Nicholas Jordan, Bryce Holley, Lei Wang

May 19, 2014

CS 411 Project 3

Our Solution

To implement the custom RAM disk device driver, we first looked up several example files. We used an example file called sbull.c, found at `http://m.blog.csdn.net/blog/zplove003/7020384`. This file contains all necessary functions to implement a basic memory device, including initialization, data structures, read/write transfer requests, and more. In our code, we define a device at first and implement all the block_device_operations. Then we set three requests. In the request, we call the osurd_transfer function to do the actual read and write. The next step was to write setup_device which includes the gendisk structure for generating a new disk. The last thing is write the init and exit function to register it to kernel module.

It also includes a modern geometry detecting function getgeo() to determine the data organization in the drive, which include its size and number of cylinders, heads, and sectors. This allows cfdisk command to detect the information in the drive and allow proper partitioning. All relevant variable names and functions in this sample were renamed to osurd to differentiate it.

We also used an example file called crypto_aes.c , found at `https://github.com/JonathanSalwan/stuffz/blob/master/lkm_samples/crypto_aes.c` This file contains an easy example for crypto-aes algorithms, it implements the basic function for alloc, setkey, free, encrypt and decrypt cipher. These functions also made direct use of the u8 type, which dev->data and buffer already are. This was much more convenient than converting to scatterlists. We looked at where the write and read transfer requests were taking place in the driver file, and inserted several printk() statements to indicate when the write/read took place, as well as for printing out raw data to verify encryption and decryption.

For writing, instead of simply using memcpy() to move the buffer data to dev->buffer, we used the crypto_cipher_encrypt_one() function to first encrypt the buffer data according to our defined cipher, to then be sent to dev->data. For reading, we used the corresponding decryption function crypto_cipher_decrypt_one() to decrypt the dev->data and move it to the buffer. We nested each of these crypto functions inside a for loop, iterating by the given cipher block size, to pass encrypted or decrypted blocks.

Testing

We wrote a simple c program named rwtest.c which opens the new device, and writes and reads several bytes to and from the device, and must be run as root. After compiling a running this test, printing out kernel messages with dmesg will show the read/write activity that has taken place. This is best visible when outputted to a file.