Assignment 3 - Write-Up

Tyler Jones, Claude Maimon

November 15th 2017

1 The design you plan to use to implement the LOOK algorithms

In order to implement an encrypted block device, we will use an existing implementation for the block device and manipulate it to match our requirements. Our block device will encrypt and decrypt the data before writing it to the disk and after reading it from the disk. We will use the simple block cipher API for the encryption.

2 Version control log

Date	Name	Commit	Message
11/07	Tyler	9a83b6abaebab4d6441c392784ef42c47b524a43	First attempt at block device driver
11/11	Tyler	a9f5919195a0899f9f5d59e9a061db0361d97e2e	adding makefile and Kconfig, and working version of RAM disk device driver
11/11	Tyler	21e558fbb963e2bd6e75c413b2cc87b44a5a7a90	adding mounting instructions
11/11	Tyler	4a5b251f8fe67e480ec315cafb0984fc2b4685ca	Update readme.txt
11/12	Claude	b9c18115b5cc3f7631057dbe8b02099ac9e95d2f	Added encryption to block. Made a coppy of the block file
11/12	Claude	9cb00df11d0c235120e05b41454e43ad2dd3e699	fixed qemu command.
10/15	Claude	760ed0d9feedd589f8cd5ab96677962e5aeffe73	Fixed bugs in tjmc_a3_crypto.c
11/15	Claude	37d95e5af6a8580bd5e6479192d5732871ac9185	final changes to tjmc_a3_crypto.c

3 A work log

This assignment was mostly searching for information about the crypto library and the disk driver. The version log shows the actual code changes made but we spent much more time searching for information. We started by reading the old implementation of the kernel given to us in the assignment instructions. After that, we moved on to searching for a good implementation of a RAM disk device driver that we could base our implementation on. Once we implemented the disk driver, we searched the crypto API for a very long time.Since it is not well documented, it took us a few day to understand how to use it properly.

4 Questions

4.1 What do you think the main point of this assignment is?

We think that the main point of this assignment was to teach us how to use a badly documented API. The crypto API documentation is very unhelpful and examples for using it are hard to understand. There is not man page and you have to read through a lot of information to understand how to use it.

4.2 How did you personally approach the problem? Design decisions, algorithm, etc.

We started by going over the old kernel implementation given to us in the assignment. After that, when we realized we don't need to write our own device driver from scratch we looked for a good implementation of one. We found a simple block driver (http://blog.superpat.com/2010/05/04/a-simple-block-driver-for-linux-kernel-2-6-31/) and we based our disk drive using it. For the encrypting part, we researched the crypto API for a very long time looking at the examples and descriptions. We chose to use the Single Block Cipher API with the AES option. For the implemetation of the encryption, we ise the example code in the Linux Kernel Crypto API (https://kernel.readthedocs.io/en/sphinx-samples/crypto-API.html#single-block-cipher-api)

4.3 How did you ensure your solution was correct? Testing details, for instance.

4.4 What did you learn?

We learned how to use a library when it's documentation is very bad. We learned to look at kernel examples and figure out how to use those examples in our implementation. We also learned how to use other peopel's implementation (the disk drive) and use it for out implementation.

4.5 How should the TA evaluate your work? Provide detailed steps to prove correctness

In order to evalute our work, our TA should do the following:

1) First and foremost, create a file called sstf-iosched.c by typing "cp noop-iosched.c sstf-iosched.c" within the block directory. This will create a copy of noop-iosched.c with the proper name that our patch file will now operate on. Do not change the contents within this file, the patch file will do that. Next, run our provided patch file on a clean version of the linux kernel. This patch file should be run within the block directory, and can be run as "patch ¡ sstf.patch". There will be changes to Kconfig.iosched sstf-iosched.c and Makefile, all within the block directory.

2) Next, our TA should compile a new image of the kernel with "make -j4 all" or an equivalent in order to compile the patched changes.

3) Next, our TA should run the VM with the following command:

qemu-system-i386 -gdb tcp::5531 -S -nographic -kernel linux-yocto-3.19/arch/x86/boot/bzImage -drive file=core-image-lsb-sdk-qemux86.ext4 -enable-kvm -net none -usb -localtime –no-reboot –append "root=/dev/hda rw console=ttyS0 debug"

Note that "if=virtio" and "vda" was deleted and changed to "hda" respectively from the command used in assignment 1.

4) Once in the kernel, the TA should do the following:

echo sstf ¿ /sys/block/hda/queue/scheduler

This will change the i/o scheduler from the CFQ default, to sstf, which is our CLOOK implementation.

5) From here, the TA should generate i/o in order to test that our solution is correct. Our code that we changed contains printk statements which will verify visually that requests are being processeed in a CLOOK fashion. They will be sorted, and various print statements will show for various special cases.

A very rudimentary way that we did this was to simply "echo test_text ¿ test_io_file", and to examine the output.

This output is also stored in /var/log/messages.

Essentially, verifying that the request queue is in order from least to greatest was sufficient evidence for proof of our correct implementation.