

第7章 域和协议

7.1 引言

在本章中，我们讨论支持同时操作多个网络协议的 Net/3数据结构。用Internet协议来说明在系统初始化时这些数据结构的构造和初始化。本章为我们讨论 IP协议处理层提供必要的背景资料，IP协议处理层在第8章讨论。

Net/3组把协议关联到一个域中，并且用一个协议族常量来标识每个域。Net/3还通过所使用的编址方法将协议分组。回忆图 3-19，地址族也有标识常量。当前，在一个域中的每个协议使用同类地址，并且每种地址只被一个域使用。作为结果，一个域能通过它的协议族或地址族常量唯一标识。图 7-1列出了我们讨论的协议和常量。

协 议 族	地 址 族	协 议
<i>PF_INET</i>	<i>AF_INET</i>	Internet
<i>PF_OSI, PF_ISO</i>	<i>AF_OSI, AF_ISO</i>	OSI
<i>PF_LOCAL, PF_UNIX</i>	<i>AF_LOCAL, AF_UNIX</i>	本地IPC(Unix)
<i>PF_ROUTE</i>	<i>AF_ROUTE</i>	路由表
<i>n/a</i>	<i>AF_LINK</i>	链路层(例如以太网)

图7-1 公共的协议和地址族常量

*PF_LOCAL*和*AF_LOCAL*是支持同一主机上进程间通信的协议的主要标识，它们是POSIX.12标准的一部分。在Net/3以前，用*PF_UNIX*和*AF_UNIX*标识这些协议。在Net/3中保留UNIX常量，用于向后兼容，并且要在本书中讨论。

*PF_UNIX*域支持在一个单独的 Unix主机上的进程间通信。细节见 [Stevens 1990]。*PF_ROUTE*域支持在一个进程和内核中路由软件间的通信(第18章)。我们偶尔引用*PF_OSI*协议，它作为 Net/3特性仅支持 OSI协议，但我们不讨论它们的细节。大多数讨论是关于*PF_INET*协议的。

7.2 代码介绍

本章涉及两个头文件和两个C文件。图7-2描述了这4个文件。

文 件	说 明
netinet/domain.h	domain结构定义
netinet/protosw.h	protosw结构定义
netinet/in_proto.c	IP domain和protosw结构
kern/uipc_domain.c	初始化和查找函数

图7-2 在本章中讨论的文件

7.2.1 全局变量

图7-3描述了几个重要的全局数据结构和系统参数，它们在本章中讨论，并经常在 Net/3中引用。

变 量	数 据 类 型	说 明
domain	struct domain *	链接的域列表
inetdomain	struct domain	Internet协议的domain结构
inetsw	struct protosw[]	Internet协议的protosw结构数组
max_linkhdr	int	见图7-17
max_protohdr	int	见图7-17
max_hdr	int	见图7-17
max_datalen	int	见图7-17

图7-3 在本章中介绍的全局变量

7.2.2 统计量

除了图7-4显示的由函数ip_init分配和初始化的统计量表，本章讨论的代码没有收集其他统计量。通过一个内核调试工具是查看这个表的唯一方法。

变 量	数据类型	说 明
ip_ifmatrix	int[][]	二维数组，用来统计在任意两个接口间传送的分组数

图7-4 在本章中收集的统计量

7.3 domain结构

一个协议域由一个图7-5所示的domain结构来表示。

```

42 struct domain {
43     int      dom_family;          /* AF_xxx */
44     char     *dom_name;
45     void     (*dom_init)          /* initialize domain data structures */
46         (void);
47     int      (*dom_externalize)   /* externalize access rights */
48         (struct mbuf *);
49     int      (*dom_dispose)       /* dispose of internalized rights */
50         (struct mbuf *);
51     struct protosw *dom_protosw, *dom_protoswnPROTOSW;
52     struct domain *dom_next;
53     int      (*dom_rtattach)      /* initialize routing table */
54         (void **, int);
55     int      dom_rtoffset;        /* an arg to rtattach, in bits */
56     int      dom_maxrtkey;        /* for routing layer */
57 };

```

domain.h

domain.h

图7-5 结构domain 的定义

42-57 dom_family是一个地址族常量(例如AF_INET)，它指示在此域中协议使用的编址方式。dom_name是此域的一个文本名称(例如“internet”)。

除了程序 `fstat(1)` 在它格式化插口信息时使用 `dom_name` 外，成员 `dom_name` 不被 Net/3 内核的任何部分访问。

`dom_init` 指向初始化此域的函数。`dom_externalize` 和 `dom_dispose` 指向那些管理通过此域内通信路径发送的访问权限的函数。Unix 域实现这个特性在进程间传递文件描述符。Internet 域不实现访问权限。

`dom_protosw` 和 `dom_protoswNPROTOSW` 指向一个 `protosw` 结构的数组的起始和结束。`dom_next` 指向在一个内核支持的域链表中的下一个域。包含所有域的链表通过全局指针 `domains` 来访问。

接下来的三个成员，`dom_rtattach`、`dom_rtoffset` 和 `dom_maxrtkey` 保存此域的路由信息。它们在第 18 章讨论。

图 7-6 显示了一个 `domains` 列表的例子。

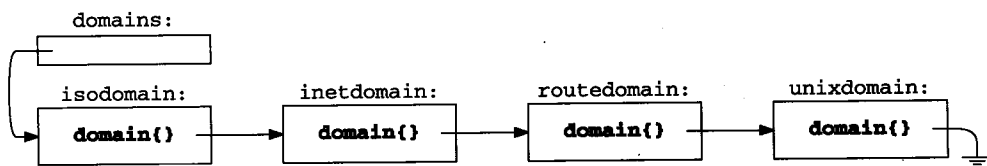


图 7-6 domains 列表

7.4 protosw 结构

在编译期间，Net/3 为内核中的每个协议分配一个 `protosw` 结构并初始化，同时将在一个域中的所有协议的这个结构组织到一个数组中。每个 `domain` 结构引用相应的 `protosw` 结构数组。一个内核可以通过提供多个 `protosw` 项为同一协议提供多个接口。Protosw 结构的定义见图 7-7。

```

57 struct protosw {
58     short    pr_type;           /* see (Figure 7.8) */
59     struct domain *pr_domain;   /* domain protocol a member of */
60     short    pr_protocol;       /* protocol number */
61     short    pr_flags;          /* see Figure 7.9 */
62 /* protocol-protocol hooks */
63     void      (*pr_input) ();    /* input to protocol (from below) */
64     int       (*pr_output) ();   /* output to protocol (from above) */
65     void      (*pr_ctlinput) (); /* control input (from below) */
66     int       (*pr_ctloutput) (); /* control output (from above) */
67 /* user-protocol hook */
68     int       (*pr_usrreq) ();   /* user request from process */
69 /* utility hooks */
70     void      (*pr_init) ();     /* initialization hook */
71     void      (*pr_fasttimo) (); /* fast timeout (200ms) */
72     void      (*pr_slowtimo) (); /* slow timeout (500ms) */
73     void      (*pr_drain) ();    /* flush any excess space possible */
74     int       (*pr_sysctl) ();   /* sysctl for protocol */
75 };

```

protosw.h

protosw.h

图 7-7 protosw 结构的定义

57-61 此结构中的前 4 个成员用来标识和表征协议。pr_type 指示协议的通信语义。图 7-8

列出了pr_type可能的值和对应的Internet协议。

pr_type	协议 语义	Internet协议
<i>SOCK_STREAM</i>	可靠的双向字节流服务	TCP
<i>SOCK_DGRAM</i>	最好的运输层数据报服务	UDP
<i>SOCK_RAW</i>	最好的网络层数据报服务	ICMP, IGMP, 原始IP
<i>SOCK_RDM</i>	可靠的数据报服务 (未实现)	n/a
<i>SOCK_SEQPACKET</i>	可靠的双向记录流服务	n/a

图7-8 pr_type 指明协议的语义

pr_domain指向相关的domain结构, pr_protocol为域中协议的编号, pr_flags标识协议的附加特征。图7-9列出了pr_flags的可能值。

pr_flags	说 明
<i>PR_ATOMIC</i>	每个进程请求映射为一个单个的协议请求
<i>PR_ADDR</i>	协议在每个数据报中都传递地址
<i>PR_CONNREQUIRED</i>	协议是面向连接的
<i>PR_WANTRCVD</i>	当一个进程接收到数据时通知协议
<i>PR_RIGHTS</i>	协议支持访问权限

图7-9 pr_flags 的值

如果一个协议支持 PR_ADDR, 必须也支持 PR_ATOMIC。PR_ADDR和 PR_CONNREQUIRED是互斥的。

当设置了PR_WANTRCVD, 并当插口层将插口接收缓存中的数据传递给一个进程时(即当在接收缓存中有更多空间可用时), 它通知协议层。

PR_RIGHTS指示访问权限控制报文能通过连接来传递。访问权限要求内核中的其他支持来确保在接收进程没有销毁报文时能完成正确的清除工作。仅 Unix域支持访问权限, 在那里它们用来在进程间传递描述符。

图7-10所示的是协议类型、协议标志和协议语义间的关系。

pr_type	PR_			是否有记录边界	可靠否	举 例	
	ADDR	ATOMIC	CONNREQUIRED			Internet	其他
<i>SOCK_STREAM</i>			•	否	•	TCP	SPP
<i>SOCK_SEQPACKET</i>		•	•	显式 隐式	• •		TP4 SPP
<i>SOCK_RDM</i>		•	•	隐式	见正文		RDP
<i>SOCK_DGRAM</i> <i>SOCK_RAW</i>	• •	• •		隐式 隐式		UDP ICMP	

图7-10 协议特征和举例

图7-10不包括标志 PR_WANTRCVD和PR_RIGHTS。对于可靠的面向连接的协议, PR_WANTRCVD总是被设置。

为了理解 Net/3中一个 protosw项的通信语义, 我们必须一起考虑 PRxxx标志和 pr_type。在图7-10中, 我们用两列(“是否有记录边界”和“可靠否”)来描述由 pr_type

隐式指示的语义。图7-10显示了可靠协议的三种类型：

- 面向连接的字节流协议，如TCP和SPP(源于XNS协议族)。这些协议用SOCK_STREAM标识。
- 有记录边界的面向连接的流协议用 SOCK_SEQPACKET标识。在这种协议类型中，PR_ATOMIC指示记录是否由每个输出请求隐式地指定，或者显式地通过在输出中设置标志MSG_EOR来指定。

SPP同时支持语义SOCK_STREAM和SOCK_SEQPACKET。

- 第三种可靠协议提供一个有隐式记录边界的面向连接服务，它由 SOCK_RDM标识。RDP不保证按照记录发送的顺序接收记录。RDP在[Partridge 1987]中讨论并在RFC 115 [Partridge and Hinden 1990]中被描述。

两种不可靠协议显示在图7-10中：

- 一个运输层数据报协议，如UDP，它包括复用和检验和，由SOCK_DGRAM指定。
- 一个网络层数据报协议，如ICMP，它由SOCK_RAW指定。在Net/3中，只有超级用户进程才能创建一个SOCK_RAW插口(图15-8)。

62-68 接着的5个成员是函数指针，用来提供从其他协议对此协议的访问。pr_input处理从一个低层协议输入的数据，pr_output处理从一个高层协议输出的数据，pr_ctlinput处理来自下层的控制信息，而pr_ctloutput处理来自上层的控制信息。pr_usrreq处理来自进程的所有通信请求。如图7-11所示。

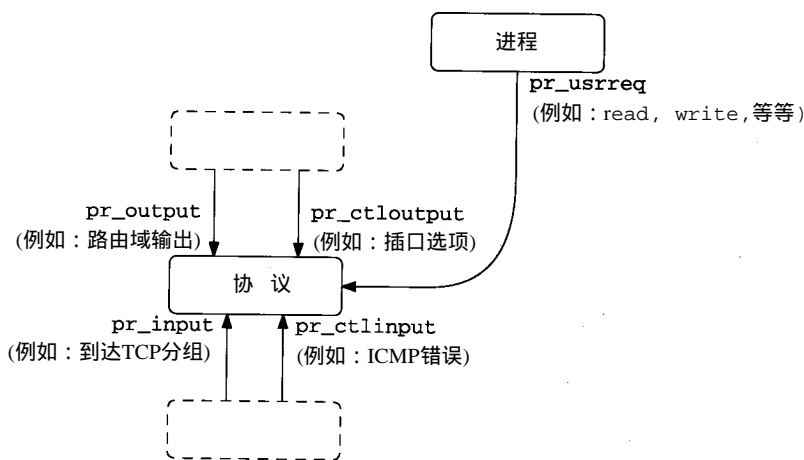


图7-11 一个协议的5个主要入口点

69-75 剩下的5个成员是协议的实用函数。pr_init处理初始化。pr_fasttimo和pr_slowtimo分别每200 ms和500 ms被调用来执行周期性的协议函数，如更新重传定时器。pr_drain在内存缺乏时被m_reclaim调用(图2-13)。它请求协议释放尽可能多的内存。pr_sysctl为sysctl(8)命令提供一个接口，一种修改系统范围的参数的方式，如允许转发分组或UDP检验和计算。

7.5 IP的domain和protosw结构

声明所有协议的结构 domain和protosw，并进行静态初始化。对于 Internet协议，inetsw数组包含protosw结构。图7-12总结了在数组inetsw中的协议信息。图7-13显示了

Internet协议的数组定义和domain结构的定义。

inetsw[]	pr_protocol	pr_type	说 明	缩 写
0	0	0	Internet协议	IP
1	IPPROTO_UDP	SOCK_DGRAM	用户数据报协议	UDP
2	IPPROTO_TCP	SOCK_STREAM	传输控制协议	TCP
3	IPPROTO_RAW	SOCK_RAW	Internet协议(原始)	IP(原始)
4	IPPROTO_ICMP	SOCK_RAW	Internet控制报文协议	ICMP
5	IPPROTO_IGMP	SOCK_RAW	Internet组管理协议	IGMP
6	0	SOCK_RAW	Internet协议(原始、默认)	IP(原始)

图7-12 Internet域协议

```

39 struct protosw inetsw[] = in_proto.c
40 {
41     {0, &inetdomain, 0, 0,
42      0, ip_output, 0, 0,
43      0,
44      ip_init, 0, ip_slowtimo, ip_drain, ip_sysctl
45     },
46     {SOCK_DGRAM, &inetdomain, IPPROTO_UDP, PR_ATOMIC | PR_ADDR,
47      udp_input, 0, udp_ctlinput, ip_ctloutput,
48      udp_usrreq,
49      udp_init, 0, 0, 0, udp_sysctl
50     },
51     {SOCK_STREAM, &inetdomain, IPPROTO_TCP, PR_CONNREQUIRED | PR_WANTRCVD,
52      tcp_input, 0, tcp_ctlinput, tcp_ctloutput,
53      tcp_usrreq,
54      tcp_init, tcp_fasttimo, tcp_slowtimo, tcp_drain,
55     },
56     {SOCK_RAW, &inetdomain, IPPROTO_RAW, PR_ATOMIC | PR_ADDR,
57      rip_input, rip_output, 0, rip_ctloutput,
58      rip_usrreq,
59      0, 0, 0, 0,
60     },
61     {SOCK_RAW, &inetdomain, IPPROTO_ICMP, PR_ATOMIC | PR_ADDR,
62      icmp_input, rip_output, 0, rip_ctloutput,
63      rip_usrreq,
64      0, 0, 0, 0, icmp_sysctl
65     },
66     {SOCK_RAW, &inetdomain, IPPROTO_IGMP, PR_ATOMIC | PR_ADDR,
67      igmp_input, rip_output, 0, rip_ctloutput,
68      rip_usrreq,
69      igmp_init, igmp_fasttimo, 0, 0,
70     },
71     /* raw wildcard */
72     {SOCK_RAW, &inetdomain, 0, PR_ATOMIC | PR_ADDR,
73      rip_input, rip_output, 0, rip_ctloutput,
74      rip_usrreq,
75      rip_init, 0, 0, 0,
76     },
77 };

78 struct domain inetdomain =
79 {AF_INET, "internet", 0, 0, 0,
80  inetsw, &inetsw[sizeof(inetsw) / sizeof(inetsw[0])], 0,
81  rn_inithread, 32, sizeof(struct sockaddr_in)};
in_proto.c

```

图7-13 Internet的domain和protosw结构

39-77 在`inetsw`数组中的3个`protosw`结构提供对IP的访问。第一个：`inetsw[0]`，标识IP的管理函数并且只能由内核访问。其他两项：`inetsw[3]`和`inetsw[6]`，除了`pr_protocol`值以外它们是一样的，都提供到IP的一个原始接口。`inetsw[3]`处理接收到的任何未识别协议的分组。`inetsw[6]`是默认的原始协议，当没有找到其他可匹配的项时，这个结构由函数`pffindproto`返回(7.6节)。

在Net/3以前的版本中，通过`inetsw[3]`传输不带IP首部的分组，由进程负责构造正确的首部。由内核通过`inetsw[6]`传输带IP首部的分组。4.3BSD Reno引入了`IP_HDRINCL`插口选项(32.8节)，这样在`inetsw[3]`和`inetsw[6]`之间的区别就不再重要了。

原始接口允许一个进程发送和接收不涉及运输层的IP分组。原始接口的一个用途是实现内核外的传输协议。一旦这个协议稳定下来，就能移植到内核中改进它的性能和对其他进程的可用性。另一个用途就是作为诊断工具，如`traceroute`，它使用原始IP接口来直接访问IP。第32章讨论原始IP接口。图7-14总结了IP `protosw`结构。

protosw	inetsw[0]	inetsw[3和6]	说 明
<code>pr_type</code>	0	<code>SOCK_RAW</code>	IP提供原始分组服务
<code>pr_domain</code>	<code>&inetdomain</code>	<code>&inetdomain</code>	两协议都是Internet域的一部分
<code>pr_protocol</code>	0	<code>IPPROTO_RAW</code> 或0	<code>IPPROTO_RAW</code> (255)和0都是预留的(RFC 1700)，并且不应在一个IP数据报中出现
<code>pr_flags</code>	0	<code>PR_ATOMIC/PR_ADDR</code>	插口层标志，IP不使用
<code>pr_input</code>	null	<code>rip_input</code>	从IP、ICMP或IGMP接收未识别的数据报
<code>pr_output</code>	<code>ip_output</code>	<code>rip_output</code>	分别准备并发送数据报到IP和硬件层
<code>pr_ctlinput</code>	null	null	IP不使用
<code>pr_ctloutput</code>	null	<code>rip_ctloutput</code>	响应来自进程的配置请求
<code>pr_usrreq</code>	null	<code>rip_usrreq</code>	响应来自进程的协议请求
<code>pr_init</code>	<code>ip_init</code>	null或 <code>rip_init</code>	<code>ip_init</code> 完成所有初始化
<code>pr_fasttimo</code>	null	null	IP不使用
<code>pr_slowtimo</code>	<code>ip_slowtimo</code>	null	用于IP重装算法的慢超时
<code>pr_drain</code>	<code>ip_drain</code>	null	如果可能，释放内存
<code>pr_sysctl</code>	<code>ip_sysctl</code>	null	修改系统范围参数

图7-14 IP `inetsw` 的条目

78-81 Internet协议的domain结构显示在图7-13的下部。Internet域使用`AF_INET`风格编址，文本名称为“`internet`”，没有初始化和控制报文函数，它的`protosw`结构在`inetsw`数组中。

Internet协议的路由初始化函数是`rn_inithead`。一个IP地址的最大有效位数为32，并且一个Internet选路键的大小为一个`sockaddr_in`结构的大小(16字节)。

`inetsw[3]`和`inetsw[6]`的唯一区别是它们的`pr_protocol`号和初始化函数`rip_init`，它仅在`inetsw[6]`中定义，因此只在初始化期间被调用一次。

domaininit函数

在系统初始化期间(图3-23),内核调用domaininit来链接结构domain和protosw。domaininit显示在图7-15中。

```

37 /* simplifies code in domaininit */
38 #define ADDDOMAIN(x) { \
39     extern struct domain __CONCAT(x,domain); \
40     __CONCAT(x,domain.dom_next) = domains; \
41     domains = &__CONCAT(x,domain); \
42 }
43 domaininit()
44 {
45     struct domain *dp;
46     struct protosw *pr;
47     /* The C compiler usually defines unix. We don't want to get
48      * confused with the unix argument to ADDDOMAIN
49      */
50 #undef unix
51     ADDDOMAIN(unix);
52     ADDDOMAIN(route);
53     ADDDOMAIN(inet);
54     ADDDOMAIN(iso);
55     for (dp = domains; dp; dp = dp->dom_next) {
56         if (dp->dom_init)
57             (*dp->dom_init) ();
58         for (pr = dp->dom_protosw; pr < dp->dom_protoswnNPROTOSW; pr++)
59             if (pr->pr_init)
60                 (*pr->pr_init) ();
61     }
62     if (max_linkhdr < 16) /* XXX */
63         max_linkhdr = 16;
64     max_hdr = max_linkhdr + max_protohdr;
65     max_datalen = MHLEN - max_hdr;
66     timeout(pffasttimo, (void *) 0, 1);
67     timeout(pfslowtimo, (void *) 0, 1);
68 }

```

uipc_domain.c

图7-15 函数domaininit

37-42 ADDDOMAIN宏声明并链接一个domain结构。例如,ADDDOMAIN(unix)展开为:

```

extern struct domain unixdomain;
unixdomain.dom_next = domains;
domains = &unixdomain;

```

宏_CONCAT定义在sys/defs.h中,并且连接两个符号名。例如 _CONCAT(unix, domain)产生unixdomain。

43-54 domaininit通过调用ADDDOMAIN为每个支持的域构造域列表。

因为符号unix常常被C预处理器预定义,因此,Net/3在这里显式地取消它的定义,使ADDDOMAIN能正确工作。

图7-16显示了链接的结构 domain 和 protosw，它们配置在内核中来支持 Internet、Unix 和 OSI 协议族。

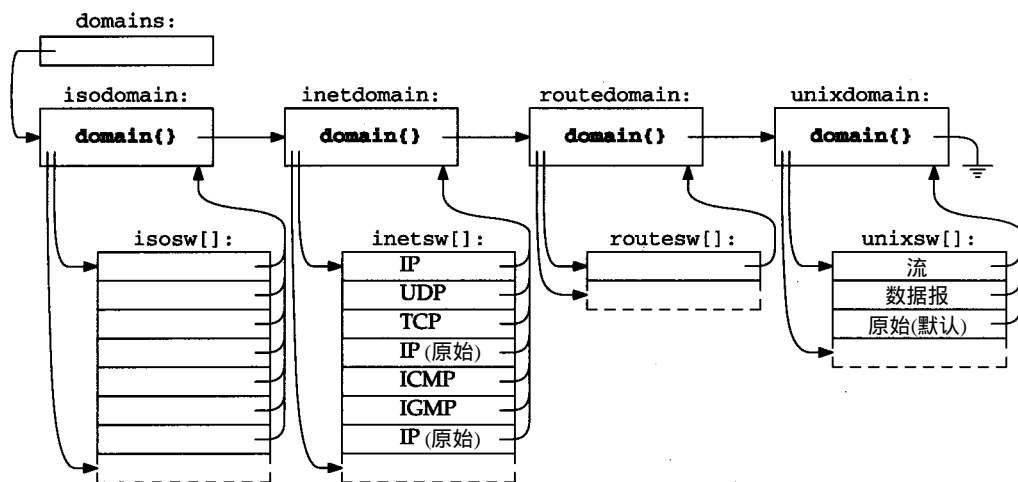


图7-16 初始化后的 domain 链表和 protosw 数组

55-61 两个嵌套的 for 循环查找内核中的每个域和协议，并且若定义了初始化函数 dom_init 和 pr_init，则调用它们。对于 Internet 协议，调用下面的函数（图7-13）：ip_init、udp_init、tcp_init、igmp_init 和 rip_init。

62-65 在 domaininit 中计算这些参数，用来控制 mbuf 中分组的格式，以避免对数据的额外复制。在协议初始化期间设置了 max_linkhdr 和 max_protohdr。这里，domaininit 将 max_linkhdr 强制设置为一个下限 16。16 字节用于给带有 4 字节边界的 14 字节以太网首部留出空间。图7-17和图7-18列出了这些参数和典型的取值。

变 量	值	说 明
max_linkhdr	16	由链路层添加的最大字节数
max_protohdr	40	由网络和运输层添加的最大字节数
max_hdr	56	max_linkhdr + max_protohdr
max_datalen	44	在计算了链路和协议首部后的分组首部 mbuf 中的可用数据字节数

图7-17 用来减少协议数据复制的参数

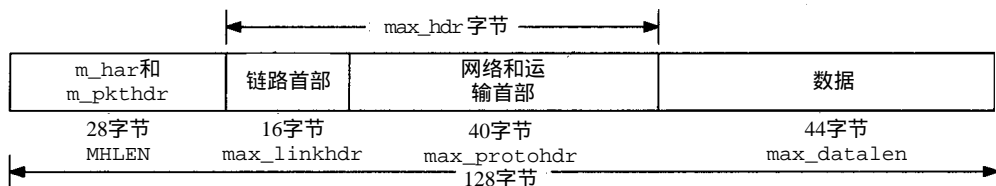


图7-18 mbuf 和相关的最大首部长度

max_protohdr 是一个软限制，估算预期的协议首部大小。在 Internet 域中，IP 和 TCP 首部长度通常为 20 字节，但都可达到 60 字节。长度超过 max_protohdr 的代价是花时间将数据向后移动，以留出比预期的协议首部更大的空间。

66-68 domaininit通过调用timeout启动pfslowtimo和pffasttimo。第3个参数指明何时内核应该调用这个函数，在这里是在1个时钟滴答内。两个函数都显示在图7-19中。

uipc_domain.c

```

153 void
154 pfslowtimo(arg)
155 void *arg;
156 {
157     struct domain *dp;
158     struct protosw *pr;

159     for (dp = domains; dp; dp = dp->dom_next)
160         for (pr = dp->dom_protosw; pr < dp->dom_protoswnPROTOSW; pr++)
161             if (pr->pr_slowtimo)
162                 (*pr->pr_slowtimo) ();
163     timeout(pfslowtimo, (void *) 0, hz / 2);
164 }

165 void
166 pffasttimo(arg)
167 void *arg;
168 {
169     struct domain *dp;
170     struct protosw *pr;

171     for (dp = domains; dp; dp = dp->dom_next)
172         for (pr = dp->dom_protosw; pr < dp->dom_protoswnPROTOSW; pr++)
173             if (pr->pr_fasttimo)
174                 (*pr->pr_fasttimo) ();
175     timeout(pffasttimo, (void *) 0, hz / 5);
176 }

```

uipc_domain.c

图7-19 函数pfslowtimo 和pffasttimo

153-176 这两个相近的函数用两个 for 循环分别为每个协议调用函数 pr_slowtimo 和 pr_fasttimo，前提是如果定义了这两个函数。这两个函数每 500 ms 和 200 ms 通过调用 timeout 调度自己一次，timeout 在图3-43中讨论过。

7.6 pffindproto和pffindtype函数

如图7-20所示，函数pffindproto和pffindtype通过编号(例如IPPROTO_TCP)或类型(例如SOCK_STREAM)来查找一个协议。如我们在第15章要看到的，当进程创建一个插口时，这两个函数被调用来查找相应的 protosw 项。

69-84 pffindtype 线性搜索 domains，查找指定的族，然后在域内搜索第一个为此指定类型的协议。

85-107 pffindproto 像 pffindtype 一样搜索 domains，查找由调用者指定的族、类型和协议。如果 pffindproto 在指定的协议族中没有发现一个 (protocol, type) 匹配项，并且 type 为 SOCK_RAW，而此域有一个默认的原始协议 (pr_protocol 等于 0)，则 pffindproto 选择默认的原始协议而不是完全失败。例如，一个调用如下：

```
pffindproto(PF_INET, 27, SOCK_RAW);
```

它返回一个指向 inetsw[6] 的指针，默认的原始 IP 协议，因为 Net/3 不包括对协议 27 的支持。通过访问原始 IP，一个进程可以使用内核来管理 IP 分组的发送和接收，从而自己实现协议 27

服务。

协议27预留给可靠的数据报协议(RFC 1151)。

两个函数都返回一个所选协议的 `protosw` 结构的指针；或者，如果没有找到匹配项，则返回一个空指针。

```

69 struct protosw *
70 pffindtype(family, type)
71 int      family, type;
72 {
73     struct domain *dp;
74     struct protosw *pr;
75     for (dp = domains; dp; dp = dp->dom_next)
76         if (dp->dom_family == family)
77             goto found;
78     return (0);
79 found:
80     for (pr = dp->dom_protosw; pr < dp->dom_protoswnNPROTOSW; pr++)
81         if (pr->pr_type == type)
82             return (pr);
83     return (0);
84 }

85 struct protosw *
86 pffindproto(family, protocol, type)
87 int      family, protocol, type;
88 {
89     struct domain *dp;
90     struct protosw *pr;
91     struct protosw *maybe = 0;
92     if (family == 0)
93         return (0);
94     for (dp = domains; dp; dp = dp->dom_next)
95         if (dp->dom_family == family)
96             goto found;
97     return (0);
98 found:
99     for (pr = dp->dom_protosw; pr < dp->dom_protoswnNPROTOSW; pr++) {
100         if ((pr->pr_protocol == protocol) && (pr->pr_type == type))
101             return (pr);
102         if (type == SOCK_RAW && pr->pr_type == SOCK_RAW &&
103             pr->pr_protocol == 0 && maybe == (struct protosw *) 0)
104             maybe = pr;
105     }
106     return (maybe);
107 }

```

uipc_domain.c

图7-20 函数 `pffindproto` 和 `pffindtype`

举例

我们在15.6节中会看到，当一个应用程序进行下面的调用时：

```
socket(PF_INET, SOCK_STREAM, 0);    /* TCP 插口 */
```

`pffindtype`被调用如下：

```
pffindtype(PF_INET, SOCK_STREAM);
```

图7-12显示pffindtype会返回一个指向inetsw[2]的指针，因为TCP是此数组中第一个SOCK_STREAM协议。同样，

```
socket(PF_INET, SOCK_DGRAM, 0); /* UCP 插口 */
```

会导致

```
pffindtype(PF_INET, SOCK_DGRAM);
```

它返回一个指向inetsw[1]中UDP的指针。

7.7 pfctlinput函数

函数pfctlinput给每个域中的每个协议发送一个控制请求(图7-21)。当可能影响每个协议的事件发生时，使用这个函数，例如一个接口被关闭，或路由表发生改变。当一个ICMP重定向报文到达时，ICMP调用pfctlinput(图11-14)，因为重定向会影响所有Internet协议(例如UDP和TCP)。

```

142 pfctlinput(cmd, sa)                                     uipc_domain.c
143 int      cmd;
144 struct sockaddr *sa;
145 {
146     struct domain *dp;
147     struct protosw *pr;

148     for (dp = domains; dp; dp = dp->dom_next)
149         for (pr = dp->dom_protosw; pr < dp->dom_protoswnPROTOSW; pr++)
150             if (pr->pr_ctlinput)
151                 (*pr->pr_ctlinput) (cmd, sa, (caddr_t) 0);
152 }

```

uipc_domain.c

图7-21 函数pfctlinput

142-152 两个嵌套的for循环查找每个域中的每个协议。pfctlinput通过调用每个协议的pr_ctlinput函数来发送由cmd指定的协议控制命令。对于UDP，调用udp_ctlinput；而对于TCP，调用tcp_ctlinput。

7.8 IP初始化

如图7-13所示，Internet域没有一个初始化函数，但单个Internet协议有。现在，我们仅查看IP初始化函数ip_init。在第23章和第24章中，我们讨论UDP和TCP初始化函数。在讨论这些代码前，需要说明一下数组ip_prottox。

7.8.1 Internet传输分用

一个网络层协议像IP必须分用输入数据报，并将它们传递到相应的运输层协议。为了完成这些，相应的protosw结构必须通过一个在数据报中出现的协议编号得到。对于Internet协议，这由数组ip_prottox来完成，如图7-22所示。

数组ip_prottox的下标是来自IP首部的协议值(ip_p，图8-8)。被选项是inetsw数组中处理此数据报的协议的下标。例如，一个协议编号为6的数据报由inetsw[2]处理，协议为TCP。内核在协议初始化时构造ip_prottox，如图7-23所示。

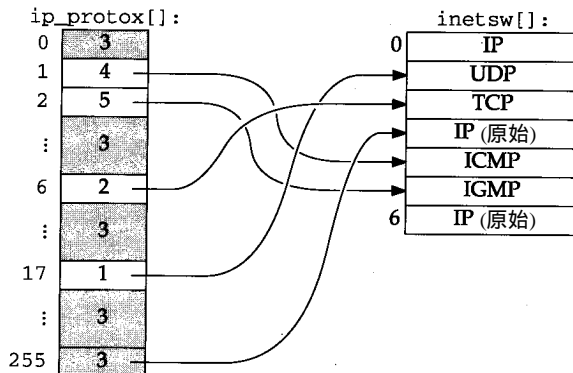


图7-22 数组ip_prottox 将协议编号映射到数组 inetsw 中的一项

```

71 void
72 ip_init()
73 {
74     struct protosw *pr;
75     int i;

76     pr = pffindproto(PF_INET, IPPROTO_RAW, SOCK_RAW);
77     if (pr == 0)
78         panic("ip_init");
79     for (i = 0; i < IPPROTO_MAX; i++)
80         ip_prottox[i] = pr - inetsw;
81     for (pr = inetdomain.dom_protoswNPROTOSW; pr++)
82         if (pr->pr_domain->dom_family == PF_INET &&
83             pr->pr_protocol && pr->pr_protocol != IPPROTO_RAW)
84             ip_prottox[pr->pr_protocol] = pr - inetsw;
85     ipq.next = ipq.prev = &ipq;
86     ip_id = time.tv_sec & 0xffff;
87     ipintrq.ifq_maxlen = ipqmaxlen;
88     i = (if_index + 1) * (if_index + 1) * sizeof(u_long);
89     ip_ifmatrix = (u_long *) malloc(i, M_RTABLE, M_WAITOK);
90     bzero((char *) ip_ifmatrix, i);
91 }
92 }

```

ip_input.c

图7-23 函数ip_init

7.8.2 ip_init函数

domaininit (图7-15)在系统初始化期间调用函数 ip_init。

71-78 pffindproto返回一个指向原始协议(inetsw[3], 图7-14)的指针。如果找不到原始协议, Net/3就调用panic, 因为这是内核要求的部分。如果找不到原始协议, 内核一定被错误配置了。IP将传输到一个未知传输协议的到达分组传递给此协议, 在那里它们由内核外部的一个进程来处理。

79-85 接着的两个循环初始化数组 ip_prottox。第一个循环将数组中的每项设置为 pr, 即默认协议的下标(图7-22中为3)。第二个循环检查 inetsw中的每个协议(而不是协议编号为0或IPPROTO_RAW的项), 并且将 ip_prottox中的匹配项设置为引用相应的 inetsw项。为此, 每个 protosw结构中的 pr_protocol必须是期望出现在输入数据报中的协议编号。

86-92 ip_init初始化IP重装队列ipq(10.6节),用系统时钟植入ip_id,并将IP输入队列(ipintrq)的最大长度设置为50(ipqmaxlen)。ip_id用系统时钟设置,为数据报标识符提供一个随机起点(10.6节)。最后,ip_init分配一个二维数组ip_ifmatrix,统计在系统接口之间路由的分组数。

在Net/3中,有很多变量可以被一个系统管理员修改。为了允许在运行时改变这些变量而不需重新编译内核,一个常量(在此例中是IFQ_MAXLEN)表示的默认值在编译时指派给一个变量(ipqmaxlen)。一个系统管理员能使用一个内核调试器如adb,来修改ipqmaxlen,并用新值重启内核。如果图7-23直接使用IFQ_MAXLEN,它会要求内核重新编译来改变这个限制。

7.9 sysctl系统调用

系统调用sysctl访问并修改Net/3系统范围参数。系统管理员通过程序sysctl(8)修改这些参数。每个参数由一个分层的整数列表来标识,并有一个相应的类型。此系统调用的原型为:

```
int sysctl(int * name, u_int namelen, void *old, size_t * oldlenp, void *new, size_t newlen);
```

*name指向一个包含namelen个整数的数组。*old指向在此范围内返回的旧值,*new指向在此范围内传递的新值。

图7-24总结了关于联网名称的组织。

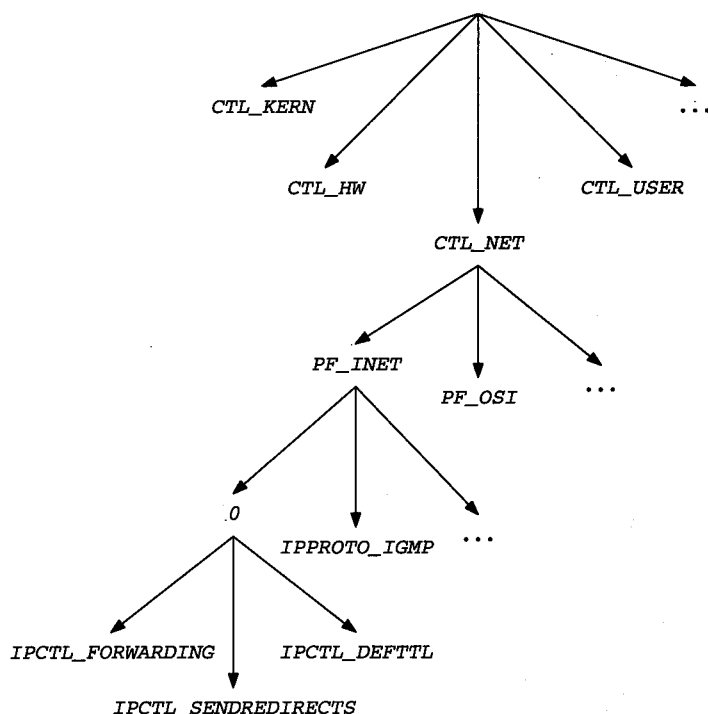


图7-24 sysctl 的名称组织

在图7-24中,IP转发标志的全名为

CTL_NET、PF_INET、0、IPCTL_FORWARDING

用4个整数存储在一个数组中。

net_sysctl函数

每层的sysctl命名方案通过不同函数处理。图 7-25显示了处理这些Internet参数的函数。

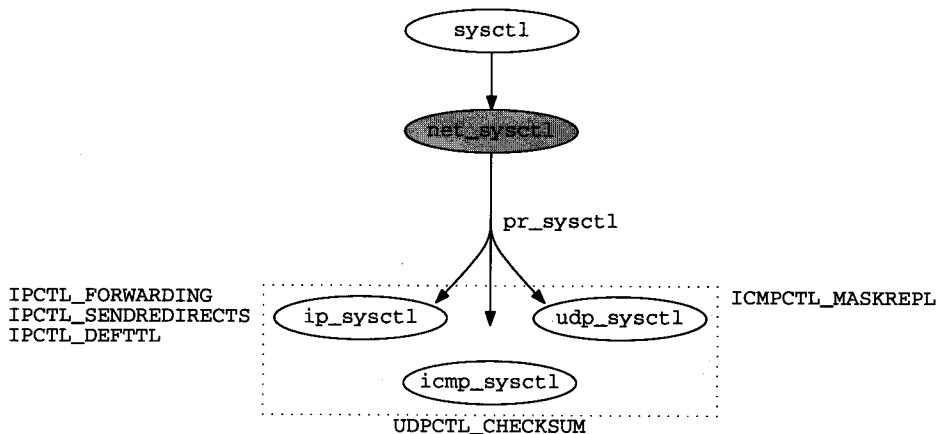


图7-25 处理Internet参数的sysctl函数

顶层名称由sysctl处理。网络层名称由net_sysctl处理，它根据族和协议将控制转给此协议的protosw项指定的pr_sysctl函数。

sysctl在内核中通过_sysctl函数实现，函数_sysctl不在本书中讨论。它包含将sysctl参数传给内核和从内核取出sysctl参数的代码及一个switch语句，这个switch语句选择相应的函数来处理这些参数，在这里是net_sysctl。

图7-26所示的是函数net_sysctl。

```

108 net_sysctl(name, namelen, oldp, oldlenp, newp, newlen, p)
109 int      *name;
110 u_int     namelen;
111 void      *oldp;
112 size_t    oldlenp;
113 void      *newp;
114 size_t    newlen;
115 struct proc *p;
116 {
117     struct domain *dp;
118     struct protosw *pr;
119     int     family, protocol;
120     /*
121      * All sysctl names at this level are nonterminal;
122      * next two components are protocol family and protocol number,
123      * then at least one additional component.
124      */
125     if (namelen < 3)
126         return (EISDIR);          /* overloaded */
127     family = name[0];
128     protocol = name[1];
129     if (family == 0)

```

图7-26 函数net_sysctl

```

130         return (0);
131     for (dp = domains; dp; dp = dp->dom_next)
132         if (dp->dom_family == family)
133             goto found;
134     return (ENOPROTOOPT);
135 found:
136     for (pr = dp->dom_protosw; pr < dp->dom_protoswnPROTOSW; pr++)
137         if (pr->pr_protocol == protocol && pr->pr_sysctl)
138             return ((*pr->pr_sysctl) (name + 2, namelen - 2,
139                                     olddp, oldlenp, newp, newlen));
140     return (ENOPROTOOPT);
141 }

```

uipc_domain.c

图7-26 (续)

108-119 net_sysctl的参数除了增加了p外，同系统调用sysctl一样，p指向当前进程结构。

120-134 在名称中接下来的两个整数被认为是在结构domain和protosw中指定的协议族和协议编号成员的值。如果没有指定族，则返回0。如果指定了族，for循环在域列表中查找一个匹配的族。如果没有找到，则返回ENOPROTOOPT。

135-141 如果找到匹配域，第二个for循环查找第一个定义了函数pr_sysctl的匹配协议。当找到匹配项，将请求传递给此协议的pr_sysctl函数。注意，把(name+2)指向的整数传递给下一级。如果没有找到匹配的协议，返回ENOPROTOOPT。

图7-27所示的是为Internet协议定义的pr_sysctl函数。

pr_protocol	inetsw[]	pr_sysctl	说 明	参 考
0	0	ip_sysctl	IP	8.9节
IPPROTO_UDP	1	udp_sysctl	UDP	23.11节
IPPROTO_ICMP	4	icmp_sysctl	ICMP	11.14节

图7-27 Internet协议族的pr_sysctl 函数

在路由选择域中，pr_sysctl指向函数sysctl_rtable，它在第19章中讨论。

7.10 小结

本章从说明结构domain和protosw开始，这两个结构在Net/3内核中描述及组织协议。我们看到一个域的所有protosw结构在编译时分配在一个数组中，inetdomain和数组inetsw描述Internet协议。我们仔细查看了三个描述IP协议的inetsw项：一个用于内核访问IP，其他两个用于进程访问IP。

在系统初始化时，domainint将域链接到domains列表中，调用域和协议初始化函数，并调用快速和慢速超时函数。

两个函数pffindproto和pffindtype通过协议号或类型搜索域和协议列表。pfctlinput发送一个控制命令给所有协议。

最后，我们说明了IP初始化程序，它通过数组ip_protox完成传输分用。

习题

7.1 由谁调用pfsfindproto会返回一个指向inetsw[6]指针？