

ASSIGNMENT 6 - Device Driver

Description:

The encryptor.c file is an example of a simple Linux kernel module that implements a character device driver named "encryptor." This device driver allows user programs to interact with it using standard file operations such as open, close, read, write, and also includes custom IOCTL (input/output control) commands for managing a Caesar cipher-based encryption/decryption scheme.

Approach:

I approached this project by first researching a device driver and understanding its usage and how to interact with a driver. After about an hour of research I realized that building a driver in linux uses the open, read, and write commands and that the device driver is a c file that gives new meaning to the open, read and write commands in linux. I also came across several examples and knew that i would need the following functions:

```
ssize_t device_read(struct file *filp, char *buffer, size_t length, loff_t * offset);
ssize_t device_write(struct file *filp, const char *buffer, size_t length, loff_t * offset);
int device_open(struct inode *inode, struct file *file);
int device_release(struct inode *inode, struct file *file);
long device_ioctl(struct file *filep, unsigned int cmd, unsigned long arg);

struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = device_read,
    .write = device_write,
    .open = device_open,
    .release = device_release,
    .unlocked_ioctl = device_ioctl,
};
```

In order to interact with the device driver I would use normal linux read, open , and write methods that will then use the driver commands device_read, device_open, device_write.

Analysis:

Driver (encryptor.c)

Header Inclusions: The code includes various header files required for Linux kernel development, such as `<linux/init.h>`, `<linux/module.h>`, `<linux/kernel.h>`, and others.

Device and IOCTL Definitions: The code defines the device name as "encryptor" and custom IOCTL commands for getting the key, setting a negative key, and resetting the key.

Character Device Initialization: The `encryptor_init` function is the module's initialization routine. It registers the character device driver with the kernel using `register_chrdev`, obtaining a major device number. The file operations structure (`encryptor_fops`) is initialized with functions for opening, releasing, reading, writing, and handling IOCTL commands.

File Operation Functions:

`encryptor_open`: Called when a user program opens the device file. It prints a message indicating the device has been opened.

`encryptor_release`: Called when a user program closes the device file. It prints a message indicating the device has been released.

`encryptor_read`: Reads encrypted data from the device and copies it to the user buffer after decrypting it using the Caesar cipher.

`encryptor_write`: Accepts data from the user, decrypts it using the Caesar cipher, stores it, and prints the received input.

IOCTL Implementation:

`encryptor_ioctl`: Handles custom IOCTL commands. It interprets the command, performs actions like getting or setting the Caesar key, and communicates with user space through the `copy_to_user` and `copy_from_user` functions.

Module Initialization and Exit:

`module_init`: Specifies the function (`encryptor_init`) to be executed when the module is loaded.

`module_exit`: Specifies the function (`encryptor_exit`) to be executed when the module is unloaded.

Kernel Parameter: The code defines a kernel parameter `caesar_key` that allows setting the initial Caesar key when the module is loaded.

Module Information: The module provides information such as its license, author, and description.

Test file (Mcglothen_Christian_HW6_main.c):

Opening the Device File (open):

The program starts by opening the device file using the open system call. It provides the path to the device file /dev/encryptor and requests read and write access (O_RDWR). If the open call is successful, a file descriptor (fd) is returned, which is used to identify the open device file.

Getting the Caesar Key (ioctl with GET_KEY):

The program uses the ioctl system call to retrieve the current Caesar key from the driver. It sends an IOCTL command GET_KEY to the driver.

The value of the Caesar key is returned to the program through the caesar_key variable.

User Interaction and Input:

The program displays a menu for the user to choose between encryption and decryption.

The user inputs their choice (1 for encryption, 2 for decryption) using the scanf function.

Encryption (Option 1):

If the user chooses encryption, the program prompts the user to enter a message to encrypt using scanf.

The program then copies the entered message to the buffer array and encrypts it by copying the characters to the result array after applying the Caesar cipher.

The encrypted message is written to the device driver using the write system call.

The program then reads the encrypted message back from the device using the read system call and displays it.

Decryption (Option 2):

If the user chooses decryption, the program first sets a negative Caesar key by sending an IOCTL command SET_NEGATIVE_KEY to the driver. This prepares the driver to decrypt the message.

The program prompts the user to enter a message to decrypt using scanf.

Similar to encryption, the program copies the entered message to the buffer array and writes it to the device using the write system call.

The program reads the decrypted message back from the device using the read system call and displays it.

Finally, the program resets the Caesar key to its original value by sending an IOCTL command RESET_KEY to the driver.

Closing the Device File (close):

After completing the chosen operation, the program closes the device file using the close system call, releasing the associated resources.

Issues And Resolutions:

Originally, The test file (Mcglothen_Christian_HW6_main.c) was handling the decryption while the driver handled encryption. Also another issue was hard coding the caesar key in both the driver and the test file.

In order to resolve these issues, the first step was allowing the user to set the caesar key when doing an insmod or echo to the encryptor file. This was done by including the below statements:

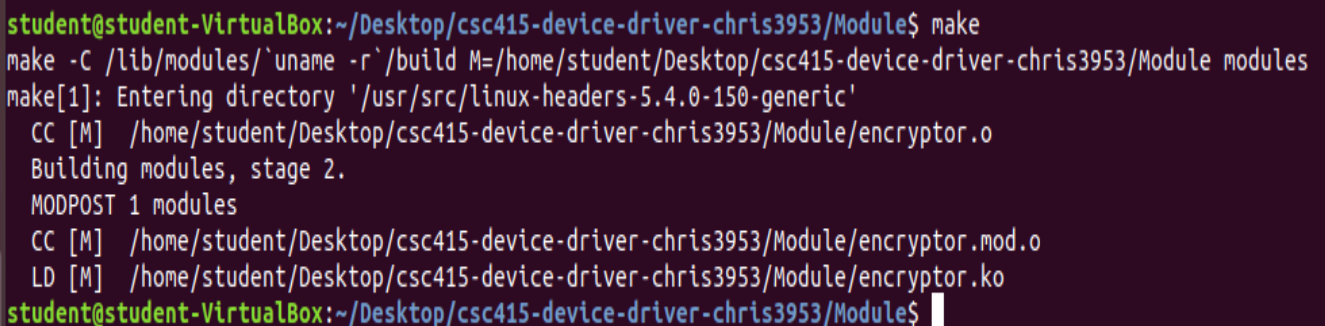
Encryptor.c

```
// Set Caesar key as a kernel parameter
module_param(caesar_key, int, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
MODULE_PARM_DESC(caesar_key, "Caesar key for encryption/decryption");
```

As stated previously the second issue was fixing the driver in order to allow decryption. The encryptor_write command is the function that encrypts the message but only uses the positive version of the caesar key. In order to decrypt a message I needed to flip the sign of the caesar key which then turns the encryptor_write command into a decrypting write. I achieved this by adding to the ioctl method which triggers flipping the sign.

Screenshot Of Compilation:

1. In the Module folder run “make”



```
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$ make
make -C /lib/modules/`uname -r`/build M=/home/student/Desktop/csc415-device-driver-chris3953/Module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-150-generic'
CC [M] /home/student/Desktop/csc415-device-driver-chris3953/Module/encryptor.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/student/Desktop/csc415-device-driver-chris3953/Module/encryptor.mod.o
LD [M] /home/student/Desktop/csc415-device-driver-chris3953/Module/encryptor.ko
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$
```

2. Now do sudo insmod encryptor.ko caesar_key=<number>

```
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$ sudo insmod encryptor.ko caesar_key=5
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$
```

3. Run sudo dmesg to get major number

```
[ 71.165696] rfkill: input handler disabled
[ 1187.104222] Encryptor device driver loaded with major number 240
[ 1187.104224] Caesar key set to: 5
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$
```

4. Now run sudo mknod /dev/encryptor c <majorNumber> 0

```
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$ sudo mknod /dev/encryptor c 240 0
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$
```

5. Now run sudo chmod 666 /dev/encryptor

```
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$ sudo mknod /dev/encryptor c 240 0
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$ ls -l /dev/encryptor
crw-r--r-- 1 root root 240, 0 Aug  5 20:35 /dev/encryptor
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$ sudo chmod 666 /dev/encryptor
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$ ls -l /dev/encryptor
crw-rw-rw- 1 root root 240, 0 Aug  5 20:35 /dev/encryptor
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$
```

6. Cd into the Test folder and do make run

```
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Module$ cd ..
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953$ cd Test
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Test$ make run
gcc -c -o Mcglothen_Christian_HW6_main.o Mcglothen_Christian_HW6_main.c -g -I.
gcc -o Mcglothen_Christian_HW6_main Mcglothen_Christian_HW6_main.o -g -I. -l pthread
./Mcglothen_Christian_HW6_main
You have set the key to 5
Choose an option:
1. Encrypt a message
2. Decrypt a message
Enter your choice:
```

Screenshot Of Output:

(option 1 encryption)

```
student@student-VirtualBox: ~/Desktop/csc415-device-driver-chris3953/Test$ make run
gcc -c -o Mcglothen_Christian_HW6_main.o Mcglothen_Christian_HW6_main.c -g -I.
gcc -o Mcglothen_Christian_HW6_main Mcglothen_Christian_HW6_main.o -g -I. -l pthread
./Mcglothen_Christian_HW6_main
You have set the key to 5
Choose an option:
1. Encrypt a message
2. Decrypt a message
Enter your choice: 1
Enter a message to encrypt: HOpe my grade is good
Encrypted message: MTuj rd lwfij nx ltti
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Test$
```

(Option 2 decryption)

```
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Test$ make run
gcc -c -o Mcglothen_Christian_HW6_main.o Mcglothen_Christian_HW6_main.c -g -I.
gcc -o Mcglothen_Christian_HW6_main Mcglothen_Christian_HW6_main.o -g -I. -l pthread
./Mcglothen_Christian_HW6_main
You have set the key to 5
Choose an option:
1. Encrypt a message
2. Decrypt a message
Enter your choice: 1
Enter a message to encrypt: HOpe my grade is good
Encrypted message: MTuj rd lwfij nx ltti
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Test$ make run
./Mcglothen_Christian_HW6_main
You have set the key to 5
Choose an option:
1. Encrypt a message
2. Decrypt a message
Enter your choice: 2
Enter a message to decrypt: MTuj rd lwfij nx ltti
Decrypted message: HOpe my grade is good
student@student-VirtualBox:~/Desktop/csc415-device-driver-chris3953/Test$
```