第14章　设备驱动程序

操作系统的一个功能就是将用户和系统的硬件特性隔离。例如，虚拟文件系统（ VFS）使得系统上挂接的文件系统具有相同的接口，使用户不必关心底层的硬件设备。本章介绍 Linux 系统内核是如何管理系统中的硬件设备的。

14.1　硬件设备的管理

计算机系统中的物理设备都有自己的硬件控制器。例如键盘、鼠标和串行口是由 SuperIO 芯片控制的；IDE硬盘是由IDE控制器控制的，等等。每一个硬件控制器都有自己的控制及状态寄存器（CSR）,而且随设备的不同而不同。CSR用来启动和停止设备、初始化设备和诊断设备错误。设备驱动程序一般集成在操作系统内核，这样不同的应用程序就可以共享这些代码。Linux系统内核中的设备驱动程序在本质上是一些特殊的，常驻内存的低级硬件处理程序的共享库。正是Linux系统的设备驱动程序处理系统中的硬件设备的具体细节。

设备驱动程序的一个基本特点就是对设备的抽象处理。系统中的所有硬件设备看起来都和一般的文件一样，它们可以使用处理文件的标准系统调用来打开、关闭和读写。系统中的每一个设备都由一个设备文件来代表，例如，主 IDE硬盘的设备文件是 /dev/had。对于块设备和字符设备来说，这些设备文件可以使用 mknod命令创建。新建的设备文件使用主设备号和从设备号来描述此设备。网络设备的设备文件是当系统查找到网络设备并初始化网络控制器之后才建立的。一个设备驱动程序控制的所有设备有一个相同的主设备号，通过不同的从设备号来区分设备和它们的控制器。例如，主 IDE硬盘的每一个分区都有一个不同的从设备号，这样主 IDE 硬盘的第二个分区的设备文件是 /dev/hda2。Linux系统使用主设备号和系统中的一些表来将系统调用中使用的设备文件映射到设备驱动程序中。

Linux 系统支持三种类型的硬件设备：字符设备、块设备和网络设备。字符设备是直接读取的，不必使用缓冲区。例如，系统的串行口 /dev/cua0 和 /dev/cua1。块设备每次只能读取一定大小的块的倍数，通常一块是 512或者1024字节。块设备通过缓冲区读写，并且可以随机地读写。块设备可以通过它们的设备文件存取，但通常是通过文件系统存取。只有块设备支持挂接的文件系统。网络设备是通过BSD 套接字界面存取的。

Linux系统支持多种设备，这些设备的驱动程序之间有一些共同的特点：

- 内核代码：设备驱动程序是系统内核的一部分，所以如果驱动程序出现错误的话，将可能严重地破坏整个系统。
- 内核接口：设备驱动程序必须为系统内核或者它们的子系统提供一个标准的接口。例如，一个终端驱动程序必须为 Linux内核提供一个文件I/O接口；一个SCSI设备驱动程序应该为SCSI子系统提供一个SCSI设备接口，同时SCSI子系统也应为系统内核提供文件 I/O和缓冲区。
- 内核机制和服务：设备驱动程序利用一些标准的内核服务，例如内存分配等。
- 可装入：大多数的Linux设备驱动程序都可以在需要时装入内核，在不需要时卸载。
- 可设置：Linux系统设备驱动程序可以集成为系统内核的一部分，至于哪一部分需要集成到内核中，可以在系统编译时设置。

- 动态性：当系统启动并且各个设备驱动程序初始化以后，驱动程序将维护其控制的设备。如果设备驱动程序控制的设备并不存在，也并不妨碍系统的运行。在这种情况下，设备的驱动程序只是多占用了一点系统的内存。

14.2 轮询和中断

每当设备发出一个命令，例如“移动读写头到软盘的第 42 扇区”，设备驱动程序可以选择如何确定命令是否完成。设备驱动程序即可以选择轮询设备，也可以选择使用中断。

轮询设备是指每隔一定的时间驱动程序就读取一次设备的状态寄存器，直到状态寄存器的值有所改变为止。但当设备驱动程序是系统内核的一部分时，轮询将会使系统的效率变得极低，因为在设备的请求完成之前，系统将无法进行任何其他的工作。使用轮询的设备驱动程序通过系统计时器来使系统内核在稍后的时间调用设备驱动程序的一个子过程。 Linux系统中的软盘就是这样工作的。比此方法更好的方法是使用中断。

中断驱动的设备驱动程序是指每当硬件设备需要服务时，它就会产生一个硬件中断。例如，每当一个以太网设备从网络中接收一个以太网数据包时会产生一个中断。 Linux系统内核能够将中断从产生中断的设备传递到相应的设备驱动程序中，这可以通过设备驱动程序在系统内核中登记中断的使用来实现。设备驱动程序将中断处理过程的地址和设备希望使用的中断号登记在内核中。通过/proc/interrupts可以知道设备驱动程序使用的是哪一个中断，以及每种类型的中断有几个：

```
0:     727432    计时器
1:      20534    键盘
2:          0    级联（cascade）
3:      79691 +  串口
4:      28258 +  串口
5:          1    声霸卡
11:     20868 +  aic7xxx
13:         1    数学错误
14:       247 +  ide0
15:       170 +  ide1
```

这种中断源的请求是在驱动程序初始化时完成的。系统中的一些中断是固定的，例如，软盘控制器使用中断6。其他有些中断，例如来自PCI设备的中断是在系统启动时动态分配的。在这种情况下，设备驱动程序必须在要求中断的所有权之前首先知道设备的中断号。对于 PCI中断，Linux系统支持标准的PCI BIOS反馈来确定设备在系统中的信息，包括设备的中断号。

14.3 直接内存存取

当交换的数据量很小时，中断驱动的设备驱动程序可以通过使用中断来和设备之间交换数据。但如果是高速设备，例如硬盘或者以太网设备，那么中断方式就会占用过多的系统内存。

解决此问题的方法是使用直接内存存取，也就是 DMA。一个DMA控制器允许在设备没有处理器的干预下和系统内存直接交换数据。一个 PC机的 ISA DMA控制器共有8个DMA通道，其中7个可以用于设备驱动程序。每一个DMA通道都包括一个16位的地址寄存器和一个16位的记数寄存器。在开始传输数据之前，设备驱动程序需要设置 DMA通道的地址和记数寄存器，以及数据传输的方向 —— 是读数据还是写数据。在这之后，设备就可以随时开始 DMA传输了。

当传输结束后，设备将会中断系统的CPU，但在传输过程中，CPU可以做其他工作。

设备驱动程序在使用DMA时应该小心。首先， DMA控制器不知道虚拟内存的存在，它只能存取系统的物理内存。这样 DMA使用的系统内存必须是连续的物理内存块。这意味着你不能直接使用DMA传输数据到一个进程的虚拟内存地址空间。但你可以将一个进程的物理页面锁定到内存中，这样在DMA操作期间进程的物理页面就不会被交换出系统的物理内存。第二，DMA控制器无法存取整个的物理内存。 DMA通道的地址寄存器存储的是DMA地址的后16位，而前8位则来自页面寄存器。这表明DMA请求只能使用系统最开始的16M地址。

DMA通道是稀少资源，它们只有7个，并且设备驱动器之间不能共享DMA通道。就像使用中断一样，设备驱程序也必须了解它使用的是哪一个 DMA通道。系统中一些设备有固定的DMA通道，例如，软盘使用的是 DMA通道2。有时一个设备的DMA通道可以通过跳线设置。更为灵活的设备可以通过它们的 CSR了解使用的DMA通道，这样，设备驱动程序可以随便地选择一个空闲的DMA通道。

Linux系统使用数据结构dma_chan来了解系统中DMA通道的使用情况。dma_chan结构只有两个字段，一个是指向描述 DMA通道使用者的字符串的指针，另一个用来标识DMA通道是否已被分配。你在系统中使用cat/proc/dma命令时，显示的正是dma_chan结构的数组。

14.4 内存

设备驱动程序在使用内存时也应十分小心。因为设备驱动程序是系统内核的一部分，所以它也无法使用虚拟内存。每当一个设备驱动程序运行时，当前进程都有可能改变。设备驱动程序不能依靠一个特殊的进程来运行，即使设备驱动程序代表这个进程。象系统内核的其他部分一样，设备驱动程序使用数据结构来保持和它所控制的设备的联系。这些数据结构可以作为设备驱动程序的一部分来静态地分配，但这样可能会使得系统的内核变大。大多数设备驱动程序使用内核中没有分页的内存来存储它们的数据。

系统的设备驱动程序使用Linux系统提供的内核内存分配和撤消程序。内核中的内存是以 2的乘方为单位分配的，例如， 128或者512字节。

在请求内核内存时， Linux系统可能需要做很多的额外工作。例如，如果系统中的空闲的物理内存太少，被占用的物理内存可能被放弃或者交换到系统的交换文件中。一般情况下，Linux系统将会挂起设备驱动程序的请求，把进程送入到一个等待队列中直到有足够的内存以后再继续运行。但并不是系统中所有的设备驱动程序都希望以这种方式运行。所以当系统不能立即为设备驱动程序分配足够的内存时，内核的内存分配程序将会返回一个错误。如果设备驱动程序希望使用DMA和被分配的内存之间交换数据，那么它可以将内存指定为可以直接存取的。这样，只有Linux系统内核，而不是设备驱动程序需要知道DMA是如何构成的。

14.5 设备驱动程序和内核之间的接口

Linux系统和设备驱动程序之间使用标准的交互接口。无论是字符设备、块设备还是网络设备的设备驱动程序，当系统内核请求它们的服务时，都使用同样的接口。这样， Linux系统内核可以用同样的方法来使用完全不同的各种设备。

Linux系统是一个动态的操作系统，每当Linux系统内核启动时，它都有可能检测到不同的物理设备，这样就可能需要不同的驱动程序。 Linux允许你在构建系统内核时使用设置脚本将设备驱动程序包括在系统内核中。这些驱动程序在系统启动时初始化，它们可能找不到所控制的设备。其他的设备驱动程序可以在需要时作为内核模块装入到系统内核中。为了适应设备驱

动程序的这种动态的特性，设备驱动程序在其初始化时就在系统内核中进行登记。 Linux系统使用设备驱动程序的登记表作为内核和驱动程序接口的一部分。这些表中包括指向处理程序的指针和其他信息。

14.5.1 字符设备

存取系统中的字符设备和存取系统文件一样。应用程序使用标准的系统调用来打开、读写和关闭字符设备，就像使用一个文件一样，即使是 PPP使用的调制解调器也是一样。当字符设备初始化时，设备驱动程序通过在由数据结构 device_struct组成的chrdevs数组中添加一个入口来向系统内核注册。设备的主设备号用来作为此 chrdevs的索引。一个设备的主设备号是固定的。

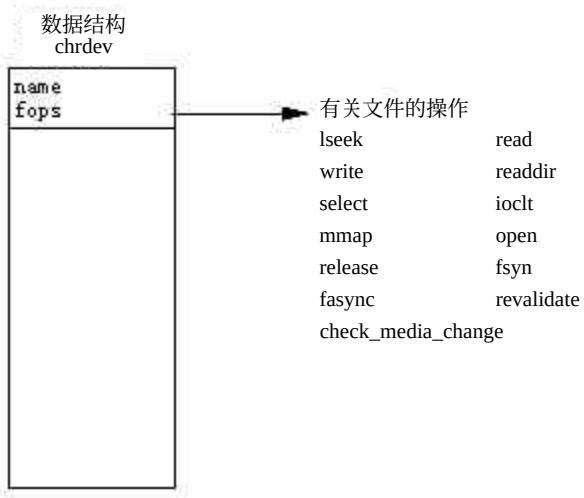

图14-1 字符设备驱动程序示意图

数组chrdevs中的每一个入口都是一个device_struct结构。如图14-1所示，device_struct结构包括两个指针元素：一个指向登记的设备驱动程序名，另一个指向一个包括各种文件操作过程的地址的数组。此数组中包括的地址指向设备驱动程序中处理文件的操作，例如，打开、读写和关闭子过程。字符设备的/proc/devices中内容就来自chrdevs数组。

当代表一个字符设备的字符设备文件打开以后，系统必须知道如何正确地调用相应字符设备的文件操作过程。和一般的文件或者目录一样，每一个字符设备文件都由 VFS索引节点来表示，VFS索引节点包括设备的主标识符和从标识符。 VFS索引节点是在文件系统检测到设备文件名时，由文件系统创建的。

每一个VFS索引节点都和一系列的文件操作相连，并且这些文件操作随索引节点代表的文件的不同而不同。每当一个VFS索引节点所代表的字符设备文件创建时，它的有关文件的操作就设置为缺省的字符设备操作。缺省的文件操作只包含一个打开文件的操作。当应用程序打开一个字符设备文件时，通用的文件操作使用设备的主标识符作为 chrdevs数组的索引，依此可以找到有关此设备的各种文件操作。它还将建立起描述此字符设备文件的文件数据结构，使得其中的文件操作指针指向此设备驱动程序中的有关文件的操作。这样，应用程序中的文件操作将会映射到字符设备的文件操作调用中。

14.5.2 块设备

对于块设备的存取和对于文件的存取一样。它的机制和字符设备使用的机制相同。 Linux系统中有一个blkdevs数组，它描述了一系列在系统中注册的块设备。数组 blkdevs也使用设备的主设备号作为索引。它的每一个入口均由数据结构 device_struct组成。和字符设备不一样的是，块设备有几种类型，例如 SCSI设备和IDE设备。每一类的块设备都在 Linux系统内核中注册，并向内核提供自己的文件操作。例如，一个 SCSI 设备驱动程序为SCSI子系统提供接口，反过来SCSI子系统使用这些接口为系统内核提供有关此设备的文件操作。

每一个块设备驱动程序必须既向缓冲区提供接口，也提供一般的文件操作接口。每一个块设备都在blk_dev数组中有一个blk_dev_struct结构的入口。数据结构blk_dev_struct 包括一个请求过程的地址和一个指向请求数据结构链表的指针，每一个请求数据结构都代表一个来自缓冲区的请求。

每当缓冲区希望和一个在系统中注册的块设备交换数据，它都会在 blk_dev_struct中添加一个请求数据结构。如图14-2所示，每一个请求都有一个指针指向一个或者多个 buffer_head数据结构，每一个buffer_head结构都是一个读写数据块的请求。每一个请求结构都在一个静态链表all_requests中。如果请求添加到了一个空的请求链表中，则调用设备驱动程序的请求函数来开始处理请求队列。否则，设备驱动程序只是简单地处理请求队列中的每一个请求。

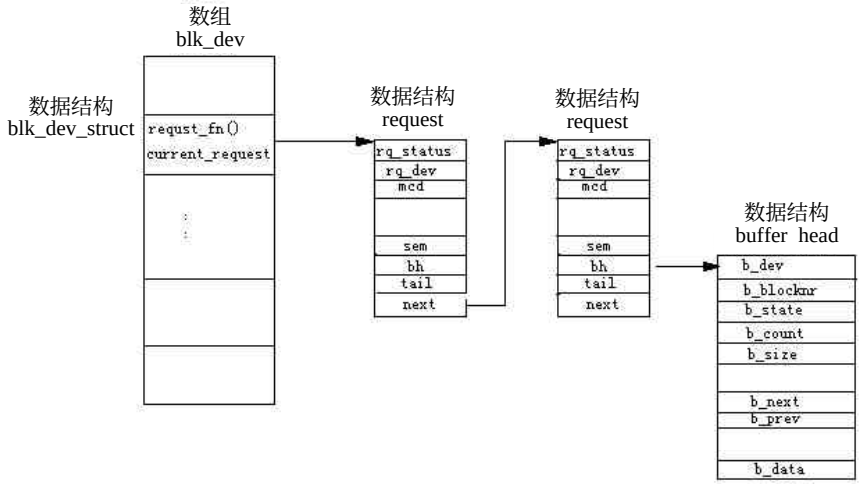

图14-2 块设备驱动程序数据结构示意图

一旦设备驱动程序完成了一个请求，它将把 buffer_head 结构从request结构中移走，并把buffer_head结构标记为已更新，同时将它解锁。这样就可以唤醒等待锁定操作完成的进程。

14.6 硬盘

硬盘驱动器提供了一个永久性存储数据的方法。硬盘一般使用它的柱面号、磁头号和扇区号来描述。例如，启动时Linux系统可能这样描述一个IDE硬盘：

hdb: Conner Peripherals 540MB - CFS540A, 516MB w/64kB Cache, CHS=1050/16/63

这说明此硬盘一共有1050个柱面，16个磁头以及每个磁道上有63个扇区。

硬盘还可以进一步地分区。很多的 Linux系统的硬盘都包括三个分区：一个是 DOS文件系统分区，一个是EXT2文件系统分区，第三个是交换分区。硬盘使用硬盘分区表描述分区，分

区表的每一个入口都包括以磁头、扇区和柱面表示的分区起始位置。

以下是包括两个主分区的硬盘：

Disk /dev/sda: 64 heads, 32 sectors, 510 cylinders

Units = cylinders of 2048 * 512 bytes

Device Boot	Begin	Start	End	Blocks	Id	System
/dev/sda1	1	1	478	489456	83	Linux native
/dev/sda2	479	479	510	32768	82	Linux swap

Expert command (m for help): p

Disk /dev/sda: 64 heads, 32 sectors, 510 cylinders

Nr	AF	Hd	Sec	Cyl	Hd	Sec	Cyl	Start	Size	ID
1	00	1	1	0	63	32	477	32	978912	83
2	00	0	1	478	63	32	509	978944	65536	82
3	00	0	0	0	0	0	0	0	0 00	
4	00	0	0	0	0	0	0	0	00	

这表明硬盘的第一个分区开始于柱面0，磁头1和扇区1，一直到柱面477，磁头63和扇区32。第二个分区从柱面478开始一直到最里面的柱面为止。

系统初始化时，Linux系统将确定硬盘的数目和类型。另外，Linux系统也检测硬盘是如何分区的。这些信息都保存在gendisk_head指针指向的由数据结构gendisk组成的链表中。每一个硬盘子系统，例如IDE硬盘系统，初始化时将会产生代表此硬盘的数据结构 gendisk，并且同时会注册有关此硬盘的文件操作并把它添加到数据结构 blk_dev中。每一个gendisk结构中都包括一个唯一的主设备号，并且和设备文件中的主设备号匹配。例如，SCSI硬盘子系统创建一个gendisk结构，其入口是sd，主设备号是8。图14-3显示了两个gendisk结构，一个是SCSI硬盘系统的gendisk结构，另一个是IDE硬盘的gendisk结构。

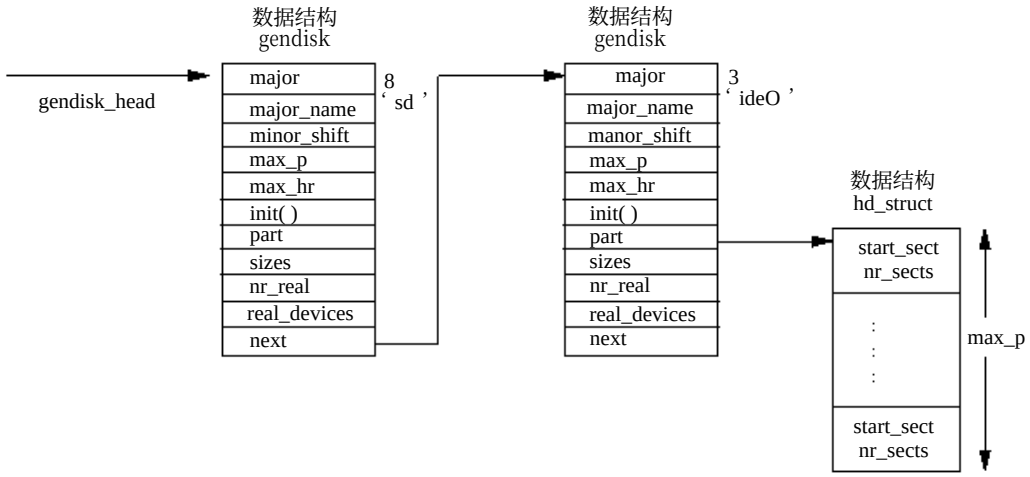

图14-3 硬盘链表示意图

虽然是硬盘子系统在初始化时创建的 gendisk结构，但只有当Linux系统进行分区检测时才会用到gendisk结构。每一个硬盘子系统都有一个自己的数据结构，允许它将设备的主设备号和从设备号映射到物理硬盘的各个分区。每当从硬盘中读取一个数据块时，无论是通过缓冲区

还是通过有关的文件操作，系统内核都将使用设备文件（例如 /dev/sda2）中的主设备号找到相应的设备，然后由设备驱动程序或者设备子系统将从设备号映射到真正的物理设备中。

14.6.1 IDE 硬盘

Linux系统中最为常用的硬盘就是IDE（Integrated Disk Electronic）硬盘。一个IDE硬盘控制器最多可以支持两个硬盘，一个是主硬盘，另一个是从硬盘。而系统中的第一个硬盘控制器叫做主硬盘控制器，第二个硬盘控制器叫做从硬盘控制器。 Linux根据系统检测到硬盘的顺序来命名硬盘。这样，主硬盘控制器上的主硬盘是 /dev/hda，而主硬盘控制器上的从硬盘是/dev/hdb。/dev/hdc则是从硬盘控制器上的主硬盘。 IDE硬盘子系统将硬盘控制器而不是硬盘登记到Linux系统中。系统中主硬盘控制器的主标识符号是 3，从硬盘控制器的主标识符是 22。这意味着如果一个系统中有两个IDE硬盘控制器，那么在blk_dev 和 blkdevs数组中的入口分别是3和22。 在硬盘的设备文件中，硬盘 /dev/hda 和 /dev/hdb都代表主硬盘控制器，它们的主标识符号都是3。任何对这两个设备文件的有关文件和缓冲区进行的操作都将转向到同一个 IDE控制器，因为系统内核使用主标识符号作为索引。当有请求产生时，由 IDE控制器使用设备标识符中的从设备号来辨别是哪一个硬盘的请求。例如，主硬盘控制器的从硬盘 /dev/hdb的设备标识符为（3，64），而此硬盘的第一个分区/dev/hdb1的设备标识符为（3，65）。

14.6.2 初始化IDE 硬盘子系统

IDE硬盘的发展已经有很长的时间了。在这期间， IDE硬盘的接口有了很大的变化，这就使得IDE硬盘的初始化变得比较复杂。

Linux系统最多支持4个IDE硬盘控制器。每一个硬盘控制器都对应数组 ide_hwifs中的数据结构ide_hwif_t，ide_hwif_t结构中包括两个数据结构 ide_drive_t，每一个代表一个主硬盘驱动器或一个从硬盘驱动器。在硬盘系统的初始化过程中， Linux系统首先查看系统CMOS中有关硬盘的信息。Linux系统使用CMOS中的硬盘信息来创建ide_hwif_t结构，以反映系统中的硬盘控制器和所连接的硬盘的信息。在硬盘操作过程中， IDE硬盘驱动程序把命令写入到系统I/O内存空间中的IDE命令寄存器中。缺省的主硬盘控制器的控制和状态寄存器的 I/O地址是0x1F0 - 0x1F7。IDE硬盘驱动程序记录每一个硬盘控制器的 Linux系统缓冲区和VFS索引节点，并把这些信息添加到blkdevs数组中的blk_dev结构内。IDE硬盘也需要适当的中断号。一般情况下，主硬盘控制器的中断号是14，从硬盘控制器的中断号是15，但这些数值可以使用内核中的命令行参数覆盖。IDE硬盘驱动程序在系统启动过程中，也会在 gendisk链表中添加gendisk入口。此链表在稍后会用来检测所有的硬盘分区表。

14.6.3 SCSI 硬盘

SCSI总线是一个十分有效的对等结构的数据总线，一条 SCSI总线可以支持多达8个设备。总线上的每一个设备都有一个唯一的标识符。总线上任何两个设备之间可以进行同步或者异步数据传输，传输的数据宽度是32位，每秒的传输速率可达40M。从源设备到目的设备的一次单独的传输交易可以分为 8个不同的阶段。你可以通过总线上的 5个信号来分辨SCSI总线当前所处的阶段。这8个阶段分别是：

- 总线空闲

没有设备在控制总线，而且当前总线上没有交易发生。

- 设备判优

如果一个SCSI设备希望获得SCSI总线的控制权，那么它将会把它的 SCSI标识符发送到地址插脚上。具有最大的SCSI标识符的设备获得总线的控制权。

• 选择设备

当一个设备获得总线的控制权以后，它将会把目地设备的 SCSI标识符发送到地址插脚中，以便通知目地设备它将发送一个命令。

• 重新选择设备

SCSI设备可以在一次请求的处理过程中中断此请求。这时目的设备可以重新选择源设备。不是所有的SCSI设备都支持这个阶段。

• 命令

可以从源设备向目的设备传送6、10 或者 12 个字节的命令。

• 数据输入和输出

在此阶段，数据在源设备和目的设备之间传送。

• 状态

所有的命令完成之后，它允许目的设备发送一个状态字节以通知源设备传送的成功或者失败。

• 信息的输入、输出

在此阶段传送一些额外的信息。

Linux系统中的SCSI硬盘子系统由两个基本元素构成，每一个都由一个数据结构来表示：

• 主机（host）

一个SCSI主机就是一个SCSI控制器，例如NCR810 PCI SCSI控制器就是一个SCSI主机。如果Linux系统中包括多于一个的同类型的SCSI控制器，则每一个都是单独的SCSI主机。这意味着一个SCSI设备驱动程序可以控制多于一个的控制器。 SCSI主机一般是SCSI命令的源设备。

• 设备（Device）

最常见的SCSI设备是SCSI硬盘，但SCSI也支持其他类型的设备：磁带， CD-ROM以及其他的SCSI设备。SCSI设备一般就是SCSI命令的目的。但不同的设备必须区别对待，例如，对于像SCSI磁带和CD-ROM等设备， Linux系统必须检测其中的读写介质是否已经被移动。不同的SCSI磁盘类型有不同的主设备号,这样Linux 系统就可以直接找到相应的SCSI类型的设备。

14.6.4 初始化 SCSI 磁盘子系统

因为SCSI总线和SCSI设备的动态特性，初始化一个 SCSI子系统相对来说特别地麻烦。Linux系统在系统启动时对SCSI子系统进行初始化。系统首先查找SCSI 控制器（也就是SCSI主机），然后逐个检测SCSI总线以便搜寻所有的SCSI设备。最后系统初始化这些设备，使得系统内核的其他部分可以通过一般的文件形式和缓冲区设备操作来使用 SCSI设备。SCSI设备的初始化可以分为4个阶段：

• 首先， Linux系统查找在Linux系统构建内核时集成到内核中的 SCSI控制器。如图14-4所示，每一个SCSI控制器都有一个包括在builtin_scsi_hosts数组中的Scsi_Host_Template结构的入口。 Scsi_Host_Template结构包括一些指向SCSI过程的指针，这些过程用来实现SCSI主机的动作，例如检测SCSI主机都带有哪些SCSI设备。SCSI子系统在设置时要调用这些过程，并且这些过程是此种SCSI主机的驱动程序的一部分。每一个带有 SCSI设备的SCSI主机的Scsi_Host_Template数据结构组成了一个scsi_hosts的链表，链表中的每一个Scsi_Host数据结构都代表一个SCSI主机类型。例如，一个系统中包括两个 NCR810 PCI

SCSI控制器，则每一个控制器都有一个Scsi_Host结构，每一个Scsi_Host结构都指向代表设备驱动程序的Scsi_Host_Template。

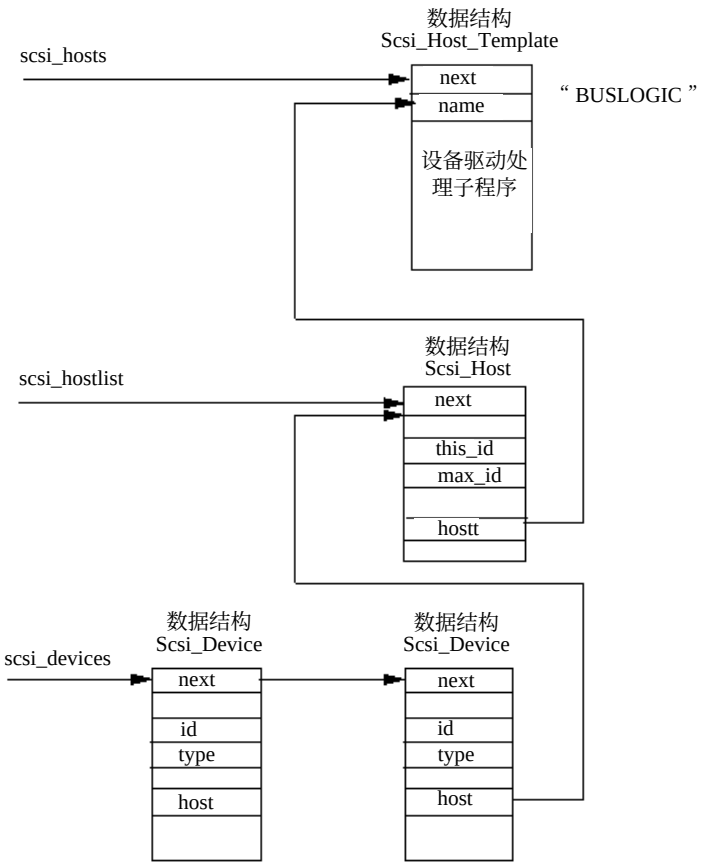

图14-4 SCSI 设备示意图

- 系统在检测到SCSI主机以后，SCSI子系统必须进一步确认每一个SCSI主机总线上所带的SCSI设备。SCSI设备的标号从0到7，而且每个设备的设备号或者标识符在设备所在的总线上都是唯一的。SCSI初始化程序使用TEST_UNIT_READY命令来检测SCSI总线上的SCSI设备。当某一个设备有回应后，初始化程序再使用 ENQUIRY命令读取设备的标识符，标识符中包含厂商名称、设备型号以及版本号。 Scsi_Cmnd数据结构代表所有的SCSI命令，而这些命令通过调用在Scsi_Host_Template结构中的设备驱动程序子过程来传递给此SCSI主机的设备驱动程序。系统中的每一个SCSI设备都有一个Scsi_Device数据结构，每一个Scsi_Device数据结构都指向Scsi_Host。所有的Scsi_Device结构组成了一个scsi_devices 链表。
- SCSI设备共有4种：磁盘，磁带， CD-ROM如通用设备。每一种类型的SCSI设备都使用不同的主设备号在系统中单独注册。每一种SCSI设备，例如SCSI 磁盘，都有自己的设备表，它使用设备表和系统中相应的设备驱动程序以及 CSI主机进行通信。每一种SCSI 类型的Scsi_Device_Template结构都包括此种SCSI设备类型的信息和执行各种任务的子程序的地址。
- SCSI子系统初始化的最后阶段是调用每一个注册的 Scsi_Device_Template结构的结束

函数。

14.6.5 传递块设备请求

Linux 系统初始化SCSI子系统之后，就可以使用 SCSI设备了。每一个可以使用的 SCSI设备都在系统内核中注册，所以 Linux系统可以直接将块设备请求传递给相应的 SCSI设备。在Linux系统中可以通过blk_dev数据结构进行缓冲区请求，或者通过 blkdevs结构进行文件操作。例如，对于一个包含多个EXT2文件系统分区的SCSI磁盘驱动器，系统内核缓冲区请求是如何传递到相应的挂接一个EXT2文件系统分区的磁盘上的？每当产生一个从 SCSI磁盘分区中读写数据块的请求时，都会创建一个新的请求结构，并将其添加到 blk_dev数组的current_request链表中。如果请求链表正在处理，那么缓冲区则不需要做任何事情，否则它将通知 SCSI处理请求队列。

系统中的每一个SCSI硬盘都由一个Scsi_Disk数据结构来代表。Scsi_Disk数据结构保存在rscsi_disks数组中，并且使用SCSI硬盘分区的从设备号作为索引。例如， /dev/sdb1的主设备号是8，从设备号是17，那么它的索引是17。每一个Scsi_Disk数据结构都包括一个指向代表此设备的Scsi_Device结构的指针，Scsi_Device结构又指向拥有此设备的SCSI主机的Scsi_Host结构。来自缓冲区的请求数据结构将会被翻译成描述SCSI命令的Scsi_Cmd结构。

14.7 网络设备

对Linux系统而言，网络设备是一个收发数据包的整体。它经常是一个物理设备，例如以太网卡。一些网络设备可以是软件的，例如用来给你自己发送数据的回馈设备。每一个网络设备都用一个数据结构来代表它。 Linux系统启动时，随着网络的初始化，网络设备驱动程序将在Linux系统中登记它所控制的设备。设备数据结构中包括有关设备的信息和允许系统支持的各种网络协议所使用的设备服务的各种函数的地址。这些函数主要用于传输数据。设备使用标准的网络服务机制来把接受到的数据传输到相应的协议层。所有传输的网络数据包都使用一个sk_buff数据结构。此sk_buff数据结构十分灵活，它可以允许网络协议头方便地添加和移走。

下面介绍device结构中所包括的有关网络设备的信息。

14.7.1 网络设备文件名

网络设备文件是在系统的网络设备被系统检测到和初始化的同时创建的。它们的名字代表了设备的类型，同种类型的多个设备则从 0开始编号。例如，以太网卡的设备文件是 /dev/eth0、/dev/eth1、/dev/eth2、以此类推。一些常见的网络设备是：

/dev/ethN	Ethernet 设备
/dev/slN	SLIP 设备
/dev/pppN	PPP 设备
/dev/lo	Loopback 设备

14.7.2 总线信息

总线信息是设备驱动程序在控制设备时所使用的信息。 irq号是设备使用的中断号，基地址是设备的控制和状态寄存器在I/O内存中的地址， DMA通道是网络设备使用的DMA通道号。所有这些信息都是在系统启动时设备初始化的过程中设置的。

14.7.3 网络接口标记

网络接口标记用来描述网络设备的特点和功能：

IFF_UP	网络接口已经打开并且正在运行 。
IFF_BROADCAST	设备中的广播地址有效 。
IFF_DEBUG	打开设备调试。
IFF_LOOPBACK	此设备是一个回馈设备。
IFF_POINTTOPOINT	这是一个点到点的连接 (SLIP 和 PPP)。
IFF_NOTRAILERS	不存在网络尾部。
IFF_RUNNING	资源分配。
IFF_NOARP	不支持ARP 协议。
IFF_PROMISC	设备在混合状态，它将接收各个地址的数据包 。
IFF_ALLMULTI	接收所有的IP贞。
IFF_MULTICAST	可以接收IP贞。

14.7.4 协议信息

每个设备中都有网络协议层如何使用设备的信息：

• mtu

指出网络可以传输的数据包的最大值，但不包括数据包上添加的链路层的数据头。

• Family

指出设备可以支持的协议族。

• Type

硬件接口类型描述了网络设备可以连接的介质类型，包括 Ethernet、X.25、Token Ring、Slip、PPP 和 Apple Localtalk。

• Addresses

和网络设备有关的一系列地址，包括IP地址。

• 数据包队列

这是sk_buff的队列，里面是等待传输的数据包。

• 支持的函数

每一个设备都提供一个标准的子函数集，这样设备的链路层协议可以调用这些子函数。这些函数包括设置子函数、帧传输子函数以及一些搜集统计信息的子函数。可以使用 ifconfig命令查看这些统计信息。

14.7.5 初始化网络设备

网络设备驱动程序象其他的Linux设备驱动程序一样，可以嵌入到Linux系统中内核中。每一个网络设备都由一个网络设备数据结构代表，这些数据结构组成一个由 dev_base指针指向的网络设备链表。如果网络各协议层需要设备执行某些操作时，它们将调用设备提供的各种设备服务子程序，这些子程序的地址保存在设备的数据结构中。但设备初始化时，设备数据结构中只包括初始化程序和检测子程序的地址。

网络设备驱动程序需要解决两个问题。第一个问题是，不是所有构建到 Linux内核中的网络设备驱动程序都有可以控制的设备。第二个问题是，系统中的以太网设备一般调用 /dev/eth0、

/dev/eth1和类似的文件，而不管该文件中的设备驱动程序到底是什么。关于第一个问题十分容易解决。每当调用一个网络设备的初始化程序时，它都将返回一个状态值用来指示它是否找到它所控制的一个设备。如果驱动程序无法找到任何设备，那么它在设备链表中的入口将会被移走。如果设备驱动程序找到一个它控制的设备，那么它将有关该设备的信息和网络设备驱动程序支持的子函数的地址添加到设备的数据结构中。

第二个问题的解决可能稍微地复杂一些。在网络设备链表中一共有 8个标准的入口，从eth0一直到 eth7。它们初始化的过程是一样的，也就是初始化程序轮流检测每一个以太网设备驱动程序，直到发现某一个驱动程序下带有设备为止。当检测到设备以后，驱动程序将会建立此设备的数据结构。同时，驱动程序也开始初始化它控制的物理设备和有关的一些信息，例如，使用的中断号，DMA通道等等。如果驱动程序检测到它控制的设备多于一个，那么它将控制所有这些设备的数据结构。一旦所有 8个标准的/dev/ethN分配完毕以后，就不再检测其他的以太网设备了。