EE516 : Homework 1

Gaurav Kalra

(Student ID: 2016 45 93)

gvkalra@kaist.ac.kr

Problem 1: Device Driver
Make your device driver with a Stack and its application program to prove the correctness.

FOLDERS
- driver
  - main.c
  - Makefile
  - stack.c
  - stack.h
  - utils.h

```
206   module_init(stack_dev_init);
207   module_exit(stack_dev_exit);
208
209   MODULE_AUTHOR("Gaurav Kalra");
210   MODULE_DESCRIPTION("HW01 device driver providing a stack");
211   MODULE_LICENSE("GPL");
```

```
144  static int __init
145  stack_dev_init(void)
146  {
147    int ret;
148
149    dbg("");
150
1      /* Register char device number */
1      ret = alloc_chrdev_region(&dev_first, CHARDEV_NR_MINOR, CHARDEV_NR_DEVICES, CHARDEV_NAME);
153    if (ret < 0) {
154      err("Failed allocating device number");
155      goto error;
156    }
157
1      /* Allocate cdev structure */
1      cdev = cdev_alloc();
160    if (cdev == NULL) {
161      err("Failed allocating character device");
162      goto error;
163    }
1      cdev->owner = THIS_MODULE;
1      cdev->ops = &fops;
166
167    /* Add char device to the system */
168    ret = cdev_add(cdev, dev_first, CHARDEV_NR_DEVICES);
169    if (ret < 0) {
170      err("Failed adding character device to the system");
171      goto error;
172    }
173
174    info("[STACK_DEVICE] allocated Major(%d) and Minor(%d)", MAJOR(dev_first), MINOR(dev_first));
175
1      /* Create /dev/stack_device node
1       * Ref: https://github.com/euspectre/kedr/blob/master/sources/examples/sample_target/cfake.c
1       */
179    class = class_create(THIS_MODULE, CHARDEV_NAME);
180    if (class == NULL) {
181      err("Failed creating device class");
182      goto error;
183    }
184    device = device_create(class, NULL, dev_first, NULL, "%s", CHARDEV_NAME);
185    if (device == NULL) {
186      err("Failed creating /dev node");
187      goto error;
188    }
189
190    return 0;
191
192  error:
193    /* we can't call stack_dev_exit()
194       because it's segment is controlled by __exit modifier
195    */
196    _stack_dev_exit();
197    return -1;
198  }
```

```
200  static void __exit
201  stack_dev_exit(void)
202  {
20     _stack_dev_exit();
204  }
```

Figure 1: Kernel Module initialization / de-initialization in main.c

Figure 1.1: Kernel Module initialization / de-initialization in main.c

A separate function *_stack_dev_exit()* is reused in *stack_dev_init()* and *stack_dev_exit()* because of different lifetimes provided by *__init* and *__exit* modifiers.

Major & Minor numbers are dynamically allocated by the Kernel using the new way of device allocation.

References:

http://stackoverflow.com/questions/8563978/what-is-kernel-section-mismatch
https://github.com/euspectre/kedr/blob/master/sources/examples/sample_target/cfake.c

```c
static struct file_operations fops = {
  .owner = THIS_MODULE,
  .open = stack_dev_open, /* open() */
  .read = stack_dev_read, /* read() */
  .write = stack_dev_write, /* write() */
  .release = stack_dev_release, /* close() */
/*
 * References:
 * [1] https://github.com/torvalds/linux/blob/master/fs/ext4/file.c#L698
 * [2] https://github.com/torvalds/linux/blob/master/fs/ext4/ioctl.c#L436
 */
  .unlocked_ioctl = stack_dev_ioctl, /* unlocked_ioctl() */
};
```

```c
static int
stack_dev_open(struct inode *inode, struct file *file)
{
  dbg("");
  return 0;
}
```

```c
static int
stack_dev_release(struct inode *inode, struct file *file)
{
  dbg("");
  return 0;
}
```

```c
static long
stack_dev_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
  dbg("");

  st_clean();

  return 0;
}
```

1

```c
static ssize_t
stack_dev_read(struct file *file, char __user *buffer, size_t length, loff_t *offset)
{
  int item, ret;

  dbg("");

  /* EOF, return 0 */
  if (st_is_empty())
    return 0;

  /* pop */
  ret = st_pop(&item);
  if (ret < 0)
    return ret;

  /* copy data into user space */
  if (copy_to_user(buffer, &item, sizeof(item)))
    return -EFAULT; /* Bad address */

  /* popped & copied to user space, return bytes */
  return sizeof(item);
}
```

2 3 4 5

```c
static ssize_t
stack_dev_write(struct file *file, const char __user *buffer, size_t length, loff_t *offset)
{
  int item, ret;

  dbg("");

  /* No more space */
  if (st_is_full())
    return -ENOMEM;

  /* copy data from user space */
  if (copy_from_user(&item, buffer, sizeof(item)))
    return -EFAULT; /* Bad address */

  /* push */
  ret = st_push(item);
  if (ret < 0)
    return ret;

  /* pushed & copied from user space, return bytes */
  return sizeof(item);
}
```

6 7 8 9

Figure 2: VFS operations implemented by Stack device in main.c

```c
#pragma once

/* True (1) if no more items can be popped and there is no top item
*/
int st_is_empty(void);

/* True (1) if no more items can be pushed
*/
int st_is_full(void);

/* Adds an item onto the stack
 * Returns < 0 on error
*/
int st_push(int item);

/* Removes the most-recently-pushed item from the stack (as out argument)
 * Returns < 0 on error
*/
int st_pop(int *item);

/* Cleans up stack
*/
void st_clean(void);
```

Figure 3: APIs exposed by Stack abstraction in stack.h

```c
#define STACK_SIZE 256

struct {
  int stack[STACK_SIZE];
  int top;
} st = {
  .top = -1, /* top is set to -1 to indicate empty stack */
};
```

```c
int st_is_empty(void)
{
  dbg("");

  if (st.top == -1)
    return 1;
  return 0;
}
```

```c
int st_push(int item)
{
  dbg("");

  /* add item */
  st.top++;
  st.stack[st.top] = item;
  return 0;
}
```

```c
int st_pop(int *item)
{
  /* invalid arguments */
  if (item == NULL)
    return -EINVAL;

  dbg("");

  /* remove item */
  *item = st.stack[st.top];
  st.top--;
  return 0;
}
```

```c
void st_clean(void)
{
  dbg("");

  /* setting top to -1 means stack is cleaned up
   * there is no way to access st
  */
  st.top = -1;
}
```

```c
int st_is_full(void)
{
  dbg("");

  if (st.top >= STACK_SIZE - 1)
    return 1;
  return 0;
}
```

Figure 4: Implementation of Stack abstraction in stack.c

```
1  MODNAME := stack_device
2  obj-m := ${MODNAME}.o
3  ${MODNAME}-objs := main.o stack.o
4
5  KDIR := /lib/modules/$(shell uname -r)/build
6
7  all:
8    $(MAKE) -C $(KDIR) M=$(PWD) modules
9
10 clean:
11   $(MAKE) -C $(KDIR) M=$(PWD) clean
```

Figure 5: Makefile for Kernel Module

```
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01/driver (master) $ make
make -C /lib/modules/4.4.0-38-generic/build M=/home/gvkalra/Desktop/EE516/HW01/driver modules
make[1]: Entering directory '/usr/src/linux-headers-4.4.0-38-generic'
  CC [M]  /home/gvkalra/Desktop/EE516/HW01/driver/main.o
  CC [M]  /home/gvkalra/Desktop/EE516/HW01/driver/stack.o
  LD [M]  /home/gvkalra/Desktop/EE516/HW01/driver/stack_device.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/gvkalra/Desktop/EE516/HW01/driver/stack_device.mod.o
  LD [M]  /home/gvkalra/Desktop/EE516/HW01/driver/stack_device.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.0-38-generic'
```

Figure 6: Generating Kernel Module (*stack_device.ko*)

```
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01/driver (master) $ sudo insmod stack_device.ko
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01/driver (master) $ dmesg
[17728.856884] <stack_dev_init:149>
[17728.856889] <stack_dev_init:174> [STACK_DEVICE] allocated Major(244) and Minor(0)
```

Figure 7: Inserting Kernel Module using *insmod*

```
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01/driver (master) $ lsmod | grep stack_device
stack_device           16384  0
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01/driver (master) $ cat /proc/devices | grep stack_device
244 stack_device
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01/driver (master) $ ls -al /dev/ | grep stack_device
crw-------   1 root root    244,   0 Sep 22 22:39 stack_device
```

Figure 8: Verifying module insertion using *lsmod*, */proc/devices* and creation of */dev/stack_device* character device. Note that Major(244) and Minor(0) have been assigned by the kernel at runtime.

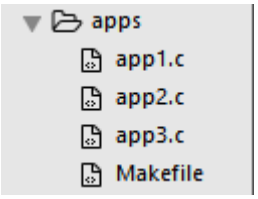
```c
#define STACK_SIZE 256

int main(int argc, const char *argv[])
{
    int item = 0, fd;
    ssize_t bytes_written;

    /* open driver node */
    fd = open("/dev/stack_device", O_WRONLY);
    if (fd < 0) {
        fprintf(stderr, "open() failed err: [%s]\n", strerror(errno));
        return -1;
    }

    /* start pushing numbers to the driver */
    while (item < STACK_SIZE) {
        bytes_written = write(fd, &item, sizeof(item));

        if (bytes_written < 0) /* error */
            fprintf(stderr, "write() failed err: [%s]\n", strerror(errno));
        else if (bytes_written != sizeof(item)) /* partial bytes written */
            fprintf(stderr, "[NOT WRITTEN] %d\n", item);
        else { /* success */
            fprintf(stdout, "[WRITTEN] %d\n", item);
            item++; /* push next item */
        }
    }

    /* cleanup */
    close(fd);
    return 0;
}
```

Figure 9: *app1* writing 256 items (0 – 255) on *stack_device*

```c
int main(int argc, const char *argv[])
{
    int item, fd;
    ssize_t bytes_read;

    /* open driver node */
    fd = open("/dev/stack_device", O_RDONLY);
    if (fd < 0) {
        fprintf(stderr, "open() failed err: [%s]\n", strerror(errno));
        return -1;
    }

    /* start poping numbers from the driver */
    while (1) {
        bytes_read = read(fd, &item, sizeof(item));

        if (bytes_read < 0) /* error */
            fprintf(stderr, "read() failed err: [%s]\n", strerror(errno));
        else if (bytes_read == 0) /* EOF */
            break;
        else if (bytes_read != sizeof(item)) /* incomplete data */
            fprintf(stderr, "[NOT READ] %d\n", item);
        else /* success */
            fprintf(stdout, "[READ] %d\n", item);
    }

    /* cleanup */
    close(fd);
    return 0;
}
```

Figure 10: *app2* reading items until EOF from *stack_device*

```
CC=gcc
CFLAGS=-Wall -Werror
BIN=app1 app2 app3

all: $(BIN)

app1:
	$(CC) -o $@ $@.c $(CFLAGS)

app2:
	$(CC) -o $@ $@.c $(CFLAGS)

app3:
	$(CC) -o $@ $@.c $(CFLAGS)

clean:
	rm -f $(BIN)
```




```
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01 (master) $ make
gcc -o app1 app1.c -Wall -Werror
gcc -o app2 app2.c -Wall -Werror
gcc -o app3 app3.c -Wall -Werror
```

Figure 11: Makefile & folder structure of applications

```
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01 (master) $ sudo ./app1 | head -n 10
[WRITTEN] 0
[WRITTEN] 1
[WRITTEN] 2
[WRITTEN] 3
[WRITTEN] 4
[WRITTEN] 5
[WRITTEN] 6
[WRITTEN] 7
[WRITTEN] 8
[WRITTEN] 9
```

Figure 12: Output of ./app1 (write) stripped to 10 lines

```
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01 (master) $ sudo ./app2 | head -n 10
[READ] 255
[READ] 254
[READ] 253
[READ] 252
[READ] 251
[READ] 250
[READ] 249
[READ] 248
[READ] 247
[READ] 246
```

Figure 13: Output of ./app2 (read) stripped to 10 lines

The read data sequence is opposite of write data sequence because the *read()* & *write()* system calls on */dev/stack_device* are implemented as a stack data structure (first-in last-out, last-in first-out) in our driver.

Problem 2: Clean Function
Make your own “clean function” that makes the stack empty.

```c
int main(int argc, const char *argv[])
{
    int fd, ret;

    /* open driver node */
    fd = open("/dev/stack_device", O_RDONLY);
    if (fd < 0) {
        fprintf(stderr, "open() failed err: [%s]\n", strerror(errno));
        return -1;
    }

    /* call ioctl() on driver */
    ret = ioctl(fd, 0);
    if (ret < 0) {
        fprintf(stderr, "ioctl() failed err: [%s]\n", strerror(errno));
        close(fd);
        return -1;
    }

    /* cleanup */
    close(fd);
    return 0;
}
```

Figure 14: *app3* cleans *stack_device* by invoking ioctl()

The structure *file_operations* provides an extension point for custom commands using *ioctl* (I/O control). I have implemented the operation of “stack clean” in Linux using *unlocked_ioctl()* [***Refer Figure 2***].

It can be invoked by an application using *ioctl()* system call.

Reference:
https://github.com/torvalds/linux/blob/master/include/linux/fs.h#L1679

```
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01 (master) $ sudo ./app1 | head -n 10
[WRITTEN] 0
[WRITTEN] 1
[WRITTEN] 2
[WRITTEN] 3
[WRITTEN] 4
[WRITTEN] 5
[WRITTEN] 6
[WRITTEN] 7
[WRITTEN] 8
[WRITTEN] 9
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01 (master) $ sudo ./app3
gvkalra@gvkalra-desktop ~/Desktop/EE516/HW01 (master) $ sudo ./app2
```

Figure 15: *app1* writes 256 items (0 – 255) on *stack_device*, *app3* cleans *stack_device*, *app2* has no remaining data to read.