第7章　开机和关机

因为操作系统正在变得越来越复杂，开机引导和关机下电的过程也越来越智能化。从简单的DOS系统转移到Windows NT系统的人们已经亲身感受到了这些变化——这已不仅仅是核心操作系统的启动引导和关闭了，还包括必须要同时启动或者关闭相当数量的服务项目。类似于Windows NT，Linux系统作为启动过程的一个组成部分需要打开的服务项目也是数量极大的。

在本章中，我们将逐步介绍Linux操作系统环境的启动和关闭过程。我们将讨论使这个过程能够自动进行的脚本程序，以及这个过程的哪些部分可以被修改。

警告　如果是在一个现实中的系统上学习应用本章中的知识，多多运用常识。当试着对启动和关机脚本程序进行修改的时候，要记住所做的修改可能会造成你的系统不能正常工作，而且无法采用重启动的方法恢复。不要在已经正常运转的系统上实验新的设置；对你准备修改的文件全部进行备份；最重要的是，在手边准备一张引导盘以防不测。

7.1　LILO

LILO是Linux Loader（Linux加载程序）的缩写，是一个引导管理程序。它可以引导多个操作系统，前提是每一种操作系统都只存在于它们自己的硬盘分区里（在基于个人电脑的系统里，引导扇区必须在前1024个磁道上）。除了引导多个操作系统之外，使用LILO还可以选择引导时使用不同的内核配置。这样，在正式对内核升级之前就可以先试用新的配置，十分方便。

LILO的概念是很容易理解的：准备一个配置文件（/etc/lilo.conf），定义哪一个硬盘分区是可以引导的（如果使用的是Linux操作系统，还可以定义引导哪一个内核）。当程序/sbin/lilo开始运行的时候，会记下这个分区信息，然后把必要的数据再写到引导扇区上，按照配置文件中的定义提供引导参数。在机器引导的时候，屏幕上会显示一个提示符（通常是 lilo :），这时可以选择使用哪一种操作系统（一般说来，倒计时结束时系统会自动选择一个操作系统来引导）。LILO从选中的分区加载必要的代码，然后把控制权全部移交给它。

注意　如果你对NT的引导过程比较熟悉，可以把 LILO想像成类似于 OS loader（NTLDR）的东西。同样地，还可以把 LILO的配置文件/etc/lilo.conf想像成类似于BOOT.INI的东西。

7.2　配置LILO

LILO的配置文件是/etc/lilo.conf。大多数情况下，用户不必对这个文件大动干戈。如果你真的需要对这个文件进行修改，其选项都相当简单明了。我们从查看一个示范性的配置文件开始。这里给出的文件可能会与你缺省的lilo.conf文件差不多。如下所示：

```
boot=/dev/hda
prompt
timeout=50
image=/boot/vmlinuz-2.2.5-15
        label=linux
        root=/dev/hda2
        read-only
other = /dev/hda1
        label = dos
        table = /dev/hda
```

第一行，boot = /dev/hda，告诉LILO把引导扇区写到哪里。它通常都是引导硬盘（对 IDE 硬盘来说就是/dev/hda；对SCSI硬盘来说就是/dev/sda）的第一个扇区。这个扇区更常用的名字是主引导扇区（Master Boot Record，MBR）。MBR的作用是告诉电脑的设计人员为了启动操作系统应该先加载哪些东西。加载以后，保存在 MBR中的程序会全面接管引导过程。

如果想使用另外一个程序（比如说 NT OS loader）来管理MBR，可以通过定义把引导扇区写到另外一个分区中去。控制将从 MBR中的代码转移到这个分区中去。

现在回的我们的配置文件上来，接下来的命令是 prompt。这条指令告诉LILO在引导的时候显示提示符“ lilo : ”。在这个提示符下，用户可以直接输入准备启动的引导映象的名字，或者按下TAB键列出可供选择的参数。缺省情况下，如果没有事先定义倒计时命令， LILO将保持等待用户输入的状态。

命令timeout = 50告诉LILO在选择缺省的引导映象开始引导过程之前要等待 50个十分之一秒（即5秒）的倒计时。

接下来的是一个小数据块，开头的一行是：

```
image = /boot/vmlinuz-2.2.5-15
```

我们定义了一个特定的引导映象。这是第一个数据块，它将是缺省的引导映象。被引导的映象是文件/boot/vmlinuz-2.2.5-15，这是一个Linux操作系统的内核。在数据块的内部有一行 label = linux，这是一个显示在“ lilo : ”提示符处引导选项清单中的名字。

在数据块中还有下面这一行：

```
root = /dev/hda2
```

它告诉LILO文件/boot/vmlinuz-2.2.5-15存放在哪个硬盘的分区上。

如果你不清楚哪个分区中保存着内核，可以先进入到内核所在的子目录，再输入：

```
df .
```

屏幕将会显示类似于如下所示的内容：

```
[root@ford /boot]# df .
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda2               108870     56119     47129  54% /
```

其中第一列数据项以 /dev打头，这是内核所保存在的设备名。在上面的例子中，内核（保存在/boot子目录中）在/dev/hda2分区上（关于硬盘分区的进一步内容请参考第 8章）。

数据块中的最后一个参数是 read-only，它告诉LILO在启动内核的时候按照只读权限挂装根文件系统。这是十分必要的，根文件系统会因此在引导过程中检查自己是否损坏。检查一

旦通过，根文件系统会以读/写权限再次自动挂装。

第一个数据块就结束了。另外一个新的数据块需要以“ image = ”，或者“ other = ”开头。在我们的例子里，下一个数据块是以“ other = /dev/hda1 ”开头的。就这个例子来说，在它打开的时候，/dev/hda1分区被设置为一个完全的DOS安装（它当然也有可能是Windows 95或者Windows NT。）与前一个数据块类似，我们还看到了一个“ label = ”语句；但是两个数据块之间的相同之处也就到此为止了。接下来的一行是 table = /dev/hda，它定义了在我们试图引导的硬盘分区上（/dev/hda1）操作系统可以在哪里找到表信息。

注意 虽然DOS和Windows是双引导配置中最常见的“其他的”操作系统，但是它们并不是你唯一的选择。只要是能够对分区进行读写的操作系统，都可以用 LILO来引导。

7.2.1 附加的LILO参数

除了刚才讨论过的参数以外，在 lilo.conf文件中还可以有一些其他的参数。本小结将讨论最常用的那些。通过它们，系统管理员就可以轻松地配置好系统中的大部分（如果不是全部的话）参数。对那些少见的特殊配置情况，可以阅读 lilo.conf的使用手册页；其中给出了全部可用参数的概括说明。

1. lilo.conf文件中的全局参数

下面介绍的全局参数的作用范围是整个配置文件，不仅仅局限于某个特定的数据块。

default = *name*	定义将要引导的缺省操作系统。如果没有此语句，第一个数据块就被认为是缺省值
message = *message-file*	“ lilo : ”提示符出现之前可以在屏幕上先显示一段信息（它的内容保存在message-file文件里）。这段信息的长度不得超过 64K；而且如果它发生了变化，映射文件也要重建（请参考后面的“运行 LILO ”一节）
prompt	强制LILO显示“ lilo : ”提示符并等待用户作出响应。如果定义了这个参数但是没有定义timeout参数，重启动就无法自动进行
timeout = *deciseconds*	以十分之一秒为单位定义一个倒计时数字，如果在倒计时期间提示符下没有输入，则使用缺省的引导映象继续完成引导过程

2. 数据块参数

本小节列出的参数其作用范围仅限于它们所在的数据块中，不同的数据块定义了不同的可引导操作系统。

image = *image file*	定义引导Linux操作系统的内核映象文件
other = *image file*	定义引导其他各种操作系统的映象文件
table = *device name*	说明本数据块的硬盘分区表保存在哪一个设备上
label = *name*	定义所在数据块的名称，当用户在“ lilo : ”提示符下查看引导菜单时，看到的就是这条语句定义的名称
password = *password*	表示只有当用户正确地输入了口令后才能引导本数据块中定义的操作系统
restricted	如果有命令行参数需要传递到内核，用户必须输入正确的口令（当需要口令来保护单用户模式的操作环境时，这个参数尤其重要）

注意 如果你打算设置password或者restricted参数，别忘了把lilo.conf文件的访问权限设置为只有根用户才能够读它。这个文件缺省的访问权限设置值是只有根用户才

能修改，但是任何人都可以读。设置根用户只读权限的命令如下所示：

```
chmod 600 /etc/lilo.conf
```

3. 内核参数

参数可以从LILO传递到内核。其中包括请求引导至单用户模式的参数（这个功能最常见的用法）。如果你只是偶尔才需要向内核传递参数（比如请求进入单用户模式），可以在“lilo:”提示符下输入那些参数。比如说，你可以在“ lilo：”提示符下输入linux s启动在label = 语句中定义为linux的内核映象，并把参数s传递给它。最终是引导Linux进入单用户模式。很明显，这些参数的作用范围仅限于Linux内核。

append = string	把string添加到用户输入的每一个命令行参数后面
literal = string	类似于append，但是string并不是附加到用户通过命令行传递的参数后面，而是完全代替之
read-only	定义根文件系统需要挂装为只读属性。在内核加载完成并使用 fsck工具程序对根文件系统进行检查之后，再把它重新挂装为读 /写属性

7.2.2 添加引导用的新内核

在第10章中，我们将讨论编译新内核并使用它进行开机引导的各个步骤。这些步骤中就包括有在lilo.conf文件中增加一些数据项，这样 LILO就能够了解新的内核。更重要的是，添加的数据项可以让你保留现有的运转正常的配置，这就可以应付新内核运转不合要求的问题。你总是有机会重启动系统并选择运转正常的老内核。

为了讨论的方便，我们假设你已经编译好 2.3.12版的内核，打算使用由 Alex Paterson博士编写的新Toxygene Network Protocol（Toxygene网络协议）提高系统的安全性。编译好的新内核vmlinuz-2.3.12保存在/boot子目录中。第一个步骤是把有关的信息添加到 /etc/lilo.conf文件中去。这个新的数据块看起来应该和下面的内容差不多：

```
image=/boot/vmlinuz-2.3.12
    label=linux-2.3.12
    root=/dev/hda2
    read-only
```

如果我们只是把这个数据块添加到 /etc/lilo.conf文件的末尾，那么vmlinuz-2.3.12还不会是缺省引导的内核。要想让它成为缺省引导的内核，我们有两种办法：一是把这个数据块移动到第一个数据块的位置，二是使用 default命令。就这个例子来说，我们准备使用 default命令。因为这个命令不能是某个数据块中的语句，我们把它插入到文件的开头，如下所示：

```
default = vmlinuz-2.3.12
```

即大功告成。下面是/etc/lilo.conf文件最终的内容：

```
default=vmlinux-2.3.12
boot=/dev/hda
prompt
timeout=50
image=/boot/vmlinuz-2.2.5-15
        label=linux
        root=/dev/hda2
```

```
        read-only
other = /dev/hda1
        label = dos
        table = /dev/hda
image=/boot/vmlinuz-2.3.12
        label=linux-2.3.12
        root=/dev/hda2
        read-only
```

在/etc/lilo.conf文件编辑完成并存盘之后，接下来的工作就是运行 LILO了。

7.3 运行LILO

你应该已经很了解/etc/lilo.conf文件了，现在是运行LILO引导管理程序的时候，我们马上就让它开工。这个过程一般说来不会出什么问题。

大多数情况下，直接运行 LILO就可以了，不需要再加上命令参数。结果看起来应该和下面这样差不多：

```
[ root@ford / boot ] # lilo
Added linux *
Added dos
```

LILO从/etc/lilo.conf文件中了解必要的信息，然后将它写入到相应的引导扇区中去。LILO的使用手册页中给出了许多命令行参数。但是它们中的大多数在 lilo.conf文件中都有对应的数据项，因此我们就不重复讨论了。下面的表格列出了 lilo命令最重要的那些参数。

LILO命令的参数	说　明
-t	对配置情况进行测试，但是不真正加载。如果单用 -t参数并不会告诉你很多东西，但是如果和 -v参数（见本表下面的定义）一起使用，就可以看到LILO到底做了些什么
-C *config-file*	缺省情况下，LILO会查找/etc/lilo.conf作为其配置文件。使用这个命令行参数，你可以为它另外指定一个配置文件
-r *root-directory*	通知LILO在开始任何操作之前使用chroot命令把根用户目录切换到指定的子目录去。chroot命令会把根用户目录切换到 root-directory定义的子目录去。这个参数的典型用法是通过软盘引导开机，以便修复发生故障的系统（举例来说，如果你是从软盘引导开机的，并且把根文件系统挂装在/mnt子目录下，就可能需要使用lilo -r /mnt命令来运行LILO）
-v	让LILO详细报告它执行的每一步操作

7.4 开机引导的步骤

在本小节中，假设你已经熟悉其他操作系统的引导过程，了解硬件的自检引导步骤。下面我们从Linux操作系统的引导加载程序开始（对个人电脑 PC而言通常是LILO）。

- **加载内核**　LILO启动之后，如果你选择了Linux作为准备引导的操作系统，第一个被加载的东西就是内核。请记住：此时的计算机内存中还不存在任何操作系统；而 PC（因为它们天然的设计缺陷）还没有办法存取机器上全部的内存。因此，内核就必须完整地加载到可用RAM的第一个兆字节之内。为了实现这个目的，内核是被压缩了的。这个文件的头部包含着必要的代码，先设置 CPU进入安全模式（以此解除内存限制）再对内核的剩余部分进行解压缩。

- **执行内核** 内核在内存中解压缩之后，它就可以开始运行了。此时的内核只知道它本身内建的各种功能，也就是说被编译为模块的内核部分还不能使用。最最基本的，内核必须有足够的代码设置它自己的虚拟内存子系统和根文件系统（通常就是 ext2文件系统）。一旦内核启动运行，对硬件的检测就会决定需要对哪些设备驱动程序进行初始化。从这里开始，内核就能够挂装根文件系统（这个过程类似于 Windows识别并存取C:盘的过程）内核挂装了根文件系统之后，将启动并运行一个叫做 init的程序。

注意 我们在这里故意略去了 Linux内核启动的许多细节；这些细节只有内核开发人员才感兴趣。如果你好奇的话，可以访问 http：//www.redhat.com:8080地址处的“ Kernel Hackers Guide（内核黑客指南）”。

- **init进程** init进程（我们将在第9章中更详细地讨论它）是非内核进程中第一个被启动运行的；因此它的进程编号 PID的值总是1。init读它的配置文件/etc/inittab，决定它需要启动的运行级别（ runlevel）。从根本上说，运行级别规定了整个系统的行为。每个级别（分别由0到6的整数表示）满足特定的目的。如果定义了 initdefault级别，这个值就直接被选中；否则需要由用户输入一个代表运行级别的数值。

代表运行级别的数值如下所示：

0	停止系统运行
1	进入单用户模式（不激活网络功能）
2	多用户模式，但是没有NFS
3	完全的多用户模式（正常操作）
4	保留
5	与运行级别3相当，但是使用 X-Windows登录程序代替文本登录程序
6	重启动系统

输入代表运行级别的数字之后， init根据/etc/inittab文件中的定义执行一个命令脚本程序。缺省的运行级别取决于在安装阶段对登录程序的选择：是使用基于文本的还是使用基于 X-Windows的登录程序。

7.4.1 rc命令脚本程序

我们在上一小节中指出：当运行级别发生改变时，将由 /etc/inittab文件定义需要运行哪一个命令脚本程序。这些命令脚本程序负责启动或者停止该运行级别特定的各种服务。

因为需要管理的服务数量很多，因此需要使用 rc命令脚本程序。其中最主要的一个是/etc/rc.d/rc，它负责为每一个运行级别按照正确的顺序调用相应的命令脚本程序。我们可以想像，这样一个命令脚本程序很容易变得难以控制！为了防止这类事件的发生，需要使用精心设计的方案。

对每一个运行级别来说，在 /etc/rc.d子目录中都有一个对应的下级目录。这些运行级别的下级子目录的命名方法是 rcX.d，其中的X就是代表运行级别的数字。比如说，运行级别 3的全部命令脚本程序都保存在/etc/rc.d/rc3.d子目录中。

在各个运行级别的子目录中，都建立有到 /etc/rc.d/init.d子目录中命令脚本程序的符号链接。但是这些符号链接并不使用命令脚本程序在 /etc/rc.d/init.d子目录中原来的名字，如果命令脚本程序是用来启动一个服务的，其符号链接的名字就以字母 S打头；如果命令脚本程序是

用来关闭一个服务的，其符号链接的名字就以字母 K打头（符号链接的内容请参考第 6章）。

许多情况下，这些命令脚本程序的执行顺序都很重要（如果没有先配置网络接口，就没有办法使用DNS服务解析主机名！）为了安排它们执行的顺序，在字母 S或者K的后面紧跟着一个两位数字。数值小的在数值大的前面执行；比如 /etc/rc.d/rc3.d/S50inet就会在/etc/rc.d/rc3.d/S55named之前执行（S50inet配置网络设置，S55named启动DNS服务器）。

存放在/etc/rc.d/init.d子目录中被符号链接上的命令脚本程序是真正的实干家；实际上是它们完成了启动或者停止各种服务的操作过程。当 /etc/rc.d/rc运行通过每个特定的运行级别子目录的时候，它会根据数字的顺序依次调用各个命令脚本程序执行。它先运行以字母 K打头的命令脚本程序，然后再运行以字母 S打头的命令脚本程序。对以字母 K打头的命令脚本程序来说，会传递stop参数；类似地，对以字母S打头的命令脚本程序来说，会传递 start参数。

我们来看看etc/rc.d/rc2.d子目录中都一些什么：

```
[root@ford rc2.d]# ls -l
total 0
lrwxrwxrwx   1 root     root           15 Aug 11  1998 K15httpd -> ../init.d/httpd
lrwxrwxrwx   1 root     root           15 Jul 29  1998 K15sound -> ../init.d/sound
lrwxrwxrwx   1 root     root           16 Jun 10 09:36 K20rstatd -> ../init.d/rstatd
lrwxrwxrwx   1 root     root           17 Jul 29  1998 K20rusersd -> ../init.d/rusersd
lrwxrwxrwx   1 root     root           15 Jul 29  1998 K20rwhod -> ../init.d/rwhod
lrwxrwxrwx   1 root     root           16 Sep 27  1998 K30ypbind -> ../init.d/ypbind
lrwxrwxrwx   1 root     root           19 Sep 27  1998 K34yppasswdd -> ../init.d/yppasswdd
lrwxrwxrwx   1 root     root           13 Jul 29  1998 K35smb -> ../init.d/smb
lrwxrwxrwx   1 root     root           16 Sep 27  1998 K35ypserv -> ../init.d/ypserv
lrwxrwxrwx   1 root     root           15 Feb 22 22:20 K45named -> ../init.d/named
lrwxrwxrwx   1 root     root           14 Jul 29  1998 K50inet -> ../init.d/inet
lrwxrwxrwx   1 root     root           16 Jul 29  1998 K55routed -> ../init.d/routed
lrwxrwxrwx   1 root     root           13 Jul 29  1998 K60atd -> ../init.d/atd
lrwxrwxrwx   1 root     root           15 Jun 10 09:32 K85netfs -> ../init.d/netfs
lrwxrwxrwx   1 root     root           17 Jun 10 09:35 K89portmap -> ../init.d/portmap
lrwxrwxrwx   1 root     root           14 Jun 10 09:29 S05apmd -> ../init.d/apmd
lrwxrwxrwx   1 root     root           17 Jul 29  1998 S10network -> ../init.d/network
lrwxrwxrwx   1 root     root           16 Jul 29  1998 S20random -> ../init.d/random
lrwxrwxrwx   1 root     root           16 Jul 29  1998 S30syslog -> ../init.d/syslog
lrwxrwxrwx   1 root     root           15 Jul 29  1998 S40crond -> ../init.d/crond
lrwxrwxrwx   1 root     root           16 Jun 10 09:33 S45pcmcia -> ../init.d/pcmcia
lrwxrwxrwx   1 root     root           13 Jul 29  1998 S60lpd -> ../init.d/lpd
lrwxrwxrwx   1 root     root           18 Jul 29  1998 S80sendmail -> ../init.d/sendmail
lrwxrwxrwx   1 root     root           13 Jun 10 09:39 S90xfs -> ../init.d/xfs
lrwxrwxrwx   1 root     root           19 Jun 10 09:34 S99linuxconf -> ../init.d/linuxconf
lrwxrwxrwx   1 root     root           11 Jul 29  1998 S99local -> ../rc.local
```

根据前面的讲解，当K35smb被调用的时候，实际执行的命令是：

```
/etc/rc.d/init.d/smb stop
```

如果调用的是S90xfs，那么实际执行的是：

```
/etc/rc.d/init.d/xfs start
```

7.4.2 编写自己的rc命令脚本程序

在维护Linux系统运转的日子里，肯定会遇到需要系统管理员对开机或者关机命令脚本进行修改的情况。有两种方法可以用来实现修改的目的：

- 如果所做的修改只在引导开机的时候起作用，并且改动不大的话，可以考虑简单地编辑一下/etc/rc.d/rc.local脚本。这个命令脚本程序是在引导过程的最后一步被执行的。
- 如果所做的修改比较细致，或者还要求关闭进程的操作必须使之明确地停止运行，则需要在/etc/rc.d/init.d子目录中添加一个命令脚本程序。这个命令脚本程序必须可以接受start和stop参数并完成相应的操作。

第一种方法，编辑/etc/rc.d/rc.local脚本，当然是两种方法中比较简单的。如果想在这个命令脚本程序中添加内容，只需要使用喜欢的编辑器程序打开它，再把打算执行的命令附加到文件的末尾就可以了。这对一两行的修改来说的确很便利。

如果确实需要使用一个命令脚本程序，这时必须选择第二个方法。编写一个 rc命令脚本程序的过程并不象想像中的那么困难。我们下面就给出一个例子，看看它是怎样实现的（顺便说一句，你可以把我们的例子当作范本，按照自己的需要进行修改和添加）。

假设你打算每隔60分钟调用一个特殊的程序弹出一条消息，提醒自己需要从键盘前面离开休息一会儿了（如果不想得心血管疾病的话！），命令脚本程序将包括下面几个部分：

- 关于这个命令脚本程序功能的说明（这样就不会在一年之后忘记它！）。
- 在试图运行它之前验证这个命令脚本程序确实存在。
- 接受start和stop参数并执行要求的动作。

参数给定，下面就是我们要编写的命令脚本程序（请注意：除第一行以外，以“ # ”符号打头的那些行是注释语句，不属于命令脚本程序的可执行部分）。

```
#!/bin/sh
#
# Carpal         Start/Stop the Carpal Notice Daemon
#
# description: Carpald is a program which wakes up every 60 minutes and
#              tells us that we need to take a break from the keyboard
#              or we'll lose all functionality of our wrists and never
#              be able to type again as long as we live.
# processname: carpald

# Source function library.
. /etc/rc.d/init.d/functions

[ -f /usr/local/sbin/carpald ] || exit 0

# See how we were called.
case "$1" in
  start)
        echo -n "Starting carpald: "
        daemon carpald

        echo
        touch /var/lock/subsys/carpald
        ;;
  stop)
        echo -n "Stopping carpald services: "
        killproc carpald
```

```
        echo
        rm -f /var/lock/subsys/carpald
        ;;
    status)
        status carpald
        ;;
    restart|reload)
        $0 stop
        $0 start
        ;;
    *)
        echo "Usage: carpald {start|stop|status|restart|reload}"
        exit 1
esac

exit 0
```

编写好新的命令脚本程序之后，再从相关的运行级别子目录中加上必要的符号链接（请参考第6章）来控制这个命令脚本程序的启动或者停止。在我们的示范脚本程序中，我们只想让它在运行级别3或者运行级别5中启动，原因是我们认为只有这两个运行级别才是日常工作的地方。最后，我们希望这个命令脚本程序在进入运行级别 6（重启动）的时候被关闭。下面是我们用来建立必要的符号链接的命令：

```
[root@ford /root]# cd /etc/rc.d/rc3.d
[root@ford rc3.d]# ln -s ../init.d/carpal S99carpal
[root@ford rc3.d]# cd ../rc5.d
[root@ford rc5.d]# ln -s ../init.d/carpal S99carpal
[root@ford rc5.d]# cd ../rc6.d
[root@ford rc6.d]# ln -s ../init.d/carpal K00carpal
```

请注意：运行级别3和5中，我们在字母S后面使用了数字99，这就保证了这个命令脚本程序将作为引导过程的组成部分，其最后阶段被执行。对运行级别 6来说，我们反其道而行之——carpald需要在其他组件之前被关闭（启动组件的顺序通常是从最关键到最不关键；而关闭组件的顺序则是从最不关键到最关键）。

看起来很不错，对吧？好消息是现在我们已经完成了这个 rc类的命令脚本程序，再也不用重新做了。更重要的是这个命令脚本程序可以在引导或者关机的时候自动运行，并且能够自己照顾自己。最初的付出换来了长远的利益。

7.4.3 激活或者禁止服务项目

有的时候，你会发现在引导的时候并不需要某个特定的服务被启动。如果你正在考虑使用Linux替换Windows NT的文件和打印服务器，就更是如此。

在前面的小节里已经介绍过：在特定的运行级别子目录中给符号链接改个名称就可以让该服务不被启动，把其名称的第一个字母由 S改为K。一旦你熟练掌握了命令行和符号链接，就会发现这是激活或者禁止服务最快的办法。

在学习本改名方法的时候，你可能会觉得图形化的操作界面 ksysv比较容易掌握。虽然它

原来是设计使用在KDE环境里的，但在Red Hat Linux 6.0下缺省安装的GNOME环境里也运行得很好。如果想启动它，只需简单地打开一个 xterm窗口并输入ksysv命令就可以了。屏幕上会出现一个窗口，其中列出了你能够修改的全部参数，需要时还包括在线帮助（如图 7-1所示）。

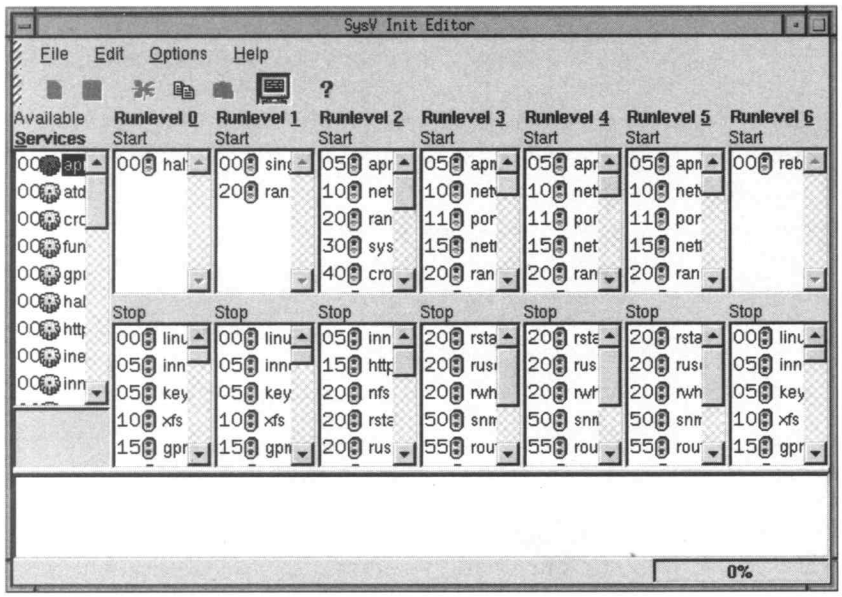

图7-1 ksysv的初始画面

7.5 小结

LILO是Linux操作系统极其灵活的引导加载程序。除了可以启动 Linux操作系统之外，LILO还可以引导其他类型的操作系统，包括 Windows系列中的许多种。本章中对 LILO和Windows NT引导过程的比较有助于你对 LILO过程的理解和掌握。 LILO如此有用的原因是它简单和可自动执行的特性。

在本章中我们主要讨论了以下关键问题：

- 使用lilo.conf配置文件。
- 运行LILO，使它可以安装到引导扇区中去。
- Linux系统的正常开机过程，包括rc命令脚本程序。
- 建立自己的rc命令脚本程序。

最后一句忠告：与其他关键的系统组件一样，不要在一个正常运转的环境中学习使用LILO，最好是在一个非正式应用的系统上来学习它。不管哪种操作系统，使用对引导过程有影响的工具程序时如果出了错，就可能很难纠正。