苏州大学实验报告

院、系	计算机学院	年级专业	19计科图灵	姓名	张昊	学号	1927405160
课程名称	操作系统课程实践					成绩	
指导教师	李培峰	同组实验者	无		实验日期	2022 年 6 月 16 日	

实 验 名 称　　实验 10　Linux 设备管理

一. 实验目的

1. 了解 Linux 字符设备管理机制。
2. 学习字符设备的基本管理方法。
3. 学会编写简单的字符设备驱动程序的方法。
4. 了解 Linux 块设备管理机制。
5. 学习块设备的基本管理方法。
6. 学会编写一个简单的块设备驱动程序。

二. 实验内容

1. （**实验 12.1：编写字符设备驱动程序**）

   （1）编写字符设备驱动程序，要求能对字符设备执行打开、读、写、I/O 控制和关闭这 5 项基本操作。

   （2）编写一个应用程序，测试添加的字符设备和块设备驱动程序的正确性。

2. （**实验 12.2：编写块设备驱动程序**）

   编写一个简单的块设备驱动程序，实现一套内存中的虚拟磁盘驱动器，并通过实际操作验证块设备驱动是否可以正常工作。

三. 操作方法和实验步骤

1. （**实验 12.1：编写字符设备驱动程序**）

   （1）编写设备驱动源程序，即编写内核模块文件 chardev.c 和 Makefile 文件。

   对字符设备的实现为：同一时间只允许被一个进程打开一次，每次打开会将内容重置为一个字符串：I already told you %d times Hello world，其中%的为打开的次数。对字符设备的读取默认会输出这个字符串，写入则会覆盖该字符串。

   chardev.c：

```c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#define SUCCESS  0
#define DEVICE_NAME  "chardev"
#define BUF_LEN  80
static int Major;
```

```c
static int Device_Open = 0;  //设备打开数量
static char msg[BUF_LEN]; //字符缓冲区
static char *msg_Ptr = NULL;  //发送的数据指针
//函数声明
static int device_open(struct inode *inode, struct file *file);
static int device_release(struct inode *inode, struct file *file);
static ssize_t device_read(struct file *filp,
                       char __user *buffer,
                       size_t length,
                       loff_t *offset);
static ssize_t device_write(struct file *filp,
                        const char __user *buffer,
                        size_t length,
                        loff_t *offset);
//定义初始化字符设备的操作方法
static struct file_operations fops =
    {.read = device_read, .write = device_write, .open = device_open, .release =
device_release};
//打开设备
static int device_open(struct inode *inode, struct file *file) {
  static int counter = 0;
  if (Device_Open)
    return -EBUSY;
  Device_Open++;
  sprintf(msg, "I already told you %d times Hello world\n", counter++);
  msg_Ptr = msg;
  try_module_get(THIS_MODULE);
  return SUCCESS;
}
//释放设备
static int device_release(struct inode *inode, struct file *file) {
  Device_Open--;
  module_put(THIS_MODULE);
  return 0;
}
//读设备
static ssize_t device_read(struct file *filp,
                       char __user *buffer,
                       size_t length,
                       loff_t *offset) {
  if (*msg_Ptr == 0)
    return 0;
  ssize_t max_len = (ssize_t) strlen(msg);
  ssize_t len = min((ssize_t) length, max_len);
  //内核空间不能直接访问用户空间的buffer
```

```
  copy_to_user(buffer, msg_Ptr, len);
  return len;
}
//写设备
static ssize_t device_write(struct file *filp,
                            const char __user *buffer,
                            size_t length,
                            loff_t *offset) {
  ssize_t len = min((int) length, BUF_LEN);
  //内核空间不能直接访问用户空间的buffer
  copy_from_user(msg_Ptr, buffer, len);
  return len;
}
//初始化字符设备
int init_chardev_module(void) {
  Major = register_chrdev(0, DEVICE_NAME, &fops);
  if (Major < 0) {
    printk("Registering the character device failed with %d \n ", Major);
    return Major;
  }
  printk("<1> I was assigned major number %d ", Major);
  printk("<1> the drive, create a dev file");
  printk("<1> mknod /dev/hello c %d 0.\n", Major);
  printk("<1> I was assigned major number %d ", Major);
  printk("<1> the device file\n");
  printk("<1> Remove the file device and module when done\n");
  printk("<1> By Hao Zhang, 1927405160.\n");
  return 0;
}
//关闭字符设备
void exit_chardev_module(void) {
  unregister_chrdev(Major, DEVICE_NAME);
}
MODULE_LICENSE("Dual BSD/GPL");
module_init(init_chardev_module);
module_exit(exit_chardev_module);
```

Makefile：

```
CONFIG_MODULE_SIG=n
obj-m := chardev.o
KDIR := /usr/src/linux-$(shell uname -r | cut -d '-' -f1)
PWD := $(shell pwd)
all: modules
modules:
   $(MAKE) -C $(KDIR) M=$(PWD) modules
```

```
clean:
	$(MAKE) -C $(KDIR) M=$(PWD) clean
```

（2）使用 make 命令编译驱动模块。

```
# make
```

（3）使用 insmod 命令安装驱动模块。

```
# insmod chardev.ko
# lsmod | grep chardev
```

（4）创建字符设备文件，

语法格式：mknod /dev/文件名 c 主设备号 次设备号

```
# mknod /dev/hello c <major> 0
```

（5）编写测试程序 test.c，访问创建的字符设备文件，测试其打开、关闭、读取和写入。

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#define data_buffer_len 4096
char read_buffer[data_buffer_len];

int main() {
  int fd = open("/dev/hello", O_RDWR);  //以可读可写方式打开
  ssize_t ret;
  size_t length;
  //读取
  ret = read(fd, read_buffer, sizeof(char) * data_buffer_len);
  read_buffer[ret] = '\0';
  printf("Read: %s (size=%ld)\n", read_buffer, ret);
  //写入
  const char *write_buffer = "Hello, here is Hao Zhang!";
  length = strlen(write_buffer);
  ret = write(fd, write_buffer, sizeof(char) * length);
  printf("Write: %s (ret=%ld)\n", write_buffer, ret);
  //再读取
  ret = read(fd, read_buffer, sizeof(char) * length);
  read_buffer[ret] = '\0';
  printf("Read: %s (size=%ld)\n", read_buffer, ret);
  close(fd);
  return 0;
}
```

并使用 gcc 编译这个字符设备文件，然后运行。

```
# gcc test,c -o chardev-test
# ./chardev-test
```

（6）使用 rmmod 卸载模块。

```
# rmmod chardev
```

（7）使用 rm 命令删除所创建的字符设备文件。

```
# rm /dev/hello
```

2. （**实验 12.2：编写块设备驱动程序**）

（1）编写设备驱动源程序，即编写内核模块文件 simp_blkdev.c 和 Makefile 文件。

simp_blkdev.c：

```
#include <linux/module.h>
#include <linux/blkdev.h>
#include <linux/genhd.h>
#include <linux/bio.h>

#define SIMP_BLKDEV_DISKNAME "simp_blkdev"
#define SIMP_BLKDEV_DEVICEMAJOR COMPAQ_SMART2_MAJOR
#define SIMP_BLKDEV_BYTES (50*1024*1024)
#define SECTOR_SIZE_SHIFT 9
static struct gendisk *simp_blkdev_disk;
static struct block_device_operations simp_blkdev_fops = {
    .owner = THIS_MODULE,
};
static struct request_queue *simp_blkdev_queue;
unsigned char simp_blkdev_data[SIMP_BLKDEV_BYTES];
/*磁盘块设备数据请求处理函数*/
static void simp_blkdev_do_request(struct request_queue *q) {
  struct request *req;    //正在处理的请求队列中的请求
  struct bio *req_bio;    //当前请求的bio
  struct bio_vec *bvec;    //当前请求bio 的段
  char *disk_mem;       //需要读/写的磁盘区域
  char *buffer;       //磁盘块设备的请求在内存中的缓冲区
  int i = 0;
  while ((req = blk_fetch_request(q)) != NULL) {
    //判断当前req 是否合法
    if ((blk_rq_pos(req) << SECTOR_SIZE_SHIFT) + blk_rq_bytes(req)
        > SIMP_BLKDEV_BYTES) {
      printk(KERN_ERR SIMP_BLKDEV_DISKNAME": bad request:block=%llu, count=%u \n",
            (unsigned long long) blk_rq_pos(req),
            blk_rq_sectors(req));
      blk_end_request_all(req, -EIO);
      continue;
    }
    //获取需要操作的内存位置
    disk_mem = simp_blkdev_data + (blk_rq_pos(req) << SECTOR_SIZE_SHIFT);
    req_bio = req->bio; //获取当前请求的bio
    switch (rq_data_dir(req)) {
      case READ:
        while (req_bio != NULL) {
          for (i = 0; i < req_bio->bi_vcnt; i++) {
            bvec = &(req_bio->bi_io_vec[i]);
            buffer = kmap(bvec->bv_page) + bvec->bv_offset;
```

```
          memcpy(buffer, disk_mem, bvec->bv_len);
          kunmap(bvec->bv_page);
          disk_mem += bvec->bv_len;
        }
        req_bio = req_bio->bi_next;
      }
      __blk_end_request_all(req, 0);
      break;
    case WRITE:
      while (req_bio != NULL) {
        for (i = 0; i < req_bio->bi_vcnt; i++) {
          bvec = &(req_bio->bi_io_vec[i]);
          buffer = kmap(bvec->bv_page) + bvec->bv_offset;
          memcpy(disk_mem, buffer, bvec->bv_len);
          kunmap(bvec->bv_page);
          disk_mem += bvec->bv_len;
        }
        req_bio = req_bio->bi_next;
      }
      __blk_end_request_all(req, 0);
    default:
      break;
   }
  }
}
/* 模块入口函数 */
static int __init simp_blkdev_init(void) {
  int ret;
  //1.添加设备之前，先申请设备的资源
  simp_blkdev_disk = alloc_disk(1);
  if (!simp_blkdev_disk) {
    ret = -ENOMEM;
    goto err_alloc_disk;
  }
  //2.设置设备相关属性（设备名，设备号，请求队列）
  strcpy(simp_blkdev_disk->disk_name, SIMP_BLKDEV_DISKNAME);
  simp_blkdev_disk->major = SIMP_BLKDEV_DEVICEMAJOR;
  simp_blkdev_disk->first_minor = 0;
  simp_blkdev_disk->fops = &simp_blkdev_fops;
  //将块设备请求处理函数的地址传入 blk_init_queue 函数，初始化一个请求队列
  simp_blkdev_queue = blk_init_queue(simp_blkdev_do_request, NULL);
  if (!simp_blkdev_queue) {
    ret = -ENOMEM;
    goto err_init_queue;
  }
```

```
    simp_blkdev_disk->queue = simp_blkdev_queue;
    set_capacity(simp_blkdev_disk, SIMP_BLKDEV_BYTES >> 9);
    //3.入口处添加磁盘块设备
    add_disk(simp_blkdev_disk);
    return 0;
    err_alloc_disk:
    return ret;
    err_init_queue:
    return ret;
}
/* 模块出口函数 */
static void __exit simp_blkdev_exit(void) {
    del_gendisk(simp_blkdev_disk);    //释放磁盘块设备
    put_disk(simp_blkdev_disk);    //释放申请的设备资源
    blk_cleanup_queue(simp_blkdev_queue);    //清除请求队列
}
MODULE_LICENSE("GPL");
module_init(simp_blkdev_init);
module_exit(simp_blkdev_exit);
```

Makefile：

```
CONFIG_MODULE_SIG=n
obj-m := simp_blkdev.o
KDIR := /usr/src/linux-$(shell uname -r | cut -d '-' -f1)
PWD := $(shell pwd)
all: modules
modules:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean
```

（2）使用 make 命令编译驱动模块。

```
# make
```

（3）使用 insmod 命令安装驱动模块。

```
# insmod simp_blkdev.ko
```

（4）使用 lsblk 命令列出当前的块设备信息。

```
# lsblk
```

（5）格式化设备 simp_blkdev。

```
# mkfs.ext3 /dev/simp_blkdev
```

（6）创建挂载点并挂载块设备。

```
# mkdir -p /mnt/temp1
# mount /dev/simp_blkdev /mnt/temp1
```

（7）查看模块使用情况。

```
# lsmod | grep simp_blkdev
```

（8）对块设备驱动进行调用测试。

```
# cp /etc/init.d/* /mnt/temp1/
# df -h
```

```
# rm -rf /mnt/temp1/*
# df -h
```

（9）取消挂载，查看模块调用结果。

```
# umount /mnt/temp1/
```

（10）使用 rmmod 命令卸载模块。

```
# rmmod simp_blkdev
```

四. 实验结果和分析

1. （**实验 12.1：编写字符设备驱动程序**）

（1）使用 make 命令编译字符设备驱动模块

```
holger@hao-zhang:/codes/exp10/chardev$ make
make -C /usr/src/linux-4.16.10 M=/codes/exp10/chardev modules
make[1]: Entering directory '/usr/src/linux-4.16.10'
  CC [M]  /codes/exp10/chardev/chardev.o
/codes/exp10/chardev/chardev.c: In function ‘device_read’:
/codes/exp10/chardev/chardev.c:53:3: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
   ssize_t max_len = (ssize_t) strlen(msg);
   ^
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /codes/exp10/chardev/chardev.mod.o
  LD [M]  /codes/exp10/chardev/chardev.ko
make[1]: Leaving directory '/usr/src/linux-4.16.10'
holger@hao-zhang:/codes/exp10/chardev$
```

（2）使用 insmod chardev.ko 命令安装编译好的字符驱动模块

使用 lsmod | grep chardev 命令查看该模块是否装载成功

```
holger@hao-zhang:/codes/exp10/chardev$ sudo insmod chardev.ko
[sudo] password for holger:
holger@hao-zhang:/codes/exp10/chardev$ sudo lsmod | grep chardev
chardev                16384  0
holger@hao-zhang:/codes/exp10/chardev$
```

（3）使用 dmesg 命令查看系统分配的主设备号

（4）根据输出的主设备号，利用 mknod 命令创建设备

主设备号为 240，执行如下命令：

```
$ sudo mknod /dev/hello c 240 0
```

```
holger@hao-zhang:/codes/exp10/chardev$ sudo dmesg | tail
[ 3473.671479] perf: interrupt took too long (26495 > 26387), lowering kernel.perf_event_max_sample_rate to 7500
[ 5076.732972] chardev: loading out-of-tree module taints kernel.
[ 5076.744558] chardev: module verification failed: signature and/or required key missing - tainting kernel
[ 5076.777773] <1> I was assigned major number 240
[ 5076.777775] <1> the drive, create a dev file
[ 5076.777777] <1> mknod /dev/hello c 240 0.
[ 5076.777778] <1> I was assigned major number 240
[ 5076.777779] <1> the device file
[ 5076.777779] <1> Remove the file device and module when done
[ 5076.777780] <1> By Hao Zhang, 1927405160.
holger@hao-zhang:/codes/exp10/chardev$ sudo mknod /dev/hello c 240 0
holger@hao-zhang:/codes/exp10/chardev$
```

（5）编译并运行测试程序 test.c

```
holger@hao-zhang:/codes/exp10/chardev$ gcc test.c -o chardev-test
holger@hao-zhang:/codes/exp10/chardev$ ./chardev-test
Read:  (size=-1)
Write: Hello, here is Hao Zhang! (ret=-1)
Read:  (size=-1)
holger@hao-zhang:/codes/exp10/chardev$ sudo ./chardev-test
Read: I already told you 1 times Hello world
 (size=39)
Write: Hello, here is Hao Zhang! (ret=25)
Read: Hello, here is Hao Zhang! (size=25)
holger@hao-zhang:/codes/exp10/chardev$ sudo ./chardev-test
Read: I already told you 2 times Hello world
 (size=39)
Write: Hello, here is Hao Zhang! (ret=25)
Read: Hello, here is Hao Zhang! (size=25)
holger@hao-zhang:/codes/exp10/chardev$
```

需要 root 权限方可运行。

说明：图中为第二次和第三次使用 root 权限运行的输出，由于第一次未能截图，故图中首次运行输出为“I already told you 1 times Hello world”。

（6）使用 rmmod 卸载模块，并使用 rm 命令删除所创建的字符设备文件。

```
holger@hao-zhang:/codes/exp10/chardev$ sudo rmmod chardev
holger@hao-zhang:/codes/exp10/chardev$ sudo lsmod | grep chardev
holger@hao-zhang:/codes/exp10/chardev$ sudo rm /dev/hello
holger@hao-zhang:/codes/exp10/chardev$ ls -l /dev |grep hello
holger@hao-zhang:/codes/exp10/chardev$
```

2. （**实验 12.2：编写块设备驱动程序**）

（1）使用 make 命令编译块设备驱动模块，如图 12.6 所示。

```
# make
```

```
holger@hao-zhang:/codes/exp10/simp_blkdev$ make
make -C /usr/src/linux-4.16.10 M=/codes/exp10/simp_blkdev modules
make[1]: Entering directory '/usr/src/linux-4.16.10'
  CC [M]  /codes/exp10/simp_blkdev/simp_blkdev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /codes/exp10/simp_blkdev/simp_blkdev.mod.o
  LD [M]  /codes/exp10/simp_blkdev/simp_blkdev.ko
make[1]: Leaving directory '/usr/src/linux-4.16.10'
holger@hao-zhang:/codes/exp10/simp_blkdev$
```

（2）挂载块设备驱动模块 simp_blkdev.ko，并使用 lsmod|grep simp_bikdev 命令查看是否挂载成功：

```
# insmod simp_blkdev.ko
# lsmod | grep simp_blkdev
```

```
holger@hao-zhang:/codes/exp10/simp_blkdev$ sudo insmod simp_blkdev.ko
[sudo] password for holger:
holger@hao-zhang:/codes/exp10/simp_blkdev$ lsmod | grep simp_blkdev
simp_blkdev         52445184  0
holger@hao-zhang:/codes/exp10/simp_blkdev$
```

已经挂载成功，且使用者为 0。

（3）使用 lsblk 命令列出当前的块设备信息

```
holger@hao-zhang:/codes/exp10/simp_blkdev$ sudo lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
simp_blkdev  72:0    0   50M  0 disk
sdb           8:16   0  500M  0 disk
└─sdb1        8:17   0  497M  0 part
sr0          11:0    1 1024M  0 rom
sda           8:0    0   80G  0 disk
├─sda2        8:2    0    1K  0 part
├─sda5        8:5    0  975M  0 part [SWAP]
└─sda1        8:1    0   79G  0 part /
holger@hao-zhang:/codes/exp10/simp_blkdev$
```

可以看到刚添加的设备 simp_blkdev，且其大小为 50 MB。

（4）格式化设备 simp_blkdev，在块设备 simp-blkdev 上建立 ext3 文件系统。

```
# mkfs.ext3 /dev/simp_blkdev
```

（5）创建挂载点并挂载块设备。

```
# mkdir -p /mnt/temp1
# mount /dev/simp_blkdev /mnt/temp1
# mount | grep simp_blkdev
```

```
holger@hao-zhang:/codes/exp10/simp_blkdev$ sudo mkfs.ext3 /dev/simp_blkdev
mke2fs 1.42.13 (17-May-2015)
Creating filesystem with 51200 1k blocks and 12824 inodes
Filesystem UUID: f0a2be92-9c64-41a6-b814-e68c10181e32
Superblock backups stored on blocks:
        8193, 24577, 40961

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

holger@hao-zhang:/codes/exp10/simp_blkdev$ sudo mkdir -p /mnt/temp1
holger@hao-zhang:/codes/exp10/simp_blkdev$ sudo mount /dev/simp_blkdev /mnt/temp1
holger@hao-zhang:/codes/exp10/simp_blkdev$ sudo mount | grep simp_blkdev
/dev/simp_blkdev on /mnt/temp1 type ext3 (rw,relatime,data=ordered)
holger@hao-zhang:/codes/exp10/simp_blkdev$
```

挂载成功。

（6）再次查看模块使用情况

```
# lsmod | grep simp_blkdev
```

```
holger@hao-zhang:/codes/exp10/simp_blkdev$ lsmod | grep simp_blkdev
simp_blkdev         52445184  1
holger@hao-zhang:/codes/exp10/simp_blkdev$
```

模块已被调用，且使用者为 1。

（7）对块设备驱动进行调用测试。

进入/mnt/temp1 目录，并尝试复制文件到该目录下，以验证是否可以使用该设备。

```
# cp /etc/init.d/* /mnt/temp1/
# ls /mnt/temp1/
```

```
holger@hao-zhang:/codes/exp10/simp_blkdev$ cd /mnt/temp1/
holger@hao-zhang:/mnt/temp1$ cp /etc/init.d/* .
cp: cannot create regular file './acpid': Permission denied
cp: cannot create regular file './alsa-utils': Permission denied
cp: cannot create regular file './anacron': Permission denied
```

需使用 root 权限。

```
holger@hao-zhang:/mnt/temp1$ sudo cp /etc/init.d/* .
holger@hao-zhang:/mnt/temp1$ ls
acpid           checkfs.sh              grub-common     lightdm              network-manager  README      speech-dispatcher  urandom
alsa-utils      checkroot-bootclean.sh  halt            lost+found           ondemand         reboot      ssh                uuidd
anacron         checkroot.sh            hostname.sh     mountall-bootclean.sh plymouth        resolvconf  thermald           vmware-tools
apparmor        console-setup           hwclock.sh      mountall.sh          plymouth-log     rsync       udev               whoopsie
apport          cron                    irqbalance      mountdevsubfs.sh     pppd-dns         rsyslog     ufw                x11-common
avahi-daemon    cups                    kerneloops      mountkernfs.sh       procps           saned       umountfs
bluetooth       cups-browsed            keyboard-setup  mountnfs-bootclean.sh rc              sendsigs    umountnfs.sh
bootmisc.sh     dbus                    killprocs       mountnfs.sh          rc.local         single      umountroot
brltty          dns-clean               kmod            networking           rcS              skeleton    unattended-upgrades
holger@hao-zhang:/mnt/temp1$
```

（8）查看资源使用情况

```
# df -h
```

```
holger@hao-zhang:/mnt/temp1$ df -h | grep simp
/dev/simp_blkdev    45M  1.1M   41M   3% /mnt/temp1
holger@hao-zhang:/mnt/temp1$
```

新增的文件系统的资源使用情况为 3%。

（9）删除文件并再次查看资源使用情况

```
holger@hao-zhang:/mnt/temp1$ sudo rm ./*
rm: cannot remove './lost+found': Is a directory
holger@hao-zhang:/mnt/temp1$ sudo rm -rf ./*
holger@hao-zhang:/mnt/temp1$ ls
holger@hao-zhang:/mnt/temp1$ df -h | grep simp
/dev/simp_blkdev    45M  817K   42M   2% /mnt/temp1
holger@hao-zhang:/mnt/temp1$
```

再次查看资源使用情况，为 2%

（10）取消挂载，查看模块调用情况，

```
# umount /mnt/temp1/
# lsmod | grep simp_blkdev
```

```
holger@hao-zhang:/mnt/temp1$ sudo umount /mnt/temp1/
umount: /mnt/temp1/: target is busy
        (In some cases useful info about processes that
         use the device is found by lsof(8) or fuser(1).)
holger@hao-zhang:/mnt/temp1$ cd
holger@hao-zhang:~$ sudo umount /mnt/temp1/
holger@hao-zhang:~$ lsmod | grep simp_blkdev
simp_blkdev         52445184  0
holger@hao-zhang:~$
```

需要离开该目录。卸载后调用数从 1 变回 0。

（11）卸载模块

```
# rmmod simp_blkdev
# lsmod | grep simp_blkdev
```

```
holger@hao-zhang:~$ sudo rmmod simp_blkdev
holger@hao-zhang:~$ lsmod | grep simp_blkdev
holger@hao-zhang:~$
```

五. 讨论、心得

1. 通过本次实验，我了解了 Linux 字符设备和块设备的管理机制，掌握了字符设备和块设备的基本管理方法，学会了编写简单的字符设备驱动程序和块设备驱动程序的方法。
2. 通过分析字符设备与块设备的驱动程序，我总结了其实现的异同点。设备驱动程序编写实现的大体框架相同，都是基于内核模块来编写的，其方法与编写一个普通的内核模块并无差异，都是先是要编写内核模块代码，使用 make 命令编译成内核模块，再使用 insmod 命令安装并测试模块。而不同之处在于字符设备驱动和块设备驱动程序具体的实现方法不同，两个设备的使用方式也不同。