Po-Han Chen ID: 923446482
Github: eric915c CSC415 Operating Systems

Assignment 6 – Coding A Device Driver

Description:
This project aims to develop a device driver for Linux that enables users to encrypt messages using a Caesar cipher algorithm. The driver will be accompanied by a user-space program that facilitates interactions with the device. Device drivers are important for hardware communication, and this assignment will involve designing a driver with an application for loading and unloading. The driver will provide write and read encrypted message functionalities, making it a valuable addition to any security-conscious user's toolkit.

Approach:
1. I implemented a character device driver for Linux using the kernel's Device Model. The driver registers itself as a character device and provides file operations for opening, reading, writing, releasing, and handling ioctl commands.
2. I implemented a Caesar cipher encryption algorithm within the driver. This algorithm shifts characters in the message by a fixed amount to perform encryption.
3. I created a user-space program that interacts with the driver. This program opens the device file, writes a message to be encrypted, calls ioctl to encrypt the message, reads the encrypted message, and then manually decrypts it.
4. I tested the driver and user-space program by loading the driver into the kernel, interacting with the device, and verifying that messages were encrypted and decrypted correctly.
5. I provided comments throughout the code to explain its functionality and usage. Additionally, I wrote a brief description of the project to provide context for the code.

Issues and Resolutions:

Issue: My first issue was encrypted message Same as decrypted message. After encrypting a message, the decrypted message was the same as the original message, indicating that encryption and decryption were not functioning correctly.

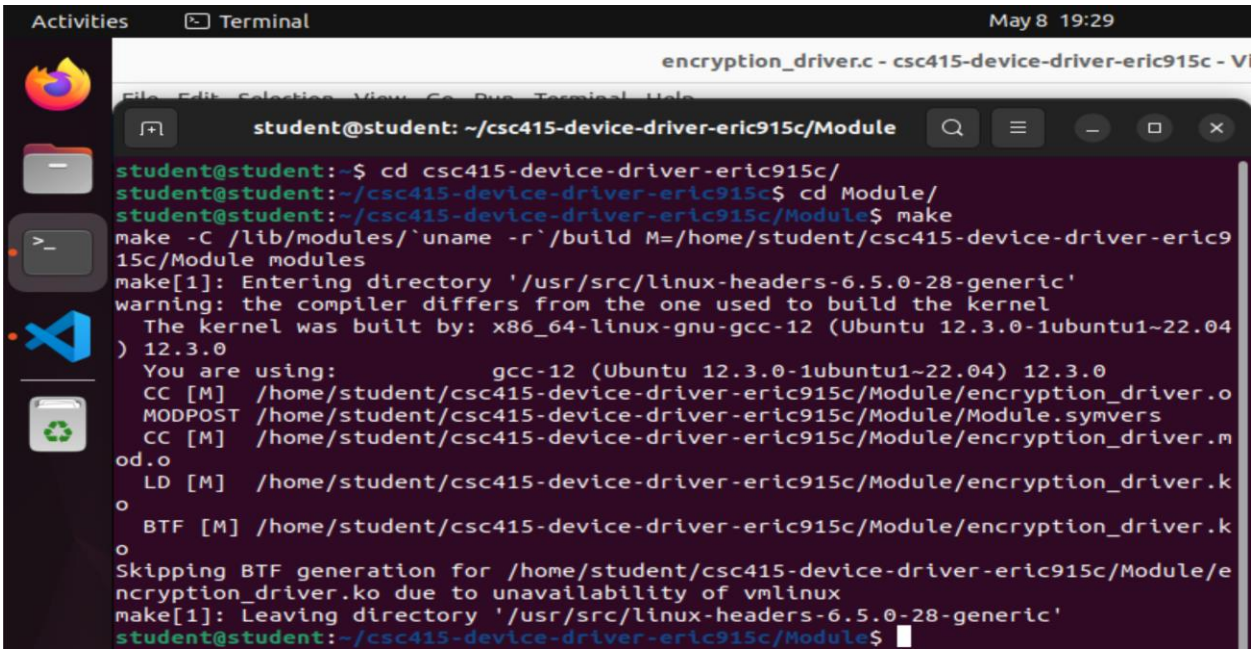
Resolution: I identified that the decryption algorithm was incorrectly implemented. Instead of using a fixed shift value of 23 for decryption, I modified the algorithm to use the opposite shift value of the encryption (in this case, 3) to correctly decrypt the message. After making this change, the encryption and decryption processes produced the expected results.

```c
//handle ioctl commands function
static long deviceIoctl(struct file *file, unsigned int ioctl_num, unsigned long ioctl_param) {
    int shift = 3; //caesar cipher shift key

    switch (ioctl_num) {
        case 0: //Encrypt
            caesarEncr(message, shift); //call encryption function
            break;
        case 1: //Decrypt
            caesarDecr(message, shift); //call decryption function
            break;
        default:
            printk(KERN_ALERT "Encryption driver: Invalid ioctl command\n");
            return -EINVAL; //return error for invalid command
    }

    printk(KERN_INFO "Encryption driver: Message processed\n"); // Log message processed
    return 0; //return success
}
```

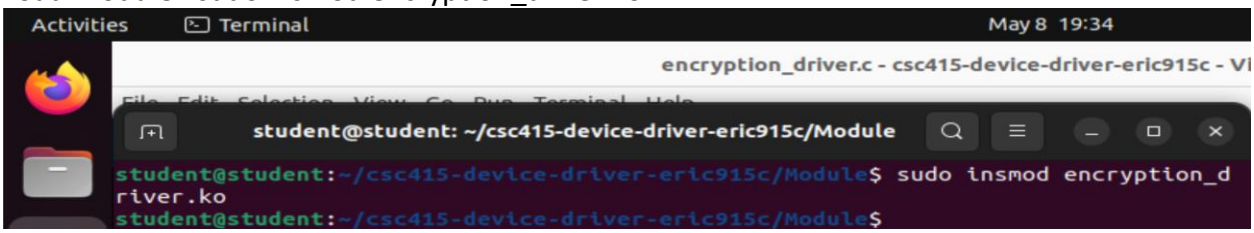
Building the Kernel Module:

Make clean

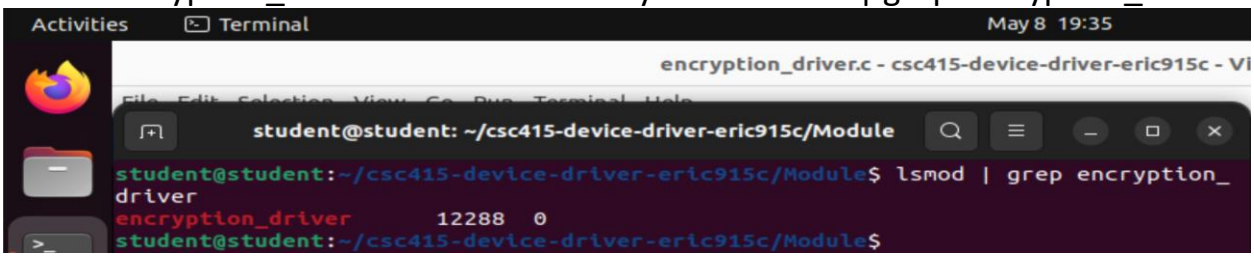
Run: make



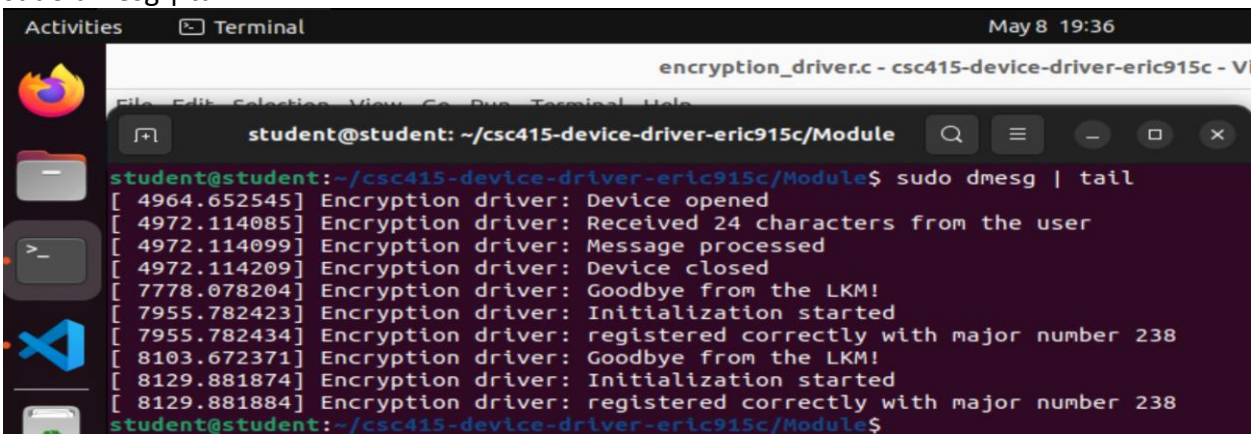
Load Module : sudo insmod encryption_driver.ko



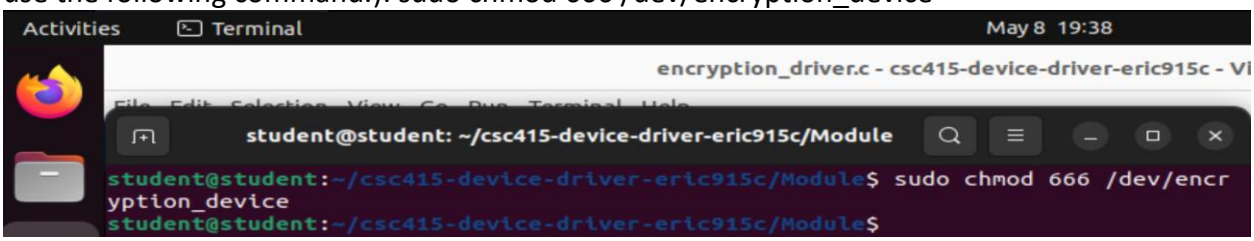
Check encryption_driver is loaded in the system : lsmod | grep encryption_driver



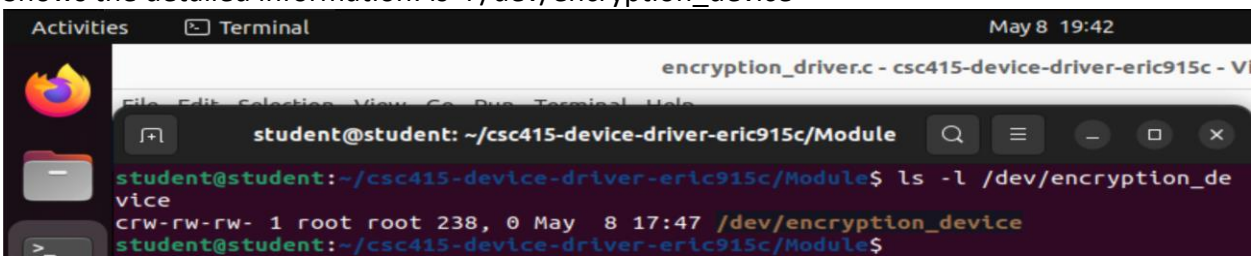
sudo dmesg | tail



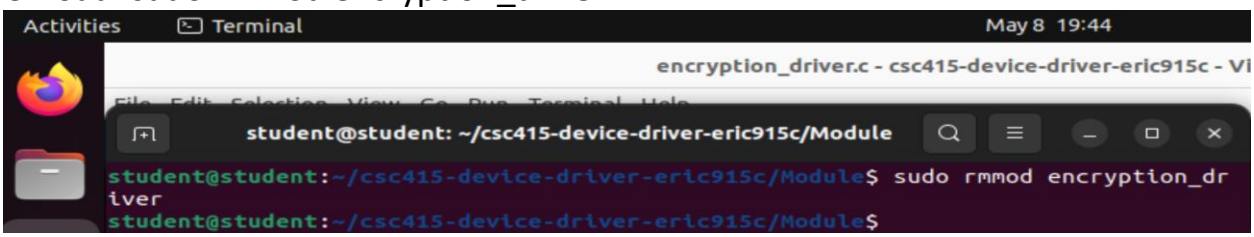
(To set permissions for the /dev/encryption_device device, allowing all users to read and write, use the following command.): sudo chmod 666 /dev/encryption_device



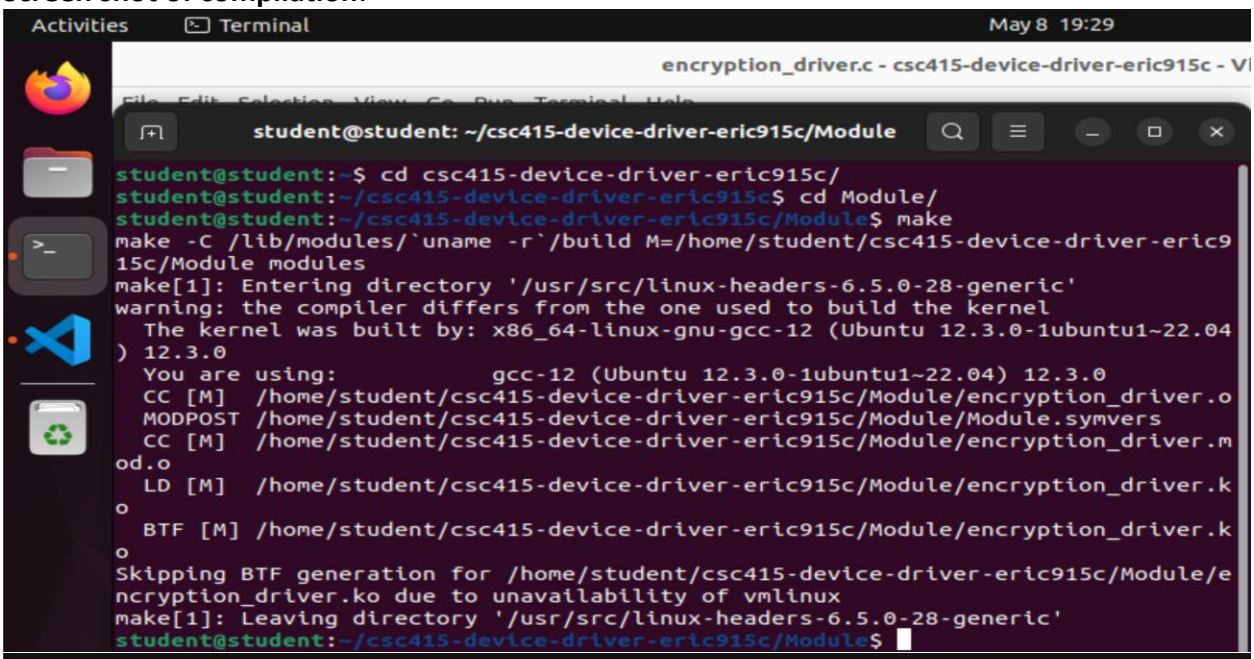
Shows the detailed information: ls -l /dev/encryption_device



Unload: sudo rmmod encryption_driver



Screen shot of compilation:



Screen shot(s) of the execution of the program: