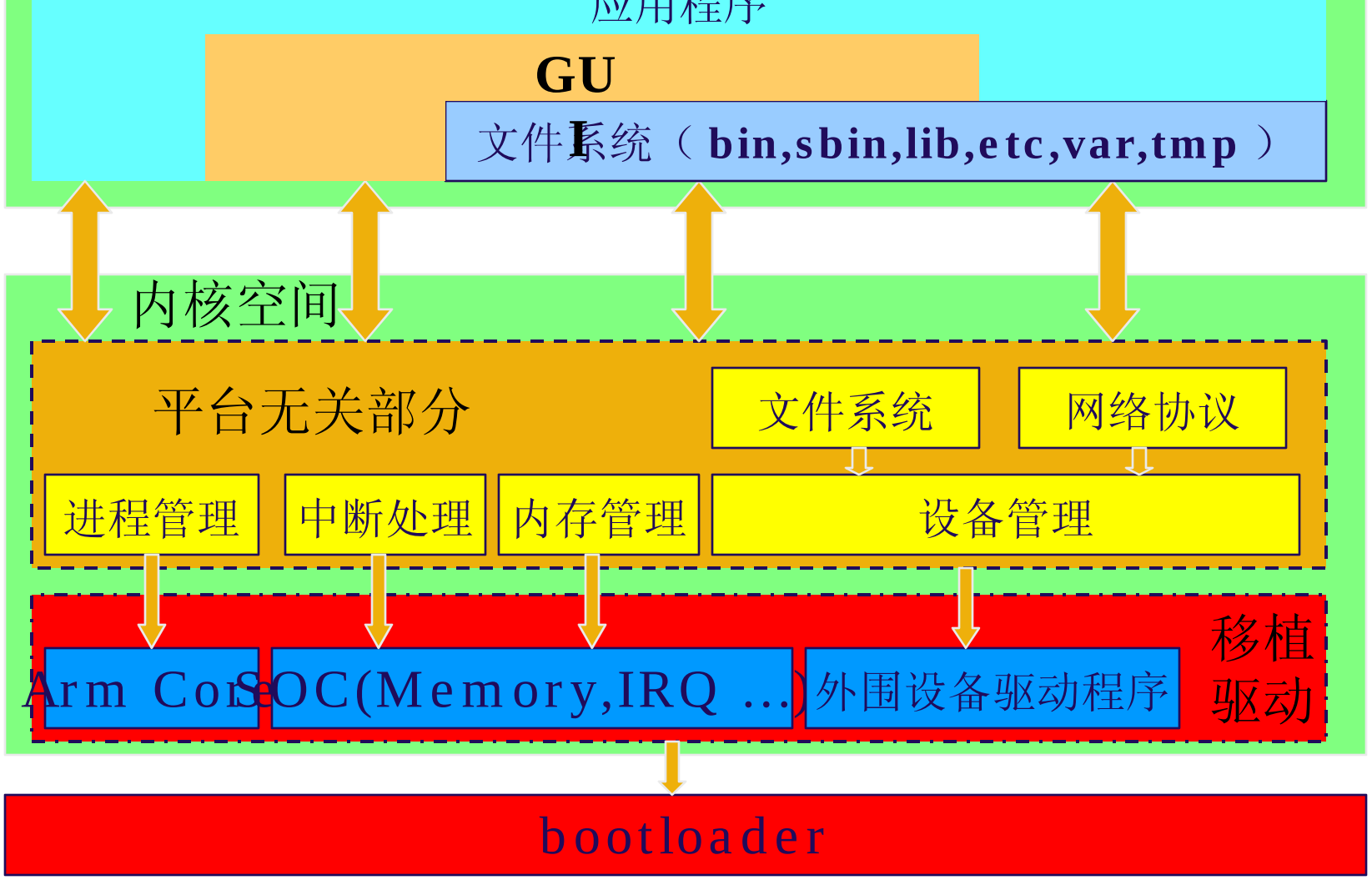
Linux 驱动程序开发

姓名：　　周余　（nack）
实验室：　　南大蔡冠深软件研发中心
研究方向：　嵌入式系统、图像处理
EMAIL:　　nackzhou@sw-linux.com

Review


用户空间
应用程序
GUI
文件系统（bin,sbin,lib,etc,var,tmp）
内核空间
平台无关部分
文件系统
网络协议
进程管理
中断处理
内存管理
设备管理
Arm Core
SOC(Memory,IRQ ...)
外围设备驱动程序
移植驱动
bootloader

Review

嵌入式 **Linux** 系统开发的主要工作

1. 建立交叉编译环境

2. 引导装载程序（ BootLoader ）编写或移植

3. Linux 内核的移植与裁减

4. 驱动程序的开发

5. 文件系统 的建立与移植

6. 图形用户界面（ GUI ）的移植

7. 应用程序的移植

报告主要内容

1. **Linux** 驱动程序基本概念

3. 字符型驱动程序

3. 时间流和中断

7. 块设备驱动程序

9. 网络驱动程序

6. 其他驱动程序体系

Linux 驱动程序基本概念

设备驱动程序的作用

系统调用是操作系统内核和应用程序之间的接口，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。设备驱动程序是内核的一部分，它完成以下的功能：

a. 对设备初始化和释放；

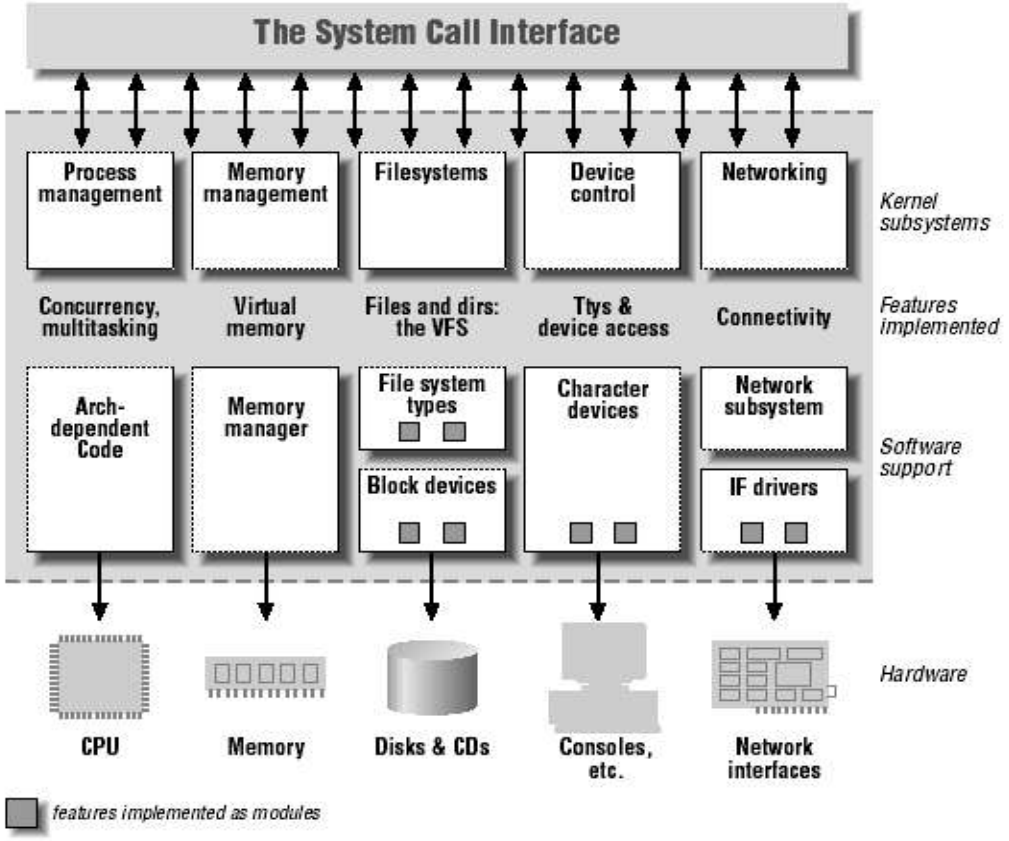
b. 把数据从内核传送到硬件和从硬件读取数据；

c. 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；

d. 检测和处理设备出现的错误；

Linux 驱动程序基本概念

系统调用、内核、驱动程序的关系



Linux 驱动程序基本概念

主要驱动类型

字符设备 (c)

块设备 (b)

网络设备 （ifconfig）

字符设备和块设备的主要区别是：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缓冲区

主设备号，从设备号

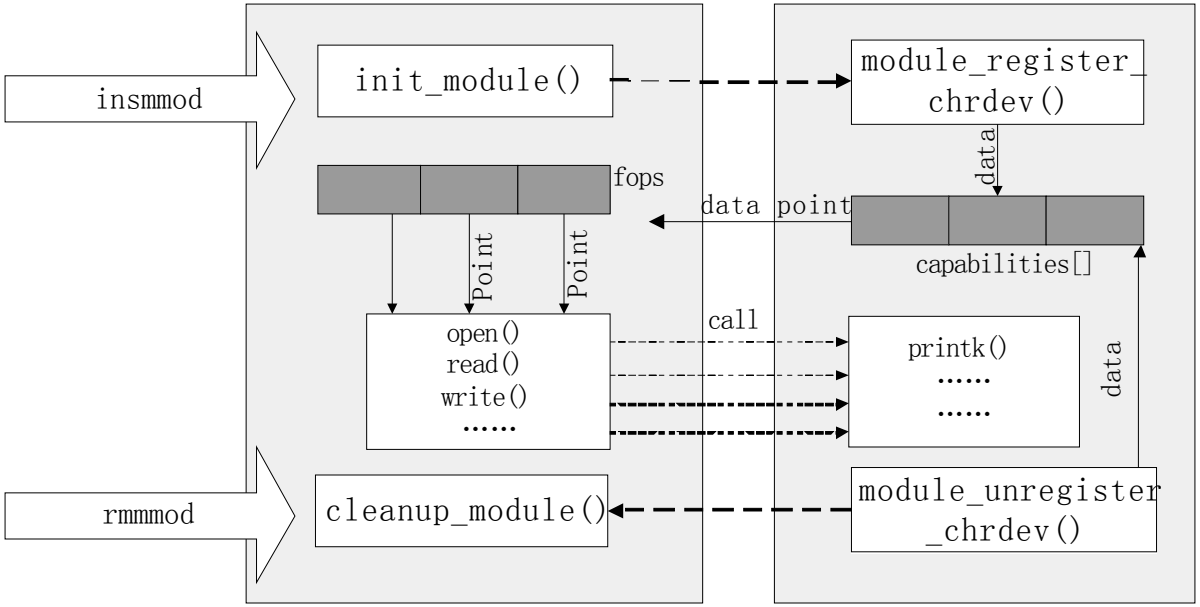
用 ll 命令可以观察，设备号规范在 docmention/devices.txt 中

Linux 驱动程序基本概念

模块

对于每一个内核模块来说，必定包含下面两个函数： int init_module ()： 这个函数在插入内核时启动，在内核中注册一定的功能函数，或者用他的代码代替内和中某些函数的内容。

int cleanup_module()：当内核模块卸载时调用，它能将模块从内核中清除。(#include <linux/module.h>)



编译 :Makefile(-D__KERNEL__ -DMODULE)

在包含 module.h 前定义 __NO_VERSION__

使用模块： insmod,lsmod,rmmod,depmod

字符型驱动程序

驱动程序的初始化、卸载

设备驱动程序所提供的入口点，在设备驱动程序初始化的时候向系统进行登记，以便系统在适当的时候调用。LINUX 系统里，通过调用 register_chrdev 向系统注册字符型设备驱动程序。register_chrdev 定义为：

```
#include <linux/fs.h>
#include <linux/errno.h>
int register_chrdev(unsigned int major, const char *name,
                    struct file_operations *fops);
```

其中，major 是为设备驱动程序向系统申请的主设备号，如果为 0 则系统为此驱动程序动态地分配一个主设备号。name 是设备名。fops 就是对各个调用的入口点的说明。此函数返回 0 表示成功。返回 -EINVAL 表示申请的主设备号非法，一般来说是主设备号大于系统所允许的最大设备号。返回 -EBUSY 表示所申请的主设备号正在被其它设备驱动程序使用。如果是动态分配主设备号成功，此函数将返回所分配的主设备号。如果 register_chrdev 操作成功，设备名就会出现在 /proc/devices 文件里。

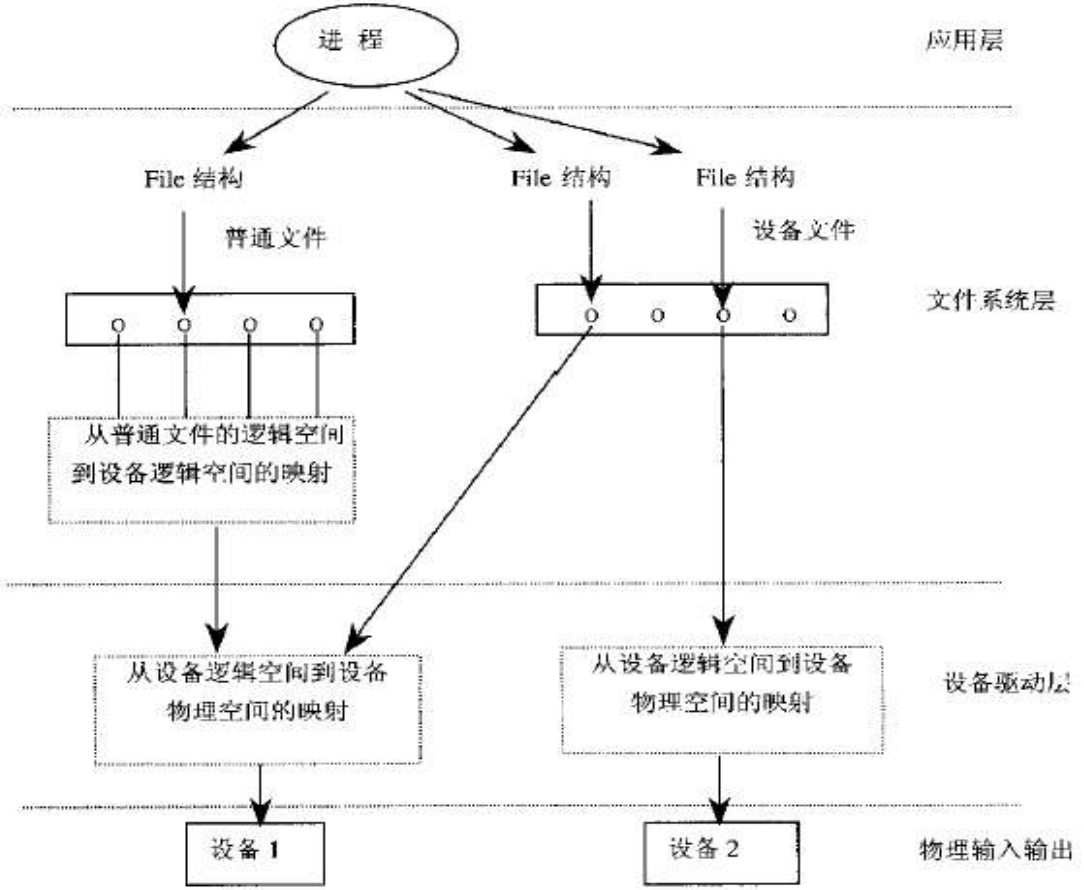
初始化部分一般还负责给设备驱动程序申请系统资源，包括内存、中断、时钟、I/O 端口等，这些资源也可以在 open 子程序或别的地方申请。在这些资源不用的时候，应该释放它们，以利于资源的共享。

字符型驱动程序

驱动程序的功能实现 -- 两个重要的数据结构：

file

file_operations



字符型驱动程序

Linux 中的 I/O 子系统向内核中的其他部分提供了一个统一的标准设备接口，这是通过 include/linux/fs.h 中的数据结构 file_operations 来完成的

(1) lseek，移动文件指针的位置，显然只能用于可以随机存取的设备。

(2) read，进行读操作，参数 buf 为存放读取结果的缓冲区，count 为所要读取的数据长度。返回值为负表示读取操作发生错误，否则返回实际读取的字节数。对于字符型，要求读取的字节数和返回的实际读取字节数都必须是 inode->i_blksize 的的倍数。

(3) write，进行写操作，与 read 类似。

(4) readdir，取得下一个目录入口点，只有与文件系统相关的设备驱动程序才使用。

(5) select，进行选择操作，如果驱动程序没有提供 select 入口，select 操作将会认为设备已经准备好进行任何的 I/O 操作。

(6) ioctl，进行读、写以外的其它操作，参数 cmd 为自定义的的命令。

(7) mmap，用于把设备的内容映射到地址空间，一般只有块设备驱动程序使用。

(8) open，打开设备准备进行 I/O 操作。返回 0 表示打开成功，返回负数表示失败。如果驱动程序没有提供 open 入口，则只要 /dev/driver 文件存在就认为打开成功。

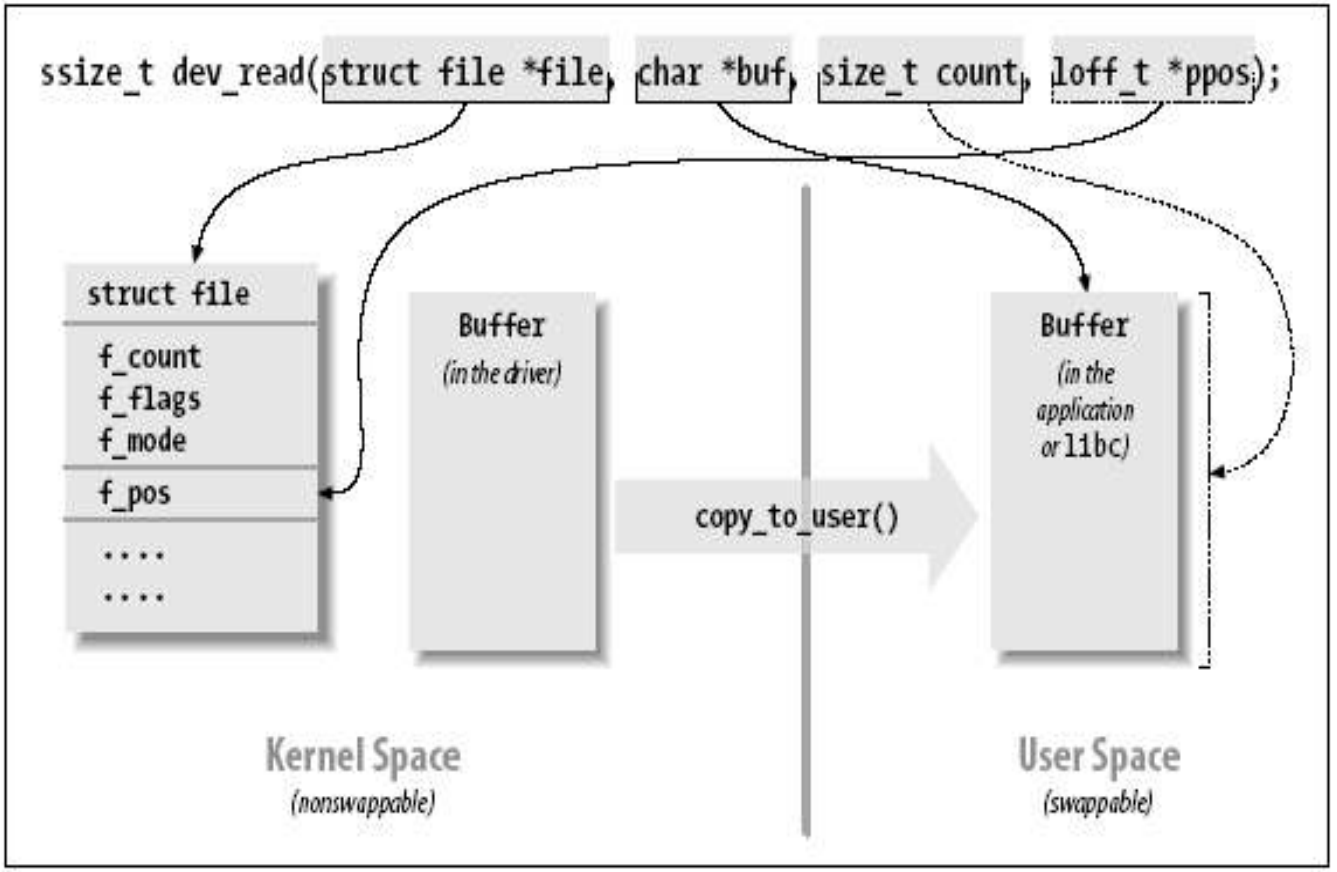
(9) release，即 close 操作。

1.open -> read -> close (open -> slect -> read -> close)

2.open-> wite -> close

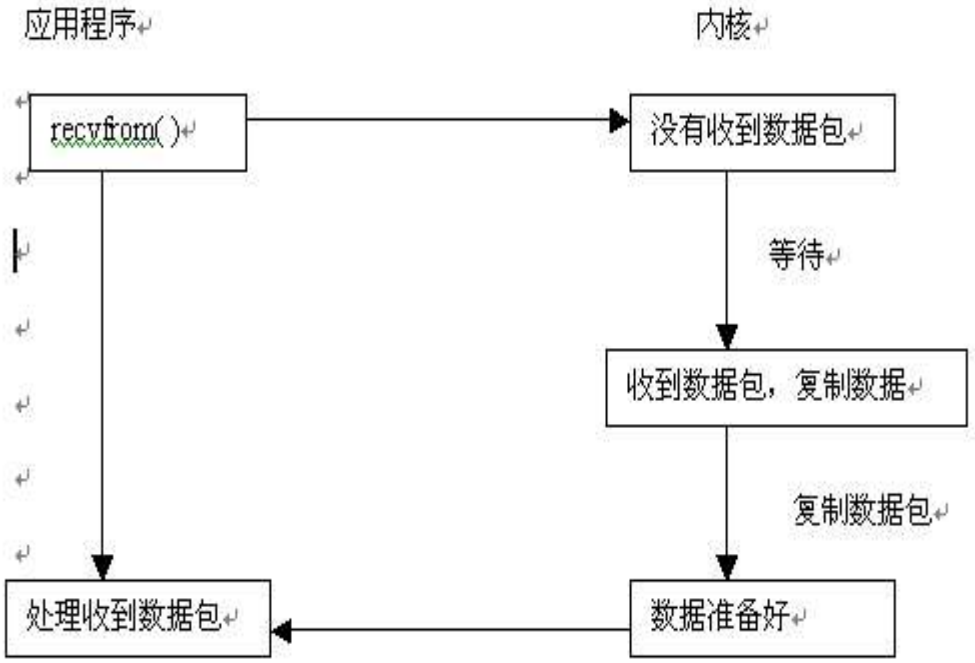
字符型驱动程序

Read 的基本方法



字符型驱动程序

阻塞 I/O 模式与非阻塞 I/O 模式
当执行读写 I/O 的系统调用时，执行或等待 I/O 操作的过程中进程阻塞，直到 I/O 操作完成，调用才结束，唤醒进程继续向下进行。缺省情况下套接口的读写操作就是阻塞 I/O 方式。阻塞 I/O 模式的流程图如下。



字符型驱动程序

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。
开关中断： disable_irq(int irq);enable_irq(int irq);
中断处理注意的方面，中断不处于进程上下文，处于中断模式
1. 不允许访问用户空间，没有到达与进程关联的用户空间路径
2.current 指针在中断模式下是无效的
3. 不能执行睡眠。不可以调用 schedule 或者 sleep_on;
   kmalloc(...,GFP_KERNEL)
4. 中断处理函数不能太长

作为系统核心的一部分，设备驱动程序在申请和释放内存时不是调用 malloc 和 free ，而代之以调用 kmalloc 和 kfree
高 1G(内核空间) 的内存分配
物理区 ||8M 隔离 ||vmalloc 区 ||8K 隔离 ||4M 的高端映射区 || 固定映射区 ||128K 保留区
kmalloc,kfree -- 最大能分配 128k 内存
vmalloc,vfree -- 分配虚礼内存空间，不一定连续 ;
get_free_page --
ioremap,iounmap

字符型驱动程序

申请与释放 IO:check_region,request_region,release_region

在申请了 I/O 端口之后，就可以如下几个函数来访问 I/O 端口：
inb,outb,inw,outw…

DMA

在设备驱动程序里，一般都需要用到计时机制。在 LINUX 系统中，时钟是由
系统接管，设备驱动程序可以向系统申请时钟。与时钟有关的系统调用有：

```
#include <asm/param.h>
#include <linux/timer.h>
void add_timer(struct timer_list * timer);
int  del_timer(struct timer_list * timer);
```

字符型驱动程序

例子 **SPI < -- > ads7846**

1. 仔细了解 ads7846, cpu 的 SPI 的文档

触摸屏坐标读取操作（中断方式，笔中断及 SPI 中断），下面的操作时序从用户点屏开始。

用户点屏 --> 笔中断到 --> 发命令读 x 坐标 --> SPI 中断（数据传输结束）到 --> 读取 x 坐标 --> 发命令读 y 坐标 --> SPI 中断到 --> 读取 y 坐标 --> 放入循环 buffer(--> 唤醒用户进程) 。

根据上目所述的坐标读取方式（中断）及顺序，驱动程序应包括两个中断服务例程，它们分别处理笔中断和 SPI 中断。

为了使中断服务例程合理，在 SPI 中断服务例程中采用了有限状态机的方式。具体地讲，就是当笔中断到来时，在笔中断服务例程中关闭笔中断，并打开 SPI 中断，置状态机状态为初态，并发命令读取某一坐标值，这样，当有数据到达 SPI 相应寄存器时，就导致进入 SPI 中断服务例程，在其中就可以读取任一坐标值了，读到就退出，并以一个计数器来计所读取的坐标组数，达四次以后，就可将这四组值送入滤波器滤波，然后唤醒用户进程来读取处理过的坐标值。

时间流和中断

Linux 的时间系统

一般 PC 机中有两个时钟，分别是 RTC （real time clock）时钟和 OS 时钟。OS 时钟产生于主板上的定时 / 计数芯片，其基本单位就是计数芯片的计数周期。在开机时通过 RTC 来初始化芯片。定时 / 计数芯片的每一个输出脉冲周期叫做一个“时钟滴答”，计算机中的时间就是以“滴答”为单位的 ，每一次滴答时间会加一。根据当前的滴答数就可以得到秒等其他单位。

计时器大概每 10ms 向 CPU 送入一个脉冲，就可以触发一个时钟中断。系统利用时钟中断维持系统时间，促使进程和环境发生切换，进行记帐等工作以确定动态优先级等等。

Linux 用全局变量 jiffies 表示系统自启动以来经过的时钟滴答数。

时间流和中断

表示系统当前时间的内核数据结构

1. 全局变量 jiffies

这是一个 32 位的无符号整数，用来表示自内核上一次启动以来的时钟滴答次数。每发生一次时钟滴答，内核的时钟中断处理函数 timer_interrupt（）都要将该全局变量 jiffies 加 1 。该变量定义在 kernel/timer.c 源文件中，如下所示：

```
unsigned long volatile jiffies;
```

2. 全局变量 xtime

它是一个 timeval 结构类型的变量，用来表示当前时间距 UNIX 时间基准 1970 － 01 － 01　00 ： 00 ： 00 的相对秒数值。结构 timeval 是 Linux 内核表示时间的一种格式（ Linux 内核对时间的表示有多种格式，每种格式都有不同的时间精度），其时间精度是微秒。该结构是内核表示时间时最常用的一种格式，它定义在头文件 include/linux/time.h 中，如下所示：

```
struct timeval {
time_t tv_sec; /* seconds */
suseconds_t tv_usec; /* microseconds */
};
```

3. 全局变量 sys_tz

它是一个 timezone 结构类型的全局变量，表示系统当前的时区信息。结构类型 timezone 定义在 include/linux/time.h 头文件中

时间流和中断

中断
arm 的中断向量表可以放在地址 0 开始，也可以指定为 0xFFFF0000 开始。

中断发生后的执行过程是：
R14_irq = 下一条指令的地址
Spsr_irq = CPSR
Cpsr[4:0] = 0b10010 ；表示进入 irq
Cpsr[7] = 1 ；关闭 irq，防止中断嵌套
If high vector table then
Pc = 0xFFFF0018
Else
Pc = 0x00000018

Linux 中的中断向量表是在 void __init trap_init(void) //
arch/arm/kernle/trap.c
中安装的，其中调用汇编函数 __trap_init((void *)vectors_base());
宏 vectors_base() 决定项量表是从 0 开始还是从 0xFFFF0000 开始。

时间流和中断

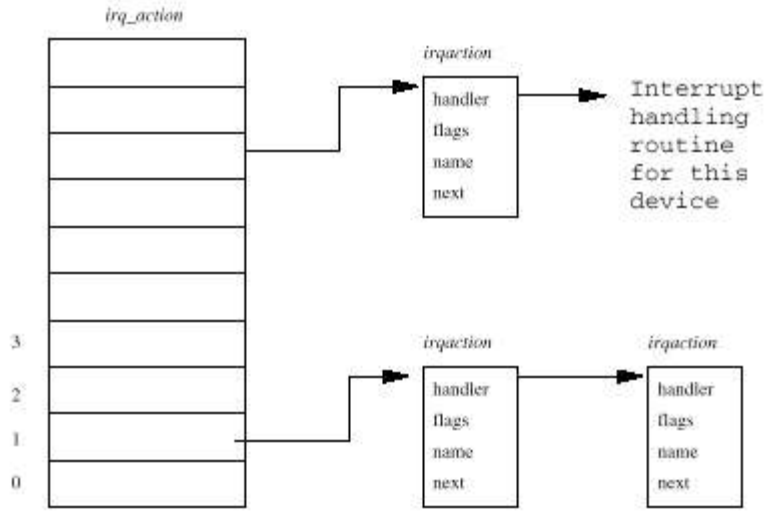
中断

1. 保存中断现场
2. 用宏 get_irqnr_and_base 取得中断号
3. 代码 do_IRQ，此时 R0 为中断号， R1 指向堆栈中保存的寄存器起始地址。
4. do { //逐个处理中断处理 handler 链表中的每个函数

```
status |= action->flags;
action->handler(irq, action->dev_id, regs);
action = action->next;
} while (action);
```

5. if (softirq_pending(cpu))

```
//执行软中断处理过程
   do_softirq();
```



时间流和中断

软中断概况

软中断是利用硬件中断的概念，用软件方式进行模拟，实现宏观上的异步执行效果。

bottom half

在 Linux 内核中， bottom half 通常用“ bh” 表示，最初用于在特权级较低的上下文中完成中断服务的非关键耗时动作，现在也用于一切可在低优先级的上下文中执行的异步动作。

task queue

显而易见，原始的 bottom half 机制有几个很大的局限，最重要的一个就是个数限制在 32 个以内，随着系统硬件越来越多，软中断的应用范围越来越大，这个数目显然是不够用的，而且，每个 bottom half 上只能挂接一个函数，也是不够用的。因此，在 2.0.x 内核里，已经在用 task queue （任务队列）的办法对其进行了扩充

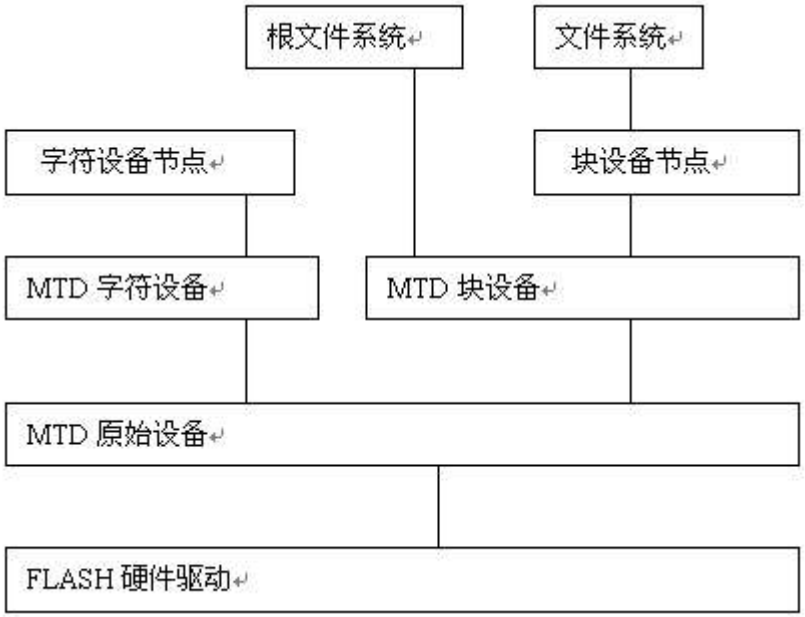
tasklet

之所以引入 tasklet ，最主要的考虑是为了更好的支持 SMP ，提高 SMP 多个 CPU 的利用率：不同的 tasklet 可以同时运行于不同的 CPU 上。

块设备驱动

MTD

在本系统中采用的两块 Flash 都是 NOR 型的 Flash 。通常市场上存在 NOR 型 Flash 和 NAND 型 Flash，这两种类型都有各自的优点，其中前者的特点是芯片内执行 (XIP， eXecute In Place)，这样应用程序可以直接在 flash 内运行，不必再把代码读到系统 RAM 中。



块设备驱动

制作 RAMdisk 具体步骤包括：

将你想要放到文件系统中的文件以适当的目录结构准备好。

将一块内存清零（这主要是为了将 RAMdisk 尽可能的压缩，因为如果未用空间是随机数的话会浪费最终产生 RAMdisk 的空间）。比如： dd if=/dev/zero of=/dev/ram bs=1k count=2048 。

在该内存空间上制作一个文件系统。比如： mke2fs -vm0 /dev/ram 2048 。

将其挂载（ mount ）到一个挂载点。比如： mount -t ext2 /dev/ram /mnt/ramdisk 。

将准备好的文件复制到它上面。

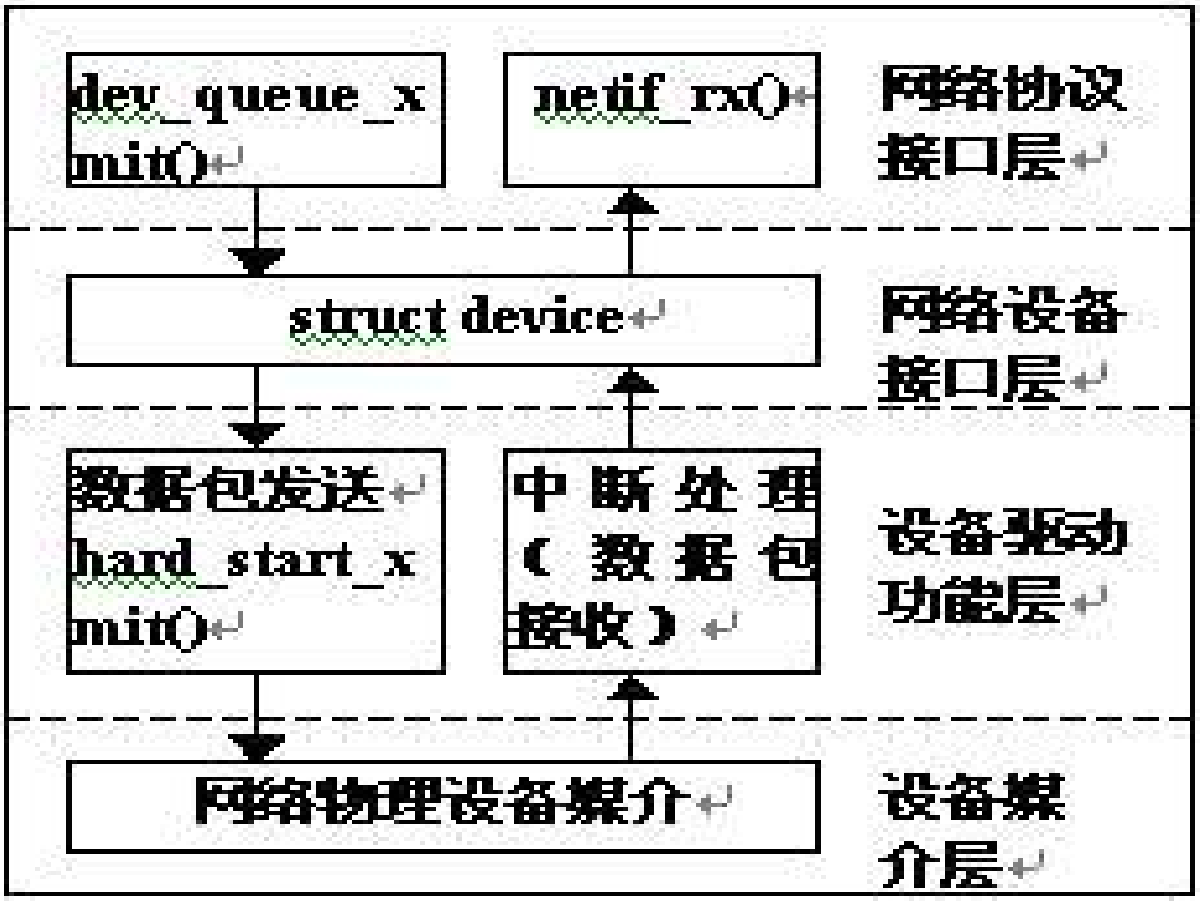
取消挂载（ umount ）。比如： umount /mnt/ramdisk 。

将其压缩制成最终的 RAMdisk 。比如： dd if=/dev/ram bs=1k count=2048 | gzip -v9 ramdisk.gz 。

CONFIG_BLK_DEV_RAM 和 CONFIG_BLK_DEV_INITRD 在"make menuconfig."需要定义为‘Y’，如果只想支持 Ramdisk，不是作为根文件系统，则 CONFIG_BLK_DEV_RAM 需定义为’ Y’，CONFIG_BLK_DEV_INITRD 不选。

网络驱动程序

Linux 网络驱动程序体系结构



网络驱动程序

1. 初始化 (initialize)

驱动程序必须有一个初始化方法。在把驱动程序载入系统的时候会调用这个初始化程序。它做以下几方面的工作：检测设备，配置和初始化硬件，向系统申请资源。

2. 打开 (open)

open 这个方法在网络设备驱动程序里是网络设备被激活的时候被调用（即设备状态由 down-->up）。所以实际上很多在 initialize 中的工作可以放到这里来做。

3. 关闭 (stop)

close 方法做和 open 相反的工作。可以释放某些资源以减少系统负担。

4. 发送 (hard_start_xmit)

所有的网络设备驱动程序都必须有这个发送方法。在系统调用驱动程序的 xmit 时，发送的数据放在一个 sk_buff 结构中。一般的驱动程序把数据传给硬件发出去

5. 接收 (reception)

驱动程序并不存在一个接收方法。有数据收到应该是驱动程序来通知系统的。一般设备收到数据后都会产生一个中断，在中断处理程序中驱动程序申请一块 sk_buff(skb)，从硬件读出数据放置到申请好的缓冲区里。

其它驱动程序体系

PCMCIA

PCMCIA 已经相当成熟，其硬件设备和驱动程序都已经标准化，各种操作系统中都内置有标准的驱动程序。 Linux 下也有完整的软件包，用户可以从 http://pcmcia-cs.sourceforge.net 免费获得，源代码遵循 GPL 公共许可协议。该软件包提供了对于多种设备的支持，稍加改动就可以移植到嵌入式系统中。

Linux PCMCIA 子系统由三层构成，如图 1 所示。最底层是接口驱动层（ socket driver ），第二层是 PC 卡服务层（ Card Services ），最上层是 PC 卡客户层 (Card Client) 。一个特殊的 PC 卡客户是叫做驱动服务（ Driver Service ） ，提供一个接口给 PC 卡应用工具（ Cardmgr, Cardctrl 等）。每一层都通过标准 API 接口对为上层提供服务。最上层的应用软件也是通过应用层 API 函数实现对 PC Cards 的读写和操作。

USB

Bluetooth

…

Sound

Video

…