Assignment #2: Encrypted “Pipe” Pseudo-Device Driver

CS 416
Professor – Ricardo Bianchini
TA – William Katsak
Group sizes: exactly 3 people per group
Due date is Nov. 3 at 11:50 PM. - Late submissions will not be accepted.
Submission: Electronic, via Sakai.

1. Overview

This assignment will give you an opportunity to gain some experience writing operating system level code. You will be required to construct a loadable kernel module (LKM) that can be dynamically loaded into a running Linux kernel. LKM code runs inside the kernel address space (i.e., not in user mode), and can interact with the running kernel in just about any way. Key OS concepts such as execution context, synchronization, and low-level memory management come into play in the construction of LKMs.

The most common usage of LKMs is in the construction of device drivers. There are three main types of device drivers in Linux: character drivers, block drivers, and network drivers. Character and block drivers are accessed via special files, known as “device files”, that live under /dev in the Linux file system, while network drivers are accessed via special APIs. For the purpose of this project, we are most interested in how character and block devices work.

Character device files look just like regular files, and can be “read from” and “written to” using the standard open/close/read/write system calls at the granularity of an individual byte. These types of drivers are used for most devices, including keyboards, mice, webcams, etc.

Block device files also appear under /dev, but these files are different in that they must be read/written at the “block level”, typically 512 bytes at a time. This type of driver is used for mass storage devices like disks, where this block-addressable scheme matches closely with how an actual disk operates.

2. Assignment

Your assignment will be to complete the implementation of a character pseudo-device driver for the Linux kernel. A pseudo-device driver is a driver that does not actually interact with hardware, but instead provides some service to userspace following the same interface as a standard device driver.

Your pseudo device implementation will provide an “encryption” service to userspace. Any bytes written into an encryption device file can be read back from the device in an encrypted form. Likewise any (encrypted) data written into a decryption file can be read back from the device in decrypted form. Your LKM will also provide a “control” driver that coordinates the creation and destruction of these individual encryption/decryption pseudo-devices.

3. Requirements

You must complete the following requirements:

1) Present a "control device" file called /dev/cryptctl.

2) The implementation of /dev/cryptctl must provide a set of ioctl() calls to support create and destroy functions.

3) When the create ioctl() is called on /dev/cryptctl, a new encrypt/decrypt file pair must appear in /dev.

   For example:
   /dev/cryptEncrypt0
   /dev/cryptDecrypt0

   The create ioctl() must return the ID of the newly created pair, in this case, 0.

4) These devices must behave as follows:
   a. Encryption must use the Vigenère cipher.
   b. You may assume that input and keys are only ASCII text.
   c. To encrypt, text is written into the /dev/cryptEncryptXX device; the encrypted text is obtained by reading back from the device.
   d. To decrypt, the process is repeated, but on the /dev/cryptDecryptXX device.

5) You must also implement additional ioctls() on /dev/cryptctl that allow you to configure each crypt device pair.

   For example, at a minimum, you will need to configure the encryption "key" for each device.

6) Implement a command line utility that uses the ioctl() interface of /dev/cryptctl to create/destroy encryption devices, and configure these devices after they are created.

In addition, members of the honors section must complete the following requirement. (The other groups can do this for 20% extra credit, but ONLY if all other requirements have been met.)

- You must implement an alternate interface to configure your devices via the Linux /sys filesystem. You do not need to implement create/destroy via /sys, only whatever configuration parameters that you require (e.g. setting the key).

4. Platform

You will be provided with access to a virtual machine (VM) with an already configured installation of Linux. It is highly recommended that you do all development and experimentation within this VM, due to the high probability that you will crash the kernel at some point.

It is REQUIRED that your final, submitted module work inside the VM to receive credit.

You will soon receive separate instructions on how to access your group’s VM.

5. Grading

The project will be graded as follows:

Minimally working LKM (cryptctl device)	10 points
Dynamic creation/destruction of encrypt and decrypt devices	30 points
Encrypt functionality	25 points
Decrypt functionality	25 points
Report	10 points
Total	**100 points**

6. Report

Write a report relating the experiences you had, what the key issues that you encountered were, and how you solved each one. The report will be a non-trivial part of your grade for the assignment, so please prepare it carefully.

At a minimum, the report must include a description of your implementation, including any attention paid to synchronization and memory management, as well as the overall control flow within the driver.

NOTE: the length of the report won't help your grade, but its content will. Be concise, write only what you need to convey.

YOUR REPORT MUST INCLUDE THE NAMES OF ALL MEMBERS OF YOUR GROUP

7. Miscellaneous Useful Links

- O'Reilly – Linux Device Drivers, 3rd Edition
  - http://lwn.net/Kernel/LDD3
- LXR – The Linux Cross Reference
  - http://lxr.linux.no
- Wikipedia: Vigenère_cipher
  - http://en.wikipedia.org/wiki/Vigenère_cipher