

第6章 IP 编 址

6.1 引言

本章讨论 Net/3 如何管理 IP 地址信息。我们从 `in_ifaddr` 和 `sockaddr_in` 结构开始，它们基于通用的 `ifaddr` 和 `sockaddr` 结构。

本章其余部分讨论 IP 地址的指派和几个查询接口数据结构与维护 IP 地址的实用函数。

6.1.1 IP 地址

虽然我们假设读者熟悉基本的 Internet 编址系统，仍然有几点值得指出。

在 IP 模型中，地址是指派给一个系统（一个主机或路由器）中的网络接口而不是系统本身。在系统有多个接口的情况下，系统有多重初始地址，并有多重 IP 地址。一个路由器被定义为有多重初始地址。如我们所看到的，这个体系特点有几个小分支。

IP 地址定义了 5 类。A、B 和 C 类地址支持单播通信。D 类地址支持 IP 多播。在一个多播通信中，一个单独的源方发送一个数据报给多个目标方。D 类地址和多播协议在第 12 章说明。E 类地址是试验用的。接收的 E 类地址分组被不参与试验的主机丢弃。

我们强调 IP 多播和硬件多播间的区别是重要的。硬件多播的特点是数据链路硬件用来将帧传输给多个硬件接口。有些网络硬件，如以太网，支持数据链路多播。其他硬件可能不支持。

IP 多播是一个在 IP 系统内实现的软件特性，将分组传输给多个可能在 Internet 中任何位置的 IP 地址。

我们假设读者熟悉 IP 网络的子网划分 (RFC 950 [Mogul and Postel 1985] 和卷 1 的第 3 章)。我们会看到每个网络接口有一个相关的子网掩码，它是判断一个分组是否到达它最后的目的或还需要被转发的关键。通常，当提及一个 IP 地址的网络部分时，我们包括任何可能定义的子网。当需要区分网络和子网时，我们就要明确地指出来。

环回网络，127.0.0.0，是一个特殊的 A 类网络。这种格式的地址是不会出现在一个主机的外部的。发送到这个网络的分组被环回并被这个主机接收。

RFC 1122 要求所有在环回网络中的地址被正确地处理。因为环回接口必须指派一个地址，很多系统选择 127.0.0.1 作为环回地址。如果系统不能正确识别，像 127.0.0.2 这样的地址可能不能被路由到环回接口而被传输到一个连接的网络，这是不允许的。有些系统可能正确地路由这个到环回接口的分组，但由于目标地址与地址 127.0.0.1 不匹配，分组被丢弃。

图 18-2 显示了一个 Net/3 系统配置为拒绝接收发送到一个不是 127.0.0.1 的环回地址的分组。

6.1.2 IP 地址的印刷规定

我们通常以点分十进制数表示法来显示一个 IP 地址。图 6-1 列出了每类 IP 地址的范围。

地址类	范 围	类 型
A	0.0.0.0到127.255.255.255	单播
B	128.0.0.0到191.255.255.255	
C	192.0.0.0到223.255.255.255	
D	224.0.0.0到239.255.255.255	多播
E	240.0.0.0到247.255.255.255	试验性

图6-1 不同IP地址类的范围

对于我们的有些例子，子网字段不按一个字节对齐（即，一个网络/子网/主机在一个B类网络中分为16/11/5）。从点分十进制数表示法很难表示这样的地址，因此我们还是用方块图来说明IP地址的内容。我们用三个部分显示每个地址：网络、子网和主机。每个部分的阴影指示它的内容。图6-2用我们网络示例(1.14节)中的主机sun的以太网接口来同时说明块表示法和点分十进制数表示法。

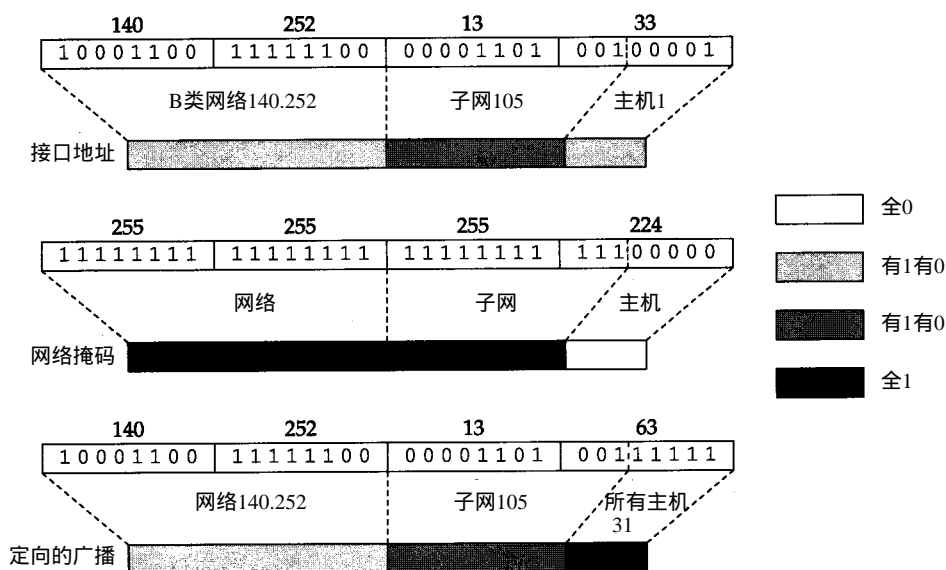


图6-2 可选的IP地址表示法

当地址的一个部分不是全为0或1时，我们使用两个中等程度的阴影。有两种中等程度的阴影，这样我们就能区分网络和子网部分或用来显示如图 6-31所示的地址组合。

6.1.3 主机和路由器

在一个Internet上的系统通常能划分为两类：主机和路由器。一个主机通常有一个网络接口，并且是一个IP分组的源或目标方。一个路由器有多个网络接口，当分组向它的目标方移动时将分组从一个网络转发到下一个网络。为执行这个功能，路由器用各种专用路由协议来交换关于网络拓扑的信息。IP路由问题比较复杂，在第18章开始讨论它们。

如果一个有多个网络接口的系统不在网络接口间路由分组，仍然叫一个主机。一个系统可能既是一个主机又是一个路由器。这种情况经常发生在当一个路由器提供运输层服务如用

于配置的 Telnet 访问，或用于网络管理的 SNMP 时。当区分一个主机和路由器间的意义并不重要时，我们使用术语系统。

不谨慎地配置一个路由器会干扰一个网络的正常运转，因此 RFC 1122 规定一个系统必须默认为一个主机来操作，并且必须显式地由一个管理员来配置作为一个路由器操作。这样做是不鼓励管理员将通用主机作为路由器来操作而没有仔细地配置。在 Net/3 中，如果全局整数 `ipforwarding` 不为 0，则一个系统作为一个路由器；如果 `ipforwarding` 为 0 (默认)，则系统作为一个主机。

在 Net/3 中，一个路由器通常称为网关，虽然术语网关现在更多的是与一个提供应用层路由的系统相关，如一个电子邮件网关，而不是转发 IP 分组的系统。我们在本书中使用术语路由器，并假设 `ipforwarding` 非 0。在编译 Net/3 内核期间，当 GATEWAY 被定义时，我们还有条件地包括所有代码，它们将 `ipforwarding` 定义为 1。

6.2 代码介绍

图 6-3 所列的两个头文件和两个 C 文件包含本章中讨论的结构定义和实用函数。

文 件	说 明
<code>netinet/in.h</code>	Internet 地址定义
<code>netinet/in_var.h</code>	Internet 接口定义
<code>netinet/in.c</code>	Internet 初始化和实用函数
<code>netinet/if.c</code>	Internet 接口实用函数

图 6-3 本章讨论的文件

全局变量

图 6-4 所列的是本章中介绍的两个全局变量。

变 量	数据类型	说 明
<code>in_ifaddr</code>	<code>struct in_ifaddr *</code>	<code>in_ifaddr</code> 结构列表的首部
<code>in_interfaces</code>	<code>int</code>	有 IP 能力的接口个数

图 6-4 在本章中介绍的全局变量

6.3 接口和地址小结

在本章讨论的所有接口和地址结构的一个例子配置如图 6-5 所示。

图 6-5 显示了我们的三个接口例子：以太网接口、SLIP 接口和环回接口。它们都有一个链路层地址作为地址列表中的第一个结点。显示的以太网接口有两个 IP 地址，SLIP 接口有一个 IP 地址，并且环回接口有一个 IP 地址和一个 OSI 地址。

注意所有的 IP 地址被链接到 `in_ifaddr` 列表中，并且所有链路层地址能从 `ifnet_addrs` 数组访问。

为了清楚起见，图 6-5 没有画出每个 `ifaddr` 结构中的指针 `ifa_ifp`。这些指针回指包含此 `ifaddr` 结构的列表的首部 `ifnet` 结构。

接下来的部分讨论图 6-5 中的数据结构及用来查看和修改这些结构的 IP 专用 `ioctl` 命令。

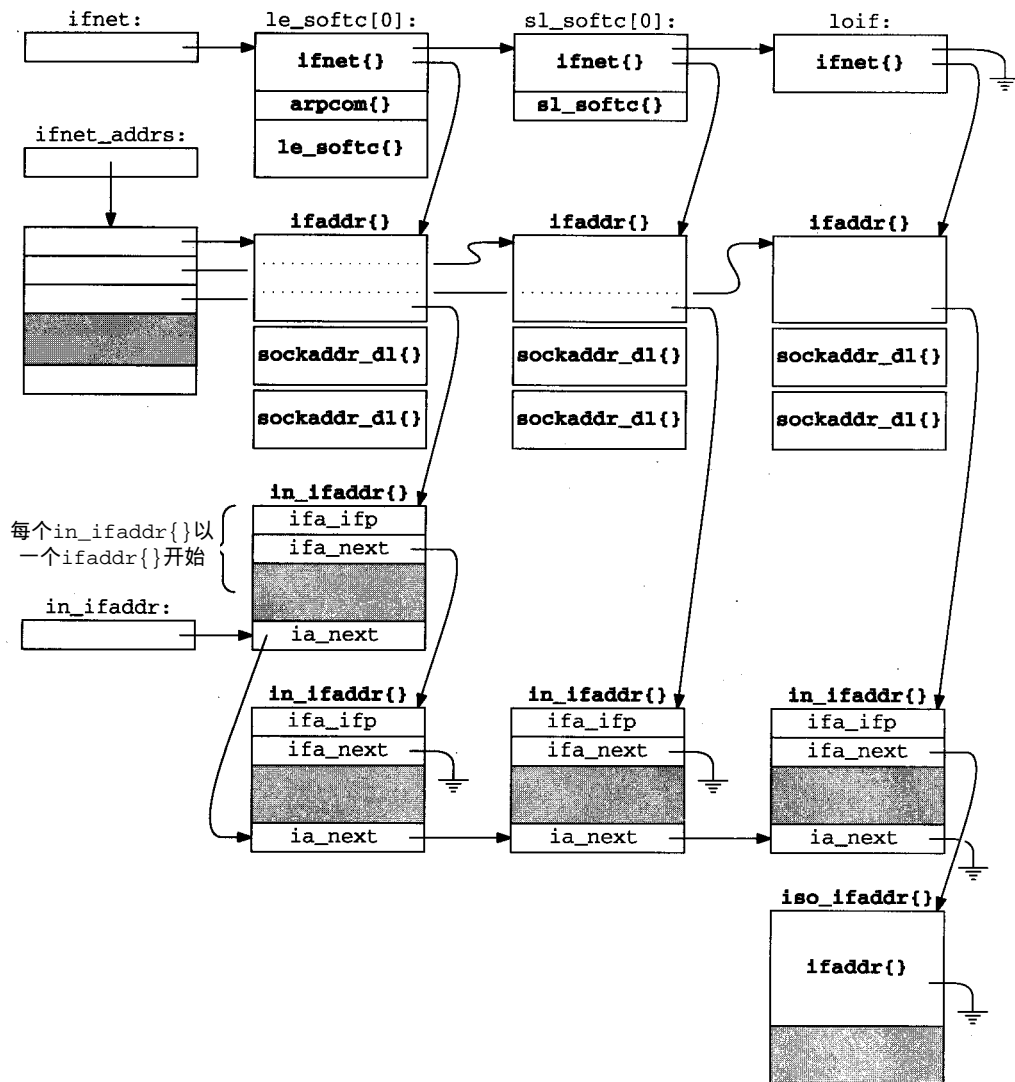


图6-5 接口和地址数据结构

6.4 sockaddr_in结构

我们在第3章讨论了通用的 `sockaddr` 和 `ifaddr` 结构。现在我们显示 IP 专用的结构：`sockaddr_in` 和 `in_ifaddr`。在 Internet 域中的地址存放在一个 `sockaddr_in` 结构：

68-70 由于历史原因，Net/3以网络字节序将Internet地址存储在一个in_addr结构中。这个结构只有一个成员s_addr，它包含这个地址。虽然这是多余和混乱的，但在Net/3中一直保持这种组织方式。

106-112 sin_len总是16(结构sockaddr_in的大小),并且sin_family为AF_INET。sin_port是一个网络字节序(不是主机字节序)的16 bit值,用来分用运输层报文。sin_addr标识一个32 bit Internet地址。

图6-6显示了sockaddr_in的成员sin_port、sin_addr和sin_zero覆盖sockaddr

的成员sa_data。在Internet域中，sin_zero未用，但必须由全0字节组成(2.7节)。将它追加到sockaddr_in结构后面，以得到与一个sockaddr结构一样的长度。

```

68 struct in_addr {
69     u_long s_addr;           /* 32-bit IP address, net byte order */
70 };

106 struct sockaddr_in {
107     u_char sin_len;           /* sizeof (struct sockaddr_in) = 16 */
108     u_char sin_family;        /* AF_INET */
109     u_short sin_port;         /* 16-bit port number, net byte order */
110     struct in_addr sin_addr;
111     char sin_zero[8];         /* unused */
112 };

```

in.h

图6-6 结构sockaddr_in

通常，当一个Internet地址存储在一个u_long中时，它以主机字节序存储，以便于地址的压缩和位操作。在in_addr结构(图6-7)中的s_addr是一个值得注意的例外。

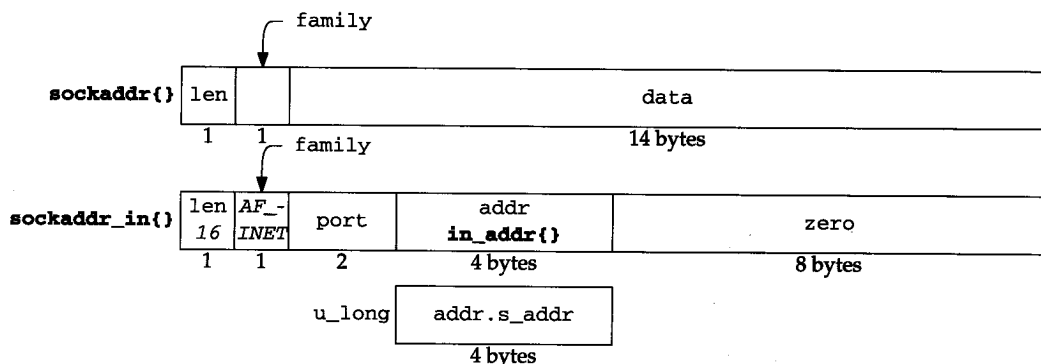


图6-7 一个sockaddr_in 结构(省略sin_)的组织

6.5 in_ifaddr结构

图6-8显示了为Internet协议定义的接口地址结构。对于每个指派给一个接口的IP地址，分配了一个in_ifaddr结构，并且添加到接口地址列表中和IP地址全局列表中(图6-5)。

41-45 in_ifaddr开始是一个通用接口地址结构ia_ifa，跟着是IP专用成员。ifaddr结构显示在图3-15中。两个宏ia_ifp和ia_flags简化了对存储在通用ifaddr结构中的接口指针和接口地址标志的访问。ia_next维护指派给任意接口的所有Internet地址的链接列表。这个列表独立于每个接口关联的链路层ifaddr结构列表，并且通过全局列表in_ifaddr来访问。

46-54 其余的成员(除了ia_multiaddrs)显示在图6-9中，它显示了在我们的B类网络例子中sun的三个接口的相应值。地址按主机字节序以u_long变量存储；变量in_addr和sockaddr_in按照网络字节序存储。sun有一个PPP接口，但显示在本表中的信息对于一个PPP或SLIP接口是一样的。

55-56 结构in_ifaddr的最后一个成员指向一个in_multi结构的列表(12.6节)，其中每项包含与此接口有关的一个IP多播地址。

```

41 struct in_ifaddr {
42     struct ifaddr ia_ifa;           /* protocol-independent info */
43 #define ia_ifp          ia_ifa.ifa_ifp
44 #define ia_flags        ia_ifa.ifa_flags
45     struct in_ifaddr *ia_next;      /* next internet addresses list */
46     u_long ia_net;                  /* network number of interface */
47     u_long ia_netmask;              /* mask of net part */
48     u_long ia_subnet;              /* subnet number, including net */
49     u_long ia_subnetmask;          /* mask of subnet part */
50     struct in_addr ia_netbroadcast; /* to recognize net broadcasts */
51     struct sockaddr_in ia_addr;     /* space for interface name */
52     struct sockaddr_in ia_dstaddr;  /* space for broadcast addr */
53 #define ia_broadaddr    ia_dstaddr
54     struct sockaddr_in ia_sockmask; /* space for general netmask */
55     struct in_multi *ia_multiaddrs; /* list of multicast addresses */
56 };

```

in_var.h

in_var.h

图6-8 结构in_ifaddr

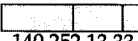


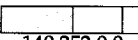

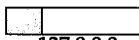


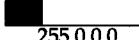
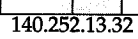

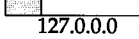
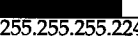

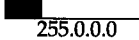



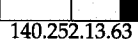
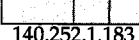

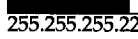
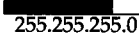
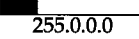
变 量	类 型	以 太 网	PPP	环 回	说 明
ia_addr	sockaddr_in	 140.252.13.33	 140.252.1.29	 127.0.0.1	网络，子网和主机号
ia_net	u_long	 140.252.0.0	 140.252.0.0	 127.0.0.0	网络号
ia_netmask	u_long	 255.255.0.0	 255.255.0.0	 255.0.0.0	网络号掩码
ia_subnet	u_long	 140.252.13.32	 140.252.1.0	 127.0.0.0	网络和子网号
ia_subnetmask	u_long	 255.255.255.224	 255.255.255.0	 255.0.0.0	网络和子网掩码
ia_netbroadcast	in_addr	 140.252.255.255	 140.252.255.255	 127.255.255.255	网络广播地址
ia_broadaddr	sockaddr_in	 140.252.13.63			定向广播地址
ia_dstaddr	sockaddr_in		 140.252.1.183	 127.0.0.1	目的地址
ia_sockmask	sockaddr_in	 255.255.255.224	 255.255.255.0	 255.0.0.0	像ia_subnetmask 但是用网络字节序

图6-9 sun上的以太网、PPP和环回in_ifaddr 结构

6.6 地址指派

在第4章中，我们显示了当接口结构在系统初始化期间被识别时的初始化。在 Internet 协议能通过这个接口进行通信前，必须指派一个 IP 地址。一旦 Net/3 内核运行，程序 ifconfig 就配置这些接口，ifconfig 通过在某个插口上的 ioctl 系统调用来发送配置命令。这通常通过 /etc/netstart 脚本来实现，这个脚本在系统引导时执行。

图6-10显示了本章中讨论的 ioctl 命令。命令相关的地址必须是此命令指定插口所支持的地址族类(即，你不能通过一个 UDP 插口配置一个 OSI 地址)。对于 IP 地址，ioctl 命令在一

个UDP插口上发送。

命 令	参 数	函 数	说 明
<i>SIOCGIFADDR</i>	<i>struct ifreq *</i>	<i>in_control</i>	获得接口地址
<i>SIOCGIFNETMASK</i>	<i>struct ifreq *</i>	<i>in_control</i>	获得接口网络掩码
<i>SIOCGIFDSTADDR</i>	<i>struct ifreq *</i>	<i>in_control</i>	获得接口目标地址
<i>SIOCGIFBRDADDR</i>	<i>struct ifreq *</i>	<i>in_control</i>	获得接口广播地址
<i>SIOCSIFADDR</i>	<i>struct ifreq *</i>	<i>in_control</i>	设置接口地址
<i>SIOCSIFNETMASK</i>	<i>struct ifreq *</i>	<i>in_control</i>	设置接口网络掩码
<i>SIOCSIFDSTADDR</i>	<i>struct ifreq *</i>	<i>in_control</i>	设置接口目标地址
<i>SIOCSIFBRDADDR</i>	<i>struct ifreq *</i>	<i>in_control</i>	设置接口广播地址
<i>SIOCDELADDR</i>	<i>struct ifreq *</i>	<i>in_control</i>	删除接口地址
<i>SIOCAIFADDR</i>	<i>struct in_aliasreq *</i>	<i>in_control</i>	添加接口地址

图6-10 接口ioctl 命令

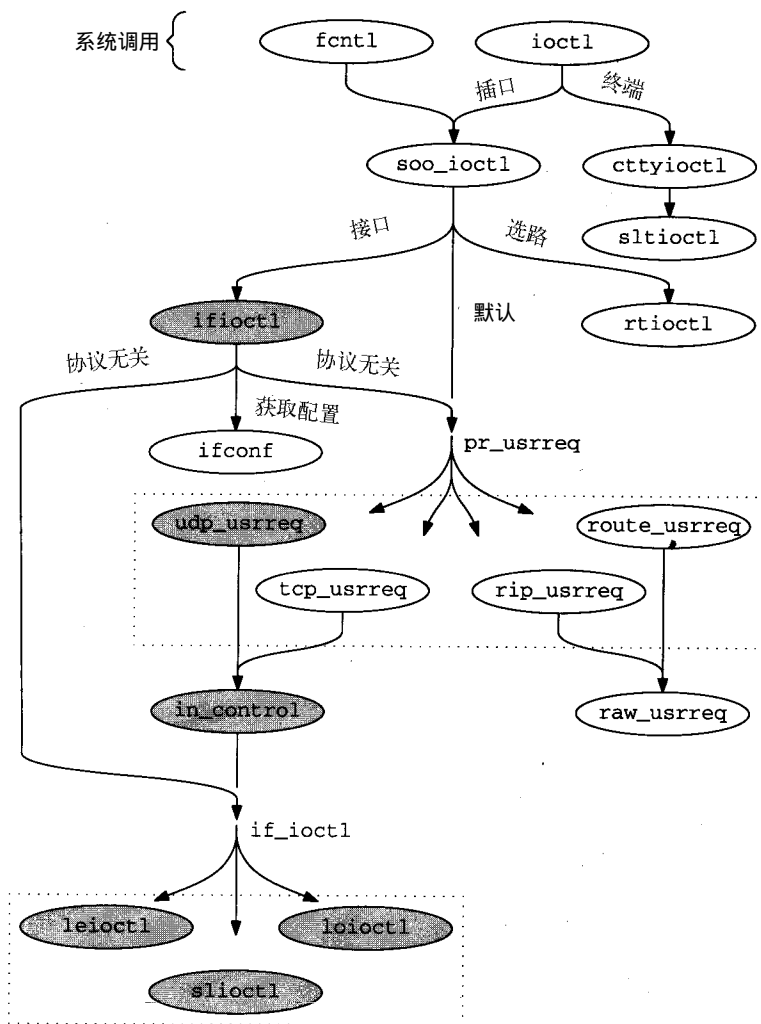


图6-11 本章中说明的ioctl 函数

获得地址信息的命令从 `SIOCG` 开始, 设置地址信息的命令从 `SIOCS` 开始。 `SIOC` 代表 socket ioctl, `G` 代表 get, 而 `S` 代表 set。

在第4章中, 我们看到了5个与协议无关的 `ioctl` 命令。图6-10中的命令修改一个接口的相关地址信息。由于地址是特定协议使用的, 因此, 命令处理是与协议相关的。图6-11强调了与这些命令关联的 `ioctl` 相关函数。

6.6.1 ifioctl函数

如图6-11所示, `ifioctl` 将协议无关的 `ioctl` 命令传递给此插口关联协议的 `pr_usrreq` 函数。将控制交给 `udp_usrreq`, 并且又立即传给 `in_control`, 在 `in_control` 中进行大部分的处理。如果在一个 TCP 插口上发送同样的命令, 控制最后也会到达 `in_control`。图6-12再次显示了 `ifioctl` 中的 `default` 代码, 第一次显示在图4-22中。

```
447     default:
448         if (so->so_proto == 0)
449             return (EOPNOTSUPP);
450         return ((*so->so_proto->pr_usrreq) (so, PRU_CONTROL,
451                                             cmd, data, ifp));
452     }
453     return (0);
454 }
```

if.c

if.c

图6-12 函数 `ifioctl` : 特定协议的命令

447-454 函数将图6-10中所列 `ioctl` 命令的所有相关数据传给与请求指定的插口相关联的协议的用户请求函数。对于一个 UDP 插口, 调用 `udp_usrreq`。23.10节讨论 `udp_usrreq` 函数的细节。现在, 我们仅需要查看 `udp_usrreq` 中的 `PRU_CONTROL` 代码:

```
if (req == PRU_CONTROL)
    return (in_control(so, (int)m, (caddr_t)addr, (struct ifnet *)control));
```

6.6.2 in_control函数

图6-11显示了通过 `soo_ioctl` 中的 `default` 或 `ifioctl` 中的与协议相关的情况, 控制能到达 `in_control`。在这两种情况中, `udp_usrreq` 调用 `in_control`, 并返回 `in_control` 的返回值。图6-13显示了 `in_control`。

132-145 `so` 指向这个 `ioctl` 命令(由第二个参数 `cmd` 标识)指定的插口。第三个参数 `data` 指向命令所用或返回的数据(图6-10的第二列)。最后一个参数 `ifp` 为空(来自 `soo_ioctl` 的无接口 `ioctl`)或指向结构 `ifreq` 或 `in_aliasreq` 中命名的接口(来自 `ifioctl` 的接口 `ioctl`)。 `in_control` 初始化 `ifr` 和 `ifra` 来访问作为一个 `ifreq` 或 `in_aliasreq` 结构的 `data`。

146-152 如果 `ifp` 指向一个 `ifnet` 结构, 这个 `for` 循环找到与此接口关联的 Internet 地址列表中的第一个地址。如果发现一个地址, `ia` 指向它的 `in_ifaddr` 结构; 否则 `ia` 为空。

若 `ifp` 为空, `cmd` 就不会匹配第一个 `switch` 中的任何情况; 或第二个 `switch` 中任何非默认情况。在第二个 `switch` 中的 `default` 情况中, 当 `ifp` 为空时, 返回 `EOPNOTSUPP`。

153-330 `in_control` 中的第一个 `switch` 确保在第二个 `switch` 处理命令之前每个命令的前提条件都满足。在后面的章节会单独说明各个情况。


```

132 in_control(so, cmd, data, ifp)                                in.c
133 struct socket *so;
134 int      cmd;
135 caddr_t data;
136 struct ifnet *ifp;
137 {
138     struct ifreq *ifr = (struct ifreq *) data;
139     struct in_ifaddr *ia = 0;
140     struct ifaddr *ifa;
141     struct in_ifaddr *oia;
142     struct in_aliasreq *ifra = (struct in_aliasreq *) data;
143     struct sockaddr_in oldaddr;
144     int      error, hostIsNew, maskIsNew;
145     u_long   i;

146     /*
147      * Find address for this interface, if it exists.
148      */
149     if (ifp)
150         for (ia = in_ifaddr; ia; ia = ia->ia_next)
151             if (ia->ia_ifp == ifp)
152                 break;

153     switch (cmd) {

        /* establish preconditions for commands */

218     }
219     switch (cmd) {

        /* perform the commands */

```

图6-13 函数in_control

如果在第二个switch中的default情况被执行，ifp指向一个接口结构；并且如果接口有一个if_ioctl函数，则in_control将ioctl命令传给这个接口进行设备的特定处理。

Net/3不定义任何会被default情况处理的接口命令。但是，一个特定设备的驱动程序可能会定义它自己的接口ioctl命令，并通过这个case来处理它们。

331-332 我们会看到这个switch语句中的很多情况都直接返回了。如果控制落到两个switch语句外，则in_control返回0。第二个switch中有几个case执行了跳出语句。

我们按照下面的顺序查看这个接口ioctl命令：

- 指派一个地址、网络掩码或目标地址；
- 指派一个广播地址；

- 取回一个地址、网络掩码、目标地址或广播地址；
- 给一个接口指派多播地址；
- 删除一个地址。

对于每组命令，在第一个 switch 语句中进行前提条件处理，然后在第二个 switch 语句中处理命令。

6.6.3 前提条件：SIOCSIFADDR、SIOCSIFNETMASK和SIOCSIFDSTADDR

图6-14显示了对SIOCSIFADDR、SIOCSIFNETMASK和SIOCSIFDSTADDR的前提条件检验。

```

166     case SIOCSIFADDR:
167     case SIOCSIFNETMASK:
168     case SIOCSIFDSTADDR:
169         if ((so->so_state & SS_PRIV) == 0)
170             return (EPERM);

171     if (ifp == 0)
172         panic("in_control");
173     if (ia == (struct in_ifaddr *) 0) {
174         oia = (struct in_ifaddr *)
175             malloc(sizeof *oia, M_IFADDR, M_WAITOK);
176         if (oia == (struct in_ifaddr *) NULL)
177             return (ENOBUFS);
178         bzero((caddr_t) oia, sizeof *oia);
179         if (ia = in_ifaddr) {
180             for (; ia->ia_next; ia = ia->ia_next)
181                 continue;
182             ia->ia_next = oia;
183         } else
184             in_ifaddr = oia;
185         ia = oia;
186         if (ifa = ifp->if_addrlist) {
187             for (; ifa->ifa_next; ifa = ifa->ifa_next)
188                 continue;
189             ifa->ifa_next = (struct ifaddr *) ia;
190         } else
191             ifp->if_addrlist = (struct ifaddr *) ia;

192         ia->ia_ifa.ifa_addr = (struct sockaddr *) &ia->ia_addr;
193         ia->ia_ifa.ifa_dstaddr
194             = (struct sockaddr *) &ia->ia_dstaddr;
195         ia->ia_ifa.ifa_netmask
196             = (struct sockaddr *) &ia->ia_sockmask;
197         ia->ia_sockmask.sin_len = 8;
198         if (ifp->if_flags & IFF_BROADCAST) {
199             ia->ia_broadaddr.sin_len = sizeof(ia->ia_addr);
200             ia->ia_broadaddr.sin_family = AF_INET;
201         }
202         ia->ia_ifp = ifp;
203         if (ifp != &loif)
204             in_interfaces++;
205     }
206     break;

```

图6-14 函数in_control：地址指派

1. 仅用于超级用户

166-172 如果这个插口不是由一个超级用户进程创建的，这些命令被禁止，并且 `in_control` 返回 `EPERM`。如果此请求没有关联的接口，内核调用 `panic`。由于如果 `ifioctl` 不能找到一个接口，它就返回(图4-22)，因此，`panic` 从来不会被调用。

当一个超级用户进程创建一个插口时，`socreate`(图15-16)设置标志 `SS_PRIV`。因为这里的检验是针对标志而不是有效的进程用户 ID 的，所以一个设置用户 ID 的根进程能创建一个插口，并且放弃它的超级用户权限，但仍然能发送有特权的 `ioctl` 命令。

2. 分配结构

173-191 如果 `ia` 为空，命令请求一个新的地址。`in_control` 分配一个 `in_ifaddr` 结构，用 `bzero` 清除它，并且将它链接到系统的 `in_ifaddr` 列表中和此接口的 `if_addrlist` 列表中。

3. 初始化结构

192-206 代码的下一部分初始化 `in_ifaddr` 结构。首先，在此结构的 `ifaddr` 部分的通用指针被初始化为指向结构 `in_ifaddr` 中的结构 `sockaddr_in`。必要时，此函数还初始化结构 `ia_sockmask` 和 `ia_broadaddr`。图6-15说明了初始化后的结构 `in_ifaddr`。

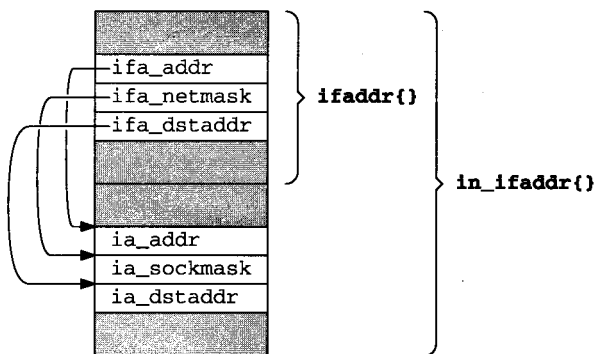


图6-15 被 `in_control` 初始化后的一个 `in_ifaddr` 结构

202-206 最后，`in_control` 建立从 `in_ifaddr` 到此接口的 `ifnet` 结构的回指指针。

`Net/3` 在 `in_interfaces` 中只统计非环回接口。

6.6.4 地址指派：SIOCSIFADDR

前提条件处理代码保证 `ia` 指向一个要被 `SIOCSIFADDR` 命令修改的 `in_ifaddr` 结构。图6-16显示了 `in_control` 第二个 `switch` 中处理这个命令的执行代码。

```

259     case SIOCSIFADDR:
260         return (in_ifinit(ifp, ia,
261                         (struct sockaddr_in *) &ifr->ifr_addr, 1));

```

in.c

图6-16 函数 `in_control` : 地址指派

159-261 `in_ifinit` 完成所有的工作。IP 地址包含在 `ifreq` 结构 (`ifr_addr`) 里传递给 `in_ifinit`。

6.6.5 in_ifinit 函数

`in_ifinit` 的主要步骤是：

- 将地址复制到此结构并将此变化通知硬件；

- 忽略原地址配置的任何路由；
- 为这个地址建立一个子网掩码；
- 建立一个默认路由到连接的网络(或主机)；
- 将此接口加入到所有主机组。

从图6-17开始分三个部分讨论这段代码。

353-357 `in_ifinit`的四个参数为：`ifp`，指向接口结构的指针；`ia`，指向要改变的`in_ifaddr`结构的指针；`sin`，指向请求的IP地址的指针；`scrub`，指示这个接口如果存在路由应该被忽略。`i`保存主机字节序的IP地址。

```

353 in_ifinit(ifp, ia, sin, scrub)                                in.c
354 struct ifnet *ifp;
355 struct in_ifaddr *ia;
356 struct sockaddr_in *sin;
357 int      scrub;
358 {
359     u_long i = ntohl(sin->sin_addr.s_addr);
360     struct sockaddr_in oldaddr;
361     int      s = splimp(), flags = RTF_UP, error, ether_output();

362     oldaddr = ia->ia_addr;
363     ia->ia_addr = *sin;
364     /*
365      * Give the interface a chance to initialize
366      * if this is its first address,
367      * and to validate the address if necessary.
368      */
369     if (ifp->if_ioctl &&
370         (error = (*ifp->if_ioctl) (ifp, SIOCSIFADDR, (caddr_t) ia))) {
371         splx(s);
372         ia->ia_addr = oldaddr;
373         return (error);
374     }
375     if (ifp->if_output == ether_output) { /* XXX: Another Kludge */
376         ia->ia_ifa.ifa_rtrequest = arp_rtrequest;
377         ia->ia_ifa.ifa_flags |= RTF_CLONING;
378     }
379     splx(s);
380     if (scrub) {
381         ia->ia_ifa.ifa_addr = (struct sockaddr *) &oldaddr;
382         in_ifscrub(ifp, ia);
383         ia->ia_ifa.ifa_addr = (struct sockaddr *) &ia->ia_addr;
384     }

```

图6-17 函数`in_ifinit`：地址指派和路由初始化

1. 指派地址并通知硬件

358-374 `in_control`将原来的地址保存在`oldaddr`中，万一发生差错时，必须恢复它。如果接口定义了一个 `if_ioctl`函数，则 `in_control`调用它。相同接口的三个函数 `leioctl`、`slioclt`和`loioctl`在下一节讨论。如果发生差错，恢复原来的地址，并且 `in_control`返回。

2. 以太网配置

375-378 对于以太网设备，`arp_rtrequest`作为链路层路由函数被选择，并且设置 `RTF_CLONING`标志。`arp_rtrequest`在21.13节讨论，而`RTF_CLONING`在19.4节的最后

讨论。如xxx注释所建议，在此加入代码以避免改变所有以太网驱动程序。

3. 忽略原来的路由

379-384 如果调用者要求已存在的路由被清除，原地址被重新连接到 `ifa_addr`，同时 `in_ifscrub` 找到并废除任何基于老地址的路由。`in_ifscrub` 返回后，新地址被恢复。

`in_ifinit` 显示在图6-18中的部分构造网络和子网掩码。

```

385     if (IN_CLASSA(i))
386         ia->ia_netmask = IN_CLASSA_NET;
387     else if (IN_CLASSB(i))
388         ia->ia_netmask = IN_CLASSB_NET;
389     else
390         ia->ia_netmask = IN_CLASSC_NET;
391     /*
392     * The subnet mask usually includes at least the standard network part,
393     * but may may be smaller in the case of supernetting.
394     * If it is set, we believe it.
395     */
396     if (ia->ia_subnetmask == 0) {
397         ia->ia_subnetmask = ia->ia_netmask;
398         ia->ia_sockmask.sin_addr.s_addr = htonl(ia->ia_subnetmask);
399     } else
400         ia->ia_netmask &= ia->ia_subnetmask;
401     ia->ia_net = i & ia->ia_netmask;
402     ia->ia_subnet = i & ia->ia_subnetmask;
403     in_socktrim(&ia->ia_sockmask);

```

in.c

图6-18 函数 `in_ifinit` : 网络和子网掩码

4. 构造网络掩码和默认子网掩码

385-400 根据地址是一个A类、B类或C类地址，在 `ia_netmask` 中构造了一个尝试性网络掩码。如果这个地址没有子网掩码，`ia_subnetmask` 和 `ia_sockmask` 被初始化为 `ia_netmask` 中的尝试性掩码。

如果指定了一个子网，`in_ifinit` 将这个尝试性网络掩码和这个已存在的子网掩码进行逻辑与运算来获得一个新的网络掩码。这个操作可能会清除该尝试性网络掩码的一些 1 bit (它从来不设置 0 bit，因为 0 逻辑与任何值都得到 0)。在这种情况下，网络掩码比所考虑的地址类型所期望的要少一些 1 bit。

这叫作超级联网，它在 RFC 1519 [Fuller et al. 1993] 中作了描述。一个超级网络是几个 A 类、B 类或 C 类网络的一个群组。卷 1 的 10.8 节也讨论了超级联网。

一个接口默认配置为不划分子网（即，网络和子网的掩码相同）。一个显式请求（用 `SIOCSIFNETMASK` 或 `SIOCAIFADDR`）用来允许子网划分（或超级联网）。

5. 构造网络和子网数量

401-403 网络和子网数量通过网络和子网掩码从新地址中获得。函数 `in_socktrim` 通过查找掩码中包含 1 bit 的最后一个字节来设置 `in_sockmask` (是一个 `sockaddr_in` 结构) 的长度。

图6-19显示了 `in_ifinit` 的最后一部分，它为接口添加了一个路由，并加入所有主机多播组。

6. 为主机或网络建立路由

404-422 下一步是为新地址所指定的网络创建一个路由。`in_control` 从接口将路由度量

复制到结构in_ifaddr中。如果接口支持广播，则构造广播地址，并且把目的地址强制为分配给环回接口的地址。如果一个点对点接口没有一个指派给链路另一端的IP地址，则in_control在试图为这个无效地址建立路由前返回。

in_ifinit将flags初始化为RTF_UP，并与环回和点对点接口的RTF_HOST进行逻辑或。rtinit为此接口给这个网络(不设置RTF_HOST)或主机(设置RTF_HOST)安装一个路由。若rtinit安装成功，则设置ia_flags中的标志IFA_ROUTE，指示已给此地址安装了一个路由。

```

404      /*
405      * Add route for the network.
406      */
407      ia->ia_ifa.ifa_metric = ifp->if_metric;
408      if (ifp->if_flags & IFF_BROADCAST) {
409          ia->ia_broadaddr.sin_addr.s_addr =
410              htonl(ia->ia_subnet | ~ia->ia_subnetmask);
411          ia->ia_netbroadcast.s_addr =
412              htonl(ia->ia_net | ~ia->ia_netmask);
413      } else if (ifp->if_flags & IFF_LOOPBACK) {
414          ia->ia_ifa.ifa_dstaddr = ia->ia_ifa.ifa_addr;
415          flags |= RTF_HOST;
416      } else if (ifp->if_flags & IFF_POINTOPOINT) {
417          if (ia->ia_dstaddr.sin_family != AF_INET)
418              return (0);
419          flags |= RTF_HOST;
420      }
421      if ((error = rtinit(&(ia->ia_ifa), (int) RTM_ADD, flags)) == 0)
422          ia->ia_flags |= IFA_ROUTE;
423      /*
424      * If the interface supports multicast, join the "all hosts"
425      * multicast group on that interface.
426      */
427      if (ifp->if_flags & IFF_MULTICAST) {
428          struct in_addr addr;
429          addr.s_addr = htonl(INADDR_ALLHOSTS_GROUP);
430          in_addmulti(&addr, ifp);
431      }
432      return (error);
433 }

```

图6-19 函数in_ifinit：路由和多播组

7. 加入所有主机组

423-433 最后，一个有多播能力的接口当它被初始化时必须加入所有主机多播组。in_addmulti完成此工作，并在12.11节讨论。

6.6.6 网络掩码指派：SIOCSIFNETMASK

图6-20显示了网络掩码命令的处理。

```

262      case SIOCSIFNETMASK:
263          i = ifra->ifra_addr.sin_addr.s_addr;
264          ia->ia_subnetmask = ntohl(ia->ia_sockmask.sin_addr.s_addr = i);
265          break;

```

图6-20 函数in_control：网络掩码指派

262-265 `in_control`从`ifreq`结构中获取网络掩码，并将它以网络字节序保存在`ia_sockmask`中，以主机字节序保存在`ia_subnetmask`中。

6.6.7 目的地址指派：`SIOCSIFDSTADDR`

对于点对点接口，在链路另一端的系统的地址用 `SIOCSIFDSTADDR`命令指定。图6-14显示了图6-21中的代码的前提条件处理。

```

236     case SIOCSIFDSTADDR:
237         if ((ifp->if_flags & IFF_POINTOPOINT) == 0)
238             return (EINVAL);
239         oldaddr = ia->ia_dstaddr;
240         ia->ia_dstaddr = *(struct sockaddr_in *) &ifr->ifr_dstaddr;
241         if (ifp->if_ioctl && (error = (*ifp->if_ioctl)
242             (ifp, SIOCSIFDSTADDR, (caddr_t) ia))) {
243             ia->ia_dstaddr = oldaddr;
244             return (error);
245         }
246         if (ia->ia_flags & IFA_ROUTE) {
247             ia->ia_ifa.ifa_dstaddr = (struct sockaddr *) &oldaddr;
248             rtinit(&(ia->ia_ifa), (int) RTM_DELETE, RTF_HOST);
249             ia->ia_ifa.ifa_dstaddr =
250                 (struct sockaddr *) &ia->ia_dstaddr;
251             rtinit(&(ia->ia_ifa), (int) RTM_ADD, RTF_HOST | RTF_UP);
252         }
253         break;

```

in.c

图6-21 函数`in_control`：目的地址指派

236-245 只有点对点网络才有目的地址，因此对于其他网络，`in_control`返回`EINVAL`。将当前目的地址保存在`oldaddr`后，代码设置新地址，并通过函数`if_ioctl`通知硬件。如果发生差错，则恢复原地址。

246-253 如果地址原来有一个关联的路由，首先调用`rtinit`删除这个路由，并再次调用`rtinit`为新地址安装一个新路由。

6.6.8 获取接口信息

图6-22显示了命令`SIOCSIFBRDADDR`的前提条件处理，它同将接口信息返回给调用进程的`ioctl`命令一样。

```

207     case SIOCSIFBRDADDR:
208         if ((so->so_state & SS_PRIV) == 0)
209             return (EPERM);
210         /* FALLTHROUGH */
211     case SIOCGIFADDR:
212     case SIOCGIFNETMASK:
213     case SIOCGIFDSTADDR:
214     case SIOCGIFBRDADDR:
215         if (ia == (struct in_ifaddr *) 0)
216             return (EADDRNOTAVAIL);
217         break;

```

in.c

图6-22 函数`in_control`：前提条件处理

207-217 广播地址只能通过一个超级用户进程创建的插口来设置。命令 `SIOCSIFBRDADDR` 和4个 `SIOCGxxx` 命令仅当已经为此接口定义了一个地址时才起作用，在这种情况下，`ia` 不会为空(`ia`被`in_control`设置，图6-13)。如果`ia`为空，返回 `EADDRNOTAVAIL`。

这5个命令(4个 `get` 命令和一个 `set` 命令)的处理显示在图6-23中。

```

220     case SIOCGIFADDR:
221         *((struct sockaddr_in *) &ifr->ifr_addr) = ia->ia_addr;
222         break;
223     case SIOCGIFBRDADDR:
224         if ((ifp->if_flags & IFF_BROADCAST) == 0)
225             return (EINVAL);
226         *((struct sockaddr_in *) &ifr->ifr_dstaddr) = ia->ia_broadaddr;
227         break;
228     case SIOCGIFDSTADDR:
229         if ((ifp->if_flags & IFF_POINTOPOINT) == 0)
230             return (EINVAL);
231         *((struct sockaddr_in *) &ifr->ifr_dstaddr) = ia->ia_dstaddr;
232         break;
233     case SIOCGIFNETMASK:
234         *((struct sockaddr_in *) &ifr->ifr_addr) = ia->ia_sockmask;
235         break;

```

in.c

```

/* processing for SIOCSIFDSTADDR command (Figure 6.21) */

```

```

254     case SIOCSIFBRDADDR:
255         if ((ifp->if_flags & IFF_BROADCAST) == 0)
256             return (EINVAL);
257         ia->ia_broadaddr = *(struct sockaddr_in *) &ifr->ifr_broadaddr;
258         break;

```

in.c

图6-23 函数 `in_control`：处理

220-235 将单播地址、广播地址、目的地址或者网络掩码复制到 `ifreq` 结构。只有网络接口支持广播，广播地址才有效；并且只有点对点接口，目的地址才有效。

254-258 仅当接口支持广播，才从结构 `ifreq` 中复制广播地址。

6.6.9 每个接口多个IP地址

`SIOCGxxx` 和 `SIOCSxxx` 命令只操作与一个接口关联的第一个 IP 地址——在 `in_control` 开头的循环找到的第一个地址（图6-25）。为支持每个接口的多个 IP 地址，必须用 `SIOCAIFADDR` 命令指派和配置其他的地址。实际上，`SIOCAIFADDR` 能完成所有 `SIOCGxxx` 和 `SIOCSxxx` 命令能完成的操作。程序 `ifconfig` 使用 `SIOCAIFADDR` 来配置一个接口的所有地址信息。

如前所述，每个接口有多个地址便于在主机或网络改号时过渡。一个容错软件系统可能使用这个特性来准许一个备份系统充当一个故障系统的 IP 地址。

Net/3 的 `ifconfig` 程序的 `-alias` 选项将存放在一个 `in_aliasreq` 中的其他地址的相关信息传递给内核，如图6-24所示。

```

59 struct in_aliasreq {
60     char    ifra_name[IFNAMSIZ];    /* interface name, e.g. "en0" */
61     struct sockaddr_in ifra_addr;
62     struct sockaddr_in ifra_broadaddr;
63 #define ifra_dstaddr ifra_broadaddr
64     struct sockaddr_in ifra_mask;
65 };

```

— in_var.h

图6-24 结构in_aliasreq

59-65 注意，不像结构 ifreq，在结构 in_aliasreq 中没有定义联合。在一个单独的 ioctl 调用中可以为 SIOCAIFADDR 指定地址、广播地址和掩码。

SIOCAIFADDR 增加一个新地址或修改一个已存在地址的相关信息。SIOCDEFADDR 删除匹配的 IP 地址的 in_ifaddr 结构。图 6-25 显示命令 SIOCAIFADDR 和 SIOCDEFADDR 的前提条件处理，它假设在 in_control (图 6-13) 开头的循环已经将 ia 设置为指向与 ifra_name (如果存在) 指定的接口关联的第一个 IP 地址。

```

154     case SIOCAIFADDR:
155     case SIOCDEFADDR:
156         if (ifra->ifra_addr.sin_family == AF_INET)
157             for (oia = ia; ia; ia = ia->ia_next) {
158                 if (ia->ia_ifp == ifp &&
159                     ia->ia_addr.sin_addr.s_addr ==
160                     ifra->ifra_addr.sin_addr.s_addr)
161                     break;
162             }
163         if (cmd == SIOCDEFADDR && ia == 0)
164             return (EADDRNOTAVAIL);
165         /* FALLTHROUGH to Figure 6.14 */

```

— in.c

图6-25 函数in_control：添加和删除地址

154-165 因为 SIOCDEFADDR 代码只查看 *ifra 的前两个成员，图 6-25 所示的代码用于处理 SIOCAIFADDR (当 ifra 指向一个 in_aliasreq 结构时) 和 SIOCDEFADDR (当 ifra 指向一个 ifreq 结构时)。结构 in_aliasreq 和 ifreq 的前两个成员是一样的。

对于这两个命令，in_control 开头的循环启动 for 循环不断地查找与 ifra->ifra_addr 指定的 IP 地址相同的 in_ifaddr 结构。对于删除命令，如果地址没有找到，则返回 EADDRNOTAVAIL。

在这个处理删除命令的循环和检验后，控制落到我们在图 6-14 中讨论的代码之外。对于添加命令，图 6-14 的代码若找不到一个与 in_aliasreq 结构中地址匹配的 IP 地址，就分配一个新 in_ifaddr 结构。

6.6.10 附加 IP 地址：SIOCAIFADDR

这时 ia 指向一个新的 in_ifaddr 结构或一个包含与请求地址匹配的 IP 地址的旧 in_ifaddr 结构。SIOCAIFADDR 的处理显示在图 6-26 中。

266-277 因为 SIOCAIFADDR 能创建一个新地址或修改一个已存在地址的相关信息，标志 maskIsNew 和 hostIsNew 跟踪变化的情况。这样，在这个函数结束时，如果必要，能更新路由。

```

266     case SIOCAIFADDR:
267         maskIsNew = 0;
268         hostIsNew = 1;
269         error = 0;
270         if (ia->ia_addr.sin_family == AF_INET) {
271             if (ifra->ifra_addr.sin_len == 0) {
272                 ifra->ifra_addr = ia->ia_addr;
273                 hostIsNew = 0;
274             } else if (ifra->ifra_addr.sin_addr.s_addr ==
275                 ia->ia_addr.sin_addr.s_addr)
276                 hostIsNew = 0;
277         }
278         if (ifra->ifra_mask.sin_len) {
279             in_ifscrub(ifp, ia);
280             ia->ia_sockmask = ifra->ifra_mask;
281             ia->ia_subnetmask =
282                 ntohl(ia->ia_sockmask.sin_addr.s_addr);
283             maskIsNew = 1;
284         }
285         if ((ifp->if_flags & IFF_POINTOPOINT) &&
286             (ifra->ifra_dstaddr.sin_family == AF_INET)) {
287             in_ifscrub(ifp, ia);
288             ia->ia_dstaddr = ifra->ifra_dstaddr;
289             maskIsNew = 1; /* We lie; but the effect's the same */
290         }
291         if (ifra->ifra_addr.sin_family == AF_INET &&
292             (hostIsNew || maskIsNew))
293             error = in_ifinit(ifp, ia, &ifra->ifra_addr, 0);
294         if ((ifp->if_flags & IFF_BROADCAST) &&
295             (ifra->ifra_broadaddr.sin_family == AF_INET))
296             ia->ia_broadaddr = ifra->ifra_broadaddr;
297         return (error);

```

图6-26 函数in_control：SIOCAIFADDR 处理

代码在默认方式下取一个新的 IP 地址指派给接口 (hostIsNew 以 1 开始)。如果新地址的长度为 0, in_control 将当前地址复制到请求中, 并将 hostIsNew 修改为 0。如果长度不是 0, 并且新地址与老地址匹配, 则这个请求不包含一个新地址, 并且 hostIsNew 被设置为 0。

278-284 如果在这个请求中指定一个网络掩码, 则任何使用此当前地址的路由被忽略, 并且 in_control 安装此新掩码。

285-290 如果接口是一个点对点接口, 并且此请求包括一个新目的地址, 则 in_scrub 忽略任何使用此地址的路由, 新目的地址被安装, 并且 maskIsNew 被设置为 1, 以强制调用 in_ifinit 来重配置接口。

291-297 如果配置了一个新地址或指派了一个新掩码, 则 in_ifinit 作适当的修改来支持新的配置 (图 6-17)。注意, in_ifinit 的最后一个参数为 0。这表示已注意到这一点, 不必刷新所有路由。最后, 如果接口支持广播, 则从 in_aliasreq 结构复制广播地址。

6.6.11 删除 IP 地址：SIOCDEFADDR

命令 SIOCDEFADDR 从一个接口删除 IP 地址, 如图 6-27 所示。记住, ia 指向要被删除的 in_ifaddr 结构 (即, 与请求匹配的)。

```

298     case SIOCIFADDR:
299         in_ifscrub(ifp, ia);
300         if ((ifa = ifp->if_addrlist) == (struct ifaddr *) ia)
301             /* ia is the first address in the list */
302             ifp->if_addrlist = ifa->ifa_next;
303         else {
304             /* ia is *not* the first address in the list */
305             while (ifa->ifa_next &&
306                 (ifa->ifa_next != (struct ifaddr *) ia))
307                 ifa = ifa->ifa_next;
308             if (ifa->ifa_next)
309                 ifa->ifa_next = ((struct ifaddr *) ia)->ifa_next;
310             else
311                 printf("Couldn't unlink inifaddr from ifp\n");
312         }
313         oia = ia;
314         if (oia == (ia = in_ifaddr))
315             in_ifaddr = ia->ia_next;
316         else {
317             while (ia->ia_next && (ia->ia_next != oia))
318                 ia = ia->ia_next;
319             if (ia->ia_next)
320                 ia->ia_next = oia->ia_next;
321             else
322                 printf("Didn't unlink inifadr from list\n");
323         }
324         IFAFREE(&oia->ia_ifa);
325         break;

```

图6-27 in_control 函数：删除地址

298-323 前提条件处理代码将ia指向要删除的地址。in_ifscrub删除任何与此地址关联的路由。第一个 if 删除接口地址列表的结构。第二个 if 删除来自 Internet 地址列表(in_ifaddr)的结构。

324-325 IFAFREE只在引用计数降到0时才释放此结构。

其他引用可能来自路由表中的各项。

6.7 接口ioctl处理

我们现在查看当一个地址被分配给接口时的专用 ioctl 处理，对于我们的每个例子接口，这个处理分别在函数leioctl、slioclt和loioctl中。

in_ifinit通过图6-16中的SIOCSIFADDR代码和图6-26中的SIOCAIFADDR代码调用。in_ifinit总是通过接口的if_ioctl函数发送SIOCSIFADDR命令(图6-17)。

6.7.1 leioctl函数

图4-31显示了 LANCE 驱动程序的 SIOCSIFFLAGS 命令的处理。图 6-28 显示了 SIOCSIFADDR 命令的处理。

614-637 在处理命令前，data 转换成是一个 ifaddr 结构指针，并且 ifp->if_unit 为此请求选择相应的 le_softc 结构。

leinit 将接口标志为启动并初始化硬件。对于 Internet 地址，IP 地址保存在 arpcom 结构

中，并且为此地址发送一个免费 ARP。免费 ARP 在 21.5 节和卷 1 的 4.7 节中讨论。

未识别命令

627-677 对于未识别命令，返回 EINVAL。

```

614 leioctl(ifp, cmd, data)                                     if_le.c
615 struct ifnet *ifp;
616 int      cmd;
617 caddr_t data;
618 {
619     struct ifaddr *ifa = (struct ifaddr *) data;
620     struct le_softc *le = &le_softc[ifp->if_unit];
621     struct lereg1 *ler1 = le->sc_r1;
622     int      s = splimp(), error = 0;
623
624     switch (cmd) {
625     case SIOCSIFADDR:
626         ifp->if_flags |= IFF_UP;
627         switch (ifa->ifa_addr->sa_family) {
628         case AF_INET:
629             leinit(ifp->if_unit); /* before arpwhoas */
630             ((struct arpcom *) ifp)->ac_ipaddr =
631                 IA_SIN(ifa)->sin_addr;
632             arpwhoas((struct arpcom *) ifp, &IA_SIN(ifa)->sin_addr);
633             break;
634         default:
635             leinit(ifp->if_unit);
636             break;
637         }
638     }
639     break;
640
641     /* SIOCSIFFLAGS command (Figure 4.31) */
642     /* SIOCADDMULTI and SIOCDELMULTI commands (Figure 12.31) */
643
644     default:
645         error = EINVAL;
646     }
647     splx(s);
648     return (error);
649 }

```

图6-28 函数leioctl

6.7.2 slioclt函数

函数slioclt(图6-29)为SLIP设备驱动器处理命令 SIOCSIFADDR 和 SIOCSIFDSTADDR。

```

653 int                                     if_sl.c
654 slioclt(ifp, cmd, data)
655 struct ifnet *ifp;
656 int      cmd;
657 caddr_t data;
658 {
659     struct ifaddr *ifa = (struct ifaddr *) data;
660     struct ifreq *ifr;
661     int      s = splimp(), error = 0;
662
663     switch (cmd) {

```

图6-29 函数slioclt : 命令 SIOCSIFADDR 和 SIOCSIFDSTADDR

```

663     case SIOCSIFADDR:
664         if (ifa->ifa_addr->sa_family == AF_INET)
665             ifp->if_flags |= IFF_UP;
666         else
667             error = EAFNOSUPPORT;
668         break;

669     case SIOCSIFDSTADDR:
670         if (ifa->ifa_addr->sa_family != AF_INET)
671             error = EAFNOSUPPORT;
672         break;

```

```

/* SIOCADMULTI and SIOCDELMULTI commands (Figure 12.29)*/

```

```

688     default:
689         error = EINVAL;
690     }
691     splx(s);
692     return (error);
693 }

```

if_sl.c

图6-29 (续)

663-672 对于这两个命令，如果地址不是一个 IP 地址，则返回 EAFNOSUPPORT。SIOCSIFADDR 命令设置 IFF_UP。

未识别命令

688-693 对于未识别命令，返回 EINVAL。

6.7.3 loioctl 函数

函数 loioctl 和它的 SIOCSIFADDR 命令的实现显示在图 6-30 中。

```

135 int
136 loioctl(ifp, cmd, data)
137 struct ifnet *ifp;
138 int cmd;
139 caddr_t data;
140 {
141     struct ifaddr *ifa;
142     struct ifreq *ifr;
143     int error = 0;

144     switch (cmd) {
145     case SIOCSIFADDR:
146         ifp->if_flags |= IFF_UP;
147         ifa = (struct ifaddr *) data;
148         /*
149          * Everything else is done at a higher level.
150          */
151         break;

```

if_loop.c

```

/* SIOCADMULTI and SIOCDELMULTI commands (Figure 12.30) */

```

图6-30 函数 loioctl : 命令 SIOCSIFADDR

```
167     default:
168         error = EINVAL;
169     }
170     return (error);
171 }
```

if_loop.c

图6-30 (续)

135-151 对于Internet地址，`loioctl`设置IFF_UP，并立即返回。

未识别命令

167-171 对于未识别命令，返回EINVAL。

注意，对于所有这三个例子驱动程序，指派一个地址会导致接口被标记为启用 (IFF_UP)。

6.8 Internet实用函数

图6-31列出了几个操作Internet地址或依赖于图6-5中ifnet结构的函数，它们通常用于发现不能单从32 bit IP地址中获得的子网信息。这些函数的实现主要包括数据结构的转换和操作位掩码。读者在netinet/in.c中可以找到这些函数。

函 数	说 明
in_netof	返回in中的网络和子网部分。主机比特被设置为0。对于D类地址，返回D类首标比特和用于多播组的0比特 <code>u_long in_netof(struct in_addr in);</code>
in_canforward	如果地址为in的IP分组有资格转发，则返回真。D类和E类地址、环回网络地址和有一个为0网络号的地址不能转发 <code>int in_canforward(struct in_addr in);</code>
in_localaddr	如果主机in被定位在一个直接连接的网路，则返回真。如果全局变量ubnetsarelocal非0，则所有直接连接的网路的子网也被认为是本地的 <code>int in_localaddr(struct in_addr in);</code>
in_broadcast	如果in是一个由ifp指向的接口所关联的广播地址，则返回真 <code>int in_broadcast(struct in_addr in, struct ifnet ifp);</code>

图6-31 Internet地址函数

Net/2在in_canforward中有一个错误：它允许转发环回地址。因为大多数Net/2系统被配置为只承认一个环回地址，如127.0.0.1，Net/2系统常沿着默认路由在环回网络中转发其他地址(例如127.0.0.2)。

一个到127.0.0.2的telnet可能不是你所希望的！（习题6.6）

6.9 ifnet实用函数

几个查找数据结构的函数显示在图6-5中。列于图6-32的函数接受任何协议族类的地址，因为它们的参数是指向一个sockaddr结构的指针，这个结构中包含有地址族类。与图6-31中的函数比较，在那里的每个函数将32 bit的IP地址作为一个参数。这些函数定义在文件net/if.c中。

函 数	说 明
<code>ifa_ifwithaddr</code>	在ifnet列表中查找有一个单播或广播地址 <i>addr</i> 的接口。返回一个指向这个匹配的ifaddr结构的指针；或者若没有找到，则返回一个空指针 <code>struct ifaddr ifa_ifwithaddr(struct sockaddr *addr);</code>
<code>ifa_ifwithdstaddr</code>	在ifnet列表中查找目的地址为 <i>addr</i> 的接口。返回一个指向这个匹配的ifaddr结构的指针；或者若没有找到，则返回一个空指针 <code>struct ifaddr *ifa_ifwithdstaddr(struct sockaddr *addr);</code>
<code>ifa_ifwithnet</code>	在ifnet列表中查找与 <i>addr</i> 同一网络的地址。返回匹配的 ifaddr 结构的指针；或者若没有找到，则返回一个空指针 <code>struct ifaddr ifa_ifwithnet(struct sockaddr *addr);</code>
<code>ifa_ifwithaf</code>	在ifnet列表中查找与 <i>addr</i> 具有相同地址族类的第一个地址。返回匹配的 ifaddr 结构的指针；或者若没有找到，则返回一个空指针 <code>struct ifaddr ifa_ifwithaf(struct sockaddr *addr);</code>
<code>ifaof_ifpforaddr</code>	在ifp列表中查找与 <i>addr</i> 匹配的地址。用于精确匹配的引用次序为：一个点对点链路上的目的地址、一个同一网络上的地址和一个在同一地址族类的地址，则返回匹配的 ifaddr 结构的指针；或者若没有找到，则返回一个空指针 <code>struct ifaddr * ifaof_ifpforaddr(struct sockaddr *addr, struct ifnet *ifp);</code>
<code>ifa_ifwithroute</code>	返回目的地址 (dst) 和网关地址 (gateway) 指定的相应的本地接口的 ifaddr 结构的指针 <code>struct ifaddr * ifa_ifwithroute(int flags, struct sockaddr *dst, struct sockaddr *gateway);</code>
<code>ifunit</code>	返回与 <i>name</i> 关联的 ifnet 结构的指针 <code>struct ifnet ifunit(char *name);</code>

图6-32 ifnet 实用函数

6.10 小结

在本章中，我们概述了 IP 编址机制，并且说明了 IP 专用的接口地址结构和协议地址结构：结构 `in_ifaddr` 和 `sockaddr_in`。

我们讨论了如何通过程序 `ifconfig` 和 `ioctl` 接口命令来配置接口的 IP 专用信息。

最后，我们总结了几个操作 IP 地址和查找接口数据结构的实用函数。

习题

- 你认为为什么在结构 `sockaddr_in` 中的 `sin_addr` 最初定义为一个结构？
- `ifunit("sl0")` 返回的指针指向图 6-5 中的哪个结构？
- 当 IP 地址已经包含在接口的地址列表中的一个 `ifaddr` 结构中时，为什么还要在 `ac_ipaddr` 中备份？
- 你认为为什么 IP 接口地址要通过一个 UDP 插口而不是一个原始的 IP 插口来访问？
- 为什么 `in_socktrim` 要修改 `sin_len` 来匹配掩码的长度，而不使用一个 `sockaddr_in` 结构的标准长度？
- 当一个 `telnet 127.0.0.1` 命令中的连接请求部分被一个 Net/2 系统错误地转发，并且最后被认可，同时还被默认路由上的一个系统所接收时，会发生什么情况？