**字符设备驱动程序结构分析**

计科1201 唐佳娣 1212300112

Linux系统将设备分为3类：字符设备、块设备和网络设备。而字符设备是指只能一个字节一个字节读写的设备，不能随机读取设备内存中的某一数据，读取数据需要按照先后数据。字符设备是面向流的设备，常见的字符设备有鼠标、键盘、串口、控制台和LED设备等。

Linux内核中使用cdev结构体（定义如下所示）来描述字符设备，通过其成员dev\_t来定义设备号（分为主、次设备号）以确定字符设备的唯一性。通过其成员file\_operations来定义字符设备驱动提供给VFS的接口函数，如常见的open()、read()、write()等。

struct cdev {

      struct kobject kobj;

      struct module \*owner;  /\*通常为THIS\_MODULE\*/

      struct file\_operations \*ops; /\*在cdev\_init()这个函数里面与cdev结构联系起来\*/

      struct  list\_head list;

      dev\_t  dev;  /\*设备号\*/

      unsigned int count;

}；

cdev 结构体的dev\_t 成员定义的设备号为32位，其中12位是主设备号，20位是次设备号，只需使用以下二个简单的宏就可以从dev\_t 中获取主设备号和次设备号：

MAJOR(dev\_t dev)

MINOR(dev\_t dev)

相反地，可以通过主次设备号来生成dev\_t：

MKDEV(int major,int minor)

 在调用cdev\_add()函数向系统注册字符设备之前，首先应向系统申请设备号，而设备号可以通过动态和静态两种方法来进行申请。静态申请设备号：

int register\_chrdev\_region(dev\_t from,unsigned count,const char \*name)

动态申请设备号：

int alloc\_chrdev\_region(dev\_t \*dev,unsigned baseminor,unsigned count,const char \*name);

       其中，静态申请是已知起始设备号的情况，如先使用cat /proc/devices 命令查得哪个设备号未事先使用；动态申请是由系统自动分配，只需设置major = 0即可。

下面将对一个简单的字符设备驱动程序进行分析。

#include <linux/kernel.h>

#include <linux/module.h>

#if CONFIG\_MODVERSIONS==1  /\* 处理 CONFIG\_MODVERSIONS \*/

#define MODVERSIONS

#include <linux/modversions.h>

#endif

/\* 下面是针对字符设备的头文件 \*/

#include <linux/fs.h>

#include <linux/wrapper.h>

/\* 对于不同的版本需要做一些必要的事情\*/

#ifndef KERNEL\_VERSION

#define KERNEL\_VERSION(a,b,c) ((a)\*65536+(b)\*256+(c))

#endif

#if LINUX\_VERSION\_CODE > KERNEL\_VERSION(2,4,0)

#include <asm/uaccess.h>

#endif

#define SUCCESS 0

/\* 声明设备 \*/

/\* 这是本设备的名字，它将会出现在 /proc/devices \*/

#define DEVICE\_NAME "char\_dev"

/\* 定义此设备消息缓冲的最大长度 \*/

#define BUF\_LEN  100

/\* 为了防止不同的进程在同一个时间使用此设备，定义此静态变量跟踪其状态 \*/

static int Device\_Open = 0；

/\* 当提出请求的时候，设备将读写的内容放在下面的数组中 \*/

static char Message[BUF\_LEN]；

/\* 在进程读取这个内容的时候，这个指针是指向读取的位置\*/

static char \*Message\_Ptr ；

/\* 在这个文件中，主设备号作为全局变量以便于这个设备在注册和释放的时候使用。\*/

static int Major;

/\*open （）函数：无论一个进程何时试图去打开这个设备都会调用这个函数。 \*/

static int device\_open(struct inode \*inode,

              struct file \*file)

{

  static int counter = 0;

#ifdef DEBUG

  printk ("device\_open(%p,%p)\n", inode, file);

#endif

  printk("Device: %d.%d\n",

    inode->i\_rdev >> 8, inode->i\_rdev & 0xFF);

/\* 这个设备是一个独占设备，为了避免同时有两个进程使用这一个设备我们需要采取一定的措施\*/

  if (Device\_Open)

    return -EBUSY;

  Device\_Open++;

/\* 下面是初始化消息 ， 注意不要使读写内容的长度超出缓冲区的长度，特别是运行在内核模式时，否则如果出现缓冲上溢则可能导致系统的崩溃。\*/

  sprintf(Message,

 "If I told you once, I told you %d times - %s",

   counter++,

    "Hello, world\n");

  Message\_Ptr = Message;

/\*当这个文件被打开的时候，必须确认该模块还没有被移走并且增加此模块的用户数目。（在移走一个模块的时候会根据这个数字去决定可否移去，如果不是 0 则表明还有进程正在使用这个模块，不能移走。）\*/

  MOD\_INC\_USE\_COUNT;

  return SUCCESS;

}

/\*release ( ) 函数： 当一个进程试图关闭这个设备特殊文件的时候调用这个函数。\*/

#if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(2,4,0)

static int device\_release(struct inode \*inode,

             struct file \*file)

#else

static void device\_release(struct inode \*inode,

              struct file \*file)

#endif

{

#ifdef DEBUG

  printk ("device\_release(%p,%p)\n", inode, file);

#endif

  /\* 为下一个使用这个设备的进程做准备。\*/

  Device\_Open --;

/\* 减少这个模块使用者的数目，否则一旦打开这个模块以后，将永远都无法被释放。\*/

  MOD\_DEC\_USE\_COUNT;

#if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(2,4,0)

  return 0;

#endif

}

/\*read ( ) 函数：当一个进程已经打开此设备文件以后并且试图去读它的时候调用这个函数。\*/

#if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(2,4,0)

static ssize\_t device\_read(struct file \*file,

    char \*buffer,    /\* 把读出的数据放到这个缓冲区\*/

    size\_t length,   /\* 缓冲区的长度\*/

    loff\_t \*offset)  /\* 文件中的偏移 \*/

#else

static int device\_read(struct inode \*inode,

                       struct file \*file,

    char \*buffer,       int length)

#endif

{

  /\* 实际上读出的字节数 \*/

  int bytes\_read = 0;

  /\* 如果读到缓冲区的末尾，则返回0 ，类似文件的结束。\*/

  if (\*Message\_Ptr == 0)

    return 0;

  /\* 将数据放入缓冲区中\*/

  while (length && \*Message\_Ptr)  {

 /\* 由于缓冲区是在用户空间而不是内核空间，所以必须使用 copu\_to\_user()函数将内核空间中的数据拷贝到用户空间。\*/

    copy\_to\_user(buffer++,\*(Message\_Ptr++), length--);

    bytes\_read ++;

  }

#ifdef DEBUG

   printk ("Read %d bytes, %d left\n",

     bytes\_read, length);

#endif

   /\* Read 函数返回一个真正读出的字节数\*/

  return bytes\_read;

}

/\*write ( )  函数：当试图将数据写入这个设备文件的时侯，这个函数被调用\*/

#if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(2,4,0)

static ssize\_t device\_write(struct file \*file,

    const char \*buffer,

    size\_t length,

    loff\_t \*offset)

#else

static int device\_write(struct inode \*inode,

                        struct file \*file,

                        const char \*buffer,

                        int length)

#endif

{

  int i;

#ifdef DEBUG

  printk ("device\_write(%p,%s,%d)", file, buffer, length);

#endif

#if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(2,4,0)

    copy\_from\_user(Message, buffer，length);

  Message\_Ptr = Message;

  /\* 返回写入的字节数 \*/

  return i;

}

/\*这个设备驱动程序提供给文件系统的接口，它的指针保存在设备表中，在init\_module（）中被传递给操作系统。\*/

struct file\_operations Fops = {

  read:    device\_read,

  write:    device\_write,

  open:     device\_open,

  release: device\_release

  };

/\*这个函数用来初始化这个模块 ——注册该字符设备。init\_module （）函数调用module\_register\_chrdev，把设备驱动程序添加到内核的字符设备驱动程序表中，它返回这个驱动程序所使用的主设备号。\*/

int init\_module()

{

  /\* 试图注册设备\*/

  Major = module\_register\_chrdev(0,

                                 DEVICE\_NAME,

                                 &Fops);

  /\* 失败的时候返回负值\*/

  if (Major < 0) {

    printk ("%s device failed with %d\n",

       "Sorry, registering the character",

       Major);

    return Major;

  }

  printk ("%s The major device number is %d.\n",

          "Registeration is a success.",

          Major);

  printk ("If you want to talk to the device driver,\n");

  printk ("you'll have to create a device file. \n");

  printk ("We suggest you use:\n");

  printk ("mknod <name> c %d <minor>\n", Major);

  printk ("You can try different minor numbers %s",

          "and see what happens.\n");

  return 0;

}

/\*这个函数的功能是卸载模块，主要是从  /proc 中 取消注册的设备特殊文件。\*/

void cleanup\_module()

{

  int ret;

  /\* 取消注册的设备\*/

  ret = module\_unregister\_chrdev(Major, DEVICE\_NAME);

  /\* 如果出错则显示出错信息 \*/

  if (ret < 0)

    printk("Error in unregister\_chrdev: %d\n", ret);

}