# 一、字符设备驱动框架

 字符设备驱动的编写主要就是驱动对应的open、close、read。。。其实就是

file\_operations结构体的成员变量的实现。

# 二、驱动模块的加载与卸载

 Linux驱动程序可以编译到kernel里面，也就是zImage，也可以编译为模块，.ko。测试的时候只需要加载.ko模块就可以。

编写驱动的时候注意事项！

1、编译驱动的时候需要用到linux内核源码！因此要解压缩linux内核源码，编译linux内核源码。得到zImage和.dtb。需要使用编译后的到的zImage和dtb启动系统。

2、从SD卡启动，SD卡烧写了uboot。uboot通过tftp从ubuntu里面获取zimage和dtb，rootfs也是通过nfs挂在。

3、设置bootcmd和bootargs

bootargs=console=ttymxc0,115200 rw root=/dev/nfs nfsroot=192.168.1.66:/home/zzk/linux/nfs/rootfs ip=192.168.1.50:192.168.1.66:192.168.1.1:255.255.255.0::eth0:off

bootcmd=tftp 80800000 zImage;tftp 83000000 imx6ull-alientek-emmc.dtb;bootz 80800000 - 83000000;

4、将编译出来的.ko文件放到根文件系统里面。加载驱动会用到加载命令：insmod，modprobe。移除驱动使用命令rmmod。对于一个新的模块使用modprobe加载的时候需要先调用一下depmod命令。

5，驱动模块加载成功以后可以使用lsmod查看一下。

6，卸载模块使用rmmod命令

# 三、字符设备的注册与注销

 1、我们需要向系统注册一个字符设备，使用函数register\_chrdev。

 2、卸载驱动的时候需要注销掉前面注册的字符设备，使用函数unregister\_chrdev，注销字符设备。

# 四、设备号

 1，Linux内核使用dev\_t。

typedef \_\_kernel\_dev\_t dev\_t;

typedef \_\_u32 \_\_kernel\_dev\_t;

typedef unsigned int \_\_u32;

2、Linux内核将设备号分为两部分：主设备号和次设备号。主设备号占用前12位，次设备号占用低20位。

3、设备号的操作函数，或宏

从dev\_t获取主设备号和次设备号，MAJOR(dev\_t)，MINOR(dev\_t)。也可以使用主设备号和次设备号构成dev\_t，通过MKDEV(major，minor)

# 五、file\_operations的具体实现

 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 (\*iterate) (struct file \*, struct dir\_context \*);

 unsigned int (\*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 \*);

 int (\*mremap)(struct file \*, struct vm\_area\_struct \*);

 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 (\*aio\_fsync) (struct kiocb \*, 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

};

# 六、字符设备驱动框架的搭建

# 七、应用程序编写

 Linux下一切皆文件，首先要open

# 八、测 试

 1、加载驱动。

 modprobe chrdevbase.ko

 2，进入/dev查看设备文件，chrdevbase。/dev/chrdevbase。但是实际没有，因为我们没有创建设备节点。

 mknod /dev/chrdevbase c 200 0

 3、测试

 ./chrdevbaseAPP /dev/chrdevbase

# 八、chrdevbase虚拟设备驱动的完善

 要求：应用程序可以对驱动读写操作，读的话就是从驱动里面读取字符串，写的话就是应用向驱动写字符串。

 1、chrdevbase\_read驱动函数编写

 驱动给应用传递数据的时候需要用到copy\_to\_user函数。