Linux系统启动流程

POST-->BIOS(Boot Sequence)-->MBR(bootloader,446)-->Kernel-->initrd-->(ROOTFS)/sbin/init(/etc/inittab)
说明：BIOS自检-->从BIOS中读取启动顺序-->读取MBR中的bootloader-->加载内核-->读取伪根-->读取根文件中的init

- BIOS自检

稍有计算机基础的人都应该听过BIOS(Basic Input / Output System)，又称基本输入输出系统，可以视为是一个永久地记录在ROM中的一个软件，是操作系统输入输出管理系统的一部分。早期的BIOS芯片确实是"只读"的，里面的内容是用一种烧录器写入的，一旦写入就不能更改，除非更换芯片。现在的主机板都使用一种叫Flash EPROM的芯片来存储系统BIOS，里面的内容可通过使用主板厂商提供的擦写程序擦除后重新写入，这样就给用户升级BIOS提供了极大的方便。

BIOS的功能由两部分组成，分别是POST码和Runtime服务。POST阶段完成后它将从存储器中被清除，而Runtime服务会被一直保留，用于目标操作系统的启动。BIOS两个阶段所做的详细工作如下：

步骤1：上电自检POST(Power-on self test)，主要负责检测系统外围关键设备（如：CPU、内存、显卡、I/O、键盘鼠标等）是否正常。例如，最常见的是内存松动的情况，BIOS自检阶段会报错，系统就无法启动起来；

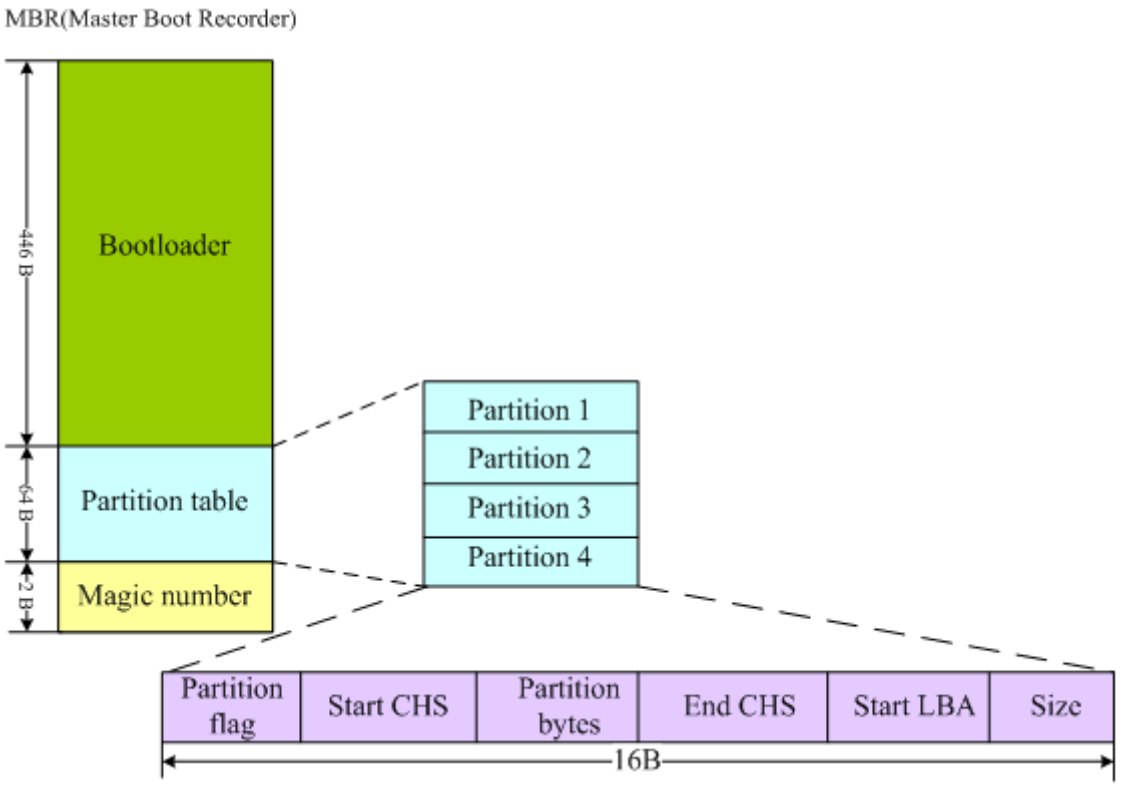
步骤2：步骤1成功后，便会执行一段小程序用来枚举本地设备并对其初始化。这一步主要是根据我们在BIOS中设置的系统启动顺序来搜索用于启动系统的驱动器，如硬盘、光盘、U盘、软盘和网络等。我们以硬盘启动为例，BIOS此时去读取硬盘驱动器的第一个扇区(MBR，512字节)，然后执行里面的代码。实际上这里BIOS并不关心启动设备第一个扇区中是什么内容，它只是负责读取该扇区内容、并执行。

至此，BIOS的任务就完成了，此后将系统启动的控制权移交到MBR部分的代码。

PS：在个人电脑中，Linux的启动是从0xFFFF0地址开始的。

- 系统引导

我们首先来了解一下MBR，它是Master Boot Record的缩写。硬盘的0柱面、0磁头、1扇区称为主引导扇区。它由三个部分组成，主引导程序(Bootloader)、 硬盘分区表DPT（Disk Partition table）和硬盘有效标志（55AA），其结构图如下所示：



磁盘分区表包含以下三部分：

1）、Partition ID （5：延申 82：Swap 83：Linux 8e：LVM fd：RAID）

2）、Partition起始磁柱

3）、Partition的磁柱数量

通常情况下，诸如lilo、grub这些常见的引导程序都直接安装在MBR中。我们以grub为例来分析这个引导过程。

grub引导也分为两个阶段stage1阶段和stage2阶段(有些较新的grub又定义了stage1.5阶段)。

1)、stage1：stage1是直接被写入到MBR中去的，这样机器一启动检测完硬件后，就将控制权交给了GRUB的代码。也就是上图所看到的前446个字节空间中存放的是stage1的代码。BIOS将stage1载入内存中0x7c00处并跳转执行。
stage1（/stage1/start.S）的任务非常单纯，仅仅是将硬盘0头0道2扇区读入内存。而0头0道2扇区内容是源代码中的/stage2/start.S，编译后512字节，它是stage2或者stage1_5的入口。而此时，stage1是没有识别文件系统的能力的。如果感觉脑子有些晕了，那么下面的过程就直接跳过，去看stage2吧!

【外传】定位硬盘的0头0道2扇区的过程：

BIOS将stage1载入内存0x7c00处并执行，然后调用BIOS INIT13中断，将硬盘0头0道2扇区内容载入内存0x7000处，然后调用copy_buffer将其转移到内存0x8000处。在定位0头0道2扇区时通常有两种寻址方式：LBA和CHS。如果你是刨根问底儿型的爱好者，那么此时去找谷哥打听打听这两种方式的来龙去脉吧。

2)、stage2：严格来说这里还应该再区分个stage1.5的，就一并把stage1.5放在这里一起介绍了，免得大家看得心里乱哄哄的。好的，我们继续说0头0到2扇区

的/stage2/start.S文件，当它的内容被读入到内存之后，它的主要作用就是负责将stage2或stage1.5从硬盘读到内存中。如果是stage2，它将被载入到0x820处；如果是stage1.5，它将被载入到0x2200处。这里的stage2或者stage1_5不是/boot分区/boot/grub目录下的文件，因为这个时候grub还没有能力识别任何文件系统。

？ 如果start.S加载stage1.5：stage1.5它存放在硬盘0头0道3扇区向后的位置，stage1_5作为stage1和stage2中间的桥梁，stage1_5有识别文件系统的能力，此后grub才有能力去访问/boot分区/boot/grub目录下的 stage2文件，将stage2载入内存并执行。

？ 如果start.S加载stage2：同样，这个stage2也不是/boot分区/boot/grub目录下的stage2，这个时候start.S读取的是存放在/boot分区Boot Sector的stage2。这种情况下就有一个限制：因为start.S通过BIOS中断方式直接对硬盘寻址（而非通过访问具体的文件系统），其寻址范围有限，限制在8GB以内。因此这种情况需要将/boot分区分在硬盘8GB寻址空间之前。

假如是情形2，我们将/boot/grub目录下的内容清空，依然能成功启动grub；假如是情形1，将/boot/grub目录下stage2删除后，则系统启动过程中grub会启动失败。

- 启动内核

当stage2被载入内存执行时，它首先会去解析grub的配置文件/boot/grub/grub.conf，然后加载内核镜像到内存中，并将控制权转交给内核。而内核会立即初始化系统中各设备并做相关的配置工作，其中包括CPU、I/O、存储设备等。
关于Linux的设备驱动程序的加载，有一部分驱动程序直接被编译进内核镜像中，另一部分驱动程序则是以模块的形式放在initrd(ramdisk)中。

Linux内核需要适应多种不同的硬件架构，但是将所有的硬件驱动编入内核又是不实际的，而且内核也不可能每新出一种硬件结构，就将该硬件的设备驱动写入内核。实际上Linux的内核镜像仅是包含了基本的硬件驱动，在系统安装过程中会检测系统硬件信息，根据安装信息和系统硬件信息将一部分设备驱动写入 initrd 。这样在以后启动系统时，一部分设备驱动就放在initrd中来加载。这里有必要给大家再多介绍一下initrd这个东东：

initrd 的英文含义是 bootloader initialized RAM disk，就是由 boot loader 初始化的内存盘。在 linu2.6内核启动前，boot loader 会将存储介质中的 initrd 文件加载到内存，内核启动时会在访问真正的根文件系统前先访问该内存中的 initrd 文件系统。在 boot loader 配置了 initrd 的情况下，内核启动被分成了两个阶段，第一阶段先执行 initrd 文件系统中的init，完成加载驱动模块等任务，第二阶段才会执行真正的根文件系统中的 /sbin/init 进程。

另外一个概念：initramfs

initramfs 是在 kernel 2.5中引入的技术，实际上它的含义就是：在内核镜像中附加一个cpio包，这个cpio包中包含了一个小型的文件系统，当内核启动时，内核将这个 cpio 包解开，并且将其中包含的文件系统释放到rootfs中，内核中的一部分初始化代码会放到这个文件系统中，作为用户层进程来执行。这样带来的明显的好处是精简了内核的初始化代码，而且使得内核的初始化过程更容易定制。

疑惑的是：我的内核是2.6.32-71.el6.i686版本，但在我的/boot分区下面却存在的是/boot/initramfs-2.6.32-71.el6.i686.img类型的文件，没搞明白，还望高人解惑。我只知道在2.6内核中支持两种格式的initrd，一种是2.4内核的文件系统镜像image-initrd，一种是cpio格式。接下来我们就来探究一下initramfs-2.6.32-71.el6.i686.img里到底放了那些东西。

```
[root@maple ~]# cp /boot/initramfs-2.6.32-71.el6.i686.img ./
[root@maple ~]# file initramfs-2.6.32-71.el6.i686.img
initramfs-2.6.32-71.el6.i686.img: gzip compressed data, from Unix, last modified
: Sat Nov 12 11:31:33 2011, max compression
[root@maple ~]# gunzip -c initramfs-2.6.32-71.el6.i686.img > initrd.img
[root@maple ~]# file initrd.img
initrd.img: ASCII cpio archive (SVR4 with no CRC)
[root@maple ~]# mkdir tmp
[root@maple ~]# cd tmp/
```

在tmp文件夹中解压initrd.img里的内容：

```
[root@maple tmp]# cpio -id < ../initrd.img
59250 blocks
[root@maple tmp]# ls
bin                 emergency  initqueue-finished  pre-pivot    sbin     usr
cmdline             etc        initqueue-settled   pre-trigger  sys      var
dev                 init       lib                 pre-udev     sysroot
dracut-004-32.el6   initqueue  mount               proc         tmp
[root@maple tmp]# file init
init: POSIX shell script text executable
[root@maple tmp]#
```

如果initrd.img文件的格式显示为“initrd.img:ISO 9660 CD-ROM filesystem data”，则可直接输入命令“mount -o loop initrd.img /mnt/test”进行挂载。

通过上的分析和我们的验证，我们确实得到了这样的结论：

grub的stage2将initrd加载到内存里，让后将其中的内容释放到内容中，内核便去执行initrd中的init脚本，这时内核将控制权交给了init文件处理。我们简单浏览一下init脚本的内容，发现它也主要是加载各种存储介质相关的设备驱动程序。当所需的驱动程序加载完后，会创建一个根设备，然后将根文件系统rootfs以只读的方式挂载。这一步结束后，释放未使用的内存，转换到真正的根文件系统上面去，同时运行/sbin/init程序，执行系统的1号进程。此后系统的控制权就全权交给/sbin/init进程了。

l 初始化系统

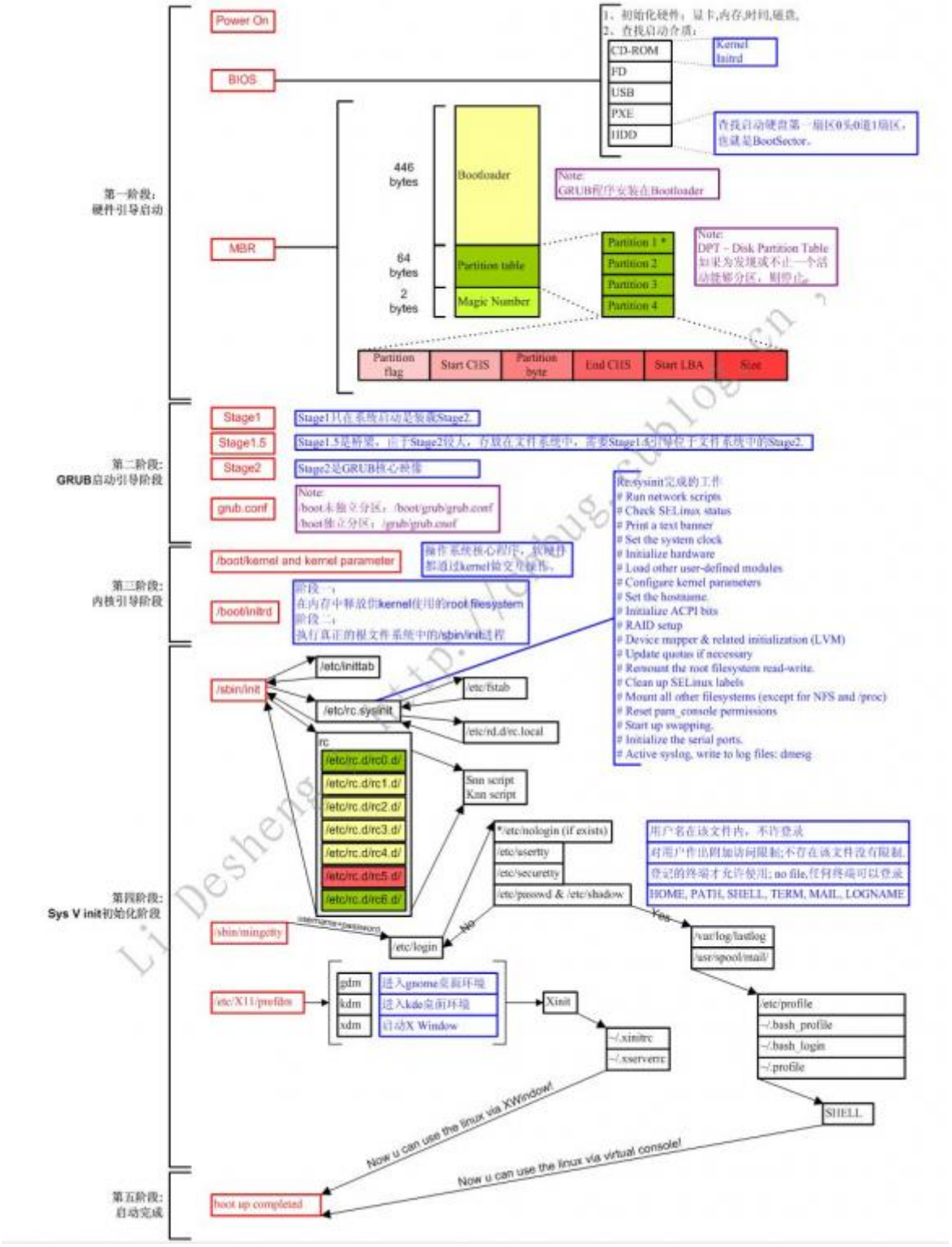
经过千辛万苦的跋涉，我们终于接近黎明的曙光了。接下来就是最后一步了：初始化系统。/sbin/init进程是系统其他所有进程的父进程，当它接管了系统的控制权先之后，它首先会去读取/etc/inittab文件来执行相应的脚本进行系统初始化，如设置键盘、字体，装载模块，设置网络等。主要包括以下工作：

1)、执行系统初始化脚本(/etc/rc.d/rc.sysinit)，对系统进行基本的配置，以读写方式挂载根文件系统及其它文件系统，到此系统算是基本运行起来了，后面需要进行运行级别的确定及相应服务的启动。rc.sysinit所做的事情(不同的Linux发行版，该文件可能有些差异)如下：

（1）获取网络环境与主机类型。首先会读取网络环境设置文件"/etc/sysconfig/network"，获取主机名称与默认网关等网络环境。

（2）测试与载入内存设备/proc及usb设备/sys。除了/proc外，系统会主动检测是否有usb设备，并主动加载usb驱动，尝试载入usb文件系统。
（3）决定是否启动SELinux。
（4）接口设备的检测与即插即用（pnp）参数的测试。
（5）用户自定义模块的加载。用户可以再"/etc/sysconfig/modules/*.modules"加入自定义的模块，此时会加载到系统中。
（6）加载核心的相关设置。按"/etc/sysctl.conf"这个文件的设置值配置功能。
（7）设置系统时间（clock）。
（8）设置终端的控制台的字形。
（9）设置raid及LVM等硬盘功能。
（10）以方式查看检验磁盘文件系统。
（11）进行磁盘配额quota的转换。
（12）重新以读取模式载入系统磁盘。
（13）启动quota功能。
（14）启动系统随机数设备（产生随机数功能）。
（15）清楚启动过程中的临时文件。
（16）将启动信息加载到"/var/log/dmesg"文件中。
当/etc/rc.d/rc.sysinit执行完后，系统就可以顺利工作了，只是还需要启动系统所需要的各种服务，这样主机才可以提供相关的网络和主机功能，因此便会执行下面的脚本。
2)、执行/etc/rc.d/rc脚本。该文件定义了服务启动的顺序是先K后S，而具体的每个运行级别的服务状态是放在/etc/rc.d/rc*.d（*=0~6）目录下，所有的文件均是指向/etc/init.d下相应文件的符号链接。rc.sysinit通过分析/etc/inittab文件来确定系统的启动级别，然后才去执行/etc/rc.d/rc*.d下的文件。
/etc/init.d-> /etc/rc.d/init.d
/etc/rc ->/etc/rc.d/rc
/etc/rc*.d ->/etc/rc.d/rc*.d
/etc/rc.local-> /etc/rc.d/rc.local
/etc/rc.sysinit-> /etc/rc.d/rc.sysinit
也就是说，/etc目录下的init.d、rc、rc*.d、rc.local和rc.sysinit均是指向/etc/rc.d目录下相应文件和文件夹的符号链接。我们以启动级别3为例来简要说明一下。
/etc/rc.d/rc3.d目录，该目录下的内容全部都是以 S 或 K 开头的链接文件，都链接到"/etc/rc.d/init.d"目录下的各种shell脚本。S表示的是启动时需要start的服务内容，K表示关机时需要关闭的服务内容。/etc/rc.d/rc*.d中的系统服务会在系统后台启动，如果要对某个运行级别中的服务进行更具体的定制，通过chkconfig命令来操作，或者通过setup、ntsys、system-config-services来进行定制。如果我们需要自己增加启动的内容，可以在init.d目录中增加相关的shell脚本，然后在rc*.d目录中建立链接文件指向该shell脚本。这些shell脚本的启动或结束顺序是由S或K字母后面的数字决定，数字越小的脚本越先执行。例如，/etc/rc.d/rc3.d /S01sysstat就比/etc/rc.d/rc3.d /S99local先执行。
3)、执行用户自定义引导程序/etc/rc.d/rc.local。其实当执行/etc/rc.d/rc3.d/S99local时，它就是在执行/etc/rc.d/rc.local。S99local是指向rc.local的符号链接。就是一般来

说，自定义的程序不需要执行上面所说的繁琐的建立shell增加链接文件的步骤，只需要将命令放在rc.local里面就可以了，这个shell脚本就是保留给用户自定义启动内容的。
4)、完成了系统所有的启动任务后，linux会启动终端或X-Window来等待用户登录。
tty1,tty2,tty3...这表示在运行等级1，2，3，4的时候，都会执行"/sbin/mingetty"，而且执行了6个，所以linux会有6个纯文本终端，mingetty就是启动终端的命令。
除了这6个之外还会执行"/etc/X11/prefdm-nodaemon"这个主要启动X-Window
至此，系统就启动完毕了。以上分析不到的地方还请各位大虾不吝指正。
关于Linux的其他分析内容下次再继续写。
最后附上一张非常完整的系统启动流程图，适合各个水平阶段的读者。



启动的服务不同：
运行级别：0-6
0：halt
1: single user mode, 直接以管理员身份切入， s,S,single
2：multi user mode, no NFS
3: multi user mode, text mode
4：reserved
5: multi user mode, graphic mode
6: reboot
详解启动过程
bootloader(MBR)
LILO: LInux LOader
GRUB: GRand Unified Bootloader
Stage1: MBR
Stage1_5:
Stage2: /boot/grub/
grub.conf
default=0 # 设定默认启动的title的编号，从0开始
timeout=5 # 等待用户选择的超时时长，单位是秒
splashimage=(hd0,0)/grub/splash.xpm.gz # grub的背景图片
hiddenmenu # 隐藏菜单
password redhat
password --md5 1HKXJ51$B9Z8A.X//XA.AtzU1.KuG.
title Red Hat Enterprise Linux Server (2.6.18-308.el5) # 内核标题，或操作系统名称，字符串，可自由修改
root (hd0,0) # 内核文件所在的设备；对grub而言，所有类型硬盘一律hd，格式为(hd#,N)；hd#, #表示第几个磁盘；最后的N表示对应磁盘的分区；
kernel /vmlinuz-2.6.18-308.el5 ro root=/dev/vol0/root rhgb quiet # 内核文件路径，及传递给内核的参数
initrd /initrd-2.6.18-308.el5.img # ramdisk文件路径
password --md5 1HKXJ51$B9Z8A.X//XA.AtzU1.KuG.
title Install Red Hat Enterprise Linux 5
root (hd0,0)
kernel /vmlinuz-5 ks=http://172.16.0.1/workstation.cfg ksdevice=eth0 noipv6
initrd /initrd-5
password --md5 1FSUEU/$uhUUc8USBK5QAXc.BfW4m.

查看运行级别：
runlevel:

who -r
查看内核release号：
uname -r

安装grub stage1：
grub
grub> root (hd0,0)
grub> set (hd0)
安装grub第二种方式：
grub-install --root-directory=/path/to/boot's_parent_dir /PATH/TO/DEVICE

grub> find
grub> root (hd#,N)
grub> kernel /PATH/TO/KERNEL_FILE
grub> initrd /PATH/TO/INITRD_FILE
grub> boot

Kernel初始化的过程：
1、设备探测
2、驱动初始化（可能会从initrd（initramfs）文件中装载驱动模块）
3、以只读挂载根文件系统；
4、装载第一个进程init（PID：1）

/sbin/init：（/etc/inittab）
upstart: ubuntu, d-bus, event-driven
systemd:
id:runlevels:action:process
id: 标识符
runlevels: 在哪个级别运行此行；
action: 在什么情况下执行此行；
process: 要运行程序；
id:3:initdefault:
si::sysinit:/etc/rc.d/rc.sysinit

ACTION:
initdefault: 设定默认运行级别
sysinit: 系统初始化
wait: 等待级别切换至此级别时执行
respawn: 一旦程序终止，会重新启动

/etc/rc.d/rc.sysinit完成的任务：

1、激活udev和selinux;
2、根据/etc/sysctl.conf文件，来设定内核参数;
3、设定时钟时钟;
4、装载键盘映射;
5、启用交换分区;
6、设置主机名;
7、根文件系统检测，并以读写方式重新挂载;
8、激活RAID和LVM设备;
9、启用磁盘配额;
10、根据/etc/fstab，检查并挂载其它文件系统;
11、清理过期的锁和PID文件;

```
for I in /etc/rc3.d/K*; do
$I stop
done
for I in /etc/rc3.d/S*; do
$I start
done
```

##: 关闭或启动的优先次序，数据越小越优先被选定
先关闭以K开头的服务，后启动以S开头的服务;

内核设计风格:
RedHat, SUSE
核心：动态加载 内核模块
内核：/lib/modules/"内核版本号命令的目录"/
vmlinuz-2.6.32
/lib/modules/2.6.32/
RedHat5: ramdisk-->initrd
RedHat6: ramfs-->initramfs

单内核：Linux (LWP)
核心：ko(kernel object)

so()

微内核：Windows, Solaris (线程)
chroot: chroot /PATH/TO/TEMPROOT [COMMAND...]
chroot /test/virrrot /bin/bash

ldd /PATH/TO/BINARY_FILE：显示二进制文件所依赖的共享库

MBR（bootloader）--> Kernel --> initrd(initramfs) --> (ROOTFS) --> /sbin/init(/etc/inittab)
/etc/inittab, /etc/init/*.conf
upstart

init /etc/inittab
id:runlevels:action:process
id:5:initdefault:
si::sysinit:/etc/rc.d/rc.sysinit
OS初始化
l0:0:wait:/etc/rc.d/rc 0
rc0.d/
K*
stop
S*
start

/etc/rc.d/init.d, /etc/init.d
服务类脚本：
start

SysV： /etc/rc.d/init.d
start|stop|restart|status
reload|configtest