

第14章 启动和关闭系统

(欢迎提出意见和建议：frebsdhandbook@163.com)

在你运行FreeBSD之前，你必须先启动它。通常的顺序是：你打开计算机，屏幕上出现许多信息，大约一分钟后屏幕上出现一个login:提示符或一个X登陆窗口。有时，处理过程可能更有趣。启动时，可以有很多个启动选项，但这也是很多问题产生的根源，所以需要花点时间了解一下。在这一章，我们将看看下面这些主题：

- 在下一节，我们将更详细地看看启动进程。
- 在第245页，我们将看看如何控制启动进程。
- 如果出现一些错误，系统无法启动，这些滚过屏幕的信息非常重要。我们将看看它们是什么意思。
- 你必须在关闭系统的时候，遵守一些规范。我们将看看这些规范和为什么。

启动系统

当你打开系统的电源时，或当你重新启动时，在系统运行起来之前，会有很多动作发生。启动系统通常被叫做“bootstrapping”（引导），这个词源自于Baron von Munchhausen书中的一个故事。下面的顺序描述了在PC架构上的启动顺序，但在其他平台上可能有一些不同。

- 首先，BIOS（更精确来讲，叫系统固件。固件在i386架构上叫做BIOS (Basic Input/Output System)，在Alpha架构上叫做SRM，在很多其他架构上叫做Open Firmware）执行检查机器是否工作正常和决定硬件配置的测试工作。这个Power On Self Test 或POST是与FreeBSD无关的。
- 接着，BIOS启动器从系统的第一个磁盘的第一个扇区（C:在BIOS中）加载Master Boot Record进入内存，然后执行它。这个步骤与所有的基于PC的操作系统是一样的。
- 直到启动引导程序决定启动哪个操作系统。在MBR中的启动程序可以是，也可以不是FreeBSD系统的一部分。FreeBSD可以安装两个不同的MBR，请看第91页。标准的MBR不需要干预就可以工作，而启动管理器可以给你选择从磁盘的任何分区启动。
- FreeBSD的启动引导程序从磁盘的后面15个扇区加载第二层引导程序BTX，然后执行它。
- 第二层引导程序又定位到第三层启动程序，叫做loader，然后加载它进入内存。loader是一个智能化的启动组件，它允许预先加载多个内核组件。看看loader的联机手册了解更多信息。默认的，loader定位到内核（在root文件系统的文件/kernel中），然后加载它进入内存。你可以在这时打断加载器loader，例如为了加载不同或额外的文件。
- 接着，只在PC架构上，内核切换机器进入32位模式，然后关闭系统BIOS。
- 内核执行它自己的测试来寻找它知道的硬件。有关这个的内容很长，然后打出有关它能找到和不能找到的硬件信息。这个操作被叫做probing。绝大多数内核在建立时需要自行选择

支持的很多种硬件，所以通常“not found”信息要比“found”信息来得多。

- 检测完之后，内核启动两个进程。第一个，进程0，是swapper，当标准的虚拟内存算法不够快时，它就负责紧急清理内存。
- 进程1被叫做init。正如名称所暗示的，它负责启动系统和守护程序。当在默认的多用户模式运行时，它调用一个shell来执行shell脚本/etc/rc。
- /etc/rc首先读取包含了许多默认的配置变量的/etc/defaults/rc.conf描述文件和包含你自己对默认配置做过修改的/etc/rc.conf文件。它接着开始执行调用系统的必须步骤，首先启动虚拟磁盘驱动程序，挂上交换空间，如果有必要，完整地检查文件系统。
- 当/etc/rc存在的时候，init读取/etc/ttys文件，然后决定在哪儿启动进程。它花费剩余的时间来寻找处理这些进程。

在启动之前你要做的事情

在你启动系统之前，你可能要做很多事情：

- 要做的最明显的事情是决定将要启动什么。启动引导器给你加载不同操作系统或不同FreeBSD内核和模块的选择。我们将在下面看到。
- 另外，你可以为内核加载器设置很多选项，包括硬件的规格和软件的特性。我们将在第248页看到这些。

你将启动什么？

如果你的系统上有多个操作系统，你可以使用第90页所描述的启动管理器来选择启动哪一个。例如，如果你有两个磁盘，第一个包含四个分区，启动的第一阶段看起来像这样：

```
F1: FreeBSD
F2: Windows
F3: Linux
F4: FreeBSD
F5: Drive 1
Default: F1
```

10秒钟以后，启动管理器试图从默认的分区分区加载启动程序。你可以通过键入相应的功能键选择任何一个启动。如果你键入F5，你可以得到第二个磁盘上显示分区的菜单，再键入F5回到第一个磁盘。如果你选择启动FreeBSD，启动加载器开始加载，你可以看到这些：

```
/ this is a `twirling baton`
BTX loader 1.00 BTX version is 1.01
BIOS drive A: is disk0
BIOS drive C: is disk1
BIOS drive D: is disk1
BIOS 639kB/130048kB available memory
```

这些信息是由BTX产生的。如果你正在从磁盘加载系统，在前一行末尾的/字符就不断变换成-、和|，然后再回到/，主要是给你一个印象，字符正在翻滚。这个显示叫做twirling baton，是指示你系统没有崩溃和死机。在它旋转之前，通常会等待几秒钟。

接着，loader显示它的提示符：

```
FreeBSD/i386 bootstrap loader, Revision 0.8
(grog@freebie.example.com, Thu Jun 13 13:06:03 CST 2002)
Loading /boot/defaults/loader.conf
Hit [Enter] to boot immediately, or any other key for command prompt.
Booting [kernel] in 6 seconds... this counts down from 10 seconds
```

这时，你通常要用boot继续，或者是键入Enter键，或者是等待10秒钟。我们将在第253页看看发生了什么。

有时，虽然你可能要改变软件或硬件的特性。在这种情况下，你可以按下任何其他键（通常是空格键），然后键入命令来启动。

Loader命令

有两种方法可以来与loader进行通信：

- 在root文件系统的/boot目录中的许多文件告诉加载器做些什么。绝大多数不会被修改，但你可以创建叫做/boot/loader.conf的文件，然后在/boot/defaults/loader.conf中对一些命令进行修改。我们将在下面看到。
- 另外，FreeBSD版本5有一个/boot/device.hints文件。这个文件取代了许多配置文件记录，允许你设置硬件特性。我们将在第264页看到。
- 你可以直接键入命令到命令提示符。

当你键入空格键时，你得到下面的提示符：

```
Type '?' for a list of commands, 'help' for more detailed help.
ok ?
Available commands:
reboot reboot the system
heap show heap usage
bcachestat get disk block cache stats
boot boot a file or loaded kernel
autoboot boot automatically after a delay
help detailed help
? list commands
show show variable(s)
set set a variable
unset unset a variable
more show contents of a file
lsdev list all devices
include read commands from a file
ls list files
load load a kernel or module
unload unload all modules
lsmod list loaded modules
pnpscan scan for PnP devices
```

这些命令中最重要的是set, show, load, unload和boot。我们将在下一节看到一些它们使用的例子。注意，如果你在启动过程中意外地键入了“任何”键，需要继续启动，你必须键入boot。

loader.conf

加载器的行为是由`/boot/defaults/loader.conf`中的记录控制的。你不应当改变这个文件：在文件`/boot/loader.conf`中加入可能不存在的记录，也有许多已经存在的记录。在

`/boot/defaults/loader.conf`中，你将看到默认的值，当前被注释掉了，因为加载器已经知道了默认值。这里是一些很有意思的信息：

```
kernel="/kernel"
userconfig_script_load="NO"
verbose_loading="NO"    # Set to YES for verbose loader output
... a little further down
#autoboot_delay="10"    # Delay in seconds before autobooting
#bootfile="kernel,kernel.old" # Set the default boot file set
#console="vidconsole"  # Set the current console
#currdev="disk1s1a"    # Set the current device
#root_disk_unit="0"    # Force the root disk unit number
#rootdev="disk1s1a"    # Set the root filesystem
```

- kernel记录给出了内核的名称。有时，可能可以修改这个值，例如测试的时候。
- Userconfig字段描述了UserConfig，我们将在下面看到。
- console=vidconsole告诉加载器从哪里输出它的信息。vidconsole是video console的缩写。如果你有一个连接到特定串行口的串行终端，你也可以选择comconsole。
- currdev指出了到哪儿寻找root文件系统。如果你在磁盘上有多个BIOS，你可以用这个值选择一个正确的。

loader有很多的选项。读一下联机手册了解更多详细信息。

UserConfig:修改启动配置

安装媒介包括*UserConfig*，这给你一个修改内核的硬件配置的机会。你也可以告诉加载器在从硬盘启动的时候键入UserConfig，虽然这通常不是必须的。

*UserConfig*对一些过时的ISA硬件不再支持了。如果你要，跳过这节，继续第253页的描述。

从安装媒介，你自动键入*UserConfig*。如果你要在一个已安装的内核上使用它，只要在

`/boot/loader.conf`中加入这行：

```
userconfig_script_load="YES"
```

在启动时，打断启动进程，然后键入：

ok **boot -c**

```
Copyright (c) 1992-2002 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
The Regents of the University of California. All rights reserved.
FreeBSD 4.6-RELEASE #0: Sun Jun 9 18:15:47 CST 2002
grog@daemon.lemis.com:/src/FreeBSD/4.6-RELEASE/src/sys/compile/DAEMON
Timecounter "i8254" frequency 1193182 Hz
CPU: Pentium II/Pentium II Xeon/Celeron (467.73-MHz 686-class CPU)
Origin = "GenuineIntel" Id = 0x665 Stepping = 5
Features=0x183fbff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTRR,
PGE,MCA,CMOV,P
```

```
AT,PSE36,MMX,FXSR>
real memory = 134217728 (131072K bytes)
avail memory = 125452288 (122512K bytes)
config> v go into visual mode
```

选择任何一种方法，你将看到这个菜单：

Kernel Configuration Menu

Skip kernel configuration and continue with installation

Start kernel configuration in full-screen visual mode

Start kernel configuration in CLI mode

Here you have the chance to go into kernel configuration mode, making any changes which may be necessary to properly adjust the kernel to match your hardware configuration.

If you are installing FreeBSD for the first time, select Visual Mode (press Down-Arrow then ENTER).

If you need to do more specialized kernel configuration and are an experienced FreeBSD user, select CLI mode.

If you are **certain** that you do not need to configure your kernel then simply press ENTER or Q now.

如果你的硬件配置与generic内核指定的相匹配，就键入ENTER。内核继续*device probes*，这个我们将在第253页看到。对于硬件支持列表，可以看看第一个CDROM上的*HARDWARE.TXT*或*HARDWARE.HTM*文件。或者是<http://www.freebsd.org/releases/release/hardware.html>，这里release是后面跟着字母R的release，例如

<http://www.freebsd.org/releases/4.6R/hardware.html>.

如果你必须修改这个配置，键入向下箭头键用全屏幕模式选择内核配置，然后键入ENTER。这可以启动*UserConfig*，内核的一部分可以在启动时被激活来修改系统的设备配置。它用全屏幕菜单的形式来指示你：

```
---Active Drivers-----25
Conflicts-----Dev---IRQ--Port--
Storage : (Collapsed)
Network : (Collapsed)
Communications : (Collapsed)
Input : (Collapsed)
Multimedia :
Miscellaneous :
---Inactive
Drivers-----Dev-----
Storage : (Collapsed)
Network : (Collapsed)
Communications : (Collapsed)
Input : (Collapsed)
Multimedia :
Miscellaneous :
```

注意屏幕顶部的关键字**Conflicts**。这是一个警告，不是一个错误。我们将在第253页看看驱动程序如何在很多配置中寻找硬件。注意所有这些配置可能是共存的，但不会有问题，除非你安装了不能共存的硬件。

关键字**(Collapsed)**并不意味着你的硬件被放弃了——它的意思是有关相应硬件的信息，

这通常有很多设备，这些硬件已经被忽略了。你可以通过在相应的行移动指针然后键入Enter来扩展它。一条指针正好显示在Storage (Collapsed)行上。

你可以用箭头键上下移动。在我们的例子中，我们有一个NE2000兼容以太网卡，它的I/O寄存器以地址0x320启动，它被设置成IRQ 9，我们要修改内核配置来识别它。我们移动滚动条到Network行，然后键入Enter。显示变成：

```
---Active Drivers-----25
Conflicts-----Dev---IRQ--Port--
Storage : (Collapsed)
Network :
NE1000,NE2000,3C503,WD/SMC80xx Ethernet adapters conf ed0 5 0x280
NE1000,NE2000,3C503,WD/SMC80xx Ethernet adapters conf ed1 5 0x300
3C509 Ethernet adapter conf ep0 10 0x300
Fujitsu MD86960A/MB869685A Ethernet adapters conf fe0 5 0x240
Intel EtherExpress Ethernet adapter conf ix0 10 0x300
DEC Etherworks 2 and 3 Ethernet adapters conf le0 5 0x300
---Inactive
Drivers-----Dev-----
Storage : (Collapsed)
Network : (Collapsed)
Communications : (Collapsed)
Input : (Collapsed)
Multimedia :
Miscellaneous :
```

高亮显示的conf意思是当前的配置可能与另一个设备相冲突：I/O地址，IRQ或内存地址也可以被另一个驱动程序探测到。如果你有两个设置相同的设备，就只有一个问题。否则，你可以跳过这个警告。在这种情况下，碰巧我们的Novell NE2000网卡已经高量显示了，所以我们需要做的是再键入Enter来编辑配置。配置被拷贝到屏幕下方的独立的区域，所以我们现在看到：

```
---Active Drivers-----25
Conflicts-----Dev---IRQ--Port--
Storage : (Collapsed)
Network :
NE1000,NE2000,3C503,WD/SMC80xx Ethernet adapters conf ed0 5 0x280
NE1000,NE2000,3C503,WD/SMC80xx Ethernet adapters conf ed1 5 0x300
3C509 Ethernet adapter conf ep0 10 0x300
Fujitsu MD86960A/MB869685A Ethernet adapters conf fe0 5 0x240
Intel EtherExpress Ethernet adapter conf ix0 10 0x300
DEC Etherworks 2 and 3 Ethernet adapters conf le0 5 0x300
---Inactive
Drivers-----Dev-----
Storage : (Collapsed)
Network : (Collapsed)
Communications : (Collapsed)
Input : (Collapsed)
Multimedia :
Miscellaneous :
-----
Port address : 0x280 Memory address : 0xd8000 Conflict allowed
IRQ number : 5
Flags : 0
-----
```

端口地址是反白显示的，这意味着我们现在可以编辑它。我们键入0x320，然后键入Tab键得到IRQ field，这里我们键入9。接着，我们键入Enter离开编辑模式，然后是q离开配置编辑器。启动程序就按普通的方式继续，而这时系统找到了我们的以太网卡。

当然，这只是一个例子。采用编辑器要比描述它更加容易。不要担心损坏磁盘上的任何东西，因为这时，只有你和计算机，计算机并不认识任何设备。

由于PC硬件的特点，IRQ 2和IRQ 9是一样的。不要给指定IRQ 2；而是使用IRQ 9。如果你使用IRQ 2，驱动程序将不能正确工作。

FreeBSD的版本5将有一个非常不同的配置方法。在写这篇文章的时候，UserConfig还没有升级，因为它只是对ISA硬件有用，UserConfig将不被FreeBSD版本5所支持。

启动时加载其他模块

默认的，loader只加载内核。那可能就是你想要的。你可能要加载一个不同的内核，或你可能要加载一个kld。

有两个方法可以来这样做。如果你只要这样做一次，你就可以通过空格键打断启动进程，然后告诉loader做什么：

```
Booting [kernel] in 6 seconds...      this counts down from 10 seconds
(space bar hit)
Type '?' for a list of commands, 'help' for more detailed help.
ok unload                             not the kernel we wanted
ok load kernel.old                   revert to previous kernel
/kernel.old text=0x2dcd35 data=0x47c7c+0x43778
syms=[0x4+0x3ded0+0x4+0x44df3]
ok load vinum                         and the kld vinum
/modules/vinum.ko text=0x14cc0 data=0x3a8+0xace90
syms=[0x4+0x1220+0x4+0xc19]
ok boot -c -s                         then start the kernel
Copyright (c) 1992-1999 FreeBSD Inc.
Copyright (c) 1982, 1986, 1989, 1991, 1993
The Regents of the University of California. All rights reserved.
FreeBSD 4.6-RELEASE #0: Fri May 3 13:06:56 CST 2002
(etc)
```

这个例子显示了两个独立的行为：一个是修改内核/kernel为/kernel.old，其他的是加载vinum kld。你不需要为了加载vinum模块而重新加载内核。

自动kld加载

如果你要在每次启动时加载kld，上面描述的方法是很麻烦的。在这种情况下，向/boot/loader.conf添加下面这行会更容易：

```
vinum_load="YES"
```

不要修改这个文件。它是设计来在升级时被替换的，当你升级时，任何修改都将丢失。而是修改/boot/loader.conf的内容。这个文件只包含与默认值不同的地方，所以很可能它不存在。

在那种情况下，只要创建它。另一种方法是确保它包含下面这行

```
vinum_load="YES" # Concatenated/mirror/raid driver
```

这告诉加载器加载除内核以外的vinum。

运行内核

在启动过程中的下一步是运行内核。如果你在Booting [kernel]提示符上没有做什么或你键入了Enter，这是默认发生的。如果你已经打断了启动进程，你可以用下面的命令继续：

```
ok boot
```

下面的例子显示了一个Abit BP6双处理器主板的启动过程。这个主板也有四个IDE控制器，系统有两个连接着的SCSI适配器。

加载器传输控制给它已经预先加载的内核。来自内核的信息是用高亮度文本显示的。这是最普通的看到它们的机会，虽然它们有时在普通的机器操作过程中出现。这些信息也获得内核信息缓存的拷贝，你可以用dmesg程序重新找回绝大多数近来的信息。经过一段时间，其他信息可能会充满缓存，你将不能再用dmesg找到启动信息，所以在启动的最后一步把启动信息的内容保存在/var/run/dmesg.boot文件中，这个文件通常保存着完整的启动信息。在使用膝上型电脑的情况下，缓存信息通常不能在关机时得到清除，即使电源关闭了也不行，所以你可能会找到多个启动的日志。

信息是这样开始的：

```
Copyright (c) 1992-2002 The FreeBSD Project.  
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994  
The Regents of the University of California. All rights reserved.  
FreeBSD 4.6-RELEASE #0: Sun Jun 9 18:15:47 CST 2002  
grog@freebie.example.org:/src/FreeBSD/4.6-RELEASE/src/sys/compile/DA  
EMON
```

第一行告诉你内核有多大。除非你遇到问题，这个信息是没有多大意义的。如果你遇到问题，最重要的信息是在这个例子的最后二行上的内核路径和编辑日期。如果你遇到这个问题，请包含进这个信息。

接着，屏幕上会显示很多滚动的信息。启动完成以后，你可以回去，检查已经滚动过去的文本：键入ScrollLock。指针重新出现，你可以使用PageUp和PageDown键滚动到先前的屏幕。要退出这个模式，再键入ScrollLock。我们将在下面几页中详细地看看这个输出方式。

一旦它完成加载，内核呼叫所有已配置的启动程序来检查机器的硬件配置。这叫做设备检测 (*probing*)。如果你有时间这样做，最好是确定它的正确性。它不是很重要，除非发生一些错误，它不能滚动屏幕。我们可能会看见这样的信息：

```
Copyright (c) 1992-2002 The FreeBSD Project.  
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994  
The Regents of the University of California. All rights reserved.  
FreeBSD 4.6-RELEASE #0: Sun Jun 9 18:15:47 CST 2002  
grog@daemon.lemis.com:/src/FreeBSD/4.6-RELEASE/src/sys/compile/DAEMO
```

```
N
Timecounter "i8254" frequency 1193182 Hz
CPU: Pentium II/Pentium II Xeon/Celeron (467.73-MHz 686-class CPU)
Origin = "GenuineIntel" Id = 0x665 Stepping = 5
Features=0x183fbff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTRR,
PGE,MCA,CMOV,P
AT,PSE36,MMX,FXSR>
The lines above identify the CPU: it's a Pentium II, Pentium II Xeon, or Celeron, and it runs at 466
MHz.
```

```
real memory = 134217728 (131072K bytes)
avail memory = 125452288 (122512K bytes)
    real memory是RAM的大小。一些系统在real模式储备1kb的RAM,但这个不会影响实际内存
    的值。内核已经加载完和初始化后,可用的内存对用户很有用。
```

在一些旧的机器上,虽然机器上有很多内存,但内核报告只有16MB内存。这是BIOS的兼容性问题引起的,经常出现在一些老的机器上。要修复它,需要建立一个定制的确指定内存大小的内核——看看MAXMEM参数的描述,它在详细的//usr/src/sys/i386/conf/NOTES配置文件中

接着,使用多处理器内核,你得到:

```
Programming 24 pins in IOAPIC #0
IOAPIC #0 intpin 2 -> irq 0
IOAPIC #0 intpin 17 -> irq 9
IOAPIC #0 intpin 18 -> irq 11
IOAPIC #0 intpin 19 -> irq 10
FreeBSD/SMP: Multiprocessor motherboard
cpu0 (BSP): apic id: 0, version: 0x00040011, at 0xfef00000
cpu1 (AP): apic id: 1, version: 0x00040011, at 0xfef00000
io0 (APIC): apic id: 2, version: 0x00170011, at 0xfef00000
```

IOAPIC指I/O Advanced Programmable Interrupt Controller,它只被SMP机器使用。它分配一些中断请求。这个信息需要在你调试内核的时候提供。

```
Preloaded elf kernel "kernel" at 0xc0508000.
Preloaded elf module "if_fxp.ko" at 0xc050809c.
Preloaded elf module "miibus.ko" at 0xc050813c.
Pentium Pro MTRR support enabled
md0: Malloc disk
Using $PIR table, 8 entries at 0xc00fdef0
```

前面三行告诉你哪个内核模块被加载。因为这些处理器是P6类处理器,所以它们有用来调

整内存使用的Memory Type Range Registers或MTRR。

接着,我们看到在主板上的其他芯片组,以处理器支持的叫做`chipset'的芯片组开始。

```
npx0: <math processor> on motherboard          numeric coprocessor, on chip
npx0: INT 16 interface
pcib0: <Intel 82443BX (440 BX) host to PCI bridge> on motherboard
pci0: <PCI bus> on pci0
pcib1: <Intel 82443BX (440 BX) PCI-PCI (AGP) bridge> at device 1.0
on pci0
pci1: <PCI bus> on pci1
```

这个主板有一个带有两个PCI总线的Intel 82443 BX芯片组。

接着，我们看到主板上的一些设备：

```
pci1: <Matrox MGA G200 AGP graphics accelerator> at 0.0
isab0: <Intel 82371AB PCI to ISA bridge> at device 7.0 on pci0
isa0: <ISA bus> on isab0                      ISA bus
atapci0: <Intel PIIX4 ATA33 controller> port 0xf000-0xf00f at device
7.1 on pci0
ata0: at 0x1f0 irq 14 on atapci0              primary IDE controller
ata1: at 0x170 irq 15 on atapci0            secondary IDE controller
uhci0: <Intel 82371AB/EB (PIIX4) USB controller> port 0xc000-0xc01f
irq 10 at device
7.2 on pci0                                  USB controller
usb0: <Intel 82371AB/EB (PIIX4) USB controller> on uhci0 USB bus
usb0: USB revision 1.0
uhub0: Intel UHCI root hub, class 9/0, rev 1.00/1.00, addr 1
uhub0: 2 ports with 2 removable, self powered
Timecounter "PIIX" frequency 3579545 Hz
chip1: <Intel 82371AB Power management controller> port 0x5000-0x500f
at device 7.3
on pci0
```

系统不知道在芯片组中哪个设备是在内部执行，哪个是主板上的独立芯片，哪个是嵌入芯片。迄今为止，已经找到了IDE控制器，但不是磁盘。它将在以后找到它们。

接着，我们找到两个Symbios SCSI主机适配器和一个以太网卡。

```
sym0: <875> port 0xc400-0xc4ff mem
0xec002000-0xec002fff,0xec003000-0xec0030ff irq 1
0 at device 9.0 on pci0
sym0: Symbios NVRAM, ID 7, Fast-20, SE, NO parity
sym0: open drain IRQ line driver, using on-chip SRAM
sym0: using LOAD/STORE-based firmware.
sym0: SCAN FOR LUNS disabled for targets 0.
sym1: <875> port 0xc800-0xc8ff mem
0xec001000-0xec001fff,0xec000000-0xec0000ff irq 9
at device 13.0 on pci0
sym1: No NVRAM, ID 7, Fast-20, SE, parity checking
第一个Symbios SCSI适配器使用IRQ 10。它是在ID 7上，像绝大多数SCSI适配器，它不支
```

持奇偶校验。第二个设备使用IRQ 9，支持奇偶校验，但它没有BIOS。这对FreeBSD来说是一个问题，但如果它自己在系统中，POST就不会找到它。在这个特殊的情况下，在其他Symbios设备上的BIOS实际上就无法找到第二个适配器。

```
dc0: <Macronix 98715AEC-C 10/100BaseTX> port 0xe000-0xe0ff mem
0xe7800000-0xe78000ff irq 11 at device 11.0 on pci0
dc0: Ethernet address: 00:80:c6:f9:a6:c8
miibus0: <MII bus> on dc0
dcpHY0: <Intel 21143 NWAY media interface> on miibus0
dcpHY0: 10baseT, 10baseT-FDX, 100baseTX, 100baseTX-FDX, auto
```

这是一个使用IRQ 11的与PHY接口关联的Macronix以太网卡。然后，我们回到板载设备，在这种情况下，两个额外的IDE控制器和留下来的ISA设备：

```
atapci1: <HighPoint HPT366 ATA66 controller> port
0xd800-0xd8ff,0xd400-0xd403,0xd000
-0xd007 irq 11 at device 19.0 on pci0
ata2: at 0xd000 on atapci1 Third IDE controller
```

The Complete FreeBSD(4th)

```
atapci2: <HighPoint HPT366 ATA66 controller> port
0xe400-0xe4ff,0xe000-0xe003,0xdc00
-0xdc07 irq 11 at device 19.1 on pci0 Fourth IDE controller
ata3: at 0xdc00 on atapci2
orm0: <Option ROMs> at iomem 0xc0000-0xc7fff,0xc8000-0xc87ff on isa0
fdc0: ready for input in output Floppy controller
fdc0: cmd 3 failed at out byte 1 of 3
```

这里的软盘驱动命令失败是由这台机器上的任何软盘驱动器的缺少引起的。

```
atkbd0: <Keyboard controller (i8042)> at port 0x60,0x64 on isa0
atkbd0: <AT Keyboard> flags 0x1 irq 1 on atkbd0 keyboard
kbd0 at atkbd0
vga0: <Generic ISA VGA> at port 0x3c0-0x3df iomem 0xa0000-0xbffff on
isa0
sc0: <System console> at flags 0x100 on isa0 system console
sc0: VGA <16 virtual consoles, flags=0x300>
sio0 at port 0x3f8-0x3ff irq 4 flags 0x10 on isa0 first serial port
sio0: type 16550A it's a buffered UART
sio1 at port 0x2f8-0x2ff irq 3 on isa0 second serial port
sio1: type 16550A
sio2 not found at 0x3e8 no more serial I/O ports
sio3 not found at 0x2e8
```

回想一下UNIX设备以0开始编号的小节，那里Microsoft系统以1开始编号。在Microsoft系统中，设备sio0到sio3被认作是COM1:到COM4:。

```
ppc0: <Parallel port> at port 0x378-0x37f irq 7 on isa0 parallel port controller
ppc0: Generic chipset (NIBBLE-only) in COMPATIBLE mode
plip0: <PLIP network interface> on ppbus0
lpt0: <Printer> on ppbus0 line printer on parallel port
lpt0: Interrupt-driven port
ppi0: <Parallel I/O> on ppbus0 alter nate parallel I/O on the same port
```

接着，在多处理器的主板上，我们得到一些特定SMP的信息。系统测试可能有时会遇到问题的IO-APIC，然后启动第二个处理器：

```
APIC_IO: Testing 8254 interrupt delivery
APIC_IO: routing 8254 via IOAPIC #0 intpin 2
SMP: AP CPU #1 Launched!
```

最后，系统检测到连接到这台机器的磁盘：

```
ad0: 19574MB <WDC WD205BA> [39770/16/63] at ata0-master UDMA33
ad4: 19574MB <WDC WD205BA> [39770/16/63] at ata0-master UDMA66
Waiting 15 seconds for SCSI devices to settle
(noperiph:sym0:0:-1:-1): SCSI BUS reset delivered.
da0 at sym1 bus 0 target 3 lun 0
da0: <SEAGATE ST15230W SUN4.2G 0738> Fixed Direct Access SCSI-2 device
da0: 20.000MB/s transfers (10.000MHz, offset 15, 16bit), Tagged Queueing
Enabled
da0: 4095MB (8386733 512 byte sectors: 255H 63S/T 522C)
da1 at sym1 bus 0 target 0 lun 0
da1: <SEAGATE ST15230W SUN4.2G 0738> Fixed Direct Access SCSI-2 device
da1: 20.000MB/s transfers (10.000MHz, offset 15, 16bit), Tagged Queueing
Enabled
da1: 4095MB (8386733 512 byte sectors: 255H 63S/T 522C)
```

这里，我们有四个磁盘，两个分别在第一个和第三个IDE控制器上，两个都作为master，两个在的第二个SCSI主机适配器上。在第一个主机适配器上没有东西。

最后，系统启动Vinum，然后加载root filesystem和swap交换分区。

```
Mounting root from ufs:/dev/ad0s1a
vinum: loaded
vinum: reading configuration from /dev/ad0s1h
vinum: updating configuration from /dev/ad4s2h
swapon: adding /dev/ad0s1b as swap device
swapon: /dev/vinum/swap: No such file or directory
Automatic reboot in progress...
```

这时，系统开始运行起来，但它仍必须启动一些服务。余下的信息来自进程，不是来自内核，所以它们很普通。

```
add net default: gateway 223.147.37.5
Additional routing options: tcp extensions=NO TCP keepalive=YES.
routing daemons:.
Mounting NFS file systems.
additional daemons: syslogd
Doing additional network setup: portmap.
Starting final network daemons: rwhod.
setting ELF ldconfig path: /usr/lib /usr/lib/compat /usr/X11R6/lib
/usr/local/lib
setting a.out ldconfig path: /usr/lib/aout /usr/lib/compat/aout
/usr/X11R6/lib/aout
starting standard daemons: inetd cron
Initial rc.i386 initialization:.
rc.i386 configuring syscons: blank_time.
Local package initialization:.
Additional TCP options:.
Tue Apr 23 13:59:05 CST 2000
```

这时，内核完成检测，它传输控制给shell脚本/etc/rc。从这点上看，显示结果是很普通的。

/etc/rc首先在/etc/defaults/rc.conf和/etc/rc.conf中读取配置信息。然后，它根据请求启动ccd和vinum。

```
vinum: loaded
vinum: reading configuration from /dev/da2h
vinum: updating configuration from /dev/da3h
vinum: updating configuration from /dev/da4h
vinum: updating configuration from /dev/da1h
```

接着，它检查相连的文件系统。通常，你将看到在/etc/fstab中针对每个文件的像这样的信息：

```
/dev/da0s1a: FILESYSTEM CLEAN; SKIPPING CHECKS
/dev/da0s1a: clean, 6311 free (367 frags, 743 blocks, 0.9% fragmentation)
/dev/da0s1e: FILESYSTEM CLEAN; SKIPPING CHECKS
/dev/da0s1e: clean, 1577 files, 31178 used, 7813 free (629 frags, 898
blocks, 1.6% fragmentation)
```

如果你的系统崩溃了，不是软件问题，就是硬件问题，或因为没有正确关机，它将执行一个文件系统的检查 (fsck)，这可能需要花一段时间，在一个大文件系统上最长需要一个小时。

你将看到像这样的信息：

```
WARNING: / was not properly dismounted
/dev/da0s1a: 6311 free (367 frags, 743 blocks, 0.9% fragmentation)
```

可以用Ctrl-C打断*fsck*。没有检查的文件系统将仍保持为“dirty”，你将进入单用户模式。当调试启动问题时，这可能是很有用的。

接着，*/etc/rc*调用三个网络中的第一个启动程序。这个初始化接口，如果必要，就设置路由和启动防火墙：

```
Doing initial network setup: hostname.
dc0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
inet 223.147.37.81 netmask 0xffffffff broadcast 223.147.37.255
inet6 fe80::280:c6ff:fef9:a6c8%dc0 prefixlen 64 scopeid 0x1
ether 00:80:c6:f9:a6:c8
media: autoselect (100baseTX <full-duplex>) status: active
supported media: autoselect 100baseTX <full-duplex> 100baseTX
10baseT/UTP <f
ull-duplex> 10baseT/UTP 100baseTX <hw-loopback> none
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
inet 127.0.0.1 netmask 0xff000000
add net default: gateway 223.147.37.5
Additional routing options:.
routing daemons:.
```

在这个例子中，没有额外的路由选项，没有路由守护程序。因此在字符:和.之间没有信息。

你将常常看到这个信息。

接着，*/etc/rc*挂上网络文件系统，清理*/var/run/*，然后启动*syslogd*：

```
Mounting NFS file systems.
Additional daemons: syslogd.
```

然后，它检查是否我们有一个存储器转储信息（core dump）。如果有，它设法把它保存在*/var/crash*中。

```
checking for core dump...savecore: no core dump
```

如果在*/var/crash*中没有足够的空间，保存存储器转储信息可能会失败。如果发生这样的情况，在你没有足够的交换空间改写存储器时，你可以清除，然后以后再保存，

接着，进行第二步启动网络，可以选择启动*named*，*ntpd*，*xntpd*，*timed*，*portmap*，*ypserv*，

rpc.ypxfr d，*rpc.yppasswdd*，*ypbind*，*ypset*，*keyser*和*rpc.yupdated*：

```
Doing additional network setup: named xntpd portmap.
starting. named 8.1.2 Sun May 9 13:04:13 CST 1999
gro@freebie.example.org:/usr
/obj/usr.sbin/named
master zone "example.org" (IN) loaded (serial 1997010902)
master zone "37.147.223.in-addr.arpa" (IN) loaded (serial 1996110801)
listening on [223.147.37.149].53 (ep0)
listening on [127.0.0.1].53 (lo0)
Forwarding source address is [0.0.0.0].1063
Ready to answer queries.
```

除了第一行，所有的信息都来自*named*。它们可能出现在第一行的中间，而不是等到行尾。

接着，*/etc/rc*按要求启用配额，然后运行第三方网络程序，启动*mountd*，*nfsd*，*rpc.lockd*，*rpc.statd*，*nfsiod*，*amd*，*rwhod*和*kerberos*：

```
Starting final network daemons: mountd nfsd rpc.statd nfsiod rwhod.
```

现在，我们已经完成了。*/etc/rc*重新建立了几个内部数据库（对用户，通过*ps*和一些其他命

令)，然后它为`ldconfig`设置默认的路径：

```
setting ELF ldconfig path: /usr/lib /usr/lib/compat /usr/X11R6/lib
/usr/local/lib
setting a.out ldconfig path: /usr/lib/aout /usr/lib/compat/aout
/usr/X11R6/lib/aout
/usr/local/lib/aout
```

接着，你可以选择启动`inetd`，`cr on`，`printer`，`sendmail`和`usbd`：

```
starting standard daemons: inetd cron sendmail.
```

`/etc/rc`做的最后一件事情是检查其他启动文件。这些可能是在用变量`local_startup`指定的文件中，或在文件`/etc/rc.local`中。在我们的例子中，没有任何文件，所以我们看到的所有东西是：

```
Local package initialization:.
```

最后，我们就完成了。`/etc/rc`停止了，`init`处理`/etc/ttys`，这在特定的终端上启动`getty`进程。

在控制台上，我们看到：

```
Mon May 13 13:52:00 CST 2002
FreeBSD (freebie.example.org) (ttyv0)
login:
```

这时，我们可以看看第13章的开始部分。

检测PCMCIA硬件

PCMCIA卡用一个重要的方法与其他总线进行区分：它们被设计来用于热插拔（*hot plugged*），在一个运行的系统上插入和卸下。这意味着传统的启动探测对检测它们不充分。而是，当一个卡被插入或卸下时，一个特定的进程检测PCMCIA硬件，然后执行相应的动作。

结果，上面描述的检测方法无法工作在PCMCIA和CardBus卡上。而是，用一个用户进程`pccardd`，监视PCMCIA硬件，然后当发生一些变化时，执行相应的动作。在启动时，随着`pccardd`启动，PCMCIA卡没有被发现。我们将在第15章看到这个是如何工作的细节。

单用户模式

有时，多个用户访问系统是不方便的。例如，如果你正在重新对磁盘分区，你不希望其他用户能访问所有的磁盘空间。即使在系统上只有你一个用户，守护程序可能在后台执行一些工作。为了避免这个问题，你可以在绝大多数守护程序启动之前停止启动进程，然后进入单用户模式。要这样做，设置`boot_single`变量，或在启动时指定`-s`标记。

```
ok boot -s
```

设备一检测完成，系统启动就被打断，你将在shell上得到一个提示符。只有`root`文件系统是可以访问的，它被挂上只读。这样做的原因是文件系统可能损坏了，要求在你写入之前先修复它。如果你必须写入到`root`文件系统，你需要先用`fsck`检查文件系统的连接性，完成之后，你可以用`-u(update)`选项挂上它。例如：

```
npx0 on motherboard
```

```
npx0: INT 16 interface          end of the probes (high intensity display)
Enter pathname of shell or RETURN for sh: hit RETURN
erase ^H, kill ^U, intr ^C
# fsck -y /dev/ad0a           check the integrity of the root file system
** /dev/ad0a
** Last Mounted on /
** Root file system
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
1064 files, 8190 used, 6913 free (61 frags, 1713 blocks, 0.4%
fragmentation)
# mount -u /                 remount root file system read/write
# mount /usr                 mount any other file systems you need
```

要退出单用户模式，进入多用户模式，只要键入Ctrl-D：

```
# ^D
Skipping file system checks...
(the rest of the boot sequence)
```

System V和Linux使用运行级别 (*run level*) 的概念，这是由init控制的。单用户模式相应运行s级别，多用户模式下，System V运行在级别3下，Linux运行在级别4下。没有其他相对应的运行级别，特别是级别2，这个级别启动一个不带网络的System V系统。网络是FreeBSD的一个集成部分，所以这是行不通的。你也不能用System V命令*init S*进入单用户模式。如果你这样做，你会得到：

```
# init S
init: already running
```

你可以用shutdown命令从一个运行的FreeBSD系统进入单用户模式 我们将在下一节看看这个情况。不幸的是，shutdown关闭一些执行的守护程序和挂上的文件系统，所以它更适合通过重新启动进入，就像上面显示的。

口令保护的单用户模式

如果你运行在一个安全的环境，你可能会比较关心不键入一个口令就可以在单用户模式启动。那是默认的——通常，如果有人可以访问你的系统控制台，一个口令不再有用，它可能是一个讨厌的东西——但你可以修改它。在*/etc/tty*中完成这个记录，修改关键字secure为insecure：

```
# If you want to be asked for password, change "secure" to "insecure"
here
console none unknown off insecure
```

如果你这样做，一旦你忘记了root口令，你将遇到麻烦。

关闭系统

FreeBSD使用许多成熟的技术来提供很高的性能。特别是，当你写数据到磁盘上，系统不

立刻把它放在磁盘上：它等待更多的数据到达，这可以显著地提高性能，因为它减少了磁盘访问的数量。

调整的结果是在数据被写入之前关闭电源。你可能会丢失数据，但如果在文件系统结构上数据是正在改变的信息，你的文件系统将被损坏。要检查这个，系统在启动时需要运行一个叫做fsck的程序。Fsck可以修复极小的损坏，但很明显通过确保系统按正确的方法关机是一个好主意。

千万不要只通过切断电源来关机。结果可能是灾难性的。

关闭系统的正确方法是用shutdown命令。可以参考shutdown的联机手册：

Shutdown为超级用户提供了一个自动关机的进程，它可以在系统关闭时通知用户。

这个命令有许多有用的选项：

- 使用-r选项重新启动计算机。你有时必须这样做，例如安装完一个新内核之后。
- 使用-h选项关闭机器。这是普通的方式，但它不是默认的。
- 没有选项，shutdown试图让机器进入单用户模式。这通常没有在启动时进入单用户模式来得好。
- shutdown加上一个时间参数，将告诉它在规定时间内关闭系统。这在一个多用户环境下是很有用的，但通常想要马上关闭，所以可以在shutdown后加上now选项。

通常情况下，你要立刻关闭机器，然后关闭电源，你可以键入：

```
# shutdown -h now
Feb 4 12:38:36 freebie shutdown: halt by grog:
Feb 4 12:38:39 freebie syslogd: exiting on signal 15
syncing disks... done
The operating system has halted.
Please press any key to reboot.
```

在你关闭切断电源前，确信看到这些信息。

重新启动

有时，你要重新启动机器，换句话说要停止它，然后重新启动它。一个典型的情况是建立一个新内核之后：为了运行它，你必须重新启动机器。那是很简单的，键入：

```
# shutdown -r now
或
# reboot
或
Ctrl-Alt-DEL
```

FreeBSD 5.0的展望

FreeBSD 5.0在内核配置文件中将包含很多与以前不同的地方，它不再包含有关IRQ, DMA, I/O地址的信息和硬件的其他设置。而是，这个信息被包含在文件/boot/device.hints中，你可以在

启动之前或在启动过程中修改这个。在写这篇文章的时候，*UserConfig*还没有被重新写，而是使用*/boot/device.hints*。可能它再也不会升级了，这个工具将永远地放弃了。

CDROM安装方法会安装文件*/boot/device.hints*，但一个内核安装就不会。你将在针对你架构的conf文件中找到它。例如，*/usr/src/sys/i386/conf*包含配置文件*GENERIC*，相应的hints文件*GENERIC.hints*。在写这篇文章的时候，FreeBSD-CURRENT还没有安装这个文件。你必须自己安装它。Hints文件包含下面特性的记录：

```
hint.sio.0.at="isa"
hint.sio.0.port="0x3F8"
hint.sio.0.flags="0x10"
hint.sio.0.irq="4"
hint.sio.1.at="isa"
hint.sio.1.port="0x2F8"
hint.sio.1.irq="3"
hint.sio.2.at="isa"
hint.sio.2.disabled="1"
hint.sio.2.port="0x3E8"
hint.sio.2.irq="5"
hint.sio.3.at="isa"
hint.sio.3.disabled="1"
hint.sio.3.port="0x2E8"
hint.sio.3.irq="9"
```

这些记录指出了串行口配置。它们取代了在内核中比较老的硬编码方法。例如，上面的hints

包含了版本4配置文件中这些行的配置信息：

```
device sio0 at isa? port IO_COM1 flags 0x10 irq 4
device sio1 at isa? port IO_COM2 irq 3
device sio2 at isa? disable port IO_COM3 irq 5
device sio3 at isa? disable port IO_COM4 irq 9
```

在版本5的配置文件中相应的这些行是：

```
device sio # 8250, 16[45]50 based serial ports
```

更重要的，这意味着如果你改变了硬件地址，你不需要重新编译内核。