dmesg命令是将缓冲区的内容读出来:

是将proc/kmsg 打开并将里面的内容读出来

# cat /etc/init.d/rcS //在我们启动的时候有mount –a这个命令

mount –a //将所有的文件系统挂载上那都包括什么东西呢？

mkdir /dev/pts

mount -t devpts devpts /dev/pts

echo /sbin/mdev > /proc/sys/kernel/hotplug

mdev –s

通过命令cat /etc/fstab 可以查看我们都挂载了哪些东西fstab 顾名思义文件系统表

# cat /etc/fstab

proc /proc proc defaults 0 0 //可以看到是将proc 挂载到 /proc下

sysfs /sys sysfs defaults 0 0

tmpfs /dev tmpfs defaults 0 0

可以查看他的挂载内容

# cat /proc/mounts

rootfs / rootfs rw 0 0

/dev/root / nfs rw,vers=2,rsize=4096,wsize=4096,hard,nolock,proto=udp,timeo=11,retrans=2,sec=sys,addr=192.168.0.105 0 0

proc /proc proc rw 0 0

sysfs /sys sysfs rw 0 0

tmpfs /dev tmpfs rw 0 0

devpts /dev/pts devpts rw 0 0

假如我们打印的信息非常多我们想要将我们的打印的内容分开打印的别的内容。

Log\_buf 里面的内容可以通过命令dmsg 将/proc/kmsg 读出来。

仿照这样的形势：

定义mylog\_buf ==> 生成/proc/mymsg ==> 通过cat 将内容打印出来。

我们驱动调用myprintk 将内容存在mylog\_buf。

查看proc\_misc.c //在内核2.6.22.6中似乎2.6.22.6patch中没有这个东西

#ifdef CONFIG\_PRINTK //如果定义了CONFIG\_PRINTK

{

struct proc\_dir\_entry \*entry;

entry = create\_proc\_entry("kmsg", S\_IRUSR, &proc\_root); //会生成/proc/kmsg

if (entry)

entry->proc\_fops = &proc\_kmsg\_operations;

}

#endif

ls -l /proc/kmsg

-r-------- 1 0 0 0 Jan 1 00:38 /proc/kmsg//在proc下也有一个kmsg

我们按照他的那样创建一个自己的mykmsg。

就可以看到有如下的信息：

# insmod mymsg.ko

# ls -l /proc/mymsg

-r-------- 1 0 0 0 Jan 1 00:53 /proc/mymsg

但是cat proc/mymsg 是说是无效的。

我们给file\_operations 填充一个read 函数。执行cat proc/mymsg 会将printk内容打印

const struct file\_operations proc\_mymsg\_operations = {

.read = mymsg\_read,

};

假如我们app:

cat /proc/mymsg 我们希望将我们自己定义的logbuf内容打印出来。

由以上的分析，cat /proc/mymsg肯定是调用proc\_mymsg\_operations read函数

所以我们要在read 函数中将copy\_to\_user不就ok了吗。

内核中的kmsg\_read函数：

static ssize\_t kmsg\_read(struct file \*file, char \_\_user \*buf,size\_t count, loff\_t \*ppos)

{

if ((file->f\_flags & O\_NONBLOCK) && !do\_syslog(9, NULL, 0))

return -EAGAIN;

return do\_syslog(2, buf, count);

}

环形缓冲区：

Buf[10] 最开始分别由一个read write 指针

W =0; R=0

1.空的时候R== w;

2.Write:

Bur[w] = val

W = (w+1)%10 //写的下一个位置

3.Read:

Val =buf[R]

R = (R+1)%10

4.满的时候：

R= (w+1)%10 //下一个要写的位置等于读位置表示满了。

具体的实现可以看我们的学习代码。

do\_syslog()//在我们2.6.22.6没找到 2.6.22.6patch中有找到。

kmsg\_read()

do\_syslog()

//如果log\_start - log\_end 相等说明没有数据休眠

wait\_event\_interruptible(log\_wait, (log\_start - log\_end));

error = \_\_put\_user(c,buf);//将得到的数据传递给用户层.

我们先以固定的十个先测试：

定义数组然后写入内容然后将里面的内容通过read 传给应用。

当我们cat proc/mdesg时候：

会有很多的东西打印出来。因为cat 他会open 文件在一个while循环中不断的读取因为我们renad 一直有返回所以就不停的打印。

因为内核中很多地方都用到printk.我们就想自己开辟一个自己的buffer.然后将信息输出到我们自己的buffer中避免了其他内容混杂在一起啊。就是实现一个myprintk.

1.首先实现一个环形缓冲区：代码

2.实现myprintf函数

在sprintf 的功能上改进一个sprintk,将打印的的内容输出到一个临时的buf然后将这个buf放入到环形缓冲区。

参考源码的kmsg\_read函数：

if ((file->f\_flags & O\_NONBLOCK) && !do\_syslog(9, NULL, 0))

return -EAGAIN;

return do\_syslog(2, buf, count);

// 如果是非阻塞的话O\_NONBLOCK 而且 buff空立马返回

do\_syslog

error = wait\_event\_interruptible(log\_wait, (log\_start - log\_end))// //第二个参数是conditon条件不成立休眠.就是等待直到不空

仿照别人的写法将得到的数据返回应用层。

这时加载驱动 不能用cat 出来东西。因为没有别人调用printk

我们将之前的first\_drv 驱动函数拿过来将printk 全部改为myprintk

要先insmod mymsg.ko 然后在insmod first\_drv.ko 因为first\_drv.ko 要用到

Mymsg的东西。然后执行我们之前的测试代码./firsttest on。

然后cat /proc/mymsg：

就可以看到打印的信息。

first\_drv\_init

first\_drv\_open : 1

first\_drv\_write : 1

first\_drv\_open : 2

first\_drv\_write : 2

但是不能cat 第二次。

因为我们将mylog\_r的位置移动了，我们要保持他的位置不动，定义一个新的变量。

而且增加open函数因为每次。cat 的时候都会调用的open.在open中我们将从新将移动的值拉回到最起始。相当于每次cat 都从开头开始读。

具体的实现参考我们的代码。思想就是我们在调用myprintk的时候写入的的那个r的头除了写的时候将他覆盖的时候其余时候不动。

我们cat的时候用另外的一个变量r\_for\_read，每次cat的时候都从r\_for\_read==r开始

注意在写的时候超过r的时候同样也要讲r\_for\_read移动。

