```
/*

* This code is to check multiple file_operations structure usage

*

*/

#include "linux/init.h"
#include "linux/device.h"
#include "linux/backing-dev.h"
#include "asm/uaccess.h"

#define HELLO_DEV_MAX 4

#define HELLO1_MINOR 1

#define HELLO2_MINOR 2

#define HELLO3_MINOR 3

dev_t hello_dev;

/* This function will be called when application opens the file, /dev/hello1 */

static int open_hello1(struct inode * inode, struct file * filp)

{

printk(" Hello1 device opened \n");

return 0 ;

}

/* This function will be called when application opens the file, /dev/hello2 */

static int open_hello2(struct inode * inode, struct file * filp)

{

printk(" Hello2 device opened \n");

return 0 ;

}

/* This function will be called when application opens the file, /dev/hello3 */

static int open_hello3(struct inode * inode, struct file * filp)
```

```
{

printk(" Hello3 device opened \n");

return 0 ;

}

/* This function will be called when application reads the file, /dev/hello1 */

static ssize_t read_hello1(struct file * file, char __user * buf, size_t count,
loff_t *ppos)

{

char * tmp = "Hi, Its Me, hello1";

printk("Inside read function of hello1 device\n");

if(copy_to_user(buf, tmp, 23 ))

{

printk("copy_to_user is returned error \n");

return -1;

}

return 0;

}

/* This function will be called when application reads the file, /dev/hello2 */

static ssize_t read_hello2(struct file * file, char __user * buf, size_t count,
loff_t *ppos)

{

char * tmp = "Hi, Its Me, hello2";

printk("Inside read function of hello2 device\n");

if(copy_to_user(buf, tmp, 23 ))

{

printk("copy_to_user is returned error \n");
```

```
return -1;

}

return 0;

}

/* This function will be called when application reads the file, /dev/hello3 */

static ssize_t read_hello3(struct file * file, char __user * buf, size_t count,
loff_t *ppos)

{

char * tmp = "Hi, Its Me, hello3";

printk("Inside read function of hello3 device\n");

if(copy_to_user(buf, tmp, 23 ))

{

printk("copy_to_user is returned error \n");

return -1;

}

return 0;

}

static const struct file_operations hello1_fops = {

.read = read_hello1,

.open = open_hello1,

};

static const struct file_operations hello2_fops = {

.read = read_hello2,

.open = open_hello2,

};
```

```
static const struct file_operations hello3_fops = {
.read = read_hello3,
.open = open_hello3,
};
static const struct {
unsigned int minor;
char *name;
umode_t mode;
const struct file_operations *fops;
} devlist[] = { /* list of minor devices */
{1, "hello1", S_IRUSR | S_IWUSR, &hello1_fops},
{2, "hello2", S_IRUGO | S_IWUSR, &hello2_fops},
{3, "hello3", S_IRUGO | S_IWUSR, &hello3_fops},
};
/*
* This function will assign file_operations to the corresponding minor
* when open function called by application
*
*/
static int hello_open(struct inode * inode, struct file * filp)
{
printk("Opened the device hello \n");
switch(iminor(inode))
{
case HELLO1_MINOR:
```

```
filp->f_op = &hello1_fops;
break;
case HELLO2_MINOR:
filp->f_op = &hello2_fops;
break;
case HELLO3_MINOR:
filp->f_op = &hello3_fops;
break;
default:
return -1;
}
return 0;
}
static const struct file_operations hello_fops = {
.open = hello_open, /* just a selector for the real open */
};
static struct class *hello_class;
static int __init hello_dev_init(void)
{
int i;
int err = 0;
err = alloc_chrdev_region(&hello_dev,1,HELLO_DEV_MAX, "hello");
if (err < 0)
{
printk(KERN_ERR "%s:failed to alloc char dev region\n", __FILE__);
```

```
}

if (register_chrdev(MAJOR(hello_dev),"hello",&hello_fops))

{

printk("unable to get major %d for hello devs\n", MAJOR(hello_dev));

}

else

{

printk(" got major %d for hello devs\n", MAJOR(hello_dev));

}

hello_class = class_create(THIS_MODULE, "hello");

for (i = 0; i < ARRAY_SIZE(devlist); i++)

{

device_create(hello_class, NULL, MKDEV(MAJOR(hello_dev), devlist[i].minor), NULL,
devlist[i].name);

}

return 0;

}

static void hello_dev_exit(void)

{

int i ;

for(i = 0; i < ARRAY_SIZE(devlist); i++)

{

device_destroy(hello_class, MKDEV(MAJOR(hello_dev), devlist[i].minor));

}

class_destroy(hello_class);

unregister_chrdev(MAJOR(hello_dev), "hello");
```

```
if(hello_dev)

{

unregister_chrdev_region(hello_dev, HELLO_DEV_MAX);

}

}

fs_initcall(hello_dev_init);

module_exit(hello_dev_exit);

MODULE_LICENSE("GPL");

MODULE_AUTHOR("RNP");
```