LINUX 字符设备管理的研究

叶 绿 (杭州应用工程技术学院信电系 310012)

摘要:LINUX 系统包含两类基本设备,块设备和字符设备。本文通过对鼠标和行式打印机这两种比较典型的字符设备的设备驱动程序的研究,来说明和分析 LINUX 系统字符设备的管理。

关键词:LINUX 字符设备 设备管理

一、引言

本文将通过分析两种比较典型的字符设备:鼠标(mouse)和行式打印机(line printer),来说明字符设备的管理。LINUX 系统设备驱动程序的主要结构有三大块(在 LINUX Kernel Hacker's Guide 中亦有介绍),分别为初始化过程 _ init、一组文件操作接口(struct file _ operatons)的定义、以及中断处理过程或者是 Polling 过程,视具体情况而定。

二、鼠标设备的管理

鼠标是一种常见的典型字符设备,在 LINUX 系统中的主设备号为 10。实际上,设备文件/dev/mouse 所对应的是一个逻辑鼠标,系统会自动地找到真正的鼠标,并对其进行操作。系统每一种鼠标的驱动程序只支持一个鼠标,而整个系统在一个时间仅支持一个鼠标。在 LINUX(version 1.3.20)中支持的鼠标种类如下所列:

```
# define BUSMOUSE _ MINOR 0 // Logitech bus mouse
# define PSMOUSE _ MINOR 1 //PS/2 mouse
# define MS _ BUSMOUSE _ MINOR 2 //Microsoft bus mouse
# define ATIXL _ BUSMOUSE _ MINOR 3 //Atixl board bus mouse ...
```

1. 逻辑鼠标

在 mouse 的设备驱动程序/drivers/char/mouse. c 中只出现了两个函数,一个是 mouse _ open 作为这个逻辑鼠标唯一的文件操作接口;另一个是 mouse _ init,作为鼠标的初始过程出现在函数 chr _ dev _ init 中。mouse _ open 根据参数次设备号判断要打开哪一类鼠标,然后简单地把指定的鼠标文件操作接口填入该逻辑设备的文件结构 file - >op;最后当然还得重新调用实际鼠标的文件操作。所以如果我们确切地知道实际的鼠标文件是哪个,例如 msbusmouse,那么对 mouse 进行操作和对 msbusmouse 进行操作的效果是一致的。函数 mouse _ init 分别初始系统配置好的各种鼠标设备,这是在编译的时候确定的。

2. msbusmouse 的驱动

(1)数据结构。设备驱动程序不可避免涉及硬件底层,因此首先我们简单了解一些底层硬件数据。

```
# define MOUSE _ IRQ 5 //中断向量
# define MS _ MSE _ DATA _ PORT 0x23d //I/O 数据端口
# define MS _ MSE _ SIGNATURE _ PORT 0x23e //信号端口
# define MS _ MSE _ CONTROL _ PORT 0x23c //控制端口
# define MS _ MSE _ CONFIG _ PORT 0x23f //配置端口,seems useless
```

对这些 I/O 端口进行操作的过程是 inb 和 outb。所有鼠标中断号都是 5,这就使系统一般只能支持一台鼠标设备。鼠标驱动程序中有一块静态数据区用来保存当前鼠标事件信息,其结构定义于文件/include/linux/busmouse. h,如下所示:

```
struct mouse _ status { //当前鼠标状态
unsigned char buttons; //鼠标按钮状态
unsigned char latch _ buttons;
int dx; // 当前坐标
int dy;
int present; // 鼠标存在状态
int ready; // 数据是否有效
int active; //means busy
struct wait _ queue * wait; //关于鼠标的等待队列
};
```

static struct mouse _ status mouse;

(2)鼠标设备工作原理。首先我们来看鼠标是如何初始化的。过程 mouse _ init 调用了这种 Microsoft bus 鼠标的初始化过程:unsigned long ms _ bus _ mouse _ init(unsigned long kmem _ start)。它开始初始化鼠标状态 mouse 中的域,清 0;然后检测当前的鼠标安装状态,通过若干次从信号端口 MS _ MSE _ SIGNATURE _ PORT 读数据,观察状态是否稳定,从而判断鼠标是否安装。最后程序禁止鼠标的中断。从这里我们可以看到,鼠标设备的检测由驱动程序完成,如果在系统启动的时候鼠标没有安装,那么即使在运行过程中再安装也无济于事。系统仍然不会承认,不过我们也可以稍稍改动驱动程序来满足要求。

现在来分析这个鼠标的文件操作接口。驱动程序并非要实现所有的文件接口,具体设备有各自的工作特点。如鼠标主要读入数据流。

```
struct file _ operations ms _ bus _ mouse _ fops = {
NULL, /* mouse _ seek,字符设备是顺序文件 */
read _ mouse, //read button, x and y coordinates.
write _ mouse, //事实上,调用本函数将返回错误信息
NULL, /* mouse _ readdir */
mouse _ select, /* mouse _ select */
NULL, /* mouse _ ioctl *
NULL, /* mouse _ mmap */
open _ mouse,
release _ mouse,
};
```

·static int open _ mouse(struct inode * inode, struct file * file)

函数首先判断鼠标是否存在或者是否有其他应用正在使用,如果没有,就标识占用鼠标,同时清数据区。然后程序向系统申请 10 号中断向量(reuqest _ irq),登记自己的中断处理程序 ms _ mouse _ interrput。如果已经有其他鼠标申请了这个中断,就失败。最后当然是往控制端口发命令启动鼠标设备并开中断。

·static void release _ mouse(struct inode * inode, struct file * file)

相对来讲释放过程要简单得多,它仅仅关鼠标中断,设置设备空闲标志并释放中断向量。

·static int read _ mouse(struct inode * inode, struct file * file, char * buffer, int count)

这是鼠标主要的功能函数。如果当前存在一个合法的鼠标事件,它就将状态数据区 mouse 中的按钮和坐标状态数据共三字节写入到用户数据区 buffer 中,并更新状态 mouse。

·static int mouse _ select(struct inode * inode, struct file * file, int sel _ type, select _ table * wait)

如果当前没有鼠标事件,就把 wait 插入到 mouse 的等待队列中,等出现事件时由中断处理程序唤醒。这个功能一般与 read _ mouse 结合使用。

LINUX 系统中鼠标工作在中断方式下,这个鼠标的中断处理程序是:

static void ms _ mouse _ interrupt(int irq, struct pt _ regs * regs);

系统在接收到硬中断后自动调用它。它的主要功能是从数据端口读入当前按钮和坐标数据,并把数据填入状态数据区 mouse。最后中断处理程序还要唤醒等待队列中的进程。值得注意的是,鼠标读操作没有阻塞过程,阻塞等待的是 select 操作。不同的鼠标有自己不同的设备驱动程序,但他们基本的工作原理和流程相视,差异只存在于硬件操作。

三、行打设备的管理

行式打印机是 UNIX 系统的一种特别的打印设备,它要比鼠标复杂,有自己的缓冲区,同时系统一般能够支持多台打印机。在 LINUX 系统中,总共能支持三台打印机。行式打印机可以分别工作在中断和 polling 两种方式下,这两种方式可以相互切换。

1. 打印机数据结构

行打在 LINUX 系统中的主设备号为 LP _ MAJOR = 6,它的中断向量根据不同的打印机分别配置。

```
#define LP _ BUFFER _ SIZE 256 //lp's own buffer ,which located in kernel
struct lp _ stats { //line printer current status
  unsigned long chars; //char count
  unsigned long sleeps;
  unsigned int maxrun;
  unsigned int maxwait; //maximum wait time
  unsigned int meanwait; //mean wait time
  unsigned int mdev;
};
struct lp _ struct {
  int base;
```

```
    unsigned int irq;
    int flags;
    unsigned int chars;
    unsigned int time;
    unsigned int wait;
    struct wait_queue * lp_wait_q;
    char * lp_buffer;
    unsigned int lastcall;
    unsigned int runchars;
    unsigned int waittime;
    struct lp_stats stats;
};
```

·lp_-struct 是行打的主要结构，支持三台设备；

·base 是行打设备的 I/O 基地址；

·irq 是中断向量，当它为 0 时，打印机工作在轮询方式

·flag 表示了打印机的当前状态；

·chars 每个字符的时限，以总线周期计算；

·time 记录了当打印机缓冲区填满时，驱动程序的等待时间；

·lp_wait_q 是打印机的等待作业队列；

·lp_buffer 是打印缓冲区；

2. 打印机驱动程序

打印机驱动程序的基本框架与鼠标差不多，所以将省略介绍相似的部分。行打的初始化过程：

long lp_init(long kmem_start);

lp_init 要在系统字符设备表中登记行打：register_chrdev(LP_MAJOR,"lp",&lp_fops)，lp_fops 是行打的文件操作接口。'lp'是设备名。然后程序分别检测三个打印机端口，如果存在打印机，就在相应的打印机结构中作初始化，标志打印机状态并通过调用 lp_reset 向打印机发送初始命令。从程序看，一般开始时打印机工作在轮询方式下而且决定了打印机的存在状态。这里还出现了关于 I/O 资源申请的问题，打印机必须向系统申请相应的 I/O 地址空间。如果要申请 I/O 空间已经被占用了，此打印机的初始过程就失败。关于资源的程序定义在文件/kernel/resource.c 中。

行打的主要文件接口数据定义：

```
static struct file_operations lp_fops = {
    lp_lseek, /* lp_lseek */
    NULL, /* lp_read */
    lp_write, /* lp_write */
    NULL, /* lp_readdir */
    NULL, /* lp_select */
    lp_ioctl, /* lp_ioctl */
    NULL, /* lp_mmap */
    lp_open, /* lp_open */
    lp_release /* lp_release */
};
```

·static int lp_open(struct inode * inode, struct file * file)

驱动程序根据 i 节点确定具体的打印机，即通过 MINOR(inode->i_rdev)确定，一般的驱动程序都使用这种方法分辨不同的设备。接着程序进行一些合法性判断，如是否存在、空闲、是否缺纸等等。我们曾经提到行打能够分别支持中断和 polling 工作方式，在中断方式下，还需申请打印机缓冲区和相应的中断号，并设置中断处理程序 lp_interrupt。最后设置忙标志。

·static void lp_release(struct inode * inode, struct file * file)

这个过程仅仅逆 open 过程，释放缓冲区和中断号，设置空闲标志。如果是 polling 方式则仅需设置空闲标志。

·static int lp_write(struct inode * inode, struct file * file, const char * buf, int count)

根据工作方式分别进行不同的写操作：int lp_write_interrupt(unsigned int minor, const char * buf, int ount) 和 int lp_write_polled(unsigned int minor, const char * buf, int count)。它们的基本功能相同，即把 buf 中的字符写入设备文件，其中在中断方式下首先把 buf 中的内容放入打印机缓冲区中，然后再一个字符一个字符的打印。两种方式主要的不同是对异常情况的处理。polling 方式调用 schedule()函数进入睡眠等待，由系统调度进程唤醒；而中断方式则调用 interruptable_sleep_on 函数在该打印机的等待队列中插入，等到打印机 ready 是由该打印机中断唤醒。总的来说，中断方式的效率更高，更灵活一些。

·static int lp_ioctl(struct inode * inode, struct file * file, unsigned int cmd, unsigned long arg)

ioctl 是一个特殊的文件接口，它的函数入口地址是 sys_ioctl，定义于文件 fs/ioctl.c 中。它主要用来对特殊文件的底层参数进行操作。如果对象是普通文件，主要对文件结构中的 f_flags 参数进行操作，如 FI/OCLEX，FI/ONCLEX 和 FI/ONBI/O 等，分别读取或设置阻塞属

性,进行 bmap 操作等。如果对象是设备文件就调用设备驱动程序的 ioctl 接口。当然并不是所有的设备文件都需要这个接口。在打印机中,该功能函数可以读取和设置 lp _ struct 结构中的参数。如果是用户超级还能重新设置该打印机的中断向量号,中断向量号为 0 即意味着工作在 polling 方式下。

当工作在中断方式下的时候,中断处理程序 lp _ interrupt 根据 irq 找到相应的打印机,并唤醒该打印机上睡眠的进程。每台打印机的中断处理程序都相同,而 polling 方式则不需要这个过程。其实从程序看,两种工作方式比较类似,轮询方式也用 schedule 进行进程切换,只不过中断方式更有目标,挑选该事件队列中的等待进程进行调度。打印机的缓冲区分配在核心空间中,中断方式下用来暂存用户打印数据。而轮询方式下打印机并没有使用这个缓冲区,它直接从用户数据区读取。

参考文献

[1] LINUX Kernel Hacker's Guide,1998 年。

[2] 金连甫,UNIX 系统代码分析,浙江大学出版社,1997 年。

[3] 胡希明等,UNIX 结构分析,浙江大学出版社,1998 年。

(来稿时间:1999 年 5 月)