河南工業大學

课 程 设 计

课程设计名称：linux 设备驱动程序安装

专 业 班 级 ： 软件 1305

学 生 姓 名 ： 邢志鹏

学 号 ： 201316920315

指 导 教 师 ： 刘扬

课程设计时间： 2015 年 7 月 6-11 日

软件工程 专业课程设计任务书

学生姓名	邢志鹏	专业班级	软件 1305	学号	201316920315
题 目	设备管理—linux 设备驱动程序安装				
课题性质	其它		课题来源	自拟课题	
指导教师	刘扬		同组姓名	无	
主要内容	在 Linux 系统中，编写一个简单的字符型设备驱动程序模块，设备具有独占特性，可执行读和写操作，相关系统调用为 open，close，read，write。Open 和 close 分别相当于请求和释放设备，read 和 write 将内容保存在设备模块内的缓冲区中。设备模块可动态注册和卸载，并建立与之对应的特殊文件/dev/mydev。				
任务要求	认识 Linux 设备的种类和设备工作方式，理解设备驱动程序的工作原理，掌握设备驱动程序的编写规范，能编写并安装简单的设备驱动程序。				
参考文献	张红光 蒋跃军等《UNIX 操作系统实验教程》机械工业出版社 2006 任满杰等《操作系统原理实用教程》 电子工业出版社 2006 汤子瀛 《计算机操作系统》（修订版）西安电子科技大学出版社 2001 张尧学 史美林《计算机操作系统教程》实验指导 清华大学出版社 2000 罗宇等 《操作系统课程设计》机械工业出版社 2005				
审查意见	指导教师签字： 教研室主任签字： 年 月 日				

1 需求分析

1.1 驱动程序介绍

驱动程序负责将应用程序如读、写等操作正确无误的传递给相关的硬件，并使硬件能够做出正确反应的代码。驱动程序像一个黑盒子，它隐藏了硬件的工作细节，应用程序只需要通过一组标准化的接口实现对硬件的操作。

1.2 Linux 设备驱动程序分类

Linux 设备驱动程序在 Linux 的内核源代码中占有很大的比例，源代码的长度日益增加，主要是驱动程序的增加。虽然 Linux 内核的不断升级，但驱动程序的结构还是相对稳定。

Linux 系统的设备分为字符设备(char device)，块设备(block device)和网络设备(network device)三种。字符设备是指在存取时没有缓存的设备，而块设备的读写都有缓存来支持，并且块设备必须能够随机存取(random access)。典型的字符设备包括鼠标，键盘，串行口等。块设备主要包括硬盘软盘设备，CD-ROM 等。

网络设备在 Linux 里做专门的处理。Linux 的网络系统主要是基于 BSD unix 的 socket 机制。在系统和驱动程序之间定义有专门的数据结构(sk_buff)进行数据传递。系统有支持对发送数据和接收数据的缓存，提供流量控制机制，提供对多协议的支持。

1.3 驱动程序的结构

驱动程序的结构如图 3.1 所示，应用程序经过系统调用，进入核心层，内核要控制硬件需要通过驱动程序实现，驱动程序相当于内核与硬件之间的“系统调用”。

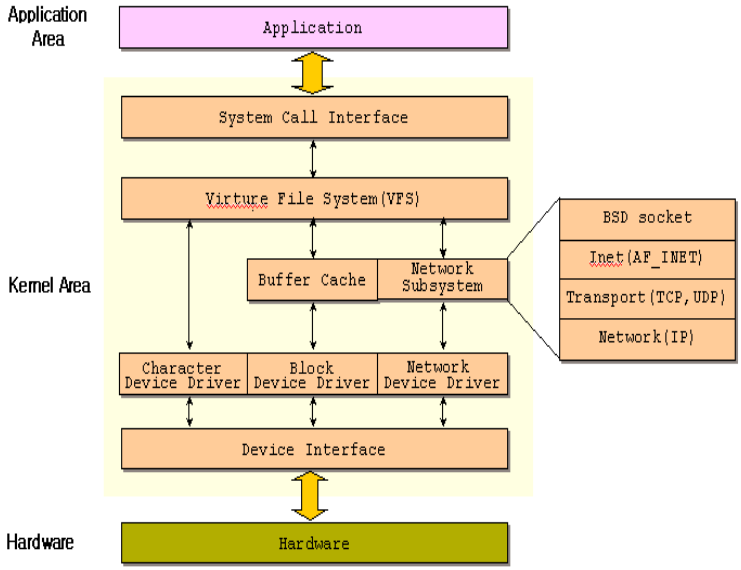


图 1.3 驱动程序的结构

1.3.1 内核模块

内核模块是 Linux 内核的重要组成要素，内核模块能在 Linux 系统启动之后能够动态进行装载和卸载，因此不需对内核进行重新编译或重启系统就可将内核的一部分替换掉，Linux 内核的所有设备驱动，文件系统，网络协议等可做成模块的形式来提供。在所有的模块中需记录编译的内核版本信息，并与当前执行的内核版本一致。即，模块具有版本依赖性，如果不一样就会出错，当然可以在模块程序中的 include<linux/module.h>之前通过宏定义#define__NO_VERSION__表明不定义模块的版本信息。

内核模块程序与一般应用程序之间主要不同之处是，模块程序没有 main()函数，模块程序在装载时调用 init_module(void)函数添加到内核中，在卸载时调用 void cleanup_module( )函数从内核中卸载。另外一个应用程序从头到尾只执行一个任务，但一个模块可以把响应未来请求的事务登记到内核中，然后等待系统调用，内核模块程序结构如图 3.2 所示。

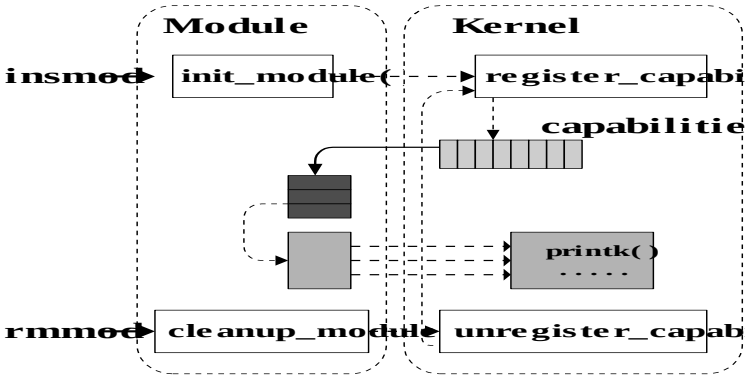


图 3.2 内核模块程序结构

1.4 主、从设备号

应用程序通过设备文件系统（devfs）的名字(或节点)访问硬件设备，所有的设备节点在/dev 目录下。利用 mknod 命令生成设备文件系统的节点，但只有超级用户才能生成设备文。Mknod 命令必须要有设备名和设备类型，主设备号（Major Number），次设备号（Minor Number）等 3 个参数。主设备号用于内核区分设备驱动，次设备号用于设备驱动区分设备。一个设备驱动可能控制多个设备。新的设备驱动要有新的主设备号。在内核源代码的 Documentation/devices.txt 中定义了所有设备的主设备号。在创建设备的时候不要与常用的设备好冲突。

1.5 驱动程序基本框架

如果采用模块方式编写设备驱动程序时，通常至少要实现设备初始化模块、设备打开模块、数据读写与控制模块、中断处理模块（有的驱动程序没有）、设备释放模块和、设备卸载模块等几个部分。

2 概要设计

2.1 重要结构体

打开的设备在内核内部由 file 结构标识，内核使用 file_operation 结构访问驱动程序函数。file_operation 结构是一个定义在<linux/fs.h>中的函数指针数组。每个文件都与它自己的函数集相关联。这个结构中的每一个字段都必须指向驱动程序中实现特定操作的函数。结构如下，详细内容可查阅相关文档。

```
Struct file_operations
{
   struct module *owner;
```

```
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char *, size_t, loff_t
*);
    int (*readdir) (struct file *, void *, filldir_t);
    unsignedint (*poll) (struct file *, structpoll_table_struct *);
    int (*ioctl) (structinode *, struct file *, unsigned int,
unsigned long);
    int (*mmap) (struct file *, structvm_area_struct *);
    int (*open) (structinode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (structinode *, struct file *);
    int (*fsync) (struct file *, structdentry *, intdatasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, structfile_lock *);
    ssize_t (*readv) (struct file *, conststructiovec *, unsigned
long, loff_t *);
    ssize_t (*writev) (struct file *, conststructiovec *, unsigned
long, loff_t *);
    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);
}
```

2.2 在对设备驱动的有了充分的学习后，字符设备的驱动程序确定采用虚拟设备的驱动程序实现，其中确定该设备主要的结构体为：

```
    struct mem_dev
    {
        char *data;
```

```
    unsigned long size;
  };
```

2.3.实现平台为 linux 系统，借助 linux 内核对设备驱动程序的抽象结构体和内核函数，要调用的内核抽象体有：

```
 struct cdev cdev;  //表示一个字符设备的内核设备的抽象体
 static const struct file_operations mem_fops =
{
    .owner = THIS_MODULE,
    .llseek = mem_llseek,
    .read = mem_read,
    .write = mem_write,
    .open = mem_open,
    .release = mem_release,
};
```

2.4 要明确定义虚拟设备的的设备结构体

```
  struct mem_dev
  {
        char *data;
        unsigned long size;
   };
```

2.5 实现模块加载函数和卸载函数

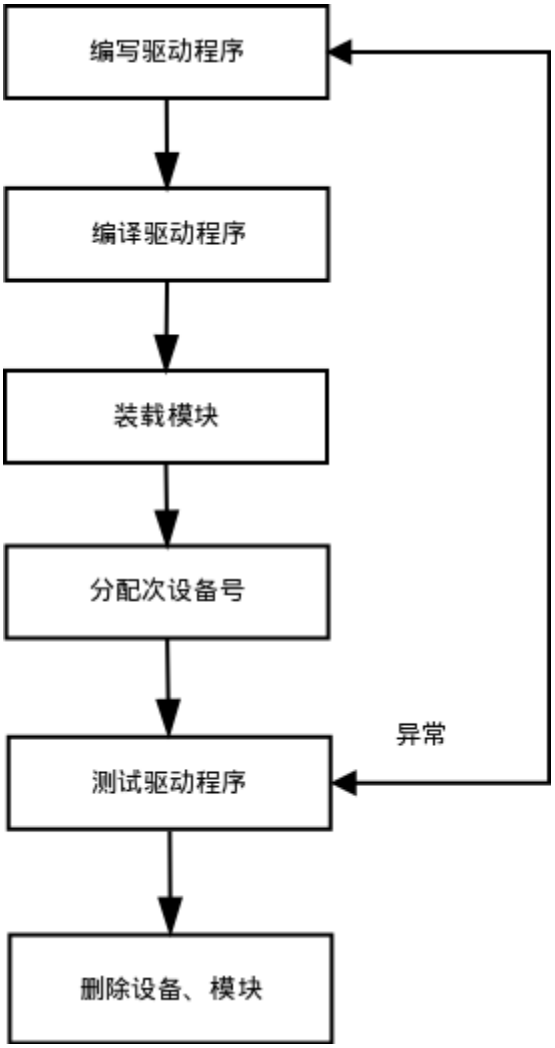
```
static int memdev_init(void){}
static int memdev_exit(void){}
```

2.6 实现 open(),close(),lseek(),write(),read()函数

```
int mem_open(struct inode *inode, struct file *filp);
int mem_release(struct inode *inode, struct file *filp);
static ssize_t mem_read(struct file *filp, char __user *buf,
size_t size, loff_t *ppos);
```

```
static ssize_t mem_write(struct file *filp, const char __user
*buf, size_t size, loff_t *ppos)
static loff_t mem_llseek(struct file *filp, loff_t offset, int
whence);
```

2.7 具体实验流程图



3 运行环境

软件环境：Deepin_2014.3_amd64.iso 安装包

硬件环境：

Processor:	Intel(R) Core(TM) i5-4200H CPU @ 2.80GHz 2.80 GHz
Installed memory (RAM):	4.00 GB (3.89 GB usable)
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

4 开发工具和编程语言

开发工具：vim + gcc

编程语言：C

使用的版本控制 ：GITHUB

5 详细设计

程序主要模块的源代码

```
1. /*设备驱动模块加载函数*/
2. static int memdev_init(void)
3. {
4.   int result;
5.   int i;
6.
7.   dev_t devno = MKDEV(mem_major, 0);      /*通过主设备号得到dev_t
   类型的设备号*/
8.
9.   /* 静态申请设备号*/
10.  if (mem_major)
11.    result = register_chrdev_region(devno, 2, "memdev");
12.  else  /* 动态分配设备号 */
13.  {
14.    result = alloc_chrdev_region(&devno, 0, 2, "memdev");
```

```
15.     mem_major = MAJOR(devno);
16.   }
17.
18.   if (result < 0)
19.     return result;
20.
21.   /*初始化cdev结构*/
22.   cdev_init(&cdev, &mem_fops);//使cdev与mem_fops联系起来
23.   cdev.owner = THIS_MODULE;//owner成员表示谁拥有这个驱动程序，
      使“内核引用模块计数”加 1；THIS_MODULE表示现在这个模块被内核使用，
      这是内核定义的一个宏
24.   cdev.ops = &mem_fops;
25.
26.   /* 注册字符设备 */
27.   cdev_add(&cdev, MKDEV(mem_major, 0), MEMDEV_NR_DEVS);
28.
29.   /* 为设备描述结构分配内存*/
30.   mem_devp = kmalloc(MEMDEV_NR_DEVS*sizeof(struct mem_dev),
      GFP_KERNEL);//目前为止我们始终用GFP_KERNEL
31.   if (!mem_devp)    /*申请失败*/
32.   {
33.     result =  - ENOMEM;
34.     goto fail_malloc;
35.   }
36.   memset(mem_devp, 0, sizeof(struct mem_dev));
37.
38.   /*为设备分配内存*/
39.   for (i=0; i < MEMDEV_NR_DEVS; i++)
40.   {
41.         mem_devp[i].size = MEMDEV_SIZE;
```

```
42.         mem_devp[i].data = kmalloc(MEMDEV_SIZE, GFP_KERNEL);//
    分配出来的地址存在此
43.         memset(mem_devp[i].data, 0, MEMDEV_SIZE);
44.     }
45.
46.     return 0;
47.
48.     fail_malloc:
49.     unregister_chrdev_region(devno, 1);
50.
51.     return result;
52. }
53.
54. /*模块卸载函数*/
55. static void memdev_exit(void)
56. {
57.     cdev_del(&cdev);   /*注销设备*/
58.     kfree(mem_devp);     /*释放设备结构体内存*/
59.     unregister_chrdev_region(MKDEV(mem_major, 0), 2); /*释放设
    备号*/
60. }
```

6. 调试分析

因源码包中已包含 makefile，故利用 make 命令交叉编译 memdev.c、test.c（已修改）等 2 个文件

模块的动态加载，以及/dev/myDevice 节点的创建

运行 drvTest 程序测试，观察结果

7. 测试结果

编译模块：# make

```
laidu@linux-pbvi:~/loop/DeviceDriver> make
make -C /lib/modules/3.16.7-21-desktop/build M=/home/laidu/loop/DeviceDri
ver modules
make[1]: Entering directory '/usr/src/linux-3.16.7-21-obj/x86_64/desktop'
make[1]: Entering directory `/usr/src/linux-3.16.7-21-obj/x86_64/desktop'
  CC [M]  /home/laidu/loop/DeviceDriver/devDrv.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/laidu/loop/DeviceDriver/devDrv.mod.o
  LD [M]  /home/laidu/loop/DeviceDriver/devDrv.ko
make[1]: Leaving directory '/usr/src/linux-3.16.7-21-obj/x86_64/desktop'
laidu@linux-pbvi:~/loop/DeviceDriver>
```

加载模块：# insmod devDrv.ko

```
linux-pbvi:/home/laidu/loop/DeviceDriver # insmod devDrv.ko
linux-pbvi:/home/laidu/loop/DeviceDriver #
```

查看系统中的模块：# lsmod

```
Module                  Size  Used by
devDrv                 12719  0
```

获取住设备号：cat /proc/devices

```
226 drm
250 myDevice
```

创建设备节点：# mknod /dev/myDevice c 250 0

```
linux-pbvi:/home/laidu/loop/DeviceDriver # mknod /dev/myDevice c 250 0
linux-pbvi:/home/laidu/loop/DeviceDriver # ls /dev/myD*
/dev/myDevice
linux-pbvi:/home/laidu/loop/DeviceDriver #
```

测试程序：# gcc drvTest.c

```
linux-pbvi:/home/laidu/loop/DeviceDriver # gcc test.c
test.c: In function ‘main’:
test.c:18:2: warning: ‘gets’ is deprecated (declared at /usr/include/stdio.h:638) [-Wdeprecated-declarations]
  gets(devName);
  ^
test.c:27:3: warning: ‘gets’ is deprecated (declared at /usr/include/stdio.h:638) [-Wdeprecated-declarations]
   gets(get);
   ^
/tmp/ccGuvg1M.o: 在函数‘main’中：
test.c:(.text+0x61): 警告：the `gets' function is dangerous and should not be used.
linux-pbvi:/home/laidu/loop/DeviceDriver #
```

./a.out

```
Please input the device's name you wanna to use :myDevice
The device was inited with a string : hello word!
Please input a string  :
hello word!
```

```
[ 4841.275591] <1>2 times to call the device

The string in the device now is : hello word!
linux-pbvi:/home/laidu/loop/DeviceDriver #
```

删除设备：rm /dev/myDevice

```
linux-pbvi:/home/laidu/loop/DeviceDriver # rm /dev/myDevice
linux-pbvi:/home/laidu/loop/DeviceDriver # ls /dev/myD*
ls: 无法访问/dev/myD*: 没有那个文件或目录
linux-pbvi:/home/laidu/loop/DeviceDriver #
```

卸载模块：# rmmod devDrv

```
linux-pbvi:/home/laidu/loop/DeviceDriver # rmmod devDrv
linux-pbvi:/home/laidu/loop/DeviceDriver # lsmod
Module                  Size  Used by
fuse                  100461  3
sr_mod                 22416  0
```

清除模块生成中间文件：# make clean

```
linux-pbvi:/home/laidu/loop/DeviceDriver # make clean
rm -rf *.o *~ core .depend .*.cmd *.ko *.mod.c .tmp_versions
linux-pbvi:/home/laidu/loop/DeviceDriver #
```

参考文献

[1] 张红光 蒋跃军等《UNIX 操作系统实验教程》机械工业出版社 2006
[2] 任满杰等《操作系统原理实用教程》 电子工业出版社 2006
[3] 汤子瀛 《计算机操作系统》（修订版）西安电子科技大学出版社 2001
[4] 张尧学 史美林《计算机操作系统教程》实验指导 清华大学出版社 2000
[5] 罗宇等 《操作系统课程设计》机械工业出版社 2005

心得体会

通过本次操作系统的课程设计，我经历了从选定课题，在到查资料自学习，到写程序和测试,可以说是一个艰难又具有挑战的过程,在这半个的月的时间里，我翻阅了大概两本关于嵌入式设备驱动开发的书籍,通过了解 linux 内核的结构到相关的内核结构和函数，在到建立一个驱动程序的模型，可以说是个漫长的过程，这是动力，在掌握了内核相关的数据结构和函数后，我开始建立我自己的驱动程序模型，对重点函数进行详细的学习后，比较 open(),read(),write()等，我开始搭建这样一个虚拟的字符设备程序,给我最大的体会就是自学习能力的重要性,在拿到一个课题后我发现我要学习的新知识很多,而且是在很短的时间内，通过参阅资料，总算完成了任务。很感谢这学期以来操作系统老师的认真指导与检查，让我对知识理解的更深了一步。