Mos Kullathon
Github: mosguinz

ID: 921425216
CSC415 Operating Systems

Assignment 6 – Device Driver

Description:

This assignment is to write a device driver in C that can be loaded and run in Linux. Then add some functionality to the device driver such as the user/application passing in a string to the device driver and the device driver returns an encrypted version of the string or passes in the encrypted string and returns the original string.

Approach:

Literally follow the lecture. Most of what I did was replicating what was done in the lecture and then — piece-by-piece — reverse engineer it to fit my purpose.

My test application is simply a command-line tool that allows the user to encrypt and decrypt text. Accepts two arguments: the mode either `e` or `d` to encrypt or decrypt, respectively, and the text. For example: `./main e abcdef` would encrypt the text `abcde` and `./main d fghijk` would decrypt `fghijk`. The output is simply the encrypted/decrypted text.

The first thing I need to figure out is how to copy data to and from the device. From the example in class, the driver utilized a buffer to store text and contained a struct that would hold other data. For my use case, I need to store the string to encrypt/decrypt and whether or not it needs to be encrypted or decrypted accordingly. So, I did away with the buffer and simply utilized the single struct instead.

Next was to decide how to encrypt the text. For this assignment, I simply chose a simple Cesar cipher which involves shifting letters by *n* characters. In C, of course, we would be shifting them by their ASCII value. To encrypt, I simply shift the values of each character by five. Then, to decrypt was to shift it back by five.

To communicate with the driver whether to encrypt or decrypt the written text, I am using my `ioctl` function to handle the command. The command to do so would be the characters `e` and `d`, respectively — which can be easily treated as an integer by their ASCII value. In `ioctl`, it can invoke the appropriate function and denote in our struct whether the string is encrypted.

To test if it worked, I simply encrypt something simple such as `abcdef` and see if it shifts everything by a certain amount. I've set the shift to five in my program, so encrypting said string should yield `fghijk`. Likewise, decrypting it should yield the original string. See the execution screenshots below.

Issues and Resolutions:

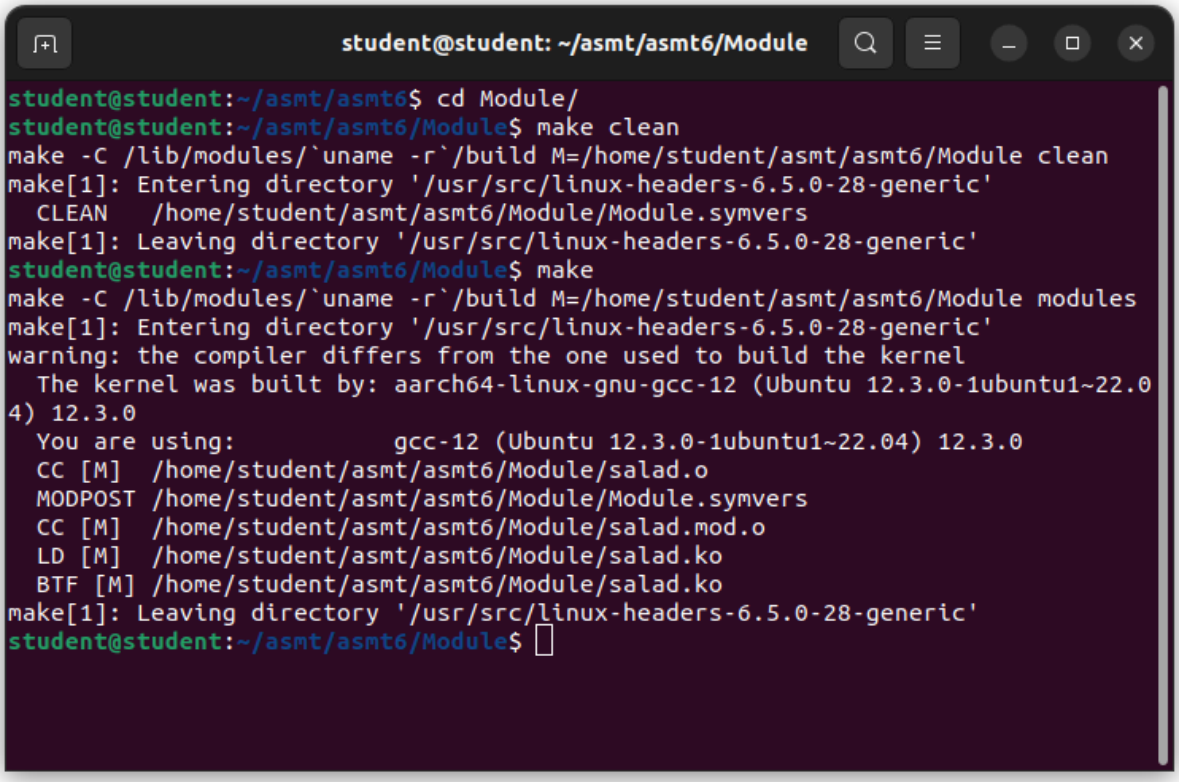
I did not really run into many issues in this assignment, since most of the implementation was already provided in class. I did run into a couple of segfault errors in the driver class due to not accounting for the offset when using `copy_to_user` and `copy_from_user`, but this was easily fixed. Another problem I ran into was opening the driver in read only mode (which was demoed in class). When reading data back from the driver, it would return nothing. The fix was simply to ensure that the driver was open with read-write mode.

Analysis:

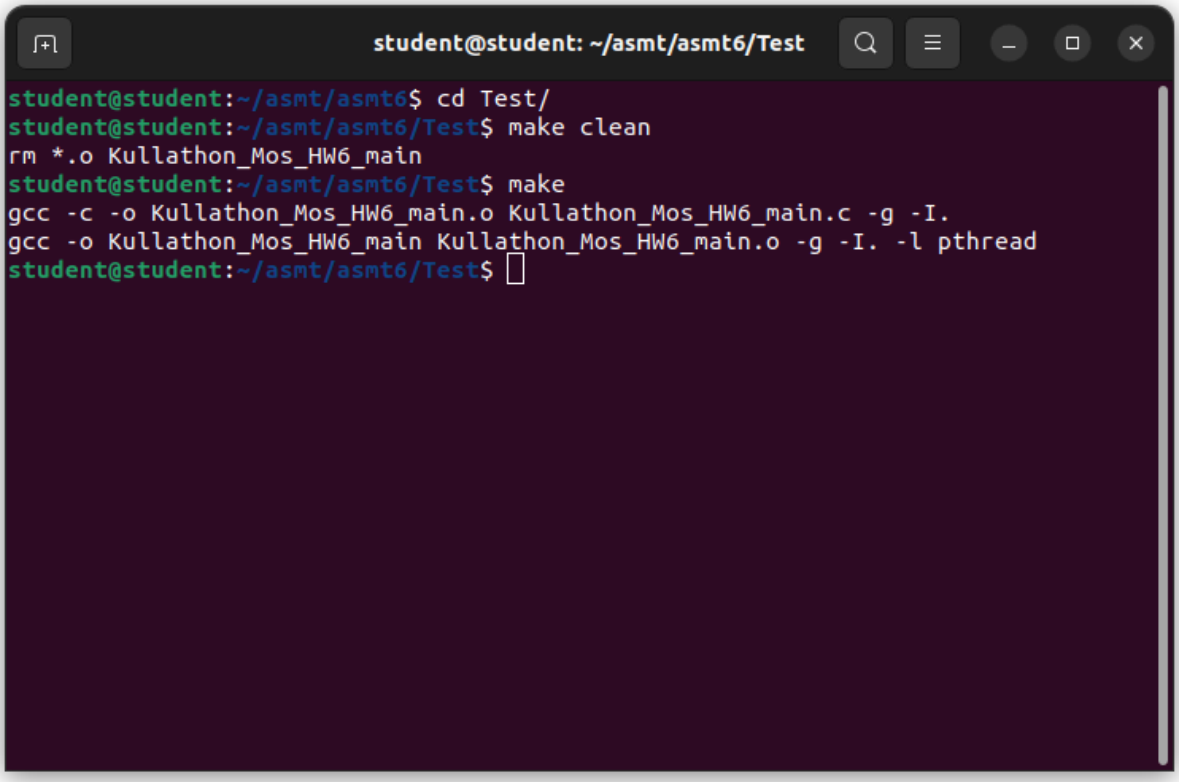
N/A

Screen shot of compilation:

Device driver

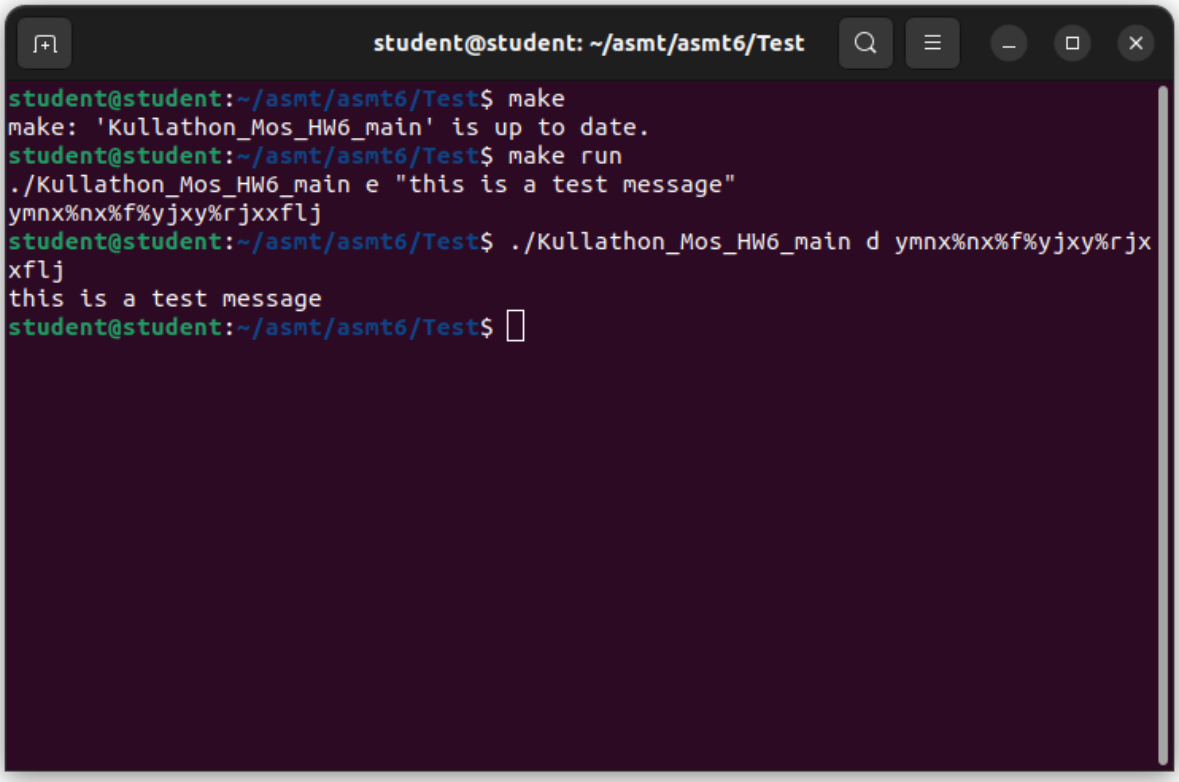


Test file

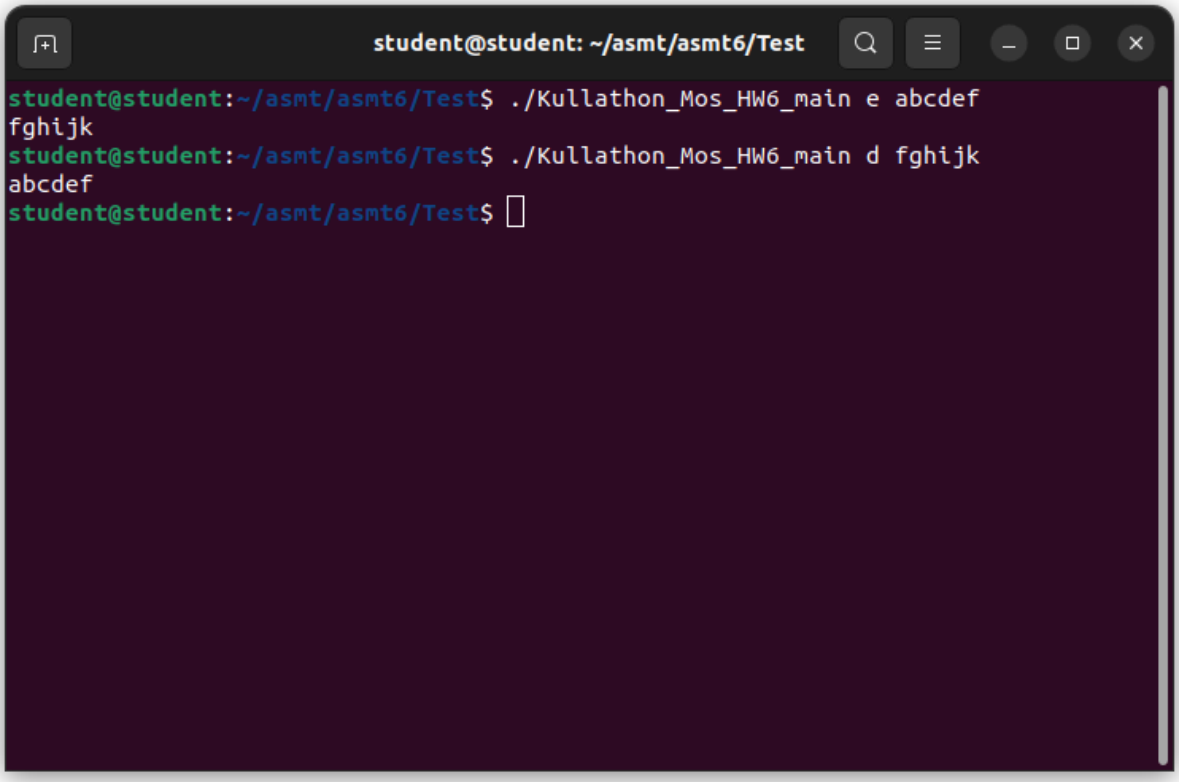


Screen shot(s) of the execution of the program:

The test file is intended to function as a command-line tool to encrypt/decrypt text. In this particular program, the shift is set to five. The default run option when using `make run` is set to encrypt the string `this is a test message`. Below is an example using the program to decrypt the encrypted string.



For a clearer example, here we are encrypting the string `abcdef`. With a shift of five, the encrypted string will be `fghijk`. Again, decrypting it yields the original string.


student@student: ~/asmt/asmt6/Test
student@student:~/asmt/asmt6/Test$ ./Kullathon_Mos_HW6_main e abcdef
fghijk
student@student:~/asmt/asmt6/Test$ ./Kullathon_Mos_HW6_main d fghijk
abcdef
student@student:~/asmt/asmt6/Test$