http://bbs.csdn.net/topics/350145608

编译内核

一、准备工作  
1、查看Linux内核版本  
# uname -a  
二、配置工作  
2.6内核和4个升级工具都下载完了  
2、下载2.6内核源码  
3、下载内核升级工具  
(1)下载module-init-tools-3.2.tar.bz2  
(2)下载mkinitrd-4.1.18-2.i386.rpm  
(3)下载lvm2-2.00.25-1.01.i386.rpm  
(4)下载device-mapper-1.00.19-2.i386.rpm  
(少一个也不行，如果没有下载齐全，请不要尝试下面的步骤，升级是不会成功的)，下面回到Linux系统中开始配置工作吧。  
4、将下载好的内核源码包和4个升级工具都拷贝到/usr/src文件夹下。  
5、拷贝完毕，开始解压新内核：  
5.1  # cd /usr/src    // (进入到/usr/src目录下，如果已经在/usr/src目录下，可不执行该命令)  
5.2  # rm –rf linux     //(删除linux文件夹。值得一提的是，如果内核先前从未编译过，则没有这个文件夹，此命令行可略过)  
5.3   # tar jvxf linux-2.6.18.tar.bz2         // (解压新内核)  
5.4   # ln -s linux-2.6.18 linux            // (重新生成linux文件夹)  
6、安装module-init-tools工具  
在/usr/src目录下，依次执行下列命令：  
6.1    # tar jvxf module-init-tools-3.2.tar.bz2      // (解压module-init-tools)   
6.2    # cd module-init-tools-3.2     // (由/usr/src目录进入module-init-tools目录下)  
6.3    #./configure --prefix=/         / /( 配置文件的默认安装路径)  
6.4    # make moveold      // (保留原来内核配置文件，防止新内核无法使用时候可以正常进入系统)  ----（前面的每一个步骤都必须执行，否则会出现这个错误：  
make: \*\*\* No rule to make target `moveold'.  Stop.！）  
6.5     # make all install         //（安装所有文件）  
6.6  #./generate-modprobe.conf /etc/modprobe.conf     // 用这个脚本生成/etc/modprobe.conf，./表示当前目录。这个文件必须要可执行。   
7、安装另外三个升级工具  
回到/usr/src目录下，依次执行下列3个命令来安装另外三个升级工具：  
7.1  # rpm -ivh --nodeps mkinitrd-4.1.18-2.i386.rpm  //(注意，这里一定要加入--nodeps参数就是指不考虑相依属性的关系，给它強制的安裝下去）  
7.2  # rpm -ivh --nodeps lvm2-2.00.25-1.01.i386.rpm  
7.3  # rpm -ivh --nodeps device-mapper-1.00.19-2.i386.rpm  
如果不更新以上几个升级包，在后面编译内核时会提示以下错误：  
mkinitrd failed  
make[1]: \*\*\* [install] Error 1  
make: \*\*\* [install] Error 2  
8、配置内核选项。  
8.1  # cd linux-2.6.18        // (进入到/usr/src/linux-2.6.18目录下)  
8.2  # make mrproper        //(该命令可确保源代码目录下没有不正确的.o文件)  
8.3  # make menuconfig      // (配置内核各选项。如果你没有安装全部的必须配置文件，那么就不会进入图形界面，就会显示错误：  
make[1]: \*\*\* [menuconfig] error 1make: \*\*\* [menuconfig] error 2。产生这个现象不仅仅是这个原因，还有就是缺少一个glibc-kernheaders-2.4-9.1.103.EL.i386.rpm)   
此时会出现一个图形界面，列出了所有的内核配置选项，有的选项下还有子选项，你可以用方向键来选择，用Y键来确定。经过多次试验，大多数选项默认就行，以下几个选项必须选择(请认真核对下面每一个选项，否则编译很有可能前功尽弃)：  
(1)Loadable Module support选项中，选上“Module unloading”和“Automatic kernel module loading”这两项；  
(2)  2.1  Device Drivers--->Block Devices中选上“Loopback device support”；  
    2.2  Device Drivers--->Multi-device support(RAID and LVM)处要选上“device mapper  support”；  
    2.3   Device Drivers--->Graphics support，一定要选上“ Support for buffer devices”；  
    2.4   Device Drivers --->USB support --->选上“USB Mass Storage support”(如果是在实环境中，想要更多USB支持，就全选吧。我的是在虚拟机中，用不着了)  
    2.5  Device Drivers --->;Network device support --->Ethernet (10 or 100Mbit) ---><\*> AMD PCnet32 PCI support   
(3)File system--->(以下9个选项是关于ext2和ext3文件系统配置，全部选上)  
    3.1   Second extended fs support  
    3.2   Ext2 extended attributes  
    3.3   Ext2 POSIX Access Control Lists  
    3.4   Ext2 Security Labels  
    3.5   Ext3 journalling file system support  
    3.6   Ext3 extended attributes  
    3.7   Ext3 POSIX Access Control Lists  
    3.8   Ext3 Security Labels  
    3.9   JBD (ext3) debugging support  
    3.10  File system--->DOS/FAT/NT Filesystems --->选上“NTFS file system support”；  
注意：  
ext2和ext3文件系统配置很重要，也是必需的，如果对Ext3、Ext2文件的支持直接编译进内核，在你重启时机器就会宕机，出现如下错误信息：  
kernel panic : no init found ,try passing init = option to kernel.....  
或者是：  
VFS:Cannot open root device "hdxy" or unknow-block(0,0)  
Please append a correct "root=" boot option  
kernel panic:VFS:Unable to mount root fs on unknown-block(0,0)   
或者是：  
mount: error 19 mounting ext3  
pivotroot: pivot\_root(/sysroot,/sysroot/initrd) failed: 2  
umount /initrd/proc fail: 2  
Freeing unused kernel memory: 244k freed  
Kernel panic – not syncing: No init found. Try passing init = option to kernel  
(在教室时，我的机器就是在重启之后出现第三种错误，进不去系统，郁闷死，只好重装了~~~如果当你重启Linux系统后，若不幸进不去2.6.18内核，你会发现你的出错信息就是上面三种了~~~哈！)  
(4)如果你在vmware下编译内核，硬盘用的是scsi的，以下三个选项必选：  
4.1   Device Drivers ---><\*>SCSI device support (此项不选的话，下面两项就选择不上)  
4.2   Device Drivers ---><\*>SCSI device support ---><\*>SCSI disk support  
4.3  Device Drivers---><8>SCSI device support--->SCSI low-level drivers---><\*>; BusLogic SCSI support  
三、编译工作  
9、开始编译啦……  
在/usr/src/linux-2.6.18目录下，执行以下命令即可编译。编译需要一段时间，耐心等候  
9.1  # make dep (建立编译时所需的从属文件。注意：如果内核从未编译过，此步可跳过)  
9.2  # make clean (清除内核编译的目标文件。注意：如果内核从未编译过，此步可跳过)  
9.3  # make bzImage (注意I大小写。这一步才是真正编译内核)  
内核编译成功后，会在/usr/src/linux/arch/i386/boot目录中生成一个新内核的映像文件bzImage。如果用 make zImage编译，内核很大的话，系统会提示你使用make bzImage命令来编译，所以我直接用make bzImage来编译。）  
PS：在执行这一步可能会出现一个错误，不能够编译内核源代码，解决的方法是 在系统上安装一个GCC编译器,如果使用的是自己定制的系统，就当我没有说过这话哈！嘿嘿  
9.4  # make modules (编译可加载模块)  
9.5  # make modules\_install (安装可加载模块)  
安装成功后，系统会在/lib/modules目录下生成一个2.6.18子目录，里面存放着新内核的所有可加载模块。  
9.6  # make install (安装新内核)  
注意：  
make install的时候可能会出现如下错误信息：  
No module BusLogic found for kernel 2.4.12  
mkinitrd failed  
此问题一般只出现在SCSI硬盘＋VMWARE+REDHAT架构中，因为BusLogic被编译进了内核而不是一个module的形式(2.4内核的Buslogic模块即使静态编译进内核也不行)。解决方式是直接将BusLogic.o文件复制过去：  
9.7   # cp /usr/src/linux-2.6.18/drivers/scsi/BusLogic.o /lib/modules/2.6.18/kernel/drivers/   
不过别忘记，复制过后再执行一下make install。这一步若卡住了，下面的都无法进行  
四、启动新内核  
10、将新内核和System.map文件拷贝到/boot目录下，依次执行以下命令：   
10.1   # cp /usr/src/linux-2.6.18/arch/i386/boot/bzImage /boot/vmlinuz-2.6.18   
10.2   # cp /usr/src/linux-2.6.18/System.map /boot/System.map-2.6.18   
10.3   # cd /boot   (进入boot目录)  
10.4   # rm –rf System.map   (将会提示是否删除原来的连接，输入yes 回车)   
10.5   # ln –s System.map-2.6.18 System.map   (提示是否重新建立连接，输入yes 回车)   
11、修改Grub启动管理器  
如果没有错误的话, 下面开始修改grub配置文件  
在/boot目录下，执行以下命令：  
11.1   # new-kernel-pkg --mkinitrd --depmod --install 2.6.18 (这时候你的/boot下会生成一个initrd-2.4.18.img，并且你的grub.conf文件也作了相应更改)  
11.2   # df (查看根目录在那个分区，下一步要用到。注意，这里根分区不时boot的那个50M的分区，而一般是你最大的那个分区，也就是“/”，千万不要搞错哦。我的为 /mnt/hgfs)  
11.3   # vi /grub/grub.conf   
进入grub.conf文件，找到如下信息：  
default=1  
timeout=10  
splashimage=(hd0,0)/grub/splash.xpm.gz  
title Red Hat Linux (2.6.18)  
root (hd0,0)  
kernel /vmlinuz-2.6.18 ro root= LABEL=/  
initrd /initrd-2.6.18.img  
做两处修改：  
(1) 将default=1改为default=0   (不改的话也可以，只不过重启之后会默认进入2.4内核)  
(2) 将kernel行的“LABEL=/”换成根目录所在的分区(上一步查看的就是)  
此步很重要，修改错误将可能导致进不去系统。  
我把我修改后的grub.conf文件列出来，不明之处，可以对照修改：  
default=0  
timeout=10  
splashimage=(hd0,0)/grub/splash.xpm.gz  
title Red Hat Linux (2.6.18)  
root (hd0,0)  
kernel /vmlinuz-2.6.18 ro root=/mnt/hgfs  
initrd /initrd-2.6.18.img  
title Red Hat Linux (2.4.20-8)  
root (hd0,0)  
kernel /vmlinuz-2.4.20-8 ro root=LABEL=/  
initrd /initrd-2.4.20-8.img  
12，重启，看看升级后的2.6.18内核吧!

**redhat9(内核版本2.4.20-8)编译内核到版本(2.4.26)**

<http://linux.chinaunix.net/techdoc/develop/2008/04/24/995698.shtml>

1.   前言  
       在我写这篇文章的时候，还是一个linux的初学者，经历了n次失败后的成功当然是兴奋的，于是很想把她写下来。  
       我的操作系统是Redhat9，其内核版本为2.4.20-8 ，需要升级到2.4.26，采用全新的2.4.26内核源码进行升级，主要的参考资料是《The Linux Kernel HOWTO》，另外还有一些参考资料我想就不提了，免得误导象我一样的初学者。  
       接下来有三个部分，“步骤索引”、“步骤说明”和“附录”，“步骤索引”真实的记录了我成功升级内核的步骤，“步骤说明”将对“步骤索引”一些注意事项进行说明，“附录”摘录自《The Linux Kernel HOWTO》的相关部分。  
2.   步骤索引  
[root@localhost src]# tar vxfj linux-2.4.26.tar.bz2  
[root@localhost src]# ln Cs linux2.4.26 linux  
[root@localhost src]# cd /usr/src/linux  
[root@localhost linux]# cp /boot/config-2.4.20-8 .config  
[root@localhost linux]# make menuconfig  
[root@localhost linux]# make dep  
[root@localhost linux]# make clean  
[root@localhost linux]# make bzImage  
[root@localhost linux]# make modules  
[root@localhost linux]# make modules\_install  
[root@localhost linux]# cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.26  
[root@localhost linux]# cp .config /boot/config-2.4.26  
[root@localhost linux]# cp System.map /boot/System.map-2.4.26  
[root@localhost linux]# mkinitrd /boot/initrd-2.4.26.img 2.4.26  
[root@localhost linux]# cd /boot  
[root@localhost boot]# rm -f System.map vmlinuz  
[root@localhost boot]# ln -s System.map-2.4.26 System.map  
[root@localhost boot]# ln -s vmlinuz-2.4.26 vmlinuz  
[root@localhost boot]# cd grub  
[root@localhost grub]# vi grub.conf  
[root@localhost grub]# reboot  
3.   步骤说明  
       源码的升级，我将其归纳为源码准备、工具准备、配置内核、编译内核、安装模块、启动设置和重启7个步骤。我是以root的身份登录的，所以省略了用户切换的过程。  
3.1. 源码准备  
       这个没有什么好说的，到  
[www.kernel.org](http://www.kernel.org/)  
去下载就是。我这里下载的是linux-2.4.26.tar.bz2，将其放到/usr/src目录下面，使用下面的方法解压缩：  
              [root@localhost src]# cd /usr/src  
[root@localhost src]# tar vxfj linux-2.4.26.tar.bz2  
[root@localhost src]# ln Cs linux2.4.26 linux（这一步我没做）  
       解压缩完成后得到linux-2.4.26目录。喜欢图形界面的朋友可以不用这样，我在gnome下执行鼠标右键弹出菜单命令“解压缩到这里…”也能达到同样的目的，不过经常出错。  
3.2. 工具准备  
       我所使用的两个内核版本跨度不大，所以没有特别安装什么工具，如果是升级到2.6.x版本会需要一些而外的工具。  
3.3. 配置内核  
       这是编译内核最麻烦的地方了，我们将要面临一大堆驱动模块的配置选项，对于我这样的菜鸟就头晕了，开始时傻呼呼的按照网站上的一些资料介绍，执行make menuconfig或make xconfig，结果碰得“头破血流”。  
       在redhat9下执行make xconfig，会提示需要qt，在我看来make menuconfig是最好用的，不过在此之前不需要着急。  内核配置的结果是在源码的顶级目录下生成一个.config文件，其中当然是保存了各种配置项的设定值，如果不想在那么多的似懂非懂的选项中选择的话，先执行下面的操作：  
[root@localhost src]# cd /usr/src/linux（我输入的是：cd /usr/src/linux-2.4.26，这个要看你解压缩后的文件名字了）  
[root@localhost linux]# cp /boot/config-2.4.20-8 .config  
       在redhat9安装完成后,/boot/config-2.4.20-8就是我们所需要的.config文件，只要将其复制到源码目录并改名成.config就可以了，这样，在执行下面的操作就能得到当前系统的配置，如果需要增加驱动模块，只需要作少量修改就可以了。  
[root@localhost linux]# make menuconfig  
[root@localhost linux]# make dep  
[root@localhost linux]# make clean  
        
3.4. 编译内核  
[root@localhost linux]# make bzImage  
       当你看到类似以下信息，说明编译成功了  
tools/build -b bbootsect bsetup compressed/bvmlinux.out CURRENT > bzImage  
Root device is (3, 8)  
Boot sector 512 bytes.  
Setup is 4886 bytes.  
System is 917 kB  
make[1]: Leaving directory `/usr/src/linux-2.4.26/arch/i386/boot'  
3.5. 安装模块  
[root@localhost linux]# make modules  
[root@localhost linux]# make modules\_install  
在make modules\_install执行完成后，如果成功了，你将会在/lib/modules目录下看到2.4.26。  
  
3.6. 启动设置  
        [root@localhost linux]# cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.26  
    [root@localhost linux]# cp .config /boot/config-2.4.26  
[root@localhost linux]# cp System.map /boot/System.map-2.4.26  
[root@localhost linux]# mkinitrd /boot/initrd-2.4.26.img 2.4.26  
       很多资料上介绍了手工生成ramdisk的过程，mkinitrd可以帮我们完成这个稍微复杂的过程，如果这个操作执行成功，mkinitrd会在/boot目录下生成initrd-2.4.26.img文件。  
[root@localhost linux]# cd /boot  
[root@localhost boot]# rm -f System.map vmlinuz  
[root@localhost boot]# ln -s System.map-2.4.26 System.map  
[root@localhost boot]# ln -s vmlinuz-2.4.26 vmlinuz  
     
       关于bzImage和System.map请参考附录。  
  
我安装redhat9时选择的是grub，所以需要改变grub.config中的设置。  
[root@localhost boot]# cd grub  
[root@localhost grub]# vi grub.conf  
    用vi打开/boot/grub/grub.conf文件，找到下面的两行，将其注释，并增加下面的内容。  
#       kernel /boot/vmlinuz-2.4.20-8 ro root=LABEL=/  
#       initrd /boot/initrd-2.4.20-8.img  
       kernel /boot/vmlinuz ro root=/dev/hda8  
       initrd /boot/initrd-2.4.26.img  
    root指定/挂载点的位置，如果不知道，可以使用df命令查看。（我朋友给我的建议是进入编辑器后不要将那两行注释掉，而是复制最后四行，粘贴，然后将复制后的四行中的2.4.20-8改成2.4.26。这样就会有两个版本供选择，万一编译失败了还可以进旧版本中。）  
       这里有一点需要注意，整个启动设置可以使用一个命令make install来完成，不过会有很多问题，对grub.conf文件的设置就是其中之一，make install修改的grub.conf文件仍然使用”LABEL”符号，而这个符号在2.4.26的内核下已经不起作用了。  
  
3.7. 重启  
[root@localhost grub]# reboot  
    但愿你也成功！  
4.   总结  
       网上的资料虽然丰富，但多数不系统，也没有随着环境的升级而得到及时的更正，很多时候找下来的资料都不知道实用于那个版本的操作系统或内核。举个小例子，在  
[www.kernel.org](http://www.kernel.org/)  
官方网站上有一个readme文件，其中说明如何编译内核，其中有这么几步操作：  
              cd include  
              rm Crf asm linux scsi  
              ln Cs asm-i386 asm  
              ln Cs linux linux  
              ln Cs scsi scsi  
  
如果执行了这些操作，在编译2.4.26的内核时会出现错误。  
在没有人指导的情况下，最好能找比较系统点的资料，或者到书店买本书。我就是这样，多亏找到了《The Linux Kernel HOWTO》才使得我成功编译内核，否则还不知道要失败多少次呢。  
5.   附录  
.config  
Everytime you compile and install the kernel image in /boot, you should also copy the corresponding config  
file to /boot area, for documentation and future reference. Do NOT touch or edit these files!!  
bzImage  
The bzImage is the compressed kernel image created with command 'make bzImage' during kernel compile. It  
important to note that bzImage is not compressed with bzip2 !! The name bz in bzImage is misleading!! It  
stands for "Big Zimage". The "b" in bzImage is "big". Both zImage and bzImage are compressed with gzip.  
The kernel includes a mini&#8722;gunzip to uncompress the kernel and boot into it. The difference is that the old  
zImage uncompresses the kernel into low memory (the first 640k), and bzImage uncompresses the kernel into  
high memory (over 1M). The only problem is that there are a very few machines where bzImage is known to  
have problems (because the machines are buggy). The bzImage actually boots faster, but other than that,  
there's no difference in the way the system \*runs\*. The rule is that if all drivers cannot fit into the zImage,  
then you need to modularize more.  
If the kernel is small, it will work as both a zImage and a bzImage, and the booted system runs the same way.  
A big kernel will work as a bzImage, but not as a zImage. Both bootimages are gzipped, (bzImage is not  
bzipped as the name would suggest), but are put together and loaded differently, that allows the kernel to load  
in higher address space, that does not limit it to lower memory in the pathetic intel architecture. So why have  
both? Backward compatability. Some older lilos and loadlins don't handle the bzImage format. Note, they  
\*boot\* differently, but \*run\* the same. There is a lot of misinformation given out about what a bzImage file is  
(mostly about it being bzip2ed).  
System.map  
System.map is a "phone directory" list of function in a particular build of a kernel. It is typically a symlink to  
the System.map of the currently running kernel. If you use the wrong (or no) System.map, debugging crashes  
is harder, but has no other effects. Without System.map, you may face minor annoyance messages.  
Do NOT touch the System.map files.