3.接口层

ifnet结构

- 系统初始化期间，为每个网络设备分配一个独立的ifnet结构
- 每个ifnet有一个列表，包含这个设备的一个或多个协议地址

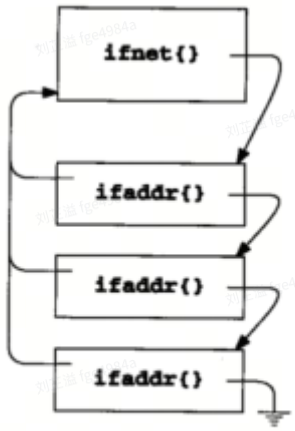

图3-5 每个ifnet 结构有一个ifaddr 结构的列表

- if_next把所有接口的ifnet结构链成一个链表；if_attach在系统初始化期间构造这个链表；if_addrlist指向这个接口的ifaddr结构列表

if.h

```
80 struct ifnet {
81     struct ifnet *if_next;      /* all struct ifnets are chained */
82     struct ifaddr *if_addrlist; /* linked list of addresses per if */
83     char    *if_name;           /* name, e.g. 'le' or 'lo' */
84     short    if_unit;           /* sub-unit for lower level driver */
85     u_short if_index;           /* numeric abbreviation for this if  */
86     short    if_flags;          /* Figure 3.7 */
87     short    if_timer;          /* time 'til if_watchdog called */
88     int      if_pcount;         /* number of promiscuous listeners */
89     caddr_t if_bpf;             /* packet filter structure */
```

if.h

图3-6 ifnet 结构：实现信息

- if_init：初始化接口
- if_output：对要传输的输出分组进行排队
- if_start：启动分组的传输
- if_done：传输完成后的清除
- if_ioctl：处理I/O控制命令
- if_reset：复位接口设备
- if_watchdog：周期性接口例程

```
114 /* procedure handles */
115     int     (*if_init)          /* init routine */
116             (int);
117     int     (*if_output)        /* output routine (enqueue) */
118             (struct ifnet *, struct mbuf *, struct sockaddr *,
119              struct rtentry *);
120     int     (*if_start)         /* initiate output routine */
121             (struct ifnet *);
122     int     (*if_done)          /* output complete routine */
123             (struct ifnet *);   /* (XXX not used; fake prototype) */
124     int     (*if_ioctl)         /* ioctl routine */
125             (struct ifnet *, int, caddr_t);
126     int     (*if_reset)
127             (int);              /* new autoconfig will permit removal */
128     int     (*if_watchdog)      /* timer routine */
129             (int);
```
if.h

图3-11 结构ifnet：接口过程

- ifq_head：指向队列第一个分组
- ifq_tail：指向队列最后一个分组
- if_len：队列中分组的数目
- ifq_maxlen：队列中允许缓存的最大个数
- ifq_drops：统计因为队列满而丢弃的分组数

```
130     struct ifqueue {
131         struct mbuf *ifq_head;
132         struct mbuf *ifq_tail;
133         int     ifq_len;        /* current length of queue */
134         int     ifq_maxlen;     /* maximum length of queue */
135         int     ifq_drops;      /* packets dropped because of full queue */
136     } if_snd;                   /* output queue */
137 };
```
if.h

图3-13 结构ifnet：输出队列

函　数	说　明
IF_QFULL	*ifq*是否满 int **IF_QFULL**(struct ifqueue *ifq);
IF_DROP	IF_DROP仅将与*ifq*关联的ifq_drops加1。这个名字会引起误导：调用者丢弃这个分组 void **IF_DROP**(struct ifqueue *ifq);
IF_ENQUEUE	把分组*m*追加到*ifq*队列的后面。分组通过mbuf首部中的m_nextpkt链接在一起 void **IF_ENQUEUE**(struct ifqueue *ifq, struct mbuf *m);
IF_PREPEND	把分组*m*插入到*ifq*队列的前面 void **IF_PREPEND**(struct ifqueue *ifq, struct mbuf *m);
IF_DEQUEUE	从*ifq*队列中取走第一个分组。*m*指向取走的分组，若队列为空，则*m*为空值 void **IF_DEQUEUE**(struct ifqueue *ifq, struct mbuf *m);
if_qflush	丢弃队列*ifq*中的所有分组，例如，当一个接口被关闭了 void **if_qflush**(struct ifqueue *ifq);

图3-14 fiqueue 例程

ifaddr结构

- ifa_next：把分配给一个接口的所有地址链接起来
- ifa_ifp：指回接口的ifnet结构的指针

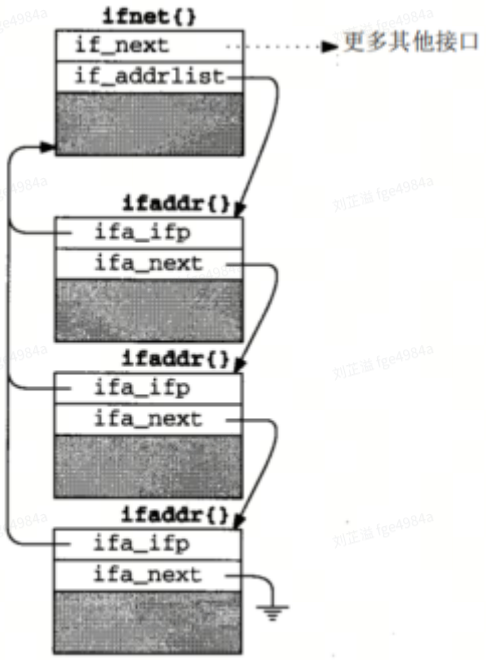

图3-16 结构ifnet 和ifaddr

- ifa_netmask指向一个位掩码，选择if_addr中的网络部分

if.h

```
217 struct ifaddr {
218         struct  ifaddr *ifa_next;       /* next address for interface */
219         struct  ifnet *ifa_ifp;         /* back-pointer to interface */
220         struct  sockaddr *ifa_addr;     /* address of interface */
221         struct  sockaddr *ifa_dstaddr;  /* other end of p-to-p link */
222 #define ifa_broadaddr   ifa_dstaddr     /* broadcast address interface */
223         struct  sockaddr *ifa_netmask;  /* used to determine subnet */
224         void    (*ifa_rtrequest)();     /* check or clean routes */
225         u_short ifa_flags;              /* mostly rt_flags for cloning */
226         short   ifa_refcnt;             /* references to this structure */
227         int     ifa_metric;             /* cost for this interface */
228 };
```

if.h

图3-15 结构ifaddr

sockaddr结构

socket.h

```
120 struct sockaddr {
121     u_char  sa_len;             /* total length */
122     u_char  sa_family;          /* address family (Figure 3.19) */
123     char    sa_data[14];        /* actually longer; address value */
124 };

271 struct osockaddr {
272     u_short sa_family;          /* address family (Figure 3.19) */
273     char    sa_data[14];        /* up to 14 bytes of direct address */
274 };
```

socket.h

图3-17 结构sockaddr 和osockaddr

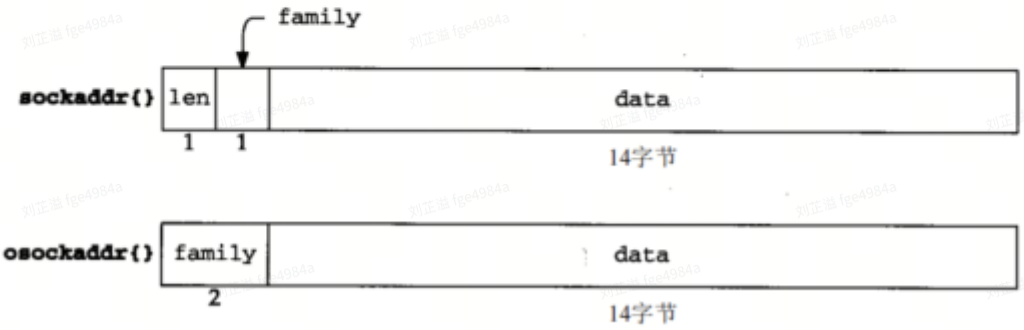

图3-18 结构sockaddr 和osockaddr（ 省略了前缀sa_）

- sa_family常量
  - AF_INET：Internet
  - AF_ISO, AF_OSI：OSI
  - AF_UNIX：Unix
  - AF_ROUTE：路由表
  - AF_LINK：链路层
  - AF_UNSPEC：以太网硬件

网络初始化

- cpu_startup查找并初始化所有连接到系统的硬件设备，包括任何网络接口
- 内核初始化硬件设备后，调用pdevinit数组中的每个pdev_attach函数
- ifinit和domaininit完成网络接口和协议的初始化，scheduler开始内核进程调度

init_main.c

```
70 main(framep)
71 void   *framep;
72 {
                    /* nonnetwork code */
96      cpu_startup();              /* locate and initialize devices */
                    /* nonnetwork code */
172     /* Attach pseudo-devices. (e.g., SLIP and loopback interfaces) */
173     for (pdev = pdevinit; pdev->pdev_attach != NULL; pdev++)
174         (*pdev->pdev_attach) (pdev->pdev_count);
175     /*
176      * Initialize protocols.  Block reception of incoming packets
177      * until everything is ready.
178      */
179     s = splimp();
180     ifinit();                   /* initialize network interfaces */
181     domaininit();               /* initialize protocol domains */
182     splx(s);
                    /* nonnetwork code */
231     /* The scheduler is an infinite loop. */
232     scheduler();
233     /* NOTREACHED */
234 }
```

init_main.c

图3-23 main 函数：网络初始化

以太网初始化

设　备	初始化函数
LANCE以太网	leattach
SLIP	slattach
环回	loopattach

图3-24 网络接口初始化函数