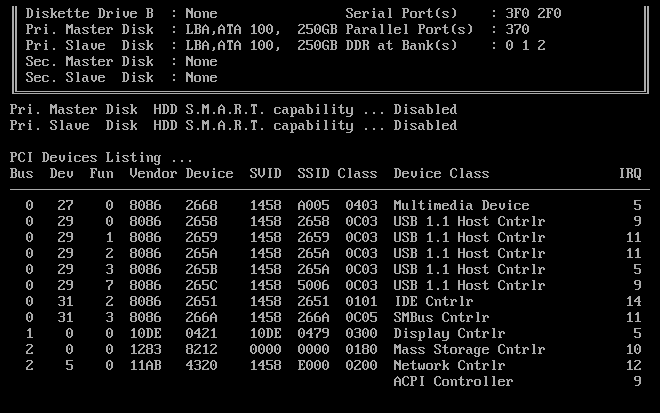


这块芯片里的程序叫做"基本輸出輸入系統"（Basic Input/Output System），简称为BIOS。

1.1 硬件自检

BIOS程序首先检查，计算机硬件能否满足运行的基本条件，这叫做"硬件自检"（Power-On Self-Test），缩写为POST。

如果硬件出现问题，主板会发出不同含义的蜂鸣，启动中止。如果没有问题，屏幕就会显示出CPU、内存、硬盘等信息。

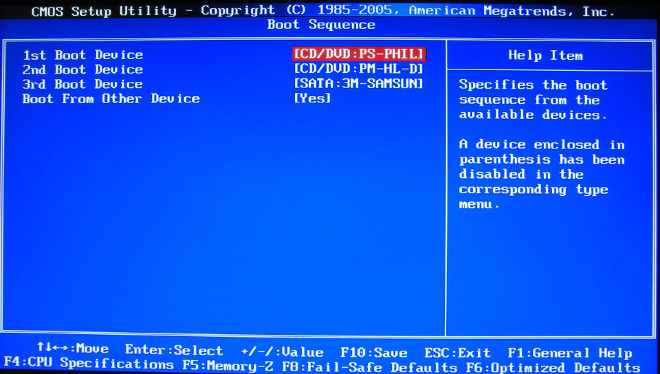


1.2 启动顺序

硬件自检完成后，BIOS把控制权转交给下一阶段的启动程序。

这时，BIOS需要知道，"下一阶段的启动程序"具体存放在哪一个设备。也就是说，BIOS需要有一个外部储存设备的排序，排在前面的设备就是优先转交控制权的设备。这种排序叫做"启动顺序"（Boot Sequence）。

打开BIOS的操作界面，里面有一项就是"设定启动顺序"。



二、第二阶段：主引导记录

BIOS按照"启动顺序"，把控制权转交给排在第一位的储存设备。

这时，计算机读取该设备的第一个扇区，也就是读取最前面的512个字节。如果这512个字节的最后两个字节是0x55和0xAA，表明这个设备可以用于启动；如果不是，表明设备不能用于启动，控制权于是被转交给"启动顺序"中的下一个设备。

这最前面的512个字节，就叫做"主引导记录"（Master boot record，缩写为MBR）。

2.1 主引导记录的结构

"主引导记录"只有512个字节，放不了太多东西。它的主要作用是，告诉计算机到硬盘的哪一个位置去找操作系统。

主引导记录由三个部分组成：

　　（1） 第1-446字节：调用操作系统的机器码。

　　（2） 第447-510字节：分区表（Partition table）。

　　（3） 第511-512字节：主引导记录签名（0x55和0xAA）。

其中，第二部分"分区表"的作用，是将硬盘分成若干个区。

2.2 分区表

硬盘分区有很多好处。考虑到每个区可以安装不同的操作系统，"主引导记录"因此必须知道将控制权转交给哪个区。

分区表的长度只有64个字节，里面又分成四项，每项16个字节。所以，一个硬盘最多只能分四个一级分区，又叫做"主分区"。

每个主分区的16个字节，由6个部分组成：

　　（1） 第1个字节：如果为0x80，就表示该主分区是激活分区，控制权要转交给这个分区。四个主分区里面只能有一个是激活的。

　　（2） 第2-4个字节：主分区第一个扇区的物理位置（柱面、磁头、扇区号等等）。

　　（3） 第5个字节：主分区类型。

　　（4） 第6-8个字节：主分区最后一个扇区的物理位置。

　　（5） 第9-12字节：该主分区第一个扇区的逻辑地址。

　　（6） 第13-16字节：主分区的扇区总数。

最后的四个字节（"主分区的扇区总数"），决定了这个主分区的长度。也就是说，一个主分区的扇区总数最多不超过2的32次方。

如果每个扇区为512个字节，就意味着单个分区最大不超过2TB。再考虑到扇区的逻辑地址也是32位，所以单个硬盘可利用的空间最大也不超过2TB。如果想使用更大的硬盘，只有2个方法：一是提高每个扇区的字节数，二是增加扇区总数。

三、第三阶段：硬盘启动

这时，计算机的控制权就要转交给硬盘的某个分区了，这里又分成三种情况。

3.1 情况A：卷引导记录

上一节提到，四个主分区里面，只有一个是激活的。计算机会读取激活分区的第一个扇区，叫做"卷引导记录"（Volume boot record，缩写为VBR）。

"卷引导记录"的主要作用是，告诉计算机，操作系统在这个分区里的位置。然后，计算机就会加载操作系统了。

3.2 情况B：扩展分区和逻辑分区

随着硬盘越来越大，四个主分区已经不够了，需要更多的分区。但是，分区表只有四项，因此规定有且仅有一个区可以被定义成"扩展分区"（Extended partition）。

所谓"扩展分区"，就是指这个区里面又分成多个区。这种分区里面的分区，就叫做"逻辑分区"（logical partition）。

计算机先读取扩展分区的第一个扇区，叫做"扩展引导记录"（Extended boot record，缩写为EBR）。它里面也包含一张64字节的分区表，但是最多只有两项（也就是两个逻辑分区）。

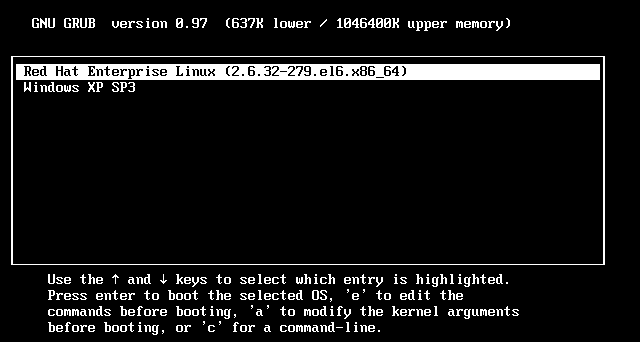
计算机接着读取第二个逻辑分区的第一个扇区，再从里面的分区表中找到第三个逻辑分区的位置，以此类推，直到某个逻辑分区的分区表只包含它自身为止（即只有一个分区项）。因此，扩展分区可以包含无数个逻辑分区。

但是，似乎很少通过这种方式启动操作系统。如果操作系统确实安装在扩展分区，一般采用下一种方式启动。

3.3 情况C：启动管理器

在这种情况下，计算机读取"主引导记录"前面446字节的机器码之后，不再把控制权转交给某一个分区，而是运行事先安装的"启动管理器"（boot loader），由用户选择启动哪一个操作系统。

Linux环境中，目前最流行的启动管理器是Grub。



四、第四阶段：操作系统

控制权转交给操作系统后，操作系统的内核首先被载入内存。

以Linux系统为例，先载入/boot目录下面的kernel。内核加载成功后，第一个运行的程序是/sbin/init。它根据配置文件（Debian系统是/etc/initab）产生init进程。这是Linux启动后的第一个进程，pid进程编号为1，其他进程都是它的后代。

然后，init线程加载系统的各个模块，比如窗口程序和网络程序，直至执行/bin/login程序，跳出登录界面，等待用户输入用户名和密码。

至此，全部启动过程完成。

（完）

半年前，我写了《计算机是如何启动的？》，探讨BIOS和主引导记录的作用。

那篇文章不涉及操作系统，只与主板的板载程序有关。今天，我想接着往下写，探讨操作系统接管硬件以后发生的事情，也就是操作系统的启动流程。

这个部分比较有意思。因为在BIOS阶段，计算机的行为基本上被写死了，程序员可以做的事情并不多；但是，一旦进入操作系统，程序员几乎可以定制所有方面。所以，这个部分与程序员的关系更密切。

我主要关心的是Linux操作系统，它是目前服务器端的主流操作系统。下面的内容针对的是Debian发行版，因为我对其他发行版不够熟悉。

第一步、加载内核

操作系统接管硬件以后，首先读入 /boot 目录下的内核文件。

以我的电脑为例，/boot 目录下面大概是这样一些文件：

　　$ ls /boot

　　config-3.2.0-3-amd64

　　config-3.2.0-4-amd64

　　grub

　　initrd.img-3.2.0-3-amd64

　　initrd.img-3.2.0-4-amd64

　　System.map-3.2.0-3-amd64

　　System.map-3.2.0-4-amd64

　　vmlinuz-3.2.0-3-amd64

　　vmlinuz-3.2.0-4-amd64

第二步、启动初始化进程

内核文件加载以后，就开始运行第一个程序 /sbin/init，它的作用是初始化系统环境。

由于init是第一个运行的程序，它的进程编号（pid）就是1。其他所有进程都从它衍生，都是它的子进程。

第三步、确定运行级别

许多程序需要开机启动。它们在Windows叫做"服务"（service），在Linux就叫做"守护进程"（daemon）。

init进程的一大任务，就是去运行这些开机启动的程序。但是，不同的场合需要启动不同的程序，比如用作服务器时，需要启动Apache，用作桌面就不需要。Linux允许为不同的场合，分配不同的开机启动程序，这就叫做"运行级别"（runlevel）。也就是说，启动时根据"运行级别"，确定要运行哪些程序。

Linux预置七种运行级别（0-6）。一般来说，0是关机，1是单用户模式（也就是维护模式），6是重启。运行级别2-5，各个发行版不太一样，对于Debian来说，都是同样的多用户模式（也就是正常模式）。

init进程首先读取文件 /etc/inittab，它是运行级别的设置文件。如果你打开它，可以看到第一行是这样的：

　　id:2:initdefault:

initdefault的值是2，表明系统启动时的运行级别为2。如果需要指定其他级别，可以手动修改这个值。

那么，运行级别2有些什么程序呢，系统怎么知道每个级别应该加载哪些程序呢？......回答是每个运行级别在/etc目录下面，都有一个对应的子目录，指定要加载的程序。

　　/etc/rc0.d

　　/etc/rc1.d

　　/etc/rc2.d

　　/etc/rc3.d

　　/etc/rc4.d

　　/etc/rc5.d

　　/etc/rc6.d

上面目录名中的"rc"，表示run command（运行程序），最后的d表示directory（目录）。下面让我们看看 /etc/rc2.d 目录中到底指定了哪些程序。

　　$ ls /etc/rc2.d

　　README

　　S01motd

　　S13rpcbind

　　S14nfs-common

　　S16binfmt-support

　　S16rsyslog

　　S16sudo

　　S17apache2

　　S18acpid

　　...

可以看到，除了第一个文件README以外，其他文件名都是"字母S+两位数字+程序名"的形式。字母S表示Start，也就是启动的意思（启动脚本的运行参数为start），如果这个位置是字母K，就代表Kill（关闭），即如果从其他运行级别切换过来，需要关闭的程序（启动脚本的运行参数为stop）。后面的两位数字表示处理顺序，数字越小越早处理，所以第一个启动的程序是motd，然后是rpcbing、nfs......数字相同时，则按照程序名的字母顺序启动，所以rsyslog会先于sudo启动。

这个目录里的所有文件（除了README），就是启动时要加载的程序。如果想增加或删除某些程序，不建议手动修改 /etc/rcN.d 目录，最好是用一些专门命令进行管理（参考这里和这里）。

第四步、加载开机启动程序

前面提到，七种预设的"运行级别"各自有一个目录，存放需要开机启动的程序。不难想到，如果多个"运行级别"需要启动同一个程序，那么这个程序的启动脚本，就会在每一个目录里都有一个拷贝。这样会造成管理上的困扰：如果要修改启动脚本，岂不是每个目录都要改一遍？

Linux的解决办法，就是七个 /etc/rcN.d 目录里列出的程序，都设为链接文件，指向另外一个目录 /etc/init.d ，真正的启动脚本都统一放在这个目录中。init进程逐一加载开机启动程序，其实就是运行这个目录里的启动脚本。

下面就是链接文件真正的指向。

　　$ ls -l /etc/rc2.d

　　README

　　S01motd -> ../init.d/motd

　　S13rpcbind -> ../init.d/rpcbind

　　S14nfs-common -> ../init.d/nfs-common

　　S16binfmt-support -> ../init.d/binfmt-support

　　S16rsyslog -> ../init.d/rsyslog

　　S16sudo -> ../init.d/sudo

　　S17apache2 -> ../init.d/apache2

　　S18acpid -> ../init.d/acpid

　　...

这样做的另一个好处，就是如果你要手动关闭或重启某个进程，直接到目录 /etc/init.d 中寻找启动脚本即可。比如，我要重启Apache服务器，就运行下面的命令：

　　$ sudo /etc/init.d/apache2 restart

/etc/init.d 这个目录名最后一个字母d，是directory的意思，表示这是一个目录，用来与程序 /etc/init 区分。

第五步、用户登录

开机启动程序加载完毕以后，就要让用户登录了。

一般来说，用户的登录方式有三种：

　　（1）命令行登录

　　（2）ssh登录

　　（3）图形界面登录

这三种情况，都有自己的方式对用户进行认证。

（1）命令行登录：init进程调用getty程序（意为get teletype），让用户输入用户名和密码。输入完成后，再调用login程序，核对密码（Debian还会再多运行一个身份核对程序/etc/pam.d/login）。如果密码正确，就从文件 /etc/passwd 读取该用户指定的shell，然后启动这个shell。

（2）ssh登录：这时系统调用sshd程序（Debian还会再运行/etc/pam.d/ssh ），取代getty和login，然后启动shell。

（3）图形界面登录：init进程调用显示管理器，Gnome图形界面对应的显示管理器为gdm（GNOME Display Manager），然后用户输入用户名和密码。如果密码正确，就读取/etc/gdm3/Xsession，启动用户的会话。

第六步、进入 login shell

所谓shell，简单说就是命令行界面，让用户可以直接与操作系统对话。用户登录时打开的shell，就叫做login shell。

Debian默认的shell是Bash，它会读入一系列的配置文件。上一步的三种情况，在这一步的处理，也存在差异。

（1）命令行登录：首先读入 /etc/profile，这是对所有用户都有效的配置；然后依次寻找下面三个文件，这是针对当前用户的配置。

　　~/.bash\_profile

　　~/.bash\_login

　　~/.profile

需要注意的是，这三个文件只要有一个存在，就不再读入后面的文件了。比如，要是 ~/.bash\_profile 存在，就不会再读入后面两个文件了。

（2）ssh登录：与第一种情况完全相同。

（3）图形界面登录：只加载 /etc/profile 和 ~/.profile。也就是说，~/.bash\_profile 不管有没有，都不会运行。

第七步，打开 non-login shell

老实说，上一步完成以后，Linux的启动过程就算结束了，用户已经可以看到命令行提示符或者图形界面了。但是，为了内容的完整，必须再介绍一下这一步。

用户进入操作系统以后，常常会再手动开启一个shell。这个shell就叫做 non-login shell，意思是它不同于登录时出现的那个shell，不读取/etc/profile和.profile等配置文件。

non-login shell的重要性，不仅在于它是用户最常接触的那个shell，还在于它会读入用户自己的bash配置文件 ~/.bashrc。大多数时候，我们对于bash的定制，都是写在这个文件里面的。

你也许会问，要是不进入 non-login shell，岂不是.bashrc就不会运行了，因此bash 也就不能完成定制了？事实上，Debian已经考虑到这个问题了，请打开文件 ~/.profile，可以看到下面的代码：

　　if [ -n "$BASH\_VERSION" ]; then

　　　　if [ -f "$HOME/.bashrc" ]; then

　　　　　　. "$HOME/.bashrc"

　　　　fi

　　fi

上面代码先判断变量 $BASH\_VERSION 是否有值，然后判断主目录下是否存在 .bashrc 文件，如果存在就运行该文件。第三行开头的那个点，是source命令的简写形式，表示运行某个文件，写成"source ~/.bashrc"也是可以的。

因此，只要运行～/.profile文件，～/.bashrc文件就会连带运行。但是上一节的第一种情况提到过，如果存在～/.bash\_profile文件，那么有可能不会运行～/.profile文件。解决这个问题很简单，把下面代码写入.bash\_profile就行了。

　　if [ -f ~/.profile ]; then

　　　　. ~/.profile

　　fi

这样一来，不管是哪种情况，.bashrc都会执行，用户的设置可以放心地都写入这个文件了。

Bash的设置之所以如此繁琐，是由于历史原因造成的。早期的时候，计算机运行速度很慢，载入配置文件需要很长时间，Bash的作者只好把配置文件分成了几个部分，阶段性载入。系统的通用设置放在 /etc/profile，用户个人的、需要被所有子进程继承的设置放在.profile，不需要被继承的设置放在.bashrc。

顺便提一下，除了Linux以外， Mac OS X 使用的shell也是Bash。但是，它只加载.bash\_profile，然后在.bash\_profile里面调用.bashrc。而且，不管是ssh登录，还是在图形界面里启动shell窗口，都是如此。

