

```

**** UNREGISTERED COPY ****
Please read the License Agreement in rz.doc
[root@wbyq internship]#insmod module.ko
[ 5145.415000] module: module license 'unspecified' taints kernel.
[ 5145.415000] Disabling lock debugging due to kernel taint
[ 5145.415000] module init
[root@wbyq internship]#rm -rf /dev/beep*
[root@wbyq internship]#mknod /dev/beep c 250 0
[root@wbyq internship]#chmod 777 hjj
[root@wbyq internship]#./hjj /dev/beep
[ 5219.215000] beep_open
[ 5219.215000] beep_unlocked_ioctl
[ 5219.215000] 1
[ 5220.215000] beep_unlocked_ioctl
[ 5220.215000] 0
[ 5221.215000] beep_unlocked_ioctl
[ 5221.215000] 1
[ 5222.215000] beep_unlocked_ioctl
[ 5222.215000] 0
[ 5223.215000] beep_unlocked_ioctl
[ 5223.220000] 1
[ 5224.220000] beep_unlocked_ioctl
[ 5224.220000] 0
[ 5225.220000] beep_release
[root@wbyq internship]#rmmod module.ko
[ 5245.330000] module exit[root@wbyq internship]#

```

校徽

江西师范大学

嵌入式系统开发技术
毕业实习设计

专 业： 电子信息工程
班 级： 14 级电子信息工程（1）班
学 号： 1408071026 1408071021
姓 名： 胡玉杰 何弘
设计题目： 按键和蜂鸣器驱动程序设计

2018 年 1 月

目 录

一、概要

目前嵌入式系统已发展成为以高速处理器和嵌入式操作系统作为核心的软硬件综合系统，Linux 作为一种开源、跨平台的操作系统，受到了越来越多开发者的青睐。随着物联网和人工智能的发展，Linux 将更多地应用在嵌入式设备中，这对 Linux 内核中的驱动设计和实现也提出了更高的要求。

驱动程序在 Linux 内核里扮演着特殊的角色。它们完全隐藏了设备工作的细节，用户的活动通过一套标准化的调用来进行，这些调用与特别的驱动是独立的；设备驱动的角色就是将这些调用映射到作用于实际硬件的和设备相关的操作上。驱动可以与内核的其他部分分开建立，并在需要的时候在运行时“插入”。这种模块化使得 Linux 的驱动程序非常的灵活。

Linux 是一套可以免费使用和自由传播的类 UNIX 操作系统，其实际上只是一个操作系统的内核，主要用于 Intelx86 系列 CPU 的计算机上，谈及 Linux 的起源，起灵感来自于 UNIX。UNIX 操作系统于 1969 年由 Bell 实验室设计开发，之后 Linus Torvalds 设计了 Linux，该系统在发展初期就得到了广大程序员的帮助，逐步发展成为现今这样一个拥有自己版权的完整的系统。Linux 具有很多特点，如支持多种体系结构，支持大量的外围设备，具有完善的网络功能，开放源代码，软件资源丰富，内核稳定等，可以总结为以下几点：

(1) 强大的编程能力。由于 Linux 源自于世界各地成千上万的程序员和黑客，使得 Linux 就犹如加入到了一个感受如云的编程组织中，同时由于 GPL 的存在，Linux 开放源代码，吸引更多专业人士的加入，在这种需求的刺激下，Linux 提供的开发工具功能也越来越完善，越来越强大。

(2) 完善的组网能力。Linux 具有强大的组网能力，它对当前的 TCP/IP 协议提供了完全的支持，同时也支持下一代 Internet 协议 Ipv6. 在安全性方面，Linux 内核中包括了 IP 防火墙代码、IP 防伪及 IP 服务质量控制等特性。

(3) Linux 是自由开放的。Linux 和运行在其上的自由软件，允许成千上万的人检查软件，修改软件，最终可以按照用户自己的意愿来定义自由软件，可以定制自己的 Linux。

(4) 系统稳定。Linux 提供了完全的内存保护机制，每个进程都运行在各自的虚拟地址空间中，不会损坏进程或内核使用的地址空间。

二、设计内容

本次设计是分别实现蜂鸣器驱动和按键驱动，以及将蜂鸣器驱动和按键驱动进行组合联调，基本熟悉 Linux 字符设备开发流程，达到毕业实习预期的目的。

三、准备工作

1、开发环境的搭建

1)、安装 VMware Workstation 和 Red Hat Enterprise Linux 5
安装并打开 VMware Workstation，如图 1 所示。

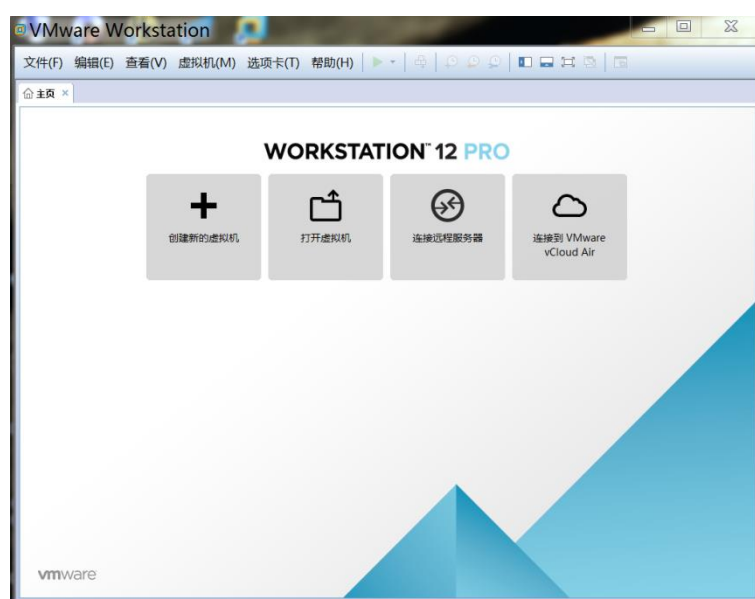


图 1 打开 VMware Workstation

解压拷贝到的已安装好的 Red Hat Enterprise Linux 5 镜像，然后在 VMware Workstation 中将其打开。



图 2



图 3

配置共享文件夹，以便于 Red Hat Enterprise Linux 5 虚拟机和宿主机交换文件。本次设置的共享文件夹目录为 D:\Flushbonading\师大物电驱动实习\Linux_redhat_未搭建环境\share。

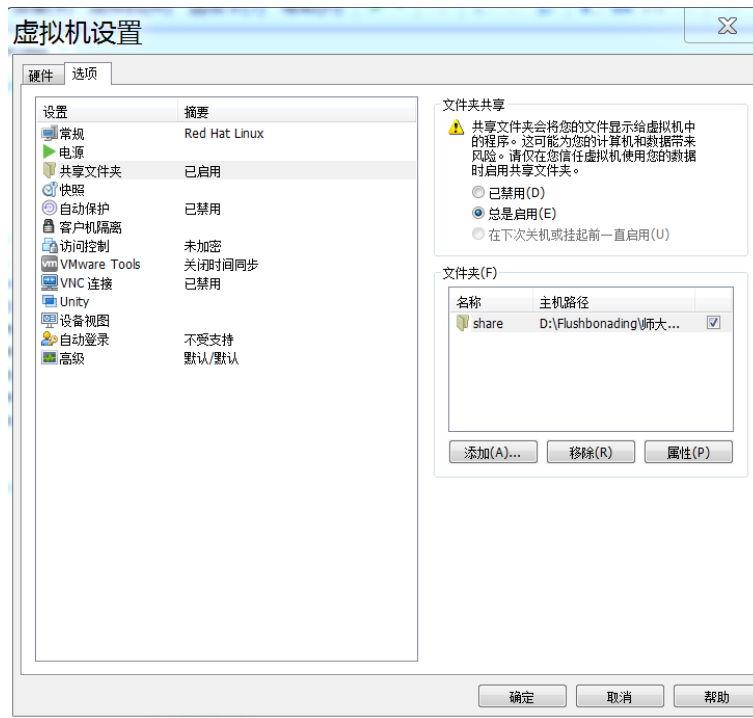


图 4

打开虚拟机，并以 root 账户登录，密码为 123456

2)、安装交叉编译工具 arm-linux-gcc

将 arm-linux-gcc-4.5.1-v6-vfp-20120301.tgz 和 linux-3.5-20170221.tgz 拷贝到共享文件夹。

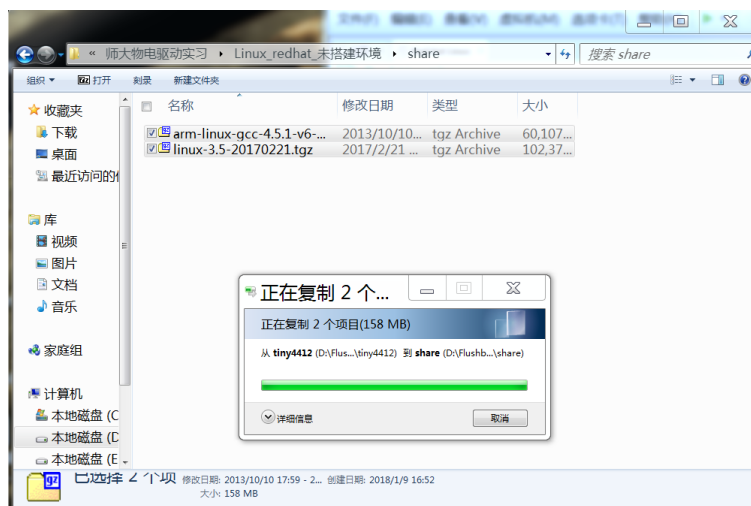


图 5

在虚拟机中打开终端，进入到共享文件夹/mnt/hgfs/share

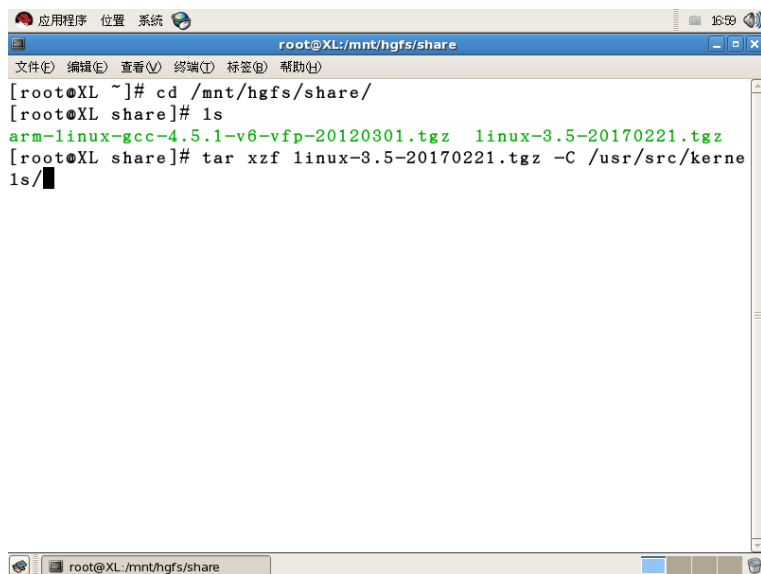


图 6

将 arm-linux-gcc-4.5.1-v6-vfp-20120301.tgz tar 使用 tar 命令解压到/opt 目录下。将 linux-3.5-20170221.tgz 使用 tar 命令解压到/usr/src/kernels 目录下。复制解压的交叉编译工具 arm-linux-gcc 的执行文件所在的目录绝对路径，并将其加入到环境变量中，且用 source 命令使环境变量生效。

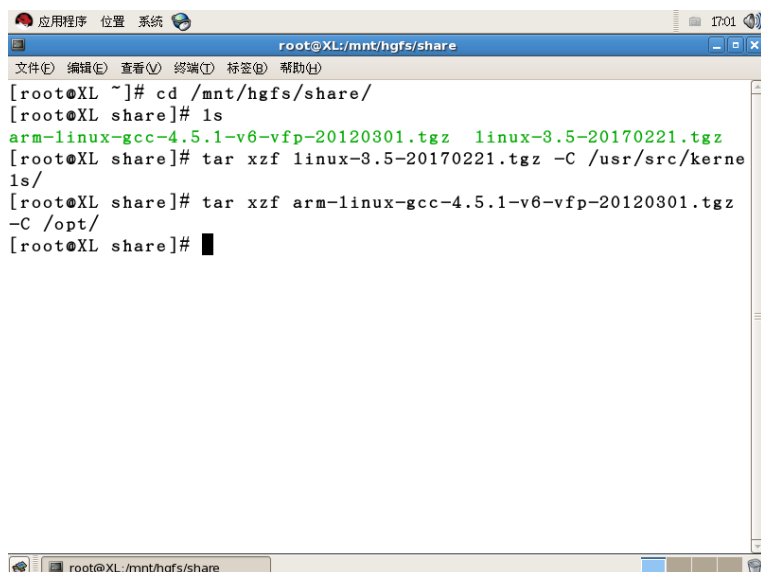


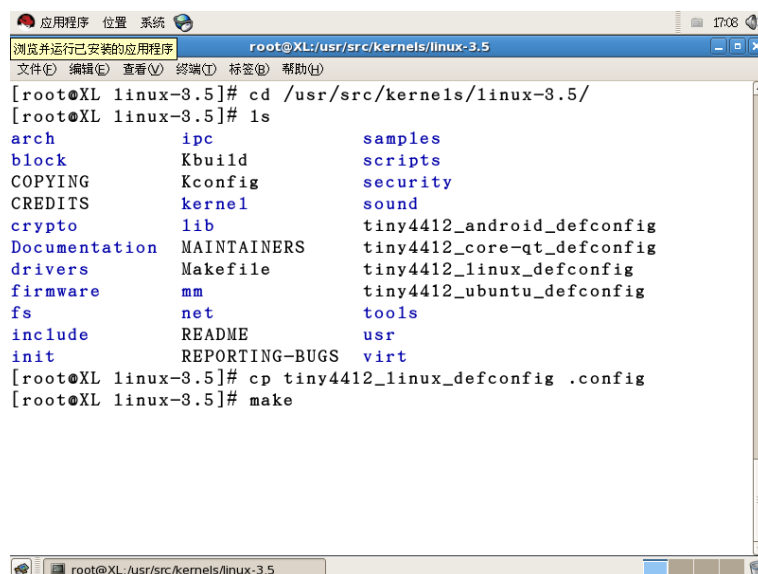
图 7

图 10

执行 `arm-linux-gcc -v` 出现如上所示的版本信息，则交叉编译工具已搭建好。

3)、构建开发板内核树

进入到前面解压的开发板内核源码文件夹(/usr/src/kernels/linux-3.5)。



```
root@XL:/usr/src/kernels/linux-3.5/
[roote@XL linux-3.5]# cd /usr/src/kernels/linux-3.5/
[roote@XL linux-3.5]# ls
arch          ipc           samples
block        Kbuild       scripts
COPYING      Kconfig      security
CREDITS      kernel       sound
crypto       lib          tiny4412_android_defconfig
Documentation MAINTAINERS  tiny4412_core-qt_defconfig
drivers       Makefile     tiny4412_linux_defconfig
firmware     mm           tiny4412_ubuntu_defconfig
fs           net          tools
include      README      usr
init         REPORTING-BUGS virt
[roote@XL linux-3.5]# cp tiny4412_linux_defconfig .config
[roote@XL linux-3.5]# make
```

图 11

由于 Linux 源码支持非常多的硬件架构，编译内核的架构是仅仅由配置参数决定，所以执行 `cp tiny4412_linux_defconfig .config`，将友善之臂配置的 Linux 配置文件作为当前构建内核树的配置文件，然后执行 `make` 命令，等待期编译完。至此，交叉编译环境已经搭建完毕了。

2、C 语言知识复习

1)、变量存储位置

栈区 stack: 由编译器自动分配释放，存放函数的参数值，函数的返回地址，局部变量等，#栈区容量有限，不能溢出。

堆区 heap: 一般由程序员分配释放。

全局区（静态区，数据区，静态数据区，数据段）static: 全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和静态变量存放在另一块相邻的区域，程序结束后由系统释放。

文字常量区: 常量字符串，程序结束后由系统释放。

程序代码区: 存放函数体的二进制代码。

2)、函数返回指针的几种方法

- a、在函数内用 `malloc`，由调用者释放
- b、在函数内使用 `static` 定义
- c、由调用者分配空间，只是把指针传给函数，函数内部把数据拷贝到内存中。（推荐）

3)、其他杂项

函数指针

返回值类型 (*指针变量名)([形参列表])。

3、Linux 基本操作命令

1)、查看文件的几种方法

<code>cat:</code>	打印文件全部内容
<code>more :</code>	分页显示（回车键），只能向下
<code>less:</code>	使用方向键向上或向下
<code>head: head -10 xx</code>	文件首 10 行
<code>tail: tail -5 xx</code>	文件尾部 5 行

2)、关于 Linux 用户管理的命令

`su:` 切换用户，默认的参数为 `root`

用户组的信息构成: 通过查看 `/etc/passwd` 文件可以看到如下信息, 第一个 500 表示 `UID`, 即用户 `ID`, 第二个 500 表示 `GID`, 即组 `ID`。

`double:x:500:500::/home/double:/bin/bash`

<code>adduser:</code>	<code>-g</code> 加入组，默认建立主目录 <code>/home/xx</code>
<code>userdel:</code>	<code>userdel xx [-rf]</code> 删除用户 <code>xx</code> [和其主目录]
<code>usermod:</code>	<code>usermod xx -g root</code> 更改用户所属组

3)、gcc 编译过程

- a、预处理 `-E xx.i`
- b、编译 语法检查 `-S xx.S`
- c、汇编 `-c xx.o`
- d、链接 `-l(library) xx`

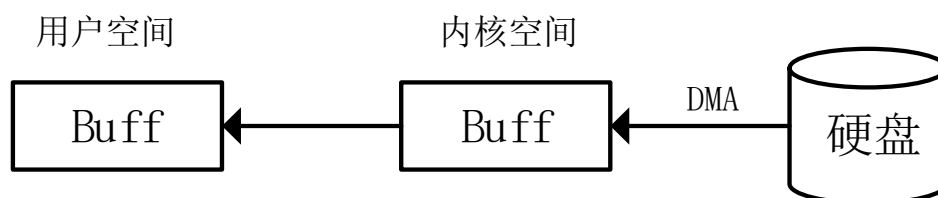
四、字符设备驱动基本结构

1、内核核心功能

- 进程管理。
- 内存管理。
- 文件管理。
- 网路管理。
- 设备管理。

2、驱动程序在系统中扮演的角色

在 Linux 系统中，Linux 使用了 ring0 模式来运行内核态的代码，ring3 来运行用户态的代码。处于 ring3 层面的应用程序不直接和硬件交互，也无法访问硬件，以一次简化的用户空间发起的磁盘读数据为例，过程如下：



其中，驱动程序一般工作在内核空间，应用程序工作在用户空间。在用户程序想要读取硬盘数据时，该进程首先通过系统调用 `read`，从用户态转移到内核态执行 `sys_read`，在内核态中通过 `sys_read` 对应到相应的 `file_operations` `xx_ops.read`，再映射到 `xx_read` 函数，然后将数据从硬盘读出，存入内核空间处的 `Buff`，再通过 `copy_to_user`，将数据传输到用户空间的 `Buff`，执行读操作。

工作在内核空间的驱动可以访问全部的硬件资源，而用户空间的程序则不行。在操作系统中，用户空间的程序出错，一般并不会导致整个系统崩溃。操作系统运行模式的存在，有效的保护了系统的稳定性。

所以说，驱动程序在用户程序与系统硬件之间扮演者不可缺少的角色。

3、驱动类型

1)、字符设备

一个字符(char)设备是一种可以当作一个字节流来存取的设备(如同一个文件)；一个字符驱动负责实现这种行为。这样的驱动常常至少实现 `open`, `close`, `read`, 和 `write` 系统调用。文本控制台(`/dev/console`)和串口(`/dev/ttySn`)是字符设备的例子,因为它们很好地展现了流的抽象。字符设备通过文件系统结点来存取,例如`/dev/tty1` 和`/dev/lp0`。在一个字符设备和一个普通文件之间唯一的不同就是,你经常可以在普通文件中移来移去,但是大部分字符设备仅仅是数据通道,你只能顺序存取。

2)、块设备

如同字符设备,块设备通过位于`/dev` 目录的文件系统结点来存取。一个块设备(例如一个磁盘)应该是可以驻有一个文件系统的。在大部分的 Unix 系统,一个

块设备只能处理这样的 I/O 操作,传送一个或多个长度经常是 512 字节(或一个更大的 2 的幂的数)的整块.相反, Linux 允许应用程序读写一个块设备像一个字符设备一样----它允许一次传送任意数目的字节。

结果就是,块和字符设备的区别仅仅在内核在内部管理数据的方式上,并且因此在内核和驱动的软件接口上不同。如同一个字符设备,每个块设备都通过一个文件系统结点被存取的,它们之间的区别对用户是透明的。块驱动和字符驱动相比,与内核的接口完全不同。

3)、网络设备

通过单独的网络接口调用任何网络事务都通过一个接口来进行,就是说,一个能够与其他主机交换数据的设备。通常,这个接口是一个硬件设备,但是它也可能是一个纯粹的软件设备,比如环回接口。一个网络接口负责发送和接收数据报文,在内核网络子系统的驱动下,不必知道单个事务是如何映射到实际的被发送的报文上的。很多网络连接(特别那些使用 TCP 的)是面向流的,但是网络设备却常常设计成处理报文的发送和接收。一个网络驱动对单个连接一无所知,它只处理报文。

4、驱动程序加载到内核的方法

1)、直接编译进内核

2)、linux 提供了 模块 Module 的机制

insmod 加载模块

rmmod 卸载模块

5、字符驱动开发步骤

1)、基本步骤

a、确定主设备号和次设备号。

b、实现字符驱动程序:

实现 file_operations 结构体,

实现初始化函数,注册字符设备,

实现销毁函数,释放字符设备,

c、创建设备文件节点。

2)、主次设备号

主设备号是内核识别一个设备的标识，是一个整数（占 12bits），范围从 0 到 4095，通常使用 1-255。次设备号由内核使用，用于正确确定设备文件所指向的设备，也是一个整数（占 20bits），范围从 0 到 1048575，通常使用 1-255。

设备编号的内部表达：dev_t 类型(32 位)，用来保存设备编号其中高 12 位为主设备号，低 20 位为次设备号。

从 dev_t 获得主次设备号：MAJOR(dev_t)取得主设备号，MINOR(dev_t)取得次设备号。

分配主设备号：

a、手动分配

```
#include <linux/fs.h>
```

```
int register_chrdev_region(dev_t first,unsigned int count,char *name)
```

b、动态分配

```
#include <linux/fs.h>
```

```
int alloc_chrdev_region(dev_t *dev,unsigned int firstminor,unsigned int count,char *name);
```

c、释放设备号

```
void unregister_chrdev_region(dev_t first,unsigned int count);
```

3)、实现字符驱动

a、初始化 cdev 结构体

```
struct cdev
{
    struct kobject kobj; //内嵌的 kobject 对象
    struct module *owner; //所属模块
    struct file_operations *ops; //文件操作结构体
    struct list_head list;
    dev_t dev; //设备号
    unsigned int count;
}
```

操作 cdev 的函数

初始化一个字符设备：

```
void cdev_init( struct cdev *, struct file_operations *);
```

向操作系统添加一个字符设备

```
int cdev_add(struct cdev *, dev_t, unsigned);
```

从操作系统删除一个字符设备

```
void cdev_del(struct cdev *);
```

b、初始化 file_operations 结构体

根据硬件设备的特点，实现某些函数。应用程序调用 open 打开设备时，内核会将 open 映射到 file_operations 里面的 open 函数指针处。主要成员有 owner，unlocked_ioctl，open，release，read，write 等。为了保证驱动的兼容性，按照 C99 的标准，以如下方式对 file_operations 结构体进行初始化。

```
static struct file_operations xx_ps =
{
    .owner    = THIS_MODULE,
    .read     = xx_read,
    .write    = xx_write,
    .open     = xx_open,
    .release  = xx_release,
    .unlocked_ioctl=xx_unlocked_ioctl,
};
```

五、蜂鸣器驱动模块

六、按键驱动模块

七、蜂鸣器模块与按键模块组合

八、总体设计

- (1) 设计思路
- (2) 设计步骤

九、小组总结

本次实习我们小组在老师的带领下学习了 Linux 字符驱动程序的基本框架。主要完成了对开发环境的搭建、字符设备驱动、蜂鸣器驱动模块、按键驱动模块以及蜂鸣器模块与按键模块的组合的学习，让我们认识了 Linux 字符驱动程序的设计，感受到了 Linux 的强大以及发展前景。

本次实习由于时间有限，只是对嵌入式有了一个大概的了解。还有许多问题还需我们进行进一步的了解和研究。特别是在实习中遇到过很多的问题，有些问题通过查资料或者在同学和老师的帮助下得以解决，也有些问题还是不太明白。也就是说，掌握的只是过于局限和片面，在遇到问题时不能快速准确的找到问题的根源。因此在今后的学习当中我们小组会努力学习，在 Linux 的世界当中遨游，争取做一名优秀的驱动工程师。

十、附录

程序代码