.o), and combine it with the default binaries for creating a module. This produces the .ko file that represents a built LKM that you can load into the operating system.

Algorithm 1 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
```

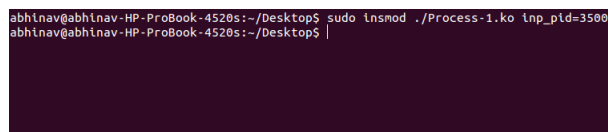
WARNING: do not use this makefile within a path that includes a space (“ ”) in its name, or the module will not build! For example, don’t create a folder called “HW M5” on your desktop and develop within it. The space between “HW” and “M5” will cause the build to fail.

3 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$ sudo insmod ./Process-1.ko lnp_pid=3508
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. **Your programs must compile and run under Xubuntu (or another variant of Ubuntu) 22.04.**

[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	PRI0	ST_PRI0	NORM_PRI0
[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	PRI0	ST_PRI0	NORM_PRI0
[21605.443063]	kworker/u16:2	14980	1	120	120	120
[21605.443063]	PARENT					
[21605.443064]	kthreadd	2	1	120	120	120
[21605.443065]						
[21605.443065]	PROCESS	PID	STATE	PRI0	ST_PRI0	NORM_PRI0
[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	PRI0	ST_PRI0	NORM_PRI0
[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	PRI0	ST_PRI0	NORM_PRI0
[21605.443078]	sudo	16711	1	120	120	120
[21605.443079]	CHILD					
[21605.443080]	sudo	16718	1	120	120	120
[21605.443080]	PARENT					

3.1 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. As you will experience, the information about the kernel is somewhat fragmented, which your write up will pull together and complete. Include the following:

- The version number of your kernel you used to find the line numbers (typically the version you found on the kernel website). [0 points but needed to grade others]
- Main source file: list each function that you defined and describe its purpose. Typically this would be functions for init and exit, plus any helpers you created.
- Task_struct: at a high level, document each of the struct attributes that you use in your program (e.g., pid, state, prio, etc).
- For each header file (except linux/module.h), document each function/macro/struct you use:
 - Functions: Include the function’s signature, what file/line it is defined, and a description (include parameters and return values). Example (self-contained, well styled, and readable):

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 substituted 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., minimal spelling or grammar errors).

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 649).
- *linux/sched/signal.h*: Defines macros for manipulating processes and threads.
 - Useful macros:
 - * *for_each_process* - A macro to loop over each currently running process (see line 601).
- *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* - purpose and line number left for you to discover.
 - Useful functions:
 - * *list_for_each* - purpose and line number left for you to discover.
 - * *list_entry* - purpose and line number left for you to discover.

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. The file(s) should be attached to the homework submission link on Canvas.

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 the makefile (e.g., "makefile") you use to build it. Be sure that your makefile uses the filename for your .c file. Do NOT include your write up or any binaries.