

Linux内核驱动

Linux内核驱动原理

- 1 Linux驱动程序概述
- 2 Linux驱动程序的相关知识
- 3 驱动程序的结构

Linux驱动程序概述

- Linux驱动程序概念
- Linux驱动程序分类

Linux驱动程序概念

在Linux中，系统调用是内核（kernel）和应用程序之间的接口，而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，这样在应用程序看来，硬件设备只是一个设备文件，应用程序可以象操作普通文件一样对硬件设备进行操作。

Linux中将所有设备视为文件。

Linux驱动程序分类

在Linux操作系统的驱动程序分成三种类型：

- 字符设备（char device）
- 块设备（block device）
- 网络设备（net device）

Linux驱动程序分类

字符设备字符设备特殊文件进行I/O操作不经过操作系统的缓冲区，进行I/O操作时每次只传输一个字符。

典型的字符设备如：鼠标、键盘、串口等。字符设备可以通过字符设备文件来访问。

Linux驱动程序分类

块设备使用随机访问的方式传输数据，并且数据总是具有固定大小的块。

为了提高数据传输效率，块设备驱动程序内部采用块缓冲技术。典型的块设备如：光盘、硬盘、软盘等。块设备可以通过网络块文件来访问。

Linux驱动程序分类

网络设备是一种特殊的设备，与字符设备和块设备不同，网络设备**并没有文件系统的节点**，也就是说网络设备没有设备文件。

网络设备最重要的特点是没有文件系统的节点。

在Linux的网络系统中，使用**UNIX的socket**机制。系统与驱动程序之间通过专有的数据结构进行访问。系统内部支持数据的收发，对网络设备的使用需要通过**socket**，而不是文件系统的节点。

Linux驱动程序的相关知识

- 内核模块的作用
- 模块的编程结构和使用
- 内核模块编写的注意事项
- Linux设备文件
- Linux对设备文件的操作
- 设备文件相关的数据结构

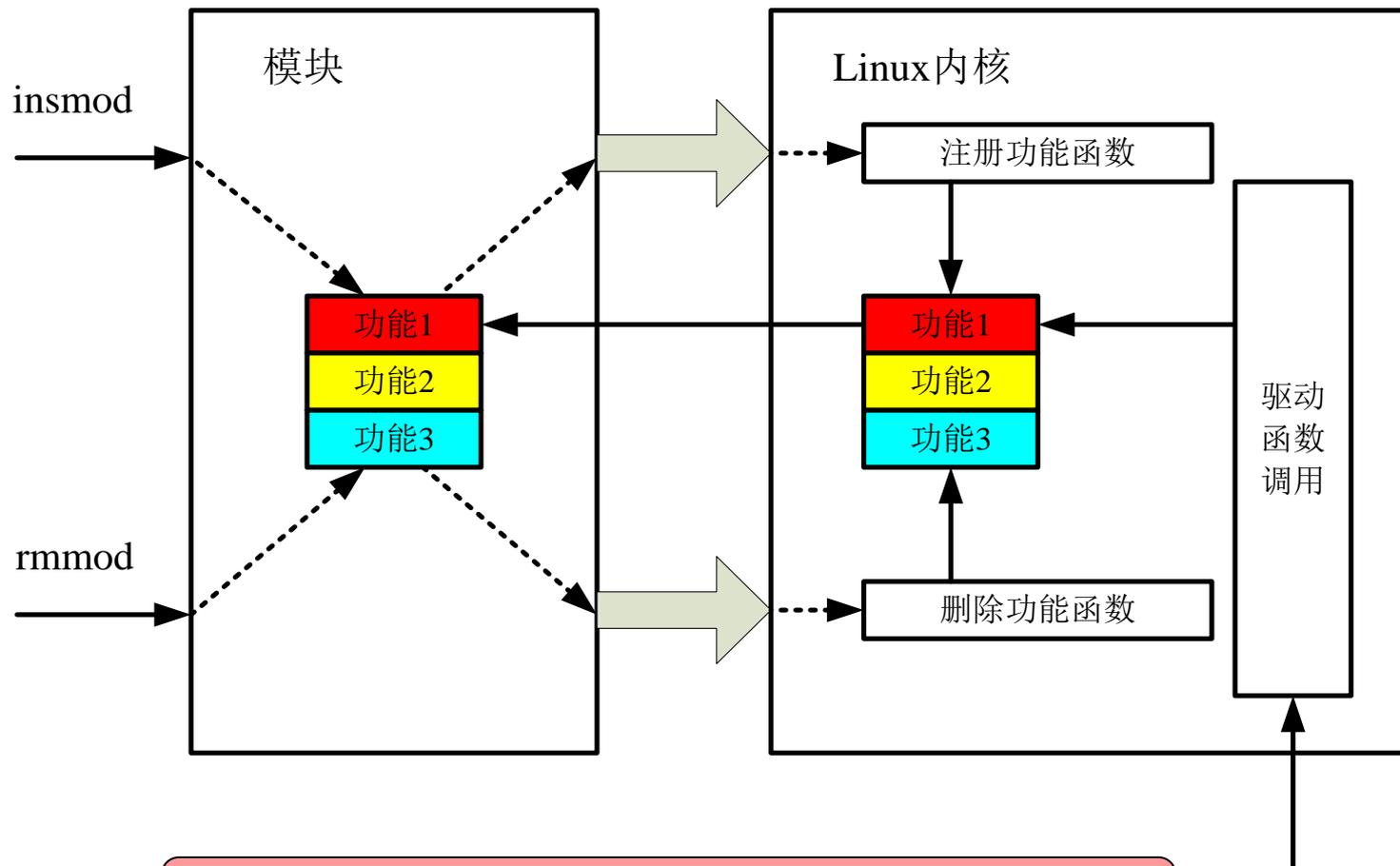
内核模块的作用

Linux设备驱动属于内核的一部分，它可以使用两种方式被编译和加载：

- 1、直接编译进Linux内核，随同Linux启动时加载,随时可以使用驱动程序；

- 2、编译成一个可加载和删除的模块，使用insmod加载，rmmod删除。

内核模块的作用



模块插入 (rmmod) 后，运行在内核空间

模块的编程结构和使用

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/sched.h>
/* 函数声明 */
static int module_init(void);
static void module_cleanup(void);
/* 注册模块函数 */
module_init(module_init);
module_exit(module_cleanup);
```

声明模块的初始化和卸载函数

使用两个宏注册模块的初始化和卸载函数

模块的编程结构和使用

```
/* 模块初始化 */
static int module_init(void)
{
    printk(KERN_CRIT "module init fails\n");
    return 0;
}
/* 模块退出 */
static void module_cleanup(void)
{
    printk(KERN_CRIT "module init exif\n");
    return;
}
```

Linux模块中的函数声明为static，它们不会被直接调用。使用注册的方式调用。

内核模块编写的注意事项

内存分配函数：

模块运行在内核空间内，它并没有链接C语言标准的函数库。因此，内存分配不能使用C语言的库函数`malloc()`和`free()`，而需要使用内核空间的对应函数。这两个函数定义在`slab.h`中。

```
#include<linux/slab.h>
```

```
void *kmalloc(size_t size, int flags);
```

比`malloc()`多一个参数。

```
void kfree(const void *);
```

内核模块编写的注意事项

由于不能使用C语言库，在内核中打印调试信息也不能使用printf。在打印调试信息的功能上，内核应该使用内核的打印函数printk。

printk在kernel/printk.c中定义和实现：

```
int printk(const char *fmt, ...)
```

```
printk(KERN_DEBUG "priority = 7\n");  
printk(KERN_INFO "priority = 6\n");  
printk(KERN_NOTICE "priority = 5\n");  
printk(KERN_WARNING "priority = 4\n");  
printk(KERN_ERR "priority = 3\n");  
printk(KERN_CRIT "priority = 2\n");  
printk(KERN_ALERT "priority = 1\n");  
printk(KERN_EMERG "priority = 0\n");
```

GCC中的两个字符串可以用这种方式实现连接。

Linux设备文件

设备文件的属性主要由三部分信息组成：

- 第一部分是文件的类型，值可能是c/b
(c代表字符设备，b代表块设备)
- 第二部分是设备的“主设备号”
- 第三部分是设备的“次设备号”

在Linux控制台上输入：

```
# ls -l /dev
```

```
crw----- 1 root  root  5, 1 3月 28 21:32 /dev/console
crw----- 1 root  root 14, 3 2003-01-30 /dev/dsp
crw----- 1 root  root 29, 0 2003-01-30 /dev/fb0
brw-rw---- 1 root  floppy  2, 0 2003-01-30 /dev/fd0
brw-rw---- 1 root  disk    3, 1 2003-01-30 /dev/hda1
crw-rw-rw- 1 root  root    1, 3 2003-01-30 /dev/null
crw--w---- 1 root  root    4, 0 2003-01-30 /dev/tty0
```

主设备号

次设备号

Linux对设备文件的操作

打开文件:

```
int open( const char * pathname, int flags);
```

关闭文件:

```
int close(int fd);
```

读:

```
ssize_t read(int fd,void * buf ,size_t count);
```

写:

```
ssize_t write(int fd,const void * buf,size_t count);
```

移动:

```
loff_t lseek(int fildes,loff_t offset ,int whence);
```

端口控制:

```
int ioctl(int fd ,unsigned int cmd,...);
```

ioctl中的cmd
用于自定义命令。

Linux对设备文件的操作

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
```

包含文件操作所需要的头文件。

```
main()
```

```
{
```

```
    int fd,size;
```

```
    char s[]="File operation\n";
```

```
    char buffer[80];
```

```
    fd=open("/tmp/tempfile",O_WRONLY|O_CREAT);
```

打开文件。

```
    write(fd,s,sizeof(s));
```

```
    close(fd);
```

关闭文件。

```
    fd=open("/tmp/tempfile",O_RDONLY);
```

```
    size=read(fd,buffer,sizeof(buffer));
```

```
    close(fd);
```

```
    printf("Display:%s",buffer);
```

设备文件相关的数据结构

struct file_operations

在Linux的include/linux目录中的fs.h文件中定义。

open/release:

在文件I/O的打开和关闭时定义，open函数也可以不定义，此时，只要文件系统中存在设备文件节点，文件就可以被成功打开。

read/write:

用于缓冲区和设备文件之间读写操作；

ioctl:

定义各种操作，参数cmd是自定义的参数，由驱动程序解释；

mmap:

将设备内容映射到内存；

内核空间和用户空间的交互

用户空间和内核空间的相互访问：

```
int access_ok(int type,unsigned long addr,unsigned long size);
unsigned long copy_to_user(void *to,
                           const void *from,
                           unsigned long len);
unsigned long copy_from_user(void *to,
                             const void *from,
                             unsigned long len);
```

将文件映射到内存空间中：

```
void *mmap(void *start,
           size_t length,
           int prot,
           int flags,
           int fd,
           loff_t offsize);
```

设备文件相关的数据结构

`struct block_device_operations`

在Linux的include/linux目录中的fs.h文件中定义。

`open/release:`

在文件I/O的打开和关闭时的操作。

`ioctl:`

定义各种操作，参数cmd是自定义的参数，由驱动程序解释；

`check_media_change:`

检查媒体（介质）的改变

`revalidate:`

重新使得媒体有效

块设备的操作中没有字符操作的read/write成员。

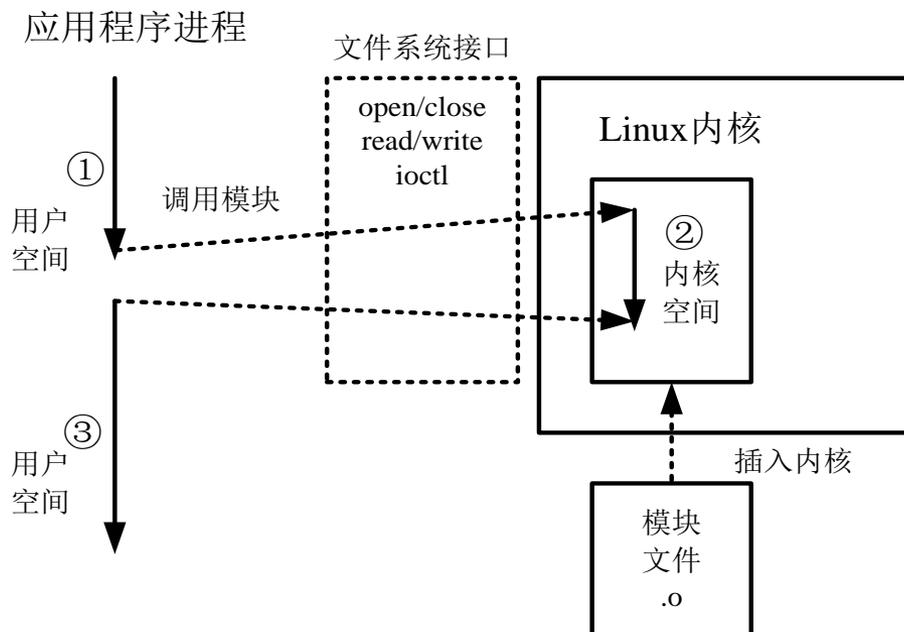
驱动程序的结构

- 字符驱动程序框架
- 块设备的驱动程序框架
- 网络设备的驱动程序框架

字符驱动程序框架

在struct file_operations中注册的函数，应该完成相应的操作：

- ❑ module_init中注册的函数执行设备文件的注册；
- ❑ module_exit中注册的函数执行设备文件的卸载。
- ❑ open中增加引用计数；close中减少引用计数
- ❑ write/read中执行读写操作，内容的传递通过缓冲区的指针buf；
- ❑ ioctl中执行驱动程序自定义的命令，根据cmd选择要执行的命令。



字符驱动程序框架

在Linux系统中，对中断的处理是属于系统核心部分，因此如果设备与系统之间以中断方式进行数据交换，就必须把该设备的驱动程序作为系统核心的一部分。

设备驱动程序通过调用`request_irq`函数来申请中断，通过`free_irq`来释放中断。它们的定义如下：

```
int request_irq(unsigned int irq,  
               irqreturn_t (*handler)(int, void *, struct pt_regs *),  
               unsigned long irq_flags,  
               const char * devname, void *dev_id);  
void free_irq(unsigned int irq, void *dev_id);
```

注册的中断号需要与内核移植的中的中断号相对应。

块设备的驱动程序框架

实现 **block_device_operations** 数据结构

块设备驱动程序的基本操作和字符设备类似。块设备的操作中没有读（**read**）和写（**write**）的函数指针。

块设备驱动具有 **check_media_change** 和 **revalidte**，前者用于判断介质是否改变，后者用于禁止改变时的操作。

网络设备的驱动程序框架

Linux下的网络设备驱动程序不需要使用文件系统的节点。网络驱动程序在Linux源代码下的`/include/linux/netdevice.h`中定义相关内容。

核心数据结构：`struct net_device`

模块初始化和卸载的时候分配和释放网络设备的数据结构（`struct net_device`）。

网络驱动程序没有文件系统的节点

课程结束