

第20章 配置内核

(欢迎提出意见和建议：frebsdhandbook@163.com)

迄今为止，我们做的所有事情都是由FreeBSD的标准GENERIC内核完成的。但，你可以在一个定制的内核中发现很多优点：

- 正如你在第2章看到的，GENERIC内核不支持FreeBSD所能支持的所有东西。例如，如果你要安装一个Yoyodyne frobulator，你必须安装对它的特定的支持。
- 实际上，在Yoyodyne上工作的开发人员已经投身于自由软件基金会。看看GNU通用公共许可了解更多细节。
- 它只需要花很少的时间就可以启动，因为它不需要花费时间来探测你没有的硬件。
- 一个定制的内核经常使用很少的内存。内核是一个必须总是出现在内存中的系统组件，所以没有用的代码会占用内存，这些将另外再占用虚拟内存系统。在一个拥有有限RAM的系统上，你可以通过建立一个定制的内核来节约一些内存。不要过高地估计节约的数量：一个小的内核可以比GENERIC内核节约500KB。
- 最后，有几个你可以调整的内核选项来符合你的需要。

在比较老的BSD版本中，你必须为你要修改的任何东西建立一个新的内核，甚至像修改一个设备的IRQ一样简单。其后，FreeBSD经过了很大的发展，它变得更好，不一定要建立一个特定的内核。如果你想启用更多内核选项，如额外的一致性检查，你当然可以这样做，但在很多情况下，你有更多灵活的做法：

- 如果你只需要添加设备支持，你可以加载一个*Kernel Loadable Module*或*kld*。看看第403页了解有关kld的更多信息。
- 许多内核选项已经被*sysctl*接口所替换。例如，GENERIC内核默认地不执行数据包路由。在FreeBSD的比较老的版本中，你必须用选项GATEWAY建立一个新的内核。现在，你可以用一个*sysctl*命令打开和关闭这个特性。我们可以在*sysctl*的联机手册了解更多信息。

自从早期的FreeBSD发布以来，配置内核已经发生了很大的变化，但它仍然没有完成。在FreeBSD的版本5中，会在2002年的晚些时候或2003年的早些时候发布，将有更多的变化。我们在这章看看这些变化。

建立一个新的内核

FreeBSD是用源代码发布的，建立一个内核主要是编译内核所需要的源代码。要建立一个内核，你可以执行下面几步：

- 安装系统源代码，如果还没有这样做。
- 在一个内核配置文件中，定义你的内核配置。这个文件定义了编译过程中将要使用的参数。我们见在第397页的开始看看如何做这个事情。

- 用`config`程序创建配置目录。我们将在第400页讨论。
- 运行`make depend`来创建内核编译的依赖关系信息。
- 运行`make`编译内核。
- 安装内核。

配置I/O设备

有许多配置文件是与可能连接到你机器的I/O设备相关的。正如我们在第2章看到的，你通常必须指定一些IRQ，DMA通道，设备内存，和你配置设备的I/O地址，特别是ISA设备。配置文件不能正确地处理IRQ2：你必须用IRQ 9来指定它们。

在FreeBSD的版本5中，我们将不再必须在配置文件中指定这些值。而是，你应当修改`/boot/device.hints`文件。这个文件不会自动安装。如果它不存在，把它从配置目录拷过来：

```
# cp -p /usr/src/sys/i386/conf/GENERIC.hints /boot/device.hints
```

内核编译目录

内核源代码被保存在`/usr/src/sys`目录中。符号连接`/sys`也会指向这个目录。这个目录中有许多的子目录，它们代表了内核的不同部分，这些当中最重要的是与架构相关的目录，`i386/conf`（针对i386架构）或`alpha/conf`（针对Alpha架构）。你可以根据你的架构情况来编辑和定制内核。有一点需要注意的是目录树的逻辑组织：每个支持的设备，文件系统，和它自己子目录的选项。这篇文章主要讲述基于i386架构的配置情况。基于Alpha架构的配置方法与此类似。

如果你的系统中没有`/usr/src/sys`目录，那内核源代码就是没有被安装。要从FreeBSD的CD-ROM安装源代码，需要执行下面几步：

```
# mkdir -p /usr/src/sys
# ln -s /usr/src/sys /sys
# cd /
# cat /cdrom/src/ssys.[a-h]* | tar xzvf -
```

针对`/usr/src/sys`的符号连接`/sys`并不是必须的，但它是一个好办法：一些软件使用它，否则你可能会用两个不同的源代码拷贝来结束。

接着，进入`i386/conf`目录，拷贝GENERIC配置文件到一个你自己命名的文件。例如：

```
# cd /usr/src/sys/i386/conf
# cp GENERIC FREEBIE
```

通常，这个名称都要用大写，如果正维护着多台不同配置的FreeBSD机器，你可以用这些机器的主机名来命名。在这个例子中，我们用FREEBIE来命名。

现在，就可以用你喜欢的文本编辑器来编辑FREEBIE。你可以在最上面一行写上你的注释，以作出区分：

```
#
```

The Complete FreeBSD(4th)

```
# FREEBIE -- My personal configuration file
#
# For more information read the handbook part System Administration ->
# Configuring the FreeBSD Kernel -> The Configuration File.
# The handbook is available in /usr/share/doc/handbook or online as
# latest version from the FreeBSD World Wide Web server
# <URL:http://www.FreeBSD.ORG/>
#
# An exhaustive list of options and more detailed explanations of the
# device lines is present in the .LINT configuration file. If you are
# in doubt as to the purpose or necessity of a line, check first in LINT.
#
# $Id: FREEBIE,v 1.101 1997/10/31 22:10:02 jseger Exp $
machine "i386"
cpu "I486_CPU"
cpu "I586_CPU"
cpu "I686_CPU"
ident FREEBIE
maxusers 10
```

如果你已经在SunOS或其他一些BSD操作系统上编译过内核，那下面的介绍你会比较熟悉。

如果你还是个新手，那GENERIC配置信息可能会让你眼花缭乱，所以下面将慢慢地对这个配置文件作一个详细的介绍：

配置文件

在/sys/i386/conf目录中，包含了许多配置文件：

- GENERIC 通用的配置文件。
- LINT 带有很多注释的完整配置文件。这个文件主要是给出了测试和参考文档，不是为编译内核准备的，它太臃肿。FreeBSD版本5将不再包含LINT配置文件。而是，它将提供一个更加清晰易懂的NOTES文件。你可以通过在那个目录中运行make来使用它创建LINT。
- PCCARD 膝上型计算机要用到的PCCARD控制器的配置文件。
- SMP-GENERIC 并行多处理器机器的一个通用配置文件。

配置文件的通用格式是很简单的。每一行包含了一个关键字和一个或多个选项。很简单，绝大多数行只包含一个选项。任何跟在一个#字符后面的语句是一个注释，将被略去。包含数字的关键字必须用双引号引起来。

这个简单规则的结果是你可以加入不受影响的选项。例如，你可以加入像下面这样的一行：
options APPLE_MAC_COMPATIBILITY

你可以用这个选项编译内核。它没什么不同。现在，你不太可能会使用一个像这样的不存在的选项，但更可能你会拼错一个选项，特别是像SYSVSHM这样很饶口的选项，结果是你无法在你要的选项中编译。如果你使用了未知的选项，*config*程序会发出警告，所以要认真地注意

这些警告。

内核选项从一个版本到一个版本不断地发生变化，因而在描述它们时没有指出来。在下面的章节中，我们将看看一些有趣的东西。需要了解一个完整的列表，可以看看LINT或在线手册。

命名内核

你编译的每个内核需要机器的关键字，cpu和ident。

例如，

```
machine "i386" For i386 architecture
machine "alpha" For alpha architecture
cpu "I486_CPU"
cpu "I586_CPU"
cpu "I686_CPU"
ident FREEBIE
```

machine

关键字machine描述了将被编译内核的机器的架构。现在，主要有两个架构：i386和AXP（alpha），不要把这两个搞错：例如，i386是指Intel 80386以及与它兼容的AMD, Cyrix和IBM处理器。

cpu cpu_type

关键字cpu描述了这个内核支持哪个CPU芯片。对于i386架构，主要是I386_CPU, I486_CPU, I586_CPU和I686_CPU，你可以同时都指定这些值。例如，你现在使用Intel Pentium处理器，就可以使用I586_CPU。

对于FreeBSD版本5，它将不能编译一个同时支持80386处理器和以后处理器的简单内核。你必须指出是I386_CPU或是其他的。

ident machine_name

关键字ident指出了内核的名称。在GENERIC文件中，它就是GENERIC。把它改为你自己起的内核名称，在这个例子中是FREEBIE。你指定的内核名称将在内核启动时显示出来，所以你自己起一个与通用的内核名称不同的名称是很有用的。

由于这个名称将被C编译器当作变量来编译，所以不要使用像DEBUG这样的名称，或者其他一种会引起编译器产生错误的机器或CPU名称，例如vax。

maxusers number

这个值设定了许多重要的系统表的大小。它大概等于你希望机器能够支持的最大并发用户数。另外，即使你是这台机器的唯一的一个用户，你也不能把这个值设成小于默认值32，特别是如果你正在使用X或编译软件。主要原因是由maxusers设定的重要的系统表是系统的最大进程的最大数目，它是 $20 + 16 * \text{maxusers}$ 这样来设定的，所以如果你把maxusers设成1，那你的机器就只能支持36个最大的并发进程，这里面包括了18个系统启动时需要的进程，15个你启动X时

The Complete FreeBSD(4th)

需要的进程。即使一个象阅读联机手册这样简单的任务也需要9个进程。把maxusers设成32将允许你支持532个并发进程，通常这些就足够用了。如果你启动另一个程序或运行一个支持有大量并发用户的服务器时，出现大量的错误，你只需要提高这个数值然后重新编译就行了。

Maxusers并不会限制登陆到你机器的用户的数量。它只是根据可能登陆到你系统的最大用户数量和它们每个运行时需要多少进程来设定不同表的大小。限制远程登陆的并发用户数量的关键字是pseudodevice `pty`。

现在，你可以指定maxusers的值为0。这允许内核自动调整32和512之间值的表，这依赖于系统上内存的数量。

config kernel_name

老版本的FreeBSD要求有一个指定内核名称的配置行。现在就不允许了。默认的，在FreeBSD版本4中，内核被放在/kernel中，在版本5中内核被放在/boot/kernel/kernel中。

Kernel options

有许多内核选项，绝大多数你不需要修改。只有一个你需要修改。默认的，makeoptions行被注释掉了。你不应当把它注释掉。

```
makeoptions DEBUG=-g # Build kernel with gdb(1) debug symbols
```

为什么要把它注释掉？FreeBSD计划不同意把它修改掉。

看看第10章，安装过程中的问题的讨论。

Multiple processors

FreeBSD 4.6支持绝大多数基于i386架构的多处理器系统。GENERIC内核默认不支持它：设置下面的选项：

```
# To make an SMP kernel, the next two are needed
options SMP                # Symmetric MultiProcessor Kernel
options APIC_IO            # Symmetric (APIC) I/O
```

一个SMP内核不能运行在一个没有IOAPIC芯片的简单的处理器上。确信在SMP内核中禁用了cpu "I386_CPU"和cpu "I486_CPU"选项。

编译和安装新的内核

编辑完配置文件，你可以用`config`命令创建编译环境。在/usr/src/sys/i386/conf目录中，你可以键入：

```
# /usr/sbin/config FREEBIE
Kernel build directory is ../../compile/FREEBIE
```

如果你在配置文件中有一个错误，很可能在运行`config`时出现错误信息。

幸运的是，`config`将打印出出错行的行号，所以你可以快速地用一个编辑器发现它。例如：

```
config: line 17: syntax error
```

The Complete FreeBSD(4th)

一个可能性是你拼错了一个关键字。可以与GENERIC或LINT内核定义中的记录相比较一下。

为了能编译内核，你将需要大概50MB的自由空间。如果你没有足够的空间，你可以通过指定DEBUG-g编译选项来减少这个值到10MB，但如果你在以后的步骤中有什么问题，要把它找出来就更加困难。

编译内核

接着，修改编译目录，编译内核：

```
# cd ../../compile/FREEBIE
# make depend
# make all
```

即使目录已经被创建了，make depend也是必须的：除了创建依赖信息，它也会创建一些编译时必须的文件。

这一步编译一个内核和一些匹配的kld。它可能要花费一些时间，在一台比较慢的机器上可能要花费一个小时。你可以通过只建立内核(make kernel)就可以加快编译。你第一次建立内核时不要这样做，但如果你发现内核无法启动，你要修改配置文件，然后从一些源代码重新编译，你可以节约一些时间。

这儿也可能会出现一些错误，不幸的是它们通常是不会自解释性的。如果make命令失败，它通常会在你的内核描述中提示一个错误，config很难捕获它。另外，检查一下你的配置文件，如果你仍不能解决这个问题，用你的内核配置发邮件到questions@FreeBSD.ORG，它会很快地进行诊断。一个如何打断这些错误的描述已经工作很长一段时间了，但当前它仍然深深地隐藏着。

最后，安装新的内核：

```
# make install
```

这将把当前的内核命名成kernel.old，把新的内核拷贝到root目录作为kernel。

FreeBSD 5将为内核引入一个新的层次结构。内核将被安装在/boot/kernel/kernel，而旧的内核将被移到/boot/kernel.old/kernel。

接着，关闭系统，重新启动引导新的内核：

```
# shutdown -r now
```

如果新的内核没有启动，或检测设备失败，不要惊慌。重新启动机器，当启动命令提示符出现时，键入空格打断启动。然后键入旧的内核：

```
Ok unload                remove the kernel that the loader has loaded
Ok load kernel.old       load the previous kernel (version 4)
Ok load /boot/kernel.old/kernel load the previous kernel (version 5)
Ok boot
```

The Complete FreeBSD(4th)

当重新配置内核的时候，在手头保留一个能工作的内核是个好主意。这里有两点要注意：如果你的内核不能启动，当你编译一个新内核时，不要把它改成`kernel.old`。这将会修改先前的正常功能的内核。不要执行`make install`，而是执行`make reinstall`，这不会重新命名旧的内核。

你可能在键入`make install`时仍有一个错误，丢掉了你以前的好的内核。很容易出现这样一种无法启动系统的情况。保留一个命名为`kernel.save`的内核是一个好主意，这个安装过程会带来很多好处。

用一个好的内核启动以后，你可以检查你的配置文件，然后再设法编译它。一个有帮助的资源是`/var/log/messages`文件，记录了所有来自每次成功启动的内核信息。另外，`dmesg`命令将显示当前启动的内核信息。

移动内核有些难以理解，实际上内核有`schg1`（不变的）标记集。你必须首先“开启”它：

```
# chflags noschg /kernel
```

如果你想“锁定”你的新内核（或有那种问题的任何文件）到一个地方以便它不能被移动或修改：

`Schg`标记使它不可能修改这个文件。这是在4.4BSD中引入的文件标记之一。它无法用在所有的UNIX类系统中。

```
# chflags schg /kernel
```

建立设备节点

如果你已经往你的内核中添加了任何新的设备，你可能在使用它们之前，必须在`/dev`目录中添加设备节点。看看第310页了解更多细节。如果你正在使用`devfs`，默认第5版中使用，内核将自动创建设备节点，你不需要为它们担心。

内核加载模块

正如我们在这章的开头所看到的，你不一定要编译一个新的内核来完成你想要的功能。而是，只要用`Kernel Loadable Module (kld)`把它加载到内核。目录`/modules`包含了许多`kld`。要加载它们，可以使用`kldload`。

例如，如果你要加载SCO UNIX兼容性，你可以键入：

```
# kldload ibcs2
```

这会加载模块`/modules/ibcs2.ko`。注意你不必指定目录的名称，也不需要`.ko`扩展。

要找到哪个模块被加载了，可以使用`kldstat`：

```
# kldstat
Id Refs Address Size Name
1 50xc0100000 1d08b0 kernel
2 20xc120d000 a000 ibcs2.ko
3 10xc121b000 3000 ibcs2_coff.ko
5 10xc1771000 e000 linux.ko
```

The Complete FreeBSD(4th)

```
6 10xc177f000 bf000 vinum.ko
```

你也可以卸载一些kld，但不是它们中的所有东西。使用*kldunload*来完成这个工作：

```
# kldunload vinum
```

sysctl

*sysctl*是一个允许在内核中指定变量的新的内核接口。这些变量中有些是只读的：你可以看看，但不能修改。其它的是可修改的。

*sysctl*变量通常被简单地参考为*sysctls*。每个*sysctl*有一个"Management Information Base"(MIB)形式的名称，包含了用点来分开的一个层次式的名称排列。名称的第一个组件指出了它与内核的哪个部分相关。

下面的例子给你提供了一个如何使用*sysctl*程序的建议：

```
$ sysctl kern.ostype
FreeBSD
$ sysctl kern
(lists all sysctls starting with kern)
$ sysctl -l
(lists all sysctls)
# sysctl net.inet.ip.forwarding=1
net.inet.ip.forwarding: 0 -> 1
```

更多有意思的*sysctl*信息：

```
kern.ostype: FreeBSD
kern.osrelease: 4.6-RELEASE
kern.osrevision: 199506
kern.version: FreeBSD 4.6-RELEASE #0: Mon May 20 17:00:28 CST 2002
root@farout.lemis.com:/src/FreeBSD/4.6-RELEASE/src/sys/compile/FREEBIE
kern.hostname: echunga.lemis.com
kern.boottime: { sec = 1007165073, usec = 570637 } Sat Dec 1 10:34:33 2001
kern.bootfile: /kernel
kern.init_path: /sbin/init:/sbin/oinit:/sbin/init.bak:/stand/sysinstall
kern.module_path: /boot;/modules/
kern.coredump: 1
kern.corefile: /var/tmp/%N.core
kern.msgbuf: nreach TCP 213.46.243.23:25370 139.130.136.138:25 in via ppp0
net.inet.ip.fw.enable: 1
hw.machine: i386
hw.model: Pentium II/Pentium II Xeon/Celeron
hw.ncpu: 1
hw.byteorder: 1234
hw.physmem: 129949696
hw.usermem: 100556800
hw.pagesize: 4096
hw.floatingpoint: 1
hw.machine_arch: i386
hw.ata.ata_dma: 1
hw.ata.wc: 1
hw.ata.tags: 0
```

The Complete FreeBSD(4th)

```
hw.ata.atapi_dma: 0
compat.linux.osname: Linux
compat.linux.osrelease: 2.2.12
compat.linux.oss_version: 198144
```

这些中有许多没有描述，但一些不太明显：

- `kern.msgbuf`显示了内核信息缓存的内容，它是由`dmesg`程序列出来的。
- `kern.corefile`指出了由于一个进程失败而产生的核心dump文件名称的模板。默认的，核心文件会在当前的工作目录中结束——无论那是什么。通过指定一个绝对的目录名称，你可以确保任何核心文件加入一个指定的目录。文本`%N`会被程序的名称所替换。
- `hw.ata.wc`指出了ATA磁盘是否执行写入缓存。在这个例子中，它们这样做了。正如我们在前面看到的，写入缓存可能会提高磁盘性能，但它可能会在系统崩溃中引起数据丢失。
- 这个`sysctl`不能在一个运行的系统上被修改。你必须在启动时设置它。