Linux kernel Module Development (2)

Target

1. Write a c/c++ program
2. To implement a Linux kernel module to read and write one specified file
3. GCC

Tools

Install GCC Software Collection

```
sudo apt-get install build-essential
```

How to use GCC

- gcc and make

Case 2: Char device based on Linux Kernel Module

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Robert W. Oliver II");
MODULE_DESCRIPTION("A simple example Linux module.");
MODULE_VERSION("0.01");

#define DEVICE_NAME "lkm_example"
#define EXAMPLE_MSG "Hello, World!\n"
#define MSG_BUFFER_LEN 15

/* Prototypes for device functions */
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *);

static int major_num;
static int device_open_count = 0;
static char msg_buffer[MSG_BUFFER_LEN];
static char *msg_ptr;

/* This structure points to all of the device functions */
static struct file_operations file_ops = {
 .read = device_read,
 .write = device_write,
 .open = device_open,
 .release = device_release
```

```
};

/* When a process reads from our device, this gets called. */
static ssize_t device_read(struct file *flip, char *buffer, size_t len, loff_t
*offset) {
 int bytes_read = 0;
  /* If we're at the end, loop back to the beginning */
  if (*msg_ptr == 0) {
   msg_ptr = msg_buffer;
  }
  /* Put data in the buffer */
  while (len && *msg_ptr) {
    /* Buffer is in user data, not kernel, so you can't just reference
     * with a pointer. The function put_user handles this for us */
    put_user(*(msg_ptr++), buffer++);
    len--;
    bytes_read++;
 }
  return bytes_read;
}

/* Called when a process tries to write to our device */
static ssize_t device_write(struct file *flip, const char *buffer, size_t len,
loff_t *offset) {
 /* This is a read-only device */
  printk(KERN_ALERT "This operation is not supported.\n");
  return -EINVAL;
}

/* Called when a process opens our device */
static int device_open(struct inode *inode, struct file *file) {
  /* If device is open, return busy */
  if (device_open_count) {
   return -EBUSY;
  }
  device_open_count++;
  try_module_get(THIS_MODULE);
  return 0;
}

/* Called when a process closes our device */
static int device_release(struct inode *inode, struct file *file) {
  /* Decrement the open counter and usage count. Without this, the module would
not unload. */
  device_open_count--;
  module_put(THIS_MODULE);
  return 0;
}

static int __init lkm_example_init(void) {
  /* Fill buffer with our message */
  strncpy(msg_buffer, EXAMPLE_MSG, MSG_BUFFER_LEN);
  /* Set the msg_ptr to the buffer */
  msg_ptr = msg_buffer;
  /* Try to register character device */
  major_num = register_chrdev(0, "lkm_example", &file_ops);
  if (major_num < 0) {
   printk(KERN_ALERT "Could not register device: %d\n", major_num);
```

```
   return major_num;
  } else {
   printk(KERN_INFO "lkm_example module loaded with device major number %d\n",
major_num);
   return 0;
  }
}

static void __exit lkm_example_exit(void) {
  /* Remember — we have to clean up after ourselves. Unregister the character
device. */
  unregister_chrdev(major_num, DEVICE_NAME);
  printk(KERN_INFO "Goodbye, World!\n");
}

/* Register module functions */
module_init(lkm_example_init);
module_exit(lkm_example_exit);
```

Compile the above Linux kernel module

1. 创建Makefile文件:

```
ModuleName=lkm_example
obj-m +=${ModuleName}.o
all:${ModuleName}.ko
${ModuleName}.ko:${ModuleName}.c
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
test:${ModuleName}.ko
    echo make test_ins
    echo make test_mk
    echo make test_test
    echo make test_rm
test_ins:${ModuleName}.ko
    sudo dmesg -C
    sudo insmod ${ModuleName}.ko
    sudo dmesg
test_mk:${ModuleName}.ko
    echo "sudo mknod /dev/${ModuleName} c MajorNum 0"
test_test:${ModuleName}.ko
    cat /dev/${ModuleName}
test_rm:${ModuleName}.ko
    sudo rmmod ${ModuleName}.ko
    sudo dmesg
```

2. 编译:

```
make test
```

3. 执行:

```
make test_ins
make test_mk
make test_test
make test_rm
```

Result: make test_test 时需要Ctrl+C来中止输出

```
sudo dmesg -C
sudo insmod lkm_example.ko
sudo dmesg
[ 5031.385259] lkm_example module loaded with device major number 237
echo "sudo mknod /dev/lkm_example c MajorNum 0"
sudo mknod /dev/lkm_example c MajorNum 0
cat /dev/lkm_example
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
...
Hello, World!
sudo rmmod lkm_example.ko
sudo dmesg
[ 5031.385259] lkm_example module loaded with device major number 237
[ 5267.758327] Goodbye, World!
```

Target & how to do

To implement a Linux kernel module to read and write one specified file

- 选择一个具体文件(xxxfile)，利用该内核机制，实现对该文件的读操作
  - cat xxxfile
- 进一步实现对该文件的写操作
  - echo "hello 学号姓名日期..." > xxxfile

    或
  - echo "hello 学号姓名日期..." >> xxxfile

Related Technology

struct file_operations

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
    ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
    int (*iopoll)(struct kiocb *kiocb, struct io_comp_batch *,
            unsigned int flags);
    int (*iterate) (struct file *, struct dir_context *);
    int (*iterate_shared) (struct file *, struct dir_context *);
    __poll_t (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    unsigned long mmap_supported_flags;
```

```
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, loff_t, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *,
int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned
long, unsigned long, unsigned long);
    int (*check_flags)(int);
    int (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *,
size_t, unsigned int);
    ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *,
size_t, unsigned int);
    int (*setlease)(struct file *, long, struct file_lock **, void **);
    long (*fallocate)(struct file *file, int mode, loff_t offset,
              loff_t len);
    void (*show_fdinfo)(struct seq_file *m, struct file *f);
#ifndef CONFIG_MMU
    unsigned (*mmap_capabilities)(struct file *);
#endif
    ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
            loff_t, size_t, unsigned int);
    loff_t (*remap_file_range)(struct file *file_in, loff_t pos_in,
                   struct file *file_out, loff_t pos_out,
                   loff_t len, unsigned int remap_flags);
    int (*fadvise)(struct file *, loff_t, loff_t, int);
    int (*uring_cmd)(struct io_uring_cmd *ioucmd, unsigned int issue_flags);
} __randomize_layout;
```