嵌入式 Linux 设备驱动程序的构建方法

陈　年

(重庆大学计算机学院　重庆 400030)　(四川理工学院　自贡 643000)

摘　要　本文在阐明 Linux 设备驱动程序工作原理的基础上,分析了嵌入式 Linux 系统设备驱动程序的开发特点,介绍了嵌入式 Linux 系统中设备驱动程序的构建方法并给出了实例。

关键词　嵌入式系统,嵌入式 Linux,设备驱动程序

Linux 由于其具有内核强大且稳定,易于扩展和裁减,效率高,丰富的硬件支持等许多优点,在嵌入式系统中得到了广泛的应用。针对应用于不同场合的嵌入式 Linux 系统,特别是一些新兴的嵌入应用,各种新的设备加入到系统中。此时开发者需要根据实际情况,为自己的特殊设备编写驱动程序。因而,越来越多的公司和个人都需要寻求为自己的设备或特殊应用设计用于嵌入式 Linux 系统的设备驱动程序。本文基于嵌入式操作系统下设备驱动程序的开发需要,阐述相关技术原理及设计要点,探求嵌入式 Linux 系统中设备驱动程序的构建方法。

1　Linux 设备驱动程序

Linux 是 Unix 操作系统的一种变种,在 Linux 下编写设备驱动程序的原理和思想完全类似于其他的 Unix 系统,即基于 I/O 设备管理采用的分层模型,如图 1 所示。从图 1 中可以看到,I/O 设备管理软件位于内核中的最底层,设备驱动程序是操作系统内核和机器硬件之间的接口,设备驱动程序为应用程序屏蔽了硬件的细节。在应用程序看来,硬件设备只是一个设备文件,应用程序可以像操作普通文件一样对硬件设备进行操作,用户感觉不到访问设备驱动程序与访问普通文件之间有什么差别。

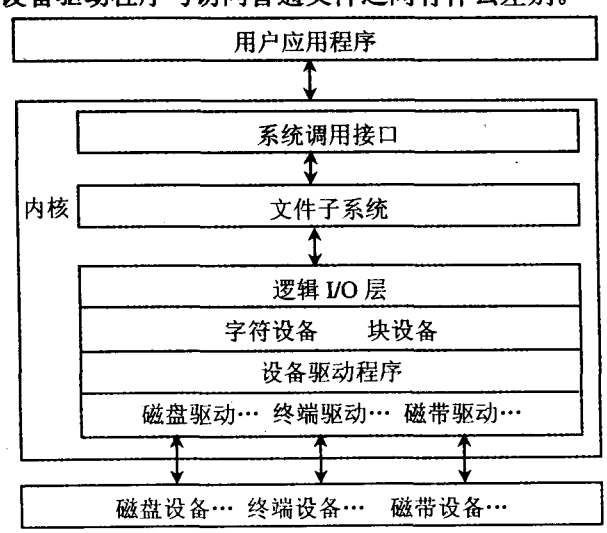

图 1　Linux I/O 管理层次结构及其相互关系

1.1　设备驱动程序的工作原理

作为内核的一部分,设备驱动程序完成对设备初始化和释放、把数据从内核传送到硬件和从硬件读取数据、读取应用程序传送给设备文件的数据和回送应用程序请求的数据和检测处理设备出现的错误的功能。Linux 设备主要分两类:字符设备和块设备,其主要区别是:在对字符设备发出读/写请求时,实际的硬件 I/O 一般就紧接着发生了,块设备则不然,它利用一块系统内存作为高速缓存,当用户进程对设备请求能满足用户的要求,就返回请求的数据,如果不能,就调用请求函数来进行实际的 I/O 操作。

组成设备驱动程序的代码分为上半部分和下半部分,这是按代码执行的时间是否确定来划分的。同步执行的代码为驱动程序的下半部分,它完成 I/O 控制和控制功能;异步执行的代码为上半部分,它直接访问硬件,实际上是一个中断处理程序,包含用户请求的 I/O 系统调用,包括读、写数据和执行等操作,如 read(),write()调用,以及启动和关闭设备等。

1.2　逻辑 I/O 与设备驱动程序的接口

逻辑 I/O 层通过内核定义的两个数据结构块设备转换表(blkdevs)和字符设备转换表(chrdevs)来实现与设备驱动程序的接口。每个设备驱动程序在设备转换表中占据一个表项。每个 Linux 设备文件都有两个设备号,第一个是主设备号,标识驱动程序,第二个是从设备号,标识使用同一个设备驱动程序的不同的硬件设备,比如有两个软盘,就可以用从设备号来相互区分。转换表中定义了每个设备驱动程序的入口点,如字符设备转换表:

```
struct device_struct{
    const char * name;
    struct file_operations * fops;
};
```

其中的 fops 就描述了字符设备驱动程序的入口点。入口点函数的功能由各个驱动程序具体实

现。把系统调用和驱动程序关联起来的关键数据结构是 file_operations：

```
struct file_operations{
int(* seek)(struct inode * ,struct file * ,off_t,int);
int(* read)(struct inode * ,struct file * ,char,int);
int(* write)(struct inode * ,struct file * ,off_t,int);
int (* readdir)(struct inode * ,struct file * ,struct dirent * ,
    int);
int (* select)(struct inode * ,struct file * ,int,select_table
    * );
int (* ioctl)(struct inode * ,struct file * ,unsined int,un-
    signed long
int (* mmap)(struct inode * ,struct file * ,struct vm_area_
    struct * );
int(* open)(struct inode * ,struct file * );
int(* release)(struct inode * ,struct file * );
int(* fsync)(struct inode * ,struct file * );
int(* fasync)(struct inode * ,struct file * ,int);
int(* check_media_change)(struct inode * ,struct file * );
int(* revalidate)(dev_t dev);
}
```

该结构中的每一个成员的名字都对应着一个系统调用。用户进程利用系统调用在对设备文件进行 read/write 等各种操作时，系统调用通过设备文件的主设备号找到相应的设备驱动程序，然后读取这个数据结构相应的函数指针，接着把控制权交给该函数。

2 嵌入式 Linux 系统设备驱动程序的特点

嵌入式 Linux 系统中的设备驱动程序和 Linux 中的大多数驱动程序一样，也是采用层次型的体系结构。在了解了 Linux 的设备驱动程序工作的基本原理的基础上，编写设备驱动程序的方法也就明确了，其主要工作就是编写子函数，并填充 file_operations 的各个域。但是嵌入式 Linux 系统的设备驱动程序还是具有其特殊之处的。

Linux 的设备驱动程序模块按照方式编译可以分为两类，一类是静态链接的设备驱动程序模块，这类模块在编制完成后要与内核一起编译，其与内核是不可分割的整体，在系统引导时与内核一起加载并驻留内存，如果需要对其进行升级或删除都需要重新编译内核。同时由于内核占用的内存是不可换页的，因此导致内核占用较大的内存空间。一般 Linux 系统中，硬盘、软驱、键盘、显示器等外设被视为系统中最基本的配置，因此这些设备的驱动程序都直接编译并链接到内核，在系统引导时自动安装并驻留内存。

另一类设备驱动程序采用可动态加载的模块。这类设备往往是不常使用的，其驱动程序代码在使用之前动态地加载到内存中，在设备使用完毕后即从内存中移去其代码。这样做的好处是可以节省内存空间，在需要使用时才将设备驱动程序加载到内核中，常驻内存的内核代码可以做得很小，同时当需要在系统中增加一个新设备的驱动程序或升级原来设备的驱动程序时，能够非常容易地实现。

对于嵌入式 Linux 系统来说，其设备驱动程序大都采用静态链接的方式，其主要原因是嵌入式系统的特殊需要决定的。

嵌入式 Linux 系统往往应用环境相对固定，系统都经过优化，尽可能地精简，其外设配置一般针对性强，大都不像通用系统那样配置，如外存空间往往比较小，同时对系统的稳定性、响应的实时性也有一定要求。动态加载内核模块虽然能够使得常驻内存的内核做得比较小，但也是有代价的。其常驻内存的核心部分要增加一个加载函数来完成内核模块装入内存后的重定位以及驱动程序的寻址工作，同时加载内核模块从磁盘等外存上读入代码会影响访问速度。嵌入式 Linux 系统大都不能够像桌面 Linux 那样灵活地使用 insmod/rmmod 加载卸载设备驱动程序。从嵌入式系统的整体性能考虑，采用静态链接模块能够使得整个系统的性能得到提高。

目前许多广泛应用的嵌入式 Linux 系统都采用静态链接的设备驱动程序模块，如 μClinux 就不支持设备驱动模块动态加载。下面就以将设备驱动程序静态编译进内核的方法来介绍嵌入式 Linux 系统设备驱动程序的具体构建方法。

3 嵌入式 Linux 系统设备驱动程序实例

下面以 μClinux 为例，编写一个仅供实验用的字符设备 test 来说明有关的技术方法。

3.1 设备驱动程序编写

编写设备驱动程序的主要工作就是编写子函数，并填充 file_operations 的各个域，以下为驱动程序 test.c 的主要内容。

函数 read_tes()是为 read 调用准备的。当调用 read 时，read_test()被调用，它把用户的缓冲区全部写 1。函数中的 buf 是 read 调用的一个参数，是用户进程空间的一个地址。但是在 read_test 被调用时，系统进入核心态，所以不能使用 buf 这个地址，必须用__put_user()，这是 kernel 提供的一个函数，用于向用户传送数据。

```
unsigned int test_major =0;
static int read_test(struct inode * node,struct file
* file,char  * buf,int count)
{int left;
if(verify_area(VERIFY_WRITE,buf,count) =
   = -EFAULT )return -EFAULT;
for(left = count ; left > 0 ; left--)
{__put_user(1,buf,1);
buf++ ;}
return count;}
```

以下是驱动程序下半部分的其他几个函数。

```
static int write_tibet(struct inode * inode,struct file * file,
const char * buf,int count)
```

```
{return count;
}
static int open_tibet(struct inode * inode,struct file * file )
{MOD_INC_USE_COUNT;
return 0;
} static void release_tibet(struct inode * inode,struct file *
file )
{
MOD_DEC_USE_COUNT;
}
```

以下为 file_operations 结构提供函数指针几个函数写作空操作。

```
struct file_operations test_fops = {NULL, read_
test,write_test,NULL,
    NULL, NULL, NULL, open_test, release_
    test, NULL, NULL,
/* nothing more, fill with NULLs */};
```

3.2 编译驱动程序到内核中

在设备驱动程序的主体 test.c 写好后，就要把驱动程序嵌入内核。将其编译进内核的基本步骤如下：

(1)改动 test.c 源带代码

第一步，将原来的：

```
#include
#include
char kernel_version[] = UTS_RELEASE;
```

改动为：

```
#ifdef MODULE
#include
#include
char kernel_version[] = UTS_RELEASE;
#else
#define MOD_INC_USE_COUNT
#define MOD_DEC_USE_COUNT
#endif
```

第二步，新建函数 int init_test(void) 将设备注册写在此处：

```
result = register_chrdev (254," test", &test_
fops);
```

(2)将 test.c 复制到/uclinux/linux/drivers/char 目录下，并且在/uclinux/linux/drivers/char 目录下 mem.c 中，int chr_dev_init( )函数中增加如下代码：

```
#ifdef CONFIG_TESTDRIVE
init_test();
#endif
```

(3)在/uclinux/linux/drivers/char 目录下 Makefile 中增加如下代码：

```
ifeq( $ (CONFIG_TESTDRIVE),y)
L_OBJS+=test.o
Endif
```

(4)在/uclinux/linux/arch/m68knommu 目录下 config.in 中字符设备段里增加如下代码：

```
bool ' support for testdrive ' CONFIG_
TESTDRIVE y
```

(5)运行 make menuconfig(在 menuconfig 的字符设备选项里可以看见刚刚添加的'support for testdrive'选项，并且已经被选中)；make dep；make linux；make linux.text；make linux.data；cat linux.text linux.data > linux.bin.

(6)在 /uclinux/romdisk/romdisk/dev/目录下创建设备：

```
mknod test.c 254 0
```

并且在/uclinux/appsrc/下运行 make，生成新的 Romdisk.s19 文件。

至此，在 μClinux 中增加设备驱动程序的工作可以说是完成了，只要将新的 linux.bin 与 Romdisk.s19 烧入目标板中，就可以使用自己的新设备 test 了。

结束语 随着嵌入式系统产品的技术和应用的迅速发展，需要为自己的嵌入式 Linux 系统添加设备驱动的需求也越来越普遍。本文介绍的嵌入式 Linux 系统中设备驱动程序构建的基本方法，为编写嵌入式系统中设备驱动程序提供了一种思路和参考。文中给出的实例仅仅是一个演示程序，真正实用的设备驱动程序还有许多问题需要解决，如中断，DMA，I/O port 等问题。

参 考 文 献

1 邹思轶. 嵌入式 Linux 设计与应用[M]. 北京：清华大学出版社，2002

2 蒋敬，徐志伟. 操作系统—原理·技术与编程[M]. 北京：机械工业出版社，2004

3 郑伟，王钦若. Linux 内核空间设备驱动程序的开发[J]. 微计算机信息，2003，19(12)：85～97

4 Writing a Linux Driver[EB/OL]. http://magguang.blog.sohu.com，2006