Linux 系统目前主要维护2.4和2.6两个内核版本，在http://www.kernel.org/ 网站上已经可以下载到最新的2.6内核linux-2.6.38.6，及最新的2.4内核linux-2.4.37.11。稳定版本号基本上是1~3月更 新一次，如：2.6.35至2.6.36，升级版本号每1~2周更新一次，如：2.6.35.1至2.6.35.2。 升级后的高版本内核并不完全兼容低版本内核，所以内核升级对于从事linux开发的技术人员来说影响很大，特别是对于那些刚刚从事linux开发的人员。

通 常，内核的升级对从事linux应用程序开发的人员来说影响较小，因为系统调用基本保持兼容，影响比较大的是驱动开发人员。每次内核的更新都可能导致许多 内核函数原型上的变化，其中既有内核本身提供的函数，也有硬件平台代码提供的函数，后者变化的更加频繁。这一点从许多经典书籍就可验证，当你按照手里的经 典著作，如：Alessandro的《linux设备驱动程序》，编写驱动时，发现并不能够成功的在你的linux平台上编译通过、或不能正常执行，原因 就在于你用的内核和书里的不一致。

本文从两个方面去解释这个问题，一方面是如何写好linux设备驱动，另一方面是如何应对不断升级的内核。

**如何写好Linux设备驱动**

Linux设备驱动是linux内核的一部分，是用来屏蔽硬件细节，为上层提供标准接口的一种技术手段。为了能够编写出质量比较高的驱动程序，要求工程师必须具备以下几个方面的知识：

●    **熟悉处理器的性能**

如：处理器的体系结构、汇编语言、工作模式、异常处理等。对于初学者来说，在还不熟悉驱动编写方法的情况下，可以先不把重心放在这一项上，因为可能因为它的枯燥、抽象而影响到你对设备驱动的兴趣。随着你不断地熟悉驱动的编写，你会很自然的意识到此项的重要性。

●    **掌握驱动目标的硬件工作原理及通讯协议**

如：串口控制器、显卡控制器、硬件编解码、存储卡控制器、I2C通讯、SPI通讯、USB通讯、SDIO通讯、I2S通讯、PCI通讯等。编写设备驱动的前提就是需要了解设备的操作方法，所以这些内容的重要程度不言而喻。但不是说要把所有设备的操作方法都熟悉了以后才可以写驱动，你只需要了解你要驱动的硬件就可以了。

●    **掌握硬件的控制方法**

如：中断、轮询、DMA 等，通常一个硬件控制器会有多种控制方法，你需要根据系统性能的需要合理的选择操作方法。初学阶段以实现功能为目的，掌握的顺序应该是，轮询->中断->DMA。随着学习的深入，需要综合考虑系统的性能需求，采取合适的方法。

●    **良好的GNU C语言编程基础**

如：C语言的指针、结构体、内存操作、链表、队列、栈、C和汇编混合编程等。这些编程语法是编写设备驱动的基础，无论对于初学者还是有经验者都非常重要。

●    **良好的linux操作系统概念**

如：多进程、多线程、进程调度、进程抢占、进程上下文、虚拟内存、原子操作、阻塞、睡眠、同步等概念及它们之间的关系。这些概念及方法在设备驱动里的使用是linux设备驱动区别单片机编程的最大特点，只有理解了它们才会编写出高质量的驱动。

●    **掌握linux内核中设备驱动的编写接口**

如：字 符设备的cdev、块设备的gendisk、网络设备的net\_device，以及基于这些基本接口的framebuffer设备的fb\_info、 mtd设备的mtd\_info、tty设备的tty\_driver、usb设备的usb\_driver、mmc设备的mmc\_host等。

Linux 内核为设备驱动编写者提供了标准的接口，驱动编写者无需精通内核的各个部分，只需要明确内核提供给我们的接口，并实现此接口就可以了。内核提供的接口采用 的是面向对象的思路，即把目标设备抽象成一个对象，通常利用一个结构体来描述这个对象。驱动工程师的任务就是实现这个对象。这个结构体中会包含设备的属性 （用变量表示）和操作方法（用函数指针表示）。如：字符设备的cdev

struct cdev {  
                struct kobject        kobj;  
                struct module        \*owner;  
                const struct file\_operations         \*ops;        // 操作方法结合，其它项都是属性   
                struct list\_head        list;  
                dev\_t                        dev;  
                unsigned int        count;  
        };

开始阶段可以以模仿为主，即套用一些固定的模板、参考例程。

**如何应对不断升级的内核**

内核升级对驱动的影响主要体现在，（1）驱动接口定义的变化；（2）内核的一些功能函数的名称、参数、头文件、宏定义的变化；（3）平台代码关于硬件操作方面封装的一些函数的变化；（4）设备模型的影响。

●    **驱动接口定义的变化**

如：2.4内核中字符设备驱动的注册接口是：

int register\_chrdev(unsigned int major, const char \* name, struct file\_operations \*fops)

而2.6内核中已经不建议使用这种方法了，改为：

int cdev\_add(struct cdev \*p, dev\_t dev, unsigned count)

这种接口定义及注册方法带来的变化，发生的并不频繁。解决方案是：参考内核中的代码。这种接口定义及注册方法在内核中非常容易找到，如：字符设备驱动的注册方法及接口定义可以参照内核driver/char/目录下的很多实例。

●    **内核的一些功能函数的名称、参数、头文件、宏定义的变化**

如：中断注册函数的格式及参数在2.4内核、2.6内核低版本和高版本之间都存在差别，在2.6.8中，中断注册函数的定义为：

int request\_irq(unsigned int irq, irqreturn\_t (\*handler)(int, void \*, struct pt\_regs \*),unsigned long irq\_flags, const char \* devname, void \*dev\_id)

irq\_flags的取值主要为下面的某一种或组合： SA\_INTERRUPT、SA\_SAMPLE\_RANDOM、SA\_SHIRQ

在2.6.26中，中断注册函数的定义为：

int request\_irq(unsigned int irq, irq\_handler\_t handler,unsigned long irqflags, const char \*devname, void \*dev\_id)

typedef irqreturn\_t (\*irq\_handler\_t)(int, void \*); irq\_flags的取值主要为下面的某一种或组合：（功能和2.6.8的对应）IRQF\_DISABLED、IRQF\_SAMPLE\_RANDOM、 IRQF\_SHARED

当 出现这些问题时，编译过程中，编译器会给我们比较明确的错误提示，根据这些提示你可以判断出是否是缺少头文件问题、是否是函数参数定义有误等。解决问题的 最好办法还是到你的目标内核中找信息。此时找问题的方法可以借助于搜索，如：你可以在新的内核中搜索request\_irq，看新内核中的驱动是如何使用 它的，这种方法非常有效。

●    **平台代码关于硬件操作方面封装的一些函数的变化**

内 核中，硬件平台相关的代码在内核更新过程中变化比较频繁，和我们的设备驱动也是息息相关，所以在针对一个新内核编写设备驱动前，一定要熟悉你的平台代码的 结构。有时平台虽然提供了内核要求的接口函数，但使用起来功能却并不完善。下面还是先举个例子说明平台代码更新对设备驱动的影响。

如： 在linux-2.6.8内核中，调用set\_irq\_type(IRQ\_EINT0,IRQT\_FALLING);去设置S3C2410的 IRQ\_EINT0的中断触发信号类型，你会发现不会有什么效果。跟踪代码发现内核的set\_irq\_type函数需要平台提供一个针对硬件平台的实现函 数

static struct irqchip s3c\_irqext\_chip = {  
                .mask = s3c\_irqext\_mask,  
                .unmask = s3c\_irqext\_unmask,  
                .ack = s3c\_irqext\_ack,  
                .type = s3c\_irqext\_type  
        };

s3c\_irqext\_type就是linux内核需要的实现函数，而s3c\_irqext\_type在2.6.8中的实现为： static int s3c\_irqext\_type(unsigned int irq, unsigned int type)  
        {  
                irqdbf("s3c\_irqext\_type: called for irq %d, type %d\n", irq, type);  
                return 0;  
        }

原来并没有实现。而在较高版本的内核，如2.6.26内核中，这个函数是实现了的。所以你一定要小心。当平台函数不好用时，一定要查查原因，或者直接操作硬件寄存器来达到目的。

●    **2.6内核设备模型对驱动的影响**

在 2.6内核中写设备驱动和在2.4内核中有着很大的不同，主要就是在设备驱动中融入了比设备驱动本身结构还复杂、还难以理解的设备模型。初学驱动时你可以 不理会设备模型，但你会发现内核里的驱动代码基本上都是融入了设备模型的了。所以很多时候你不得不面对现实，还是要弄懂它，并且它也的注册方法也会随着内 核的升级而发生变化。解决此类问题的最好方法还是参考目标内核驱动代码。

**总结：**

开始学习设备驱动时，选择一个当前比较流行的内核版本和硬件平台，不急于追赶最新潮流。这样你可以找到的网络资源会比较多，不至于有孤军奋战的感觉。我想这个过程应该不低于1年。当过了这个过程后，尝试将你编写过的驱动移植到各个目标平台上。