ioctl Function

- ioctl function
- get_ifi_info function
- ARP cache operations: e.g. Print H/W addresses

ioctl Function

#include <unistd.h>

int ioctl (int fd, int request, … /* void *arg */);

returns: 0 if OK, -1 on error

type of *arg depends on request

Category	request	Description
socket	SIOCATMARK	at out-of-band mark ?
	SIOCSPGRP	set process ID or process group ID of socket
	SIOCGPGRP	get process ID or process group ID of socket
file	FIONBIO	set/clear nonblocking flag
	FIOASYNC	set/cleat asynchronous i/o flag
	FIONREAD	get # bytes in receive buffer
	FIOSETOWN	set process ID or process group ID of file
	FIOGETOWN	get process ID or process group ID of file
ARP	SIOCSARP	create/modify ARP entry
	SIOCGARP	get ARP entry
	SIOCDARP	delete ARP entry

ioctl Function (Cont.)

Category	request	Description
interface	SIOCGIFCONF	get list of all interfaces
	SIOCSIFADDR	set interface address
	SIOCGIFADDR	get interface address
	SIOCSIFFLAGS	set interface flags
	SIOCGIFFLAGS	get interface flags
	SIOCSIFDSTADDR	set point-to-point address
	SIOCGIFDSTADDR	get point-to-point address
	SIOCGIFBRDADDR	get broadcast address
	SIOCSIFBRDADDR	set broadcast address
	SIOCGIFNETMASK	get subnet mask
	SIOCSIFNETMASK	set subnet mask
	SIOCGIFMETRIC	get interface metric
	SIOCSIFMETRIC	set interface metric
	SIOC…	(many more; implementation dependent)
routing	SIOCADDRT	add route
	SIOCDELRT	delete route

get_ifi_info Function
returns a linked list of ifi_info structures

- Create a TCP or UDP socket
- Issue SIOCGIFCONF request in ioctl in a loop
- Initialize linked list pointers
- Step to next socket address structure
- Handle aliases
- Fetch interface flags
- Allocate and initialize ifi_info structure

Figure 17.2 `ifconf` and `ifreq` structures used with various interface `ioctl` requests.

<net/if.h>

```
struct ifconf {
    int  ifc_len;                /* size of buffer, value-result */
    union {
        caddr_t ifcu_buf;        /* input from user -> kernel */
        struct ifreq *ifcu_req;  /* return from kernel -> user */
    } ifc_ifcu;
};
#define ifc_buf ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req /* array of structures returned */

#define IFNAMSIZ     16

struct ifreq {
    char    ifr_name[IFNAMSIZ];  /* interface name, e.g., "le0" */
    union {
        struct  sockaddr ifru_addr;
        struct  sockaddr ifru_dstaddr;
        struct  sockaddr ifru_broadaddr;
        short   ifru_flags;
        int     ifru_metric;
        caddr_t ifru_data;
    } ifr_ifru;
};
#define ifr_addr         ifr_ifru.ifru_addr         /* address */
#define ifr_dstaddr      ifr_ifru.ifru_dstaddr      /* other end of point-to-poir
#define ifr_broadaddr    ifr_ifru.ifru_broadaddr    /* broadcast address */
#define ifr_flags        ifr_ifru.ifru_flags        /* flags */
#define ifr_metric       ifr_ifru.ifru_metric       /* metric */
#define ifr_data         ifr_ifru.ifru_data         /* for use by interface */
```

Figure 17.3. Initialization of `ifconf` structure before `SIOCGIFCONF`.

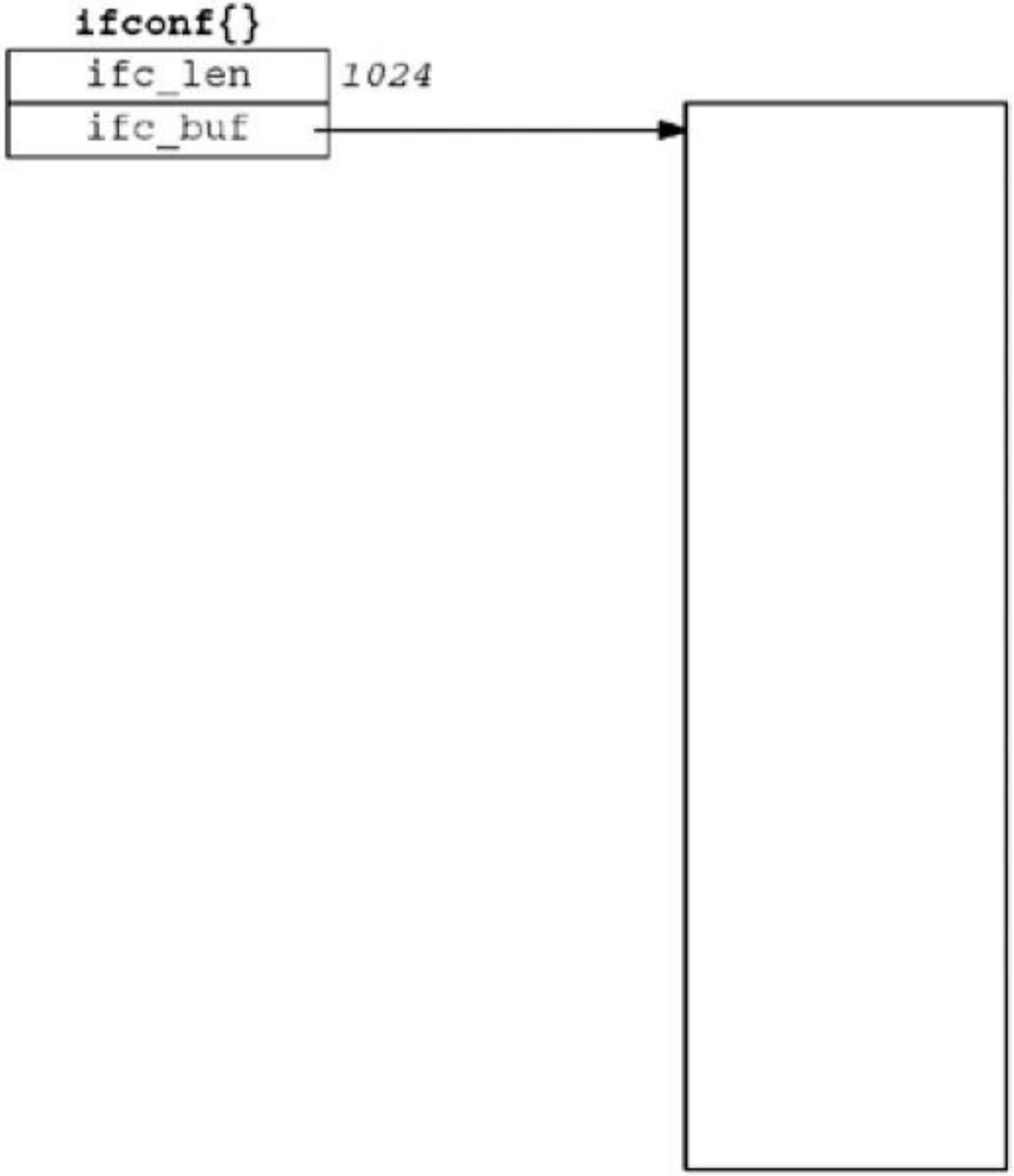
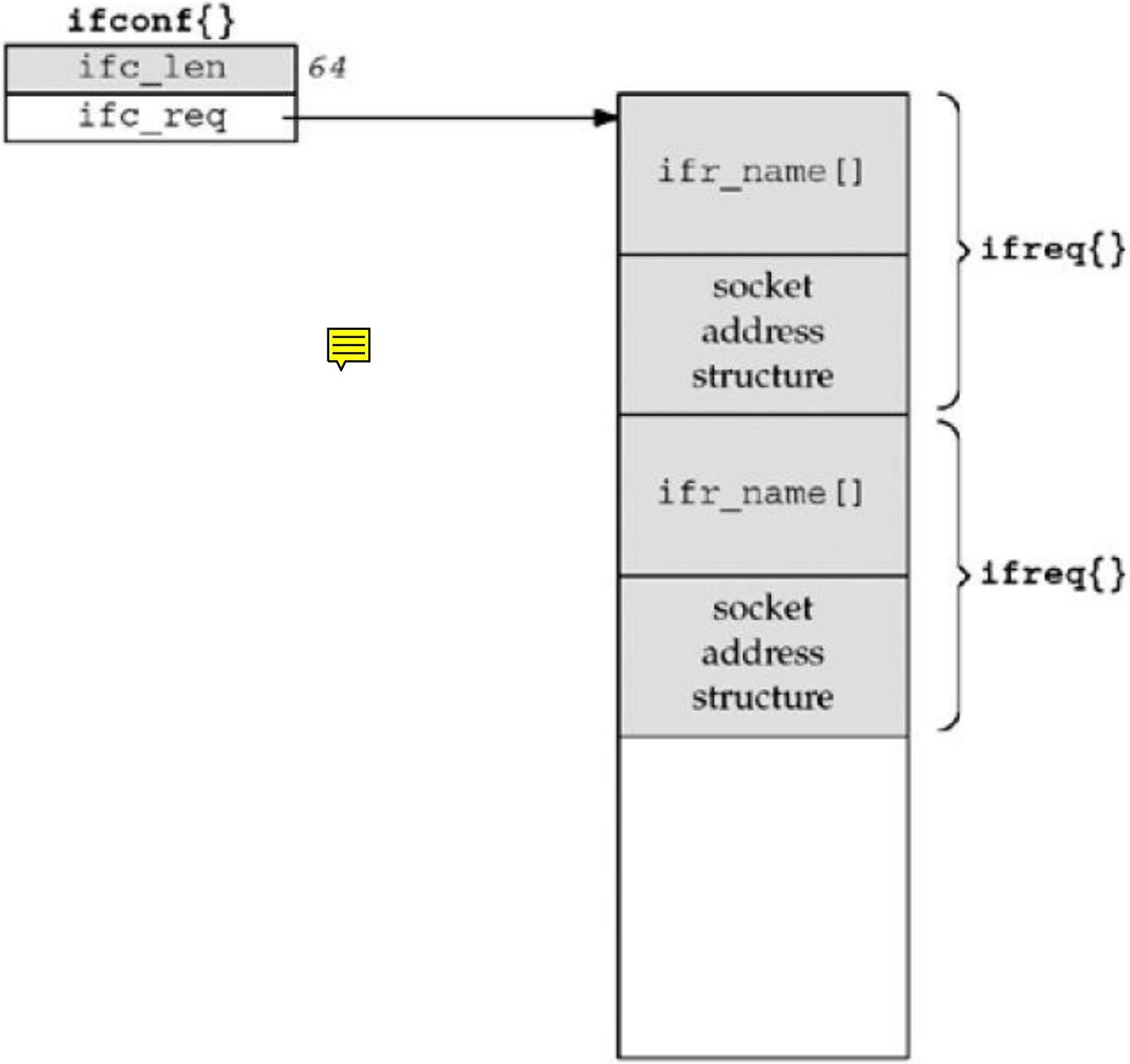


Figure 17.4. Values returned by SIOCGIFCONF.



```
struct sockaddr_dl {
  uint8_t       sdl_len;
  sa_family_t   sdl_family;    /* AF_LINK */
  uint16_t      sdl_index;     /* system assigned index, if > 0 */
  uint8_t       sdl_type;      /* IFT_ETHER, etc. from <net/if_types.h> */
  uint8_t       sdl_nlen;      /* name length, starting in sdl_data[0] */
  uint8_t       sdl_alen;      /* link-layer address length */
  uint8_t       sdl_slen;      /* link-layer selector length */
  char          sdl_data[12]; /* minimum work area, can be larger;
                                 contains i/f name and link-layer address */
};
```

Figure 18.1 Datalink socket address structure.

output:

```
macosx % prifinfo inet4 0
lo0: <UP MCAST LOOP >
  MTU: 16384
  IP addr: 127.0.0.1
en1: <UP BCAST MCAST >
  MTU: 1500
  IP addr: 172.24.37.78
  broadcast addr: 172.24.37.95
```

The first command-line argument of `inet4` specifies IPv4 adc argument of 0 specifies that no address aliases are to be return address aliases in Section A.4). Note that under MacOS X, the h Ethernet interface is not available using this method.

If we add three alias addresses to the Ethernet interface (er 80, and 81, and if we change the second command-line argumer lowing:

```
macosx % prifinfo inet4 1
lo0: <UP MCAST LOOP >
  MTU: 16384
  IP addr: 127.0.0.1
en1: <UP BCAST MCAST >
  MTU: 1500
  IP addr: 172.24.37.78                    primary IP address
  broadcast addr: 172.24.37.95
en1: <UP BCAST MCAST >
  MTU: 1500
  IP addr: 172.24.37.79                    first alias
  broadcast addr: 172.24.37.95
en1: <UP BCAST MCAST >
  MTU: 1500
  IP addr: 172.24.37.80                    second alias
  broadcast addr: 172.24.37.95
en1: <UP BCAST MCAST >
  MTU: 1500
  IP addr: 172.24.37.81                    third alias
```

Figure 17.5 `unpifi.h` header.

lib/unpifi.h

```
 1 /* Our own header for the programs that need interface configuration info.
 2    Include this file, instead of "unp.h". */

 3 #ifndef __unp_ifi_h
 4 #define __unp_ifi_h

 5 #include     "unp.h"
 6 #include     <net/if.h>

 7 #define IFI_NAME     16            /* same as IFNAMSIZ in <net/if.h> */
 8 #define IFI_HADDR     8            /* allow for 64-bit EUI-64 in future */

 9 struct  ifi_info {
10     char     ifi_name[IFI_NAME];   /* interface name, null-terminated */
11     short    ifi_index;            /* interface index */
12     short    ifi_mtu;              /* interface MTU */
13     u_char   ifi_haddr[IFI_HADDR];     /* hardware address */
14     u_short  ifi_hlen;             /* # bytes in hardware address: 0, 6, 8 */
15     short    ifi_flags;            /* IFF_xxx constants from <net/if.h> */
16     short    ifi_myflags;          /* our own IFI_xxx flags */
17     struct  sockaddr *ifi_addr;    /* primary address */
18     struct  sockaddr *ifi_brdaddr;     /* broadcast address */
19     struct  sockaddr *ifi_dstaddr;     /* destination address */
20     struct  ifi_info *ifi_next;    /* next of these structures */
21 };

22 #define IFI_ALIAS    1             /* ifi_addr is an alias */

23                      /* function prototypes */
24 struct ifi_info *get_ifi_info(int, int);
25 struct ifi_info *Get_ifi_info(int, int);
26 void    free_ifi_info(struct ifi_info *);

27 #endif  /* __unp_ifi_h */
```

)

Figure 17.6 `prifinfo` program that calls our `get_ifi_info` function.

ioctl/prifinfo.c

```
 1 #include    "unpifi.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     struct ifi_info *ifi, *ifihead;
 6     struct sockaddr *sa;
 7     u_char *ptr;
 8     int     i, family, doaliases;
 9     if (argc != 3)
10         err_quit("usage: prifinfo <inet4|inet6> <doaliases>");
11     if (strcmp(argv[1], "inet4") == 0)
12         family = AF_INET;
13     else if (strcmp (argv[1], "inet6") == 0)
14         family = AF_INET6;
15     else
16         err_quit("invalid <address-family>");
17     doaliases = atoi(argv[2]);
18     for (ifihead = ifi = Get_ifi_info(family, doaliases);
19          ifi != NULL; ifi = ifi->ifi_next) {
20         printf("%s: ", ifi->ifi_name);
21         if (ifi->ifi_index != 0)
22             printf("(%d) ", ifi->ifi_index);
23         printf("<");
24         if (ifi->ifi_flags & IFF_UP)            printf("UP ");
25         if (ifi->ifi_flags & IFF_BROADCAST)     printf("BCAST ");
26         if (ifi->ifi_flags & IFF_MULTICAST)     printf("MCAST ");
```

```
17      doaliases = atoi(argv[2]);
18      for (ifihead = ifi = Get_ifi_info(family, doaliases);
19           ifi != NULL; ifi = ifi->ifi_next) {
20          printf("%s: ", ifi->ifi_name);
21          if (ifi->ifi_index != 0)
22              printf("(%d) ", ifi->ifi_index);
23          printf("<");
24          if (ifi->ifi_flags & IFF_UP)            printf("UP ");
25          if (ifi->ifi_flags & IFF_BROADCAST)     printf("BCAST ");
26          if (ifi->ifi_flags & IFF_MULTICAST)     printf("MCAST ");
27          if (ifi->ifi_flags & IFF_LOOPBACK)      printf("LOOP ");
28          if (ifi->ifi_flags & IFF_POINTOPOINT)   printf("P2P ");
29          printf(">\n");
30          if ( (i = ifi->ifi_hlen) > 0) {
31              ptr = ifi->ifi_haddr;
32              do {
33                  printf("%s%x", (i == ifi->ifi_hlen) ? "  " : ":", *ptr++);
34              } while (--i > 0);
35              printf("\n");
36          }
37          if (ifi->ifi_mtu != 0)
38              printf("  MTU: %d\n", ifi->ifi_mtu);
39          if ( (sa = ifi->ifi_addr) != NULL)
40              printf("  IP addr: %s\n", Sock_ntop_host (sa, sizeof (*sa)));
41          if ( (sa = ifi->ifi_brdaddr) != NULL)
42              printf("  broadcast addr: %s\n",
43                     Sock_ntop_host (sa, sizeof(*sa)));
44          if ( (sa = ifi->ifi_dstaddr) != NULL)
45              printf("  destination addr: %s\n",
46                     Sock_ntop_host(sa, sizeof(*sa)));
47      }
48      free_ifi_info(ifihead);
49      exit(0);
```

Figure 17.7 Issue SIOCGIFCONF request to obtain interface configuration.

lib/get_ifi_info.c

```
 1 #include     "unpifi.h"
 2 struct ifi_info *
 3 get_ifi_info(int family, int doaliases)
 4 {
 5     struct ifi_info *ifi, *ifihead, **ifipnext;
 6     int     sockfd, len, lastlen, flags, myflags, idx = 0, hlen = 0;
 7     char    *ptr, *buf, lastname[IFNAMSIZ], *cptr, *haddr, *sdlname;
 8     struct ifconf ifc;
 9     struct ifreq *ifr, ifrcopy;
10     struct sockaddr_in *sinptr;
11     struct sockaddr_in6 *sin6ptr;

12     sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

13     lastlen = 0;
14     len = 100 * sizeof(struct ifreq);     /* initial buffer size guess */
15     for ( ; ; ) {
16         buf = Malloc(len);
17         ifc.ifc_len = len;
18         ifc.ifc_buf = buf;
19         if (ioctl(sockfd, SIOCGIFCONF, &ifc) < 0) {
20             if (errno != EINVAL || lastlen != 0)
21                 err_sys("ioctl error");
22         } else {
23             if (ifc.ifc_len == lastlen)
24                 break;          /* success, len has not changed */
25             lastlen = ifc.ifc_len;
26         }
27         len += 10 * sizeof(struct ifreq);     /* increment */
28         free(buf);
29     }
30     ifihead = NULL;
31     ifipnext = &ifihead;
32     lastname[0] = 0;
```

Figure 17.8 Process interface configuration.

lib/get_ifi_info.c

```
34      for (ptr = buf; ptr < buf + ifc.ifc_len;) {
35          ifr = (struct ifreq *) ptr;

36 #ifdef   HAVE_SOCKADDR_SA_LEN
37          len = max(sizeof(struct sockaddr), ifr->ifr_addr.sa_len);
38 #else
39          switch (ifr->ifr_addr.sa_family) {
40 #ifdef   IPV6
41          case AF_INET6:
42              len = sizeof(struct sockaddr_in6);
43              break;
44 #endif
45          case AF_INET:
46          default:
47              len = sizeof(struct sockaddr);
48              break;
49          }
50 #endif   /* HAVE_SOCKADDR_SA_LEN */
51          ptr += sizeof(ifr->ifr_name) + len; /* for next one in buffer */

52 #ifdef   HAVE_SOCKADDR_DL_STRUCT
53          /* assumes that AF_LINK precedes AF_INET or AF_INET6 */
54          if (ifr->ifr_addr.sa_family == AF_LINK) {
55              struct sockaddr_dl *sdl = (struct sockaddr_dl *) &ifr->ifr_addr;
56              sdlname = ifr->ifr_name;
57              idx = sdl->sdl_index;
58              haddr = sdl->sdl_data + sdl->sdl_nlen;
59              hlen = sdl->sdl_alen;
60          }
61 #endif

62          if (ifr->ifr_addr.sa_family != family)
63              continue;               /* ignore if not desired address family */
```

```
47              len = sizeof(struct sockaddr);
48              break;
49          }
50  #endif  /* HAVE_SOCKADDR_SA_LEN */
51          ptr += sizeof(ifr->ifr_name) + len; /* for next one in buffer */

52  #ifdef  HAVE_SOCKADDR_DL_STRUCT
53          /* assumes that AF_LINK precedes AF_INET or AF_INET6 */
54          if (ifr->ifr_addr.sa_family == AF_LINK) {
55              struct sockaddr_dl *sdl = (struct sockaddr_dl *) &ifr->ifr_addr;
56              sdlname = ifr->ifr_name;
57              idx = sdl->sdl_index;
58              haddr = sdl->sdl_data + sdl->sdl_nlen;
59              hlen = sdl->sdl_alen;
60          }
61  #endif

62          if (ifr->ifr_addr.sa_family != family)
63              continue;               /* ignore if not desired address family */

64          myflags = 0;
65          if ( (cptr = strchr(ifr->ifr_name, ':')) != NULL)
66              *cptr = 0;          /* replace colon with null */
67          if (strncmp(lastname, ifr->ifr_name, IFNAMSIZ) == 0) {
68              if (doaliases == 0)
69                  continue;       /* already processed this interface */
70              myflags = IFI_ALIAS;
71          }
72          memcpy(lastname, ifr->ifr_name, IFNAMSIZ);

73          ifrcopy = *ifr;
74          Ioctl(sockfd, SIOCGIFFLAGS, &ifrcopy);
75          flags = ifrcopy.ifr_flags;
76          if ((flags & IFF_UP) == 0)
77              continue;               /* ignore if interface not up */
```

Figure 17.9 Allocate and initialize `ifi_info` structure.

lib/get_ifi_info.c

```
78          ifi = Calloc(1, sizeof(struct ifi_info));
79          *ifipnext = ifi;          /* prev points to this new one */
80          ifipnext = &ifi->ifi_next;  /* pointer to next one goes here */

81          ifi->ifi_flags = flags; /* IFF_xxx values */
82          ifi->ifi_myflags = myflags; /* IFI_xxx values */
83  #if defined(SIOCGIFMTU) && defined(HAVE_STRUCT_IFREQ_IFR_MTU)
84          Ioctl(sockfd, SIOCGIFMTU, &ifrcopy);
85          ifi->ifi_mtu = ifrcopy.ifr_mtu;
86  #else
87          ifi->ifi_mtu = 0;
88  #endif
89          memcpy(ifi->ifi_name, ifr->ifr_name, IFI_NAME);
90          ifi->ifi_name [IFI_NAME - 1] = '\0';
91          /* If the sockaddr_dl is from a different interface, ignore it */
92          if (sdlname == NULL || strcmp (sdlname, ifr->ifr_name) != 0)
93              idx = hlen = 0;
94          ifi->ifi_index = idx;
95          ifi->ifi_hlen = hlen;
96          if (ifi->ifi_hlen > IFI_HADDR)
97              ifi->ifi_hlen = IFI_HADDR;
98          if (hlen)
99              memcpy(ifi->ifi_haddr, haddr, ifi->ifi_hlen);
```

Figure 17.10 Fetch and return interface addresses.

lib/get_ifi_info.c

```
100         switch (ifr->ifr_addr.sa_family) {
101         case AF_INET:
102             sinptr = (struct sockaddr_in *) &ifr->ifr_addr;
103             ifi->ifi_addr = Calloc(1, sizeof(struct sockaddr_in));
104             memcpy(ifi->ifi_addr, sinptr, sizeof(struct sockaddr_in));

105 #ifdef SIOCGIFBRDADDR
106             if (flags & IFF_BROADCAST) {
107                 Ioctl(sockfd, SIOCGIFBRDADDR, &ifrcopy);
108                 sinptr = (struct sockaddr_in *) &ifrcopy.ifr_broadaddr;
109                 ifi->ifi_brdaddr = Calloc(1, sizeof(struct sockaddr_in));
110                 memcpy(ifi->ifi_brdaddr, sinptr, sizeof(struct sockaddr_in));
111             }
112 #endif

113 #ifdef SIOCGIFDSTADDR
114             if (flags & IFF_POINTOPOINT) {
115                 Ioctl(sockfd, SIOCGIFDSTADDR, &ifrcopy);
116                 sinptr = (struct sockaddr_in *) &ifrcopy.ifr_dstaddr;
117                 ifi->ifi_dstaddr = Calloc(1, sizeof(struct sockaddr_in));
118                 memcpy(ifi->ifi_dstaddr, sinptr, sizeof(struct sockaddr_in));
119             }
120 #endif
121             break;

122         case AF_INET6:
123             sin6ptr = (struct sockaddr_in6 *) &ifr->ifr_addr;
124             ifi->ifi_addr = Calloc(1, sizeof(struct sockaddr_in6));
125             memcpy(ifi->ifi_addr, sin6ptr, sizeof(struct sockaddr_in6));
```

Figure 17.11 `free_ifi_info` function: frees dynamic memory allocated by `get_ifi_info`.

lib/get_ifi_info.c

```
143 void
144 free_ifi_info(struct ifi_info *ifihead)
145 {
146     struct ifi_info *ifi, *ifinext;

147     for (ifi = ifihead; ifi != NULL; ifi = ifinext) {
148         if (ifi->ifi_addr != NULL)
149             free(ifi->ifi_addr);
150         if (ifi->ifi_brdaddr != NULL)
151             free(ifi->ifi_brdaddr);
152         if (ifi->ifi_dstaddr != NULL)
153             free(ifi->ifi_dstaddr);
154         ifinext = ifi->ifi_next;     /* can't fetch ifi_next after free() */
155         free(ifi);                   /* the ifi_info{} itself */
156     }
157 }
```

ARP Cache Operations
e.g. print hardware addresses of host

- Get list of addresses and loop through each
- Print IP address
- Issue ioctl and print hardware address

```
#include <net/if_arp.h>
struct arpreq {
   struct sockaddr   arp_pa;     /* protocol address */
   struct sockaddr   arp_ha;     /* hardware address */
   int      arp_flags;   /* flags */
};
#define ATF_INUSE   0x01   /* entry in use */
#define ATF_COM   0x02   /* completed entry (hardware addr valid */
#define ATF_PERM   0x04   /* permanent entry */
#define ATF_PUBL   0x08   /* published entry (respond for other host */
```

Figure 17.12 `arpreq` structure used with `ioctl` requests for ARP cache.

<net/if_arp.h>

```
struct arpreq {
    struct sockaddr arp_pa;     /* protocol address */
    struct sockaddr arp_ha;     /* hardware address */
    int             arp_flags;  /* flags */
};

#define ATF_INUSE     0x01 /* entry in use */
#define ATF_COM       0x02 /* completed entry (hardware addr valid) */
#define ATF_PERM      0x04 /* permanent entry */
#define ATF_PUBL      0x08 /* published entry (respond for other host) */
```

Figure 17.13 Print a host's hardware addresses.

ioctl/prmac.c

```
 1 #include     "unpifi.h"
 2 #include     <net/if_arp.h>

 3 int
 4 main(int argc, char **argv)
 5 {
 6     int     sockfd;
 7     struct ifi_info *ifi;
 8     unsigned char *ptr;
 9     struct arpreq arpreq;
10     struct sockaddr_in *sin;

11     sockfd = Socket(AF_INET, SOCK_DGRAM, 0);
12     for (ifi = get_ifi_info(AF_INET, 0); ifi != NULL; ifi = ifi->ifi_next) {
13         printf("%s: ", Sock_ntop(ifi->ifi_addr, sizeof(struct sockaddr_in)));

14         sin = (struct sockaddr_in *) &arpreq.arp_pa;
15         memcpy(sin, ifi->ifi_addr, sizeof(struct sockaddr_in));

16         if (ioctl(sockfd, SIOCGARP, &arpreq) < 0) {
17             err_ret("ioctl SIOCGARP");
18             continue;
19         }

20         ptr = &arpreq.arp_ha.sa_data[0];
21         printf("%x:%x:%x:%x:%x:%x\n", *ptr, *(ptr + 1),
22                *(ptr + 2), *(ptr + 3), *(ptr + 4), *(ptr + 5));
23     }
24     exit(0);
25 }
```

```
hpux % prmac
192.6.38.100: 0:60:b0:c2:68:9b
192.168.1.1: 0:60:b0:b2:28:2b
127.0.0.1: ioctl SIOCGARP: Invalid argument
```