Anudeep Katukojwala ID: 922404701
Github: anudeepkatukojwala CSC415 Operating Systems

Assignment 6 – Device Driver

Description:
This program is to build a loadable device driver which can be loaded and unloaded into and from the kernel with some functionalities such as open, release, read, write, and IOCTL. To test the device driver functionality, we are required to write a test program to test every functionality of our device driver.

Commands present in install.sh file:
sudo insmod encryptor.ko
sudo mknod /dev/encryptor c 302 0
sudo chmod 666 /dev/encryptor

Commands present in uninstall.sh file:

sudo rm -rf /dev/encryptor
sudo rmmod encryptor

==========

Steps to Load and Unload my device driver:

1) Run make command in Module folder
2) Run make command in Test folder
3) Run install.sh in the Module folder to load our device driver into the system
   (If the install.sh has any errors (mostly invalid parameters error), it could be due to the fact that the major number I have used in the code is already in use in your system. Please change the major number in encryptor.c file at line number 35, then change the major number value in install.sh file for mknod command. Execute the make clean and then do make command again and repeat step 3)
4) Open new terminal and execute tail -f /var/log/kern.log to track the kernel logs (You can use sudo dmesg -c command as well at the end to check logs)
5) Go to Test folder and do make run
6) Program is intuitive and descriptive and should be easy to use. Use options to perform various functions. You can refer to below steps on how to run the test program.
7) While you run the test program, you can track the kern logs in the other terminal using above given tail command to verify the details
8) Once you are done using the program, please enter 4 to exit
9) Now go to Module folder and execute the uninstall.sh file to remove the device driver from the system.

==========

Steps to use my device driver using my test program:

1) Execute make run in Test folder
2) Program will enter into while loop and will only exit when 4 is entered
3) You will be presented with bunch of options to choose from
4) Since this is the first time select 2 to enter the string (You need to choose 2 first)
5) Once 2 is selected, user is prompted to enter a string.
6) Please enter a string of length 1 to 499 and make sure there are no spaces in entered string (If spaces are used, only the last part of the string will be stored)
7) If you wish to, this is the time to set the encryption key by entering 3, for this session, you cannot set the key later. (encryption key by default is set to value 5)
8) To encrypt the entered string select 0
9) To decrypt the entered string (you can only decrypt after you encrypt the string), please select 1.
10) You can encrypt the string any number of times and decrypt the string same number of times
11) At this time, you can choose to enter a new string and encrypt it and then decrypt it.
12) If you are done with the program, please enter 4 to exit the program.

Approach / What I Did:
I first cloned the repository from GitHub and spent time reading the ReadMe file. After reading through the ReadMe file, I had many questions and few answers. I then went to watch your zoom classes on device drivers and explored the pdf of klife kernel game. This gave a good start and pointed my thoughts in right direction. I then started making the plan on how to approach my program, read through a lot of man pages, which then redirected me to few more resources.

At this point, I first started writing the code about loading and unloading the device driver (with a lot of printk statements).
After confirming that my character device is getting registered and unregistered successfully, I moved on to implementing other functions such as open, release, read, write and IOCTL.

For open function of device driver, I first check if anyone else opened the device file, if yes, I return the function saying the device is busy. If no one opened the device file, we increment a variable value to set the device as opened so that no other user can open our device.
When we release the device, we decrement the variable value we incremented in open to set the device as free to open by other users.
For read and write functions, the key is to figure out how to use copy_to_user and copy_from_user functions. When the user read from device file we created, our device driver read function has to read from the kernel whatever the user has written earlier. We do that by using copy_to_user function, which copies data from kernel space to user space. Same procedure was followed for writing write function for device driver using the copy_from_user.
For every function, I have written printk statements to verify the correctness of our device driver functionality.

Coming to the IOCTL function, I used it to encrypt or decrypt a string that has already been written to our kernel buffer using our write function. The arg argument in my IOCTL function is used to send the encryptionKey for that call, and will be used to encrypt or decrypt the string. The way we encrypt is the string that was written last time using the write function will be used, we modify the same string, character by character adding a specific value called encryptionKey, to get the encrypted version of the key. We do the same for decryption but instead we subtract the each character ascii value by encyptionKey value and get the decrypted string.

The problem here is that if we want to test any device driver function other than loading and unloading the device driver, we need the help of test code.
Now for that reason, I shifted my focus on writing the test program to verify all of device drivers functionality.
In my test program I open the device file with read and write permissions and then enter into a while loop and execute various functions based on the user input value. In this user test program, user can choose to encrypt, decrypt, enter a string, set an encryption key and end the program.
After writing the test program I have used it to verify my device driver functions correctness.

Helpful Resources:

https://tuxthink.blogspot.com/2012/05/working-of-macros-majorminor-and-mkdev.html
https://stackoverflow.com/questions/9835850/what-is-the-difference-between-register_chrdev-region-and-alloc_chrdev-region-to
https://unix.stackexchange.com/questions/4711/what-is-the-difference-between-ioctl-unlocked_ioctl-and-compat_ioctl
https://tldp.org/LDP/lkmpg/2.6/html/x569.html
https://www.kernel.org/doc/htmldocs/kernel-api/API-alloc-chrdev-region.html
https://www.kernel.org/doc/htmldocs/kernel-api/API-unregister-chrdev-region.html
https://www.kernel.org/doc/htmldocs/kernel-hacking/routines-init-again.html
https://embetronicx.com/tutorials/linux/device-drivers/cdev-structure-and-file-operations-of-character-drivers/
https://www.kernel.org/doc/htmldocs/kernel-api/API-cdev-add.html
https://man.linuxreviews.org/man9/MAJOR.9.html
https://www.kernel.org/doc/htmldocs/kernel-api/API-cdev-del.html
https://courses.engr.illinois.edu/cs423/sp2021/lecture/MP1_Q-and-A.pdf
https://stackoverflow.com/questions/8107826/proper-way-to-empty-a-c-string
https://man7.org/linux/man-pages/man3/memset.3.html

Issues and Resolutions:

First issue:
Error while registering the character device

Resolution:
It looked like I was using wrong approach to register for a character device. I had to read through few of the online resources and documentation to understand the steps in making the device register into our kernel.
After resolving those issues, my device driver got successfully registered.

======

Second issue:
I was stuck on how to send the user entered encryption key value to the device driver

Resolution:
After testing out multiple failed strategies, I have decided to use the IOCTL long arg argument, to send my encryption key to the device driver, whenever user sets the mode as encryption or decryption in test program IOCTL function.

=======

Third issue:
No characters are being read

Resolution:
I have realized that I was using two different character arrays to manage read and write, and for that reason, whatever is written to device file, could not be seen when we read the device file. After correcting this mistake, everything worked.

============

Fourth issue:
When I write a long string first and then next time I write the shorter string, and then read from the device file, the shorted string just overlapped the long string I have written earlier, and part of the long string is still shown.

Resolution:
After a lot of approaches such as to strcpy “” (which did not work) and other strategies. I have come across memset function which sets a constant value for the given length. I used it to set the entire char array buffer with 0 value. After this, things worked well, and issue was resolved.

Finally, I made everything work, after resolving every issue as described above.

Analysis:
No analysis for this assignment

Screen shot of compilation:

Compilation in Module folder:

```
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Module$ make
make -C /lib/modules/`uname -r`/build M=/home/student/csc415-device-driver-anudeepkatukojwala/Module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-135-generic'
  CC [M]  /home/student/csc415-device-driver-anudeepkatukojwala/Module/encryptor.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/student/csc415-device-driver-anudeepkatukojwala/Module/encryptor.mod.o
  LD [M]  /home/student/csc415-device-driver-anudeepkatukojwala/Module/encryptor.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-135-generic'
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Module$
```

Compilation in Test folder:

```
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Test$ make
gcc -c -o katukojwala_anudeep_HW6_main.o katukojwala_anudeep_HW6_main.c -g -I.
gcc -o katukojwala_anudeep_HW6_main katukojwala_anudeep_HW6_main.o -g -I. -l pthread
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Test$
```

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

Screenshot of loading device driver:

```
student@student-VirtualBox:~$ sudo dmesg -c
[12705.754379] Character Device registered with Major number: 195
student@student-VirtualBox:~$
```

Screenshot of unloading device driver:

```
student@student-VirtualBox:~$ sudo dmesg -c
[12754.308113] Removed encryptor from the system
student@student-VirtualBox:~$
```

Execution of Test Program:

```
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Test$ make run
./katukojwala_anudeep_HW6_main

*************
If this is the first time, please select 2 to enter a string

Please enter the mode of operation desired:
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
```

```
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Test$ make run
./katukojwala_anudeep_HW6_main

*************
If this is the first time, please select 2 to enter a string

Please enter the mode of operation desired:
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
2

Enter new string for encryption
anudeep

Original String before encryption: anudeep

String written to device file is: anudeep

*************
If this is the first time, please select 2 to enter a string

Please enter the mode of operation desired:
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
```

```
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
0

Read from device file; String after 1 level encryption: fszijju

*************
If this is the first time, please select 2 to enter a string

Please enter the mode of operation desired:
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
1

Read from device file; String after decryption: anudeep, can be decrypted 0 more times

*************
If this is the first time, please select 2 to enter a string

Please enter the mode of operation desired:
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
If this is the first time, please select 2 to enter a string

Please enter the mode of operation desired:
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
3

Enter any value between 5 and 20 for setting encryption key
6

*************
If this is the first time, please select 2 to enter a string

Please enter the mode of operation desired:
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
```

```
If this is the first time, please select 2 to enter a string

Please enter the mode of operation desired:
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
3

Enter any value between 5 and 20 for setting encryption key
6

*************
If this is the first time, please select 2 to enter a string

Please enter the mode of operation desired:
0 to Encrypt the string
1 to Decrypt the string
2 to enter new string
3 to set encryption key
4 to end the program

*************
4
You chose to end the program
Exiting...
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Test$
```

Screenshots of kernel logs with sudo dmesg -c command after Test program execution:

```
student@student-VirtualBox:~$ sudo dmesg -c
[10130.377797] Encryptor character device opened successfully
[10134.077920] Successfully written to kernel: anudeep
[10134.077921] Characters written to kernel are: 7
[10135.398570] String after encryption: fszijju
[10135.398574] Total number of characters read to user space: 7
[10136.133612] String after encryption: kxnooz
[10136.133616] Total number of characters read to user space: 7
[10138.655538] Decrypted String: fszijju
[10138.655543] Total number of characters read to user space: 7
[10139.331949] Decrypted String: anudeep
[10139.331953] Total number of characters read to user space: 7
[10155.106702] Successfully written to kernel: istayinsanfrancisco
[10155.106704] Characters written to kernel are: 19
[10164.290465] String after encryption: r|}j\x82rw|jwo{jwlr|lx
[10164.290470] Total number of characters read to user space: 19
[10165.881153] Decrypted String: istayinsanfrancisco
[10165.881157] Total number of characters read to user space: 19
[10170.187547] Successfully written to kernel: abc
[10170.187548] Characters written to kernel are: 3
[10172.666461] String after encryption: jkl
[10172.666465] Total number of characters read to user space: 3
[10173.466194] Decrypted String: abc
[10173.466199] Total number of characters read to user space: 3
[10175.761693] Encryptor device driver file is released!
student@student-VirtualBox:~$
```

Execution of uninstall.sh script file:

```
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Module$ ./uninstall.sh
[sudo] password for student:
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Module$
```

Executin of install.sh script file:

```
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Module$ ./install.sh
student@student-VirtualBox:~/csc415-device-driver-anudeepkatukojwala/Module$
```