Linux 设备驱动程序庖丁解牛

写在前面的话

Linux Device Driver 尝试着看了好几遍，总感觉理解不够透彻，很多东西经不住问，才深深的意识到，要走进浩瀚无边的 Linux 世界，并不是一蹴而就的事情。我决心从理论到实践，再一次走一边这本书，同时记录下自己的心得，和大家一起分享，一起讨论，一起进步。

我意识到要写完所有学习心得及程序的实践分析，是比较困难的，但是，我将努力把自己肤浅的认识和感悟展现在大家面前，一方面敬请各位高手帮忙指出，以期获得更多的收获和进步；另一方面，也为那些如我般渴望学习，却无从下手的初学者给出一些指引。

多余的话就不再说了，我们开始吧，怀着一份横穿可可西里的勇气和激情，去探索无尽的奥秘……并希望最终能达到“庖丁解牛”技能！

必备资料：

1、《Linux Device Driver》第三版

2、《The Linux Kernel Module Programming Guide》

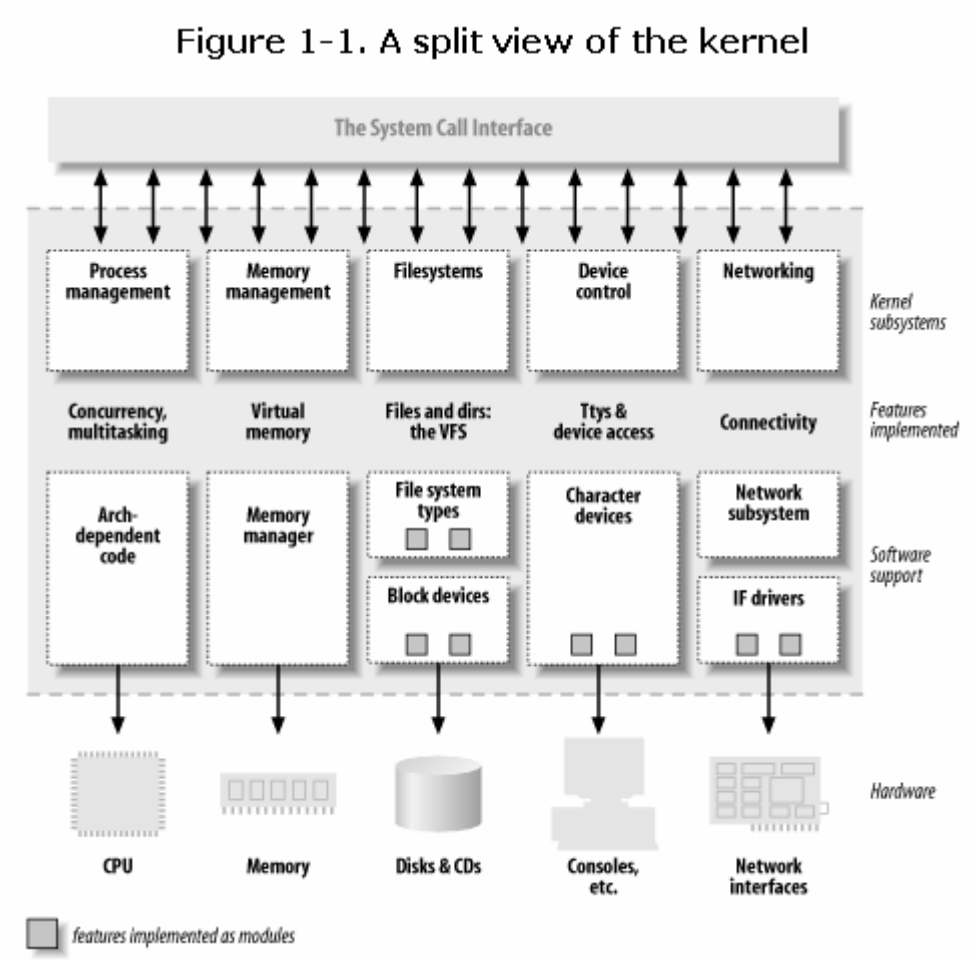
3、一份官方的 Linux 2.6 版本的源码（尽量新一点的吧）

Linux 设备驱动程序庖丁解牛之一

——开发环境的建立

工欲善其事，必先利其器。

要学习 Linux 驱动开发，首先应该搞明白 Linux 内核的架构，明白驱动程序到底扮演了一个什么样的角色。下图展示了 Linux 内核的整体架构：



具体的理论知识，希望能细心阅读《Linux Device Driver》的第一章和第二章，这里就不抄写了。

下面，我们开始介绍如何建立开发环境，由于不同的 Linux 发行版本可能存在一定的差异，我这里讲述一个普适性的方法。“如果想要学习驱动程序的编写，则标准内核是最好的”，我们就从这个“最好的”方法开始吧。

1、下载源码

从www.kernel.org上下载标准的Linux内核源码。通常有xx.tar.gz和xxx.tar.bz2 的，都可以，愿意选择哪个版本，则随你所愿了。《Linux Device Driver》第三版所讲的是 2.6.11

版的，推荐和该版本一致。标准源码下载到/usr/src/下面，然后解压：

（1）如果是 xx.tar.gz：

tar －zxvf xx.tar.gz

（2）如果是 xxx.tar.bz2：

tar －jxvf xx.tar.gz

2、编译内核源码

这一步往往是很多初学者望而生畏的，没关系，Nothing is impossible!

需要注意的是，你现在的主机安装的内核源码，尽量与你下载的源码大版本上一致，也就是说如果下载源码是 2.6 的，那主机的操作系统也最好是 2.6 的内核。不过，这也不是必要的，只是编译的时候稍微麻烦一点。

cd 到你解压的源码目录。如果都是 2.6 的内核，ok：

step1：make oldconfig （or make menuconfig）

这样做是为了简便一些，避免了无谓的配置；当然，也可以 make menuconfig，基于图形化的配置，这是比较耗时的了，推荐高手采用这种方法。对于跨版本的（主机内核版本和源码版本不一致的），可能这个选择更好一些。

Step2: make

让他编译吧，耐心等待。

Step3: make bzImage

Step4: make modules

Step5: make modules_install

Step6: make install

后面的不同的系统可能存在一些差异，我说一下 ubuntu7.04 内核 2.6.20-15

一切就绪后，查看/boot/下已经有 vmlinuz-2.6.20.3 了，但是没有对应的 initrd.img。

apt-get install initrd-tools 安装 initrd-tools 然后执行

mkinitrd -o /boot/initrd.img.2.6.20.3 /lib/modules/2.6.20.3 生成 新的 initrd.img

再修改 /boot/grub/menu.lst 用新的内核启动

title gnuser,:)

root (hd0,1)

kernel /boot/vmlinuz-2.6.20.3 root=/dev/sdb1 ro quiet splash

initrd /boot/initrd.img-2.6.20.3

savedefault

boot

重启后，成功进入新配置内核

写一个 最简单 最没用的驱动吧

我在 /home/shana/linux_q/ 目录下创建 2 个文本文件 hello.c Makefile

```
//hello.c
#include
#include
MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void)
{
printk(KERN_ALERT "Hello, world\n");
return 0;
}

static void hello_exit(void)
{
printk(KERN_ALERT"Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Makefile 文件

```
obj-m := hello.o
KERNELDIR := /lib/modules/2.6.20/build
PWD := $(shell pwd)

modules:
$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

modules_install:
$(MAKE) -C $(KERNELDIR) M=$(PWD) modules_install
```

然后执行，make，OK 了。ls，有如下文件：

hello.c hello.ko hello.mod.c hello.mod.o

执行：insmod hello.ko

然后执行 dmesg

可以看到：Hello, world 恭喜你，成功了

执行：rmmod hello.ko

可以看到：Goodbye, cruel world

就这么简单！

其它的系统也是大同小异的。

另外，我再说一下 Fedora 5 上面的建立过程，这个就不需要下载源代码了。

Fedora Core 5 与旧版本不同，不包含 kernel-source 软件包。已配置的源代码可以按照内核配置一节的步骤得到。

需要使用内核源代码的 Fedora Core 用户可以在内核 .src.rpm 软件包中找到它们。要从文件释放源码树，执行下面的命令：

以超级用户身份构建软件包是极其危险的，不应当这样做，即使是内核。下面的操作使您可以以普通用户身份构建内核。很多教程以 /usr/src/linux 作为内核的源码位置，如果想遵循这些操作，可以替换为 ~/rpmbuild/BUILD/kernel-<version>/linux-<version>。

1、在个人目录准备 RPM 软件包构建环境，运行下面的命令：

su -c 'yum install fedora-rpmdevtools'

fedora-buildrpmtree 提示时输入 root 的密码。

2、从下列来源之一获取 kernel-version.src.rpm 文件：

1）SRPMS 文件，包含在合适的 SRPMS CD iso 镜像文件中。

2）下载内核软件包的 HTTP 或 FTP 站点

3）执行这个命令：

su -c 'yum install yum-utils'

su -c 'yumdownloader --source kernel'提示时输入 root 的密码。

3、安装 kernel-<version>.src.rpm，运行命令：

rpm -Uvh kernel-<version>.src.rpm`

这个命令将 RPM 内容写到 ${HOME}/rpmbuild/SOURCES 和 ${HOME}/rpmbuild/SPECS, 这里 ${HOME} 是您的个人目录。

注：硬盘空间需求完整的内核构建过程可能需要您的个人目录有几个吉的存储空间。

4、使用这样的命令来准备内核源代码：

cd ~/rpmbuild/SPECS

rpmbuild -bp --target $(uname -m) kernel-2.6.spec

内核源码树位于 ${HOME}/rpmbuild/BUILD/kernel-<version>/ 目录。

注：这一步可能会报很多 warning，不用管它。

5、Fedora Core 附带的内核配置文件在 configs/ 目录。例如，i686 SMP 配置文件被命名为 configs/kernel-version-i686-smp.config。使用下列命令来将需要的配置文件复制到合适的位置，用来编译：

cp configs/<desired-config-file> .config

您也可以在 /lib/modules/version/build/.config 这个位置找到与您当前的内核匹配的 .config 文件。

6、每个内核的名字都包含了它的版本号，这也是 uname -r 命令显示的值。内核 Makefile 的前四行定义了内核的名字。为了保护官方的内核不被破坏，Makefile 经过了修改，以生成一个与运行中的内核不同的名字。在一个模块插入运行中的内核前，这个模块必须针对运行中的内核进行编译。为此，您必须编辑内核的 Makefile。

例如，如果 uname -r 返回字符串 2.6.15-1.1948_FC5，就将 EXTRAVERSION 定义从：

EXTRAVERSION = -prep　修改为：

EXTRAVERSION = -1.1948_FC5

也就是最后一个连字符后面的所有内容。