第5章　接口：SLIP和环回

5.1　引言

在第4章中，我们查看了以太网接口。在本章中，我们讨论　SLIP和环回接口，同样用ioctl命令来配置所有网络接口。 SLIP驱动程序使用的TCP压缩算法在29.13节讨论。环回驱动程序比较简单，在这里我们要对它进行完整地讨论。

像图4-2一样，图5-1列出了针对我们三个示例驱动程序的入口点。

ifnet	以 太 网	SLIP	环　回	说　明
if_init	leinit			初始化硬件
if_output	ether_output	sloutput	looutput	接收并将要传输的分组进行排队
if_start	lestart			开始传输帧
if_done				输出完成(未用)
if_ioctl	leioctl	slioctl	loioctl	从一个进程处理ioctl命令
if_reset	lereset			将设备重新设置为一已知状态
if_watchdog				监视设备的故障或采集统计信息

图5-1　例子驱动程序的接口函数

5.2　代码介绍

SLIP和环回驱动程序的代码文件列于图5-2中。

文　件	说　明
net/if_slvar.h	SLIP定义
net/if_sl.c	SLIP驱动程序函数
net/if_loop.c	环回驱动程序

图5-2　本章讨论的文件

5.2.1　全局变量

在本章讨论SLIP和环回接口结构。全局变量见图5-3。

变　量	数据类型	说　明
sl_softc	struct sl_softc []	SLIP接口
loif	struct ifnet	环回接口

图5-3　本章中介绍的全局变量

sl_softc是一个数组，因为可能有很多SLIP接口。loif不是一个数组，因为只可能有一个环回接口。

5.2.2 统计量

在第4章讨论的ifnet结构的统计也会被SLIP和环回驱动程序更新。采集的另一个统计量(它不在ifnet结构中)显示在图5-4中。

变 量	说 明	被SNMP使用
tk_nin	被任何串行接口(被SLIP驱动程序更新)接收的字节数	

图5-4 变量tk_nin

5.3 SLIP接口

一个SLIP接口通过一个标准的异步串行线与一个远程系统通信。像以太网一样， SLIP定义了一个标准的方法对传输在串行线上的 IP分组进行组帧。图5-5显示了将一个包含SLIP保留字符的IP分组封装到一个SLIP帧中。

分组用SLIP END字符0xc0来分割开。如果END字符出现在IP分组中，则在它前面填充SLIP ESC字符0xdb，并且在传输时将它替换为 0xdc。当ESC字符出现在IP分组中时，就在它前面填充ESC字符0xdb，并在传输时将它替换为0xdd。

因为在SLIP帧(与以太网比较)中没有类型字段， SLIP仅适用于传输IP分组。

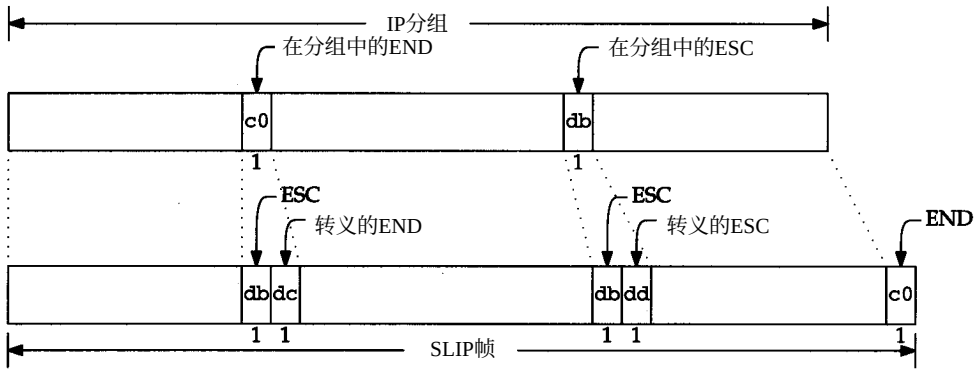

图5-5 将一个IP分组进行SLIP封装

在RFC 1055 [Romkey 1988]中讨论了SLIP，陈述了它的很多弱点和非标准情况。卷1中包含了SLIP封装的详细讨论。

点对点协议(PPP)被设计用来解决SLIP的问题，并提供一个标准方法来通过一个串行链路传输帧。PPP在RFC 1332 [McGregor 1992]和RFC 1548 [Simpson 1993]中定义。Net/3不包含一个PPP的实现，因此我们不在本书中讨论它。关于 PPP的更多信息见卷1的2.6节。附录B讨论在哪里获得一个PPP实现的参考。

5.3.1 SLIP线路规程：SLIPDISC

在Net/3中， SLIP接口依靠一个异步串行设备驱动器来发送和接收数据。传统上，这些设备驱动器称为TTY(电传机)。Net/3 TTY子系统包括一个线路规程(Line discipline)的概念，这个线路规程作为一个在物理设备和 I/O系统调用(如read和write)之间的过滤器。一个线路规

程实现以下特性：如行编辑、换行和回车处理、制表符扩展等等。 SLIP接口作为TTY子系统的一个线路规程，但它不把输入数据传给从设备读数据的进程，也不接受来自向设备写数据的进程的输出数据。 SLIP接口将输入分组传给IP输入队列，并通过SLIP的ifnet结构中的函数if_output来获得要输出的分组。内核通过一个整数常量来标识线路规程，对于 SLIP，该常量是SLIPDISC。

图5-6左边显示的是传统的线路规程，右边是 SLIP规程。我们在右边用slattach显示进程，因为它是初始化 SLIP接口的程序。 TTY子系统和线路规程的细节超出了本书的范围。我们仅介绍理解 SLIP代码工作的相关信息。对于更多关于 TTY子系统的信息见 [Leffler et al. 1989]。图5-7列出了实现SLIP驱动程序的函数。中间的列指示函数是否实现线路规程特性和(或)网络接口特性。

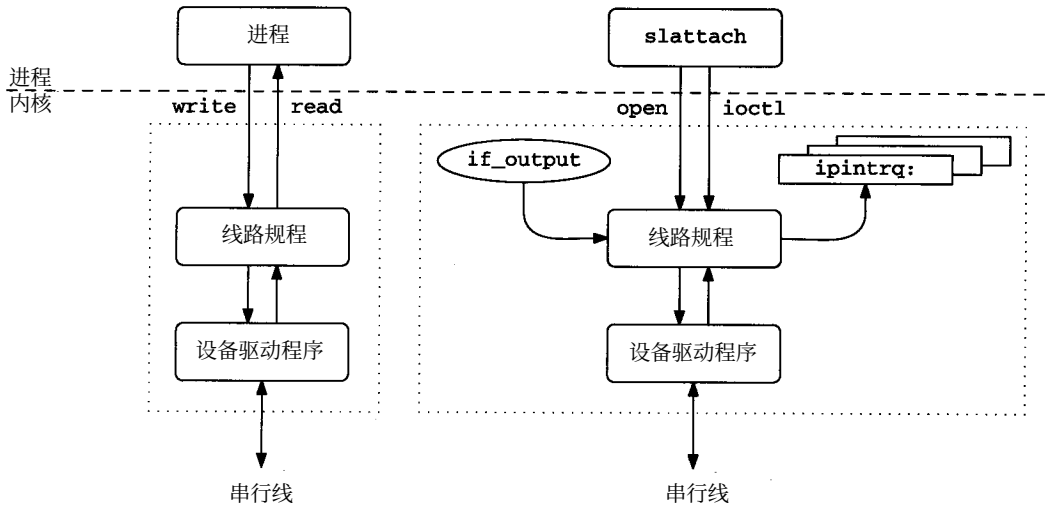

图5-6 SLIP接口作为一个线路规程

函 数	网络接口	线路规程	说 明
slattach	•		初始化sl_softc结构，并将它连接到ifnet列表
slinit	•		初始化SLIP数据结构
sloutput	•		对相关TTY设备上要传输的输出分组进行排队
slioctl	•		处理插口ioctl请求
sl_btom	•		将一个设备缓存转换成一个mbuf链表
slopen		•	将sl_softc结构连接到TTY设备，并初始化驱动程序
slclose		•	取消TTY设备与sl_softc结构的连接，标记接口为关闭，并释放存储器
sltioctl		•	处理TTY ioctl命令
slstart	•	•	从队列中取分组，并开始在TTY设备上传输数据
slinput	•	•	处理从TTY设备输入的字节，如果整个帧被接收，就排列输入的分组

图5-7 SLIP设备驱动程序的函数

在Net/3中的SLIP驱动程序通过支持TCP分组首部压缩来得到更好的吞吐量。我们在 29.13节讨论分组首部压缩，因此，图 5-7跳过实现这些特性的函数。

Net/3 SLIP接口还支持一种转义序列。当接收方检测到这个序列时，就终止 SLIP

的处理，并将对设备的控制返回给标准线路规程。我们这里的讨论忽略这个处理。

图5-8显示了作为一个线路规程的SLIP和作为一个网络接口的SLIP间的复杂关系。

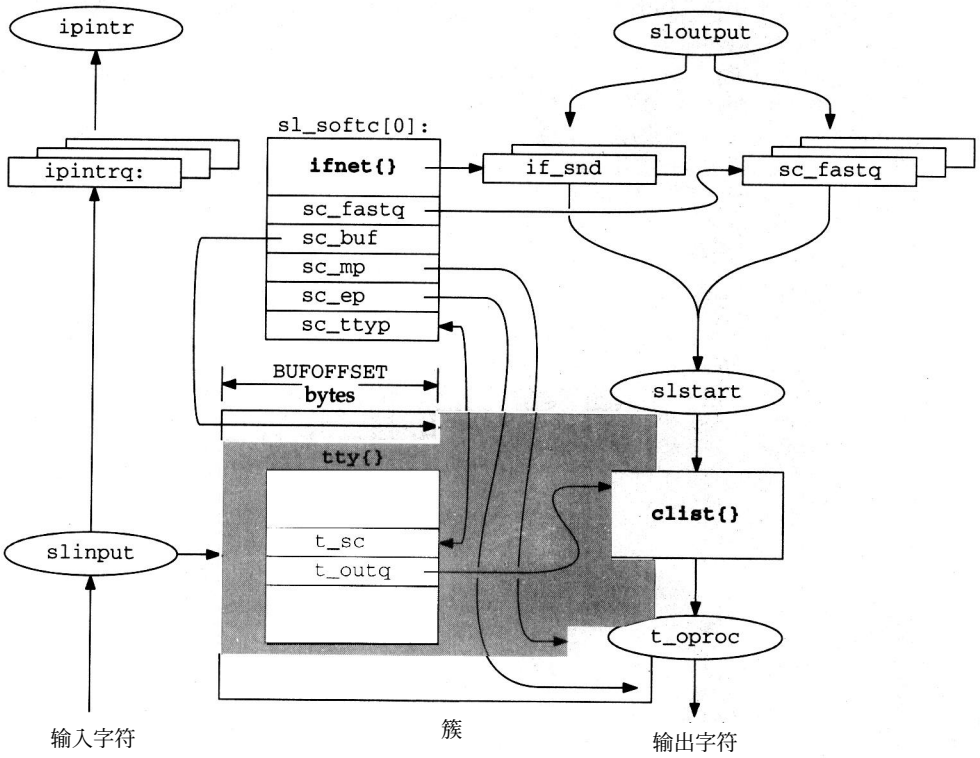

图5-8 SLIP设备驱动程序

在Net/3中，sc_ttyp和t_sc指向tty结构和sl_softc[0]结构。由于使用两个箭头会使图显得较乱，我们用一对相反的箭头表示两个指针来说明结构间的双链。

在图5-8中包含很多信息：

- 结构sl_softc表示的网络接口和结构tty表示的TTY设备。
- 输入字节存放在簇中(显示在结构tyy后面)。当一个完整的SLIP帧被接收时，封装的IP分组被slinput放到ipintrq中。
- 输出分组从if_snd或sc_fastq退队，转换成SLIP帧，并被slstart传给TTY设备。TTY缓存将字节输出到结构clist。函数t_oproc取完，并传输在clist结构中的字节。

5.3.2 SLIP初始化：slopen和slinit

我们在3.7节讨论了slattach是如何初始化sl_softc结构的。接口虽然被初始化，但还不能操作，直到一个程序(通常是slattach)打开一个TTY设备(例如：/dev/tty01)，并发送一个ioctl命令用SLIP规程代替标准的线路规程才能操作。这时， TTY子系统调用线路规程的打开函数(在此是slopen)，此函数在一个特定TTY设备和一个特定SLIP接口间建立关联。slopen显示在图5-9中。

if_sl.c

```
181 int
182 slopen(dev, tp)
183 dev_t    dev;
184 struct tty *tp;
185 {
186     struct proc *p = curproc;    /* XXX */
187     struct sl_softc *sc;
188     int      nsl;
189     int      error;

190     if (error = suser(p->p_ucred, &p->p_acflag))
191         return (error);

192     if (tp->t_line == SLIPDISC)
193         return (0);

194     for (nsl = NSL, sc = sl_softc; --nsl >= 0; sc++)
195         if (sc->sc_ttyp == NULL) {
196             if (slinit(sc) == 0)
197                 return (ENOBUFS);
198             tp->t_sc = (caddr_t) sc;
199             sc->sc_ttyp = tp;
200             sc->sc_if.if_baudrate = tp->t_ospeed;
201             ttyflush(tp, FREAD | FWRITE);
202             return (0);
203         }
204     return (ENXIO);
205 }
```

if_sl.c

图5-9 函数slopen

181-193 传递给slopen的两个参数为：dev，一个内核设备标识，slopen未用此参数；tp，一个指向此TTY设备相关tty结构的指针。最开始是一些预防处理：若进程没有超级用户权限，或TTY的线路规程已经被设置为SLIPDISC，则slopen立即返回。

194-205 for循环在sl_softc结构数组中查找第一个未用的项，调用slinit (5.10节)，通过t_sc和sc_ttyp加进结构tty和sl_softc，并将TTY输出速率(t_ospeed)复制到SLIP接口。ttyflush丢弃任何在TTY队列中追加的输入输出数据。如果一个SLIP接口结构不可用，slopen返回ENXIO。若成功，返回0。

注意，第一个变量sl_softc结构与TTY设备相关。如果系统有多个SLIP线路，在TTY设备和SLIP接口间不需要固定的映射。实际上，这个映射依赖于slattach打开和关闭TTY设备的次序。

显示在图5-10中的函数slinit初始化结构sl_softc。

156-175 函数slinit分配一个mbuf簇，并将它用三个指针连接到结构sl_softc。当一个完整的SLIP帧被接收后，输入字节存储在这个簇中。sc_buf总是指向簇中的这个分组的起始位置，sc_mp指向要接收的下一个字节的位置，并且sc_ep指向这个簇的结束。sl_compress_init为此链路初始化TCP首部的压缩状态(29.13节)。

在图5-8中，我们看到sc_buf不指向簇的第一个字节。slinit保留了148字节(BUFOFFSET)的空间，因为输入分组可能含有一个压缩了的首部，它会扩展来填充这个空间。在簇中已接收的字节用阴影表示。我们看到sc_mp指向接收的最后一个字节的下一个字节，并且sc_ep指向这个簇的结尾。图5-11显示了在几个SLIP常量间的关系。

使这个接口能运行，剩下的要做的工作就是给它分配一个 IP地址。同以太网驱动程序一样，我们将地址分配的讨论推迟到6.6节。

if_sl.c

```
156 static int
157 slinit(sc)
158 struct sl_softc *sc;
159 {
160     caddr_t p;

161     if (sc->sc_ep == (u_char *) 0) {
162         MCLALLOC(p, M_WAIT);
163         if (p)
164             sc->sc_ep = (u_char *) p + SLBUFSIZE;
165         else {
166             printf("sl%d: can't allocate buffer\n", sc - sl_softc);
167             sc->sc_if.if_flags &= ~IFF_UP;
168             return (0);
169         }
170     }
171     sc->sc_buf = sc->sc_ep - SLMAX;
172     sc->sc_mp = sc->sc_buf;
173     sl_compress_init(&sc->sc_comp);
174     return (1);
175 }
```

if_sl.c

图5-10 函数slinit

常 量	值	说 明
MCLBYTES	2048	一个mbuf簇的大小
SLBUFSIZE	2048	一个未压缩的SLIP分组的最大长度 —— 包括一个BPF首部
SLIP_HDRLEN	16	SLIP BPF首部的大小
BUFOFFSET	148	一个扩展的TCP/IP首部的最大长度加上一个BPF首部的大小
SLMAX	1900	一个存储在簇中的压缩SLIP分组的最大长度
SLMTU	296	SLIP分组的最佳长度；导致最小的时延，同时还有较高的批量吞吐量
SLIP_HIWAT	100	在TTY输出队列中排队的最大字节数
BUFOFFSET+SLMAX=SLBUFSIZE=MCLBYTES		

图5-11 SLIP常量

5.3.3 SLIP输入处理：slinput

TTY设备驱动程序每次调用slinput，都将输入字符传给SLIP线路规程。图5-12显示了函数slinput，但跳过了帧结束的处理，对于它我们分开讨论。

527-545 传递给slinput的参数为：c，下一个输入字符；tp，一个指向设备tty结构的指针。全局整数tk_nin计算所有TTY设备的输入字符数。slinput将tp->t_sc转换成sc，sc是指向一个sl_softc结构的指针。如果这个TTY设备没有相关联的接口，slinput立即返回。

slinput的第一个参数是一个整数。除了接收的字符，c还包含从TTY设备驱动程序以高位在前的比特序发送的控制字符。如果在c中指示了一个差错，或调制解调器控制线禁用并且不应该被忽略，则SC_ERROR被置位，并且slinput返回。之后，当slinput处理END字符时，此帧被丢弃。标志CLOCAL指示系统应该把这个线路视为一个本地线路 (即不是一个拨号线路)，并且不应该看到调制解调器的控制信号。

if_sl.c

```
527 void
528 slinput(c, tp)
529 int     c;
530 struct tty *tp;
531 {
532     struct sl_softc *sc;
533     struct mbuf *m;
534     int     len;
535     int     s;
536     u_char  chdr[CHDR_LEN];

537     tk_nin++;
538     sc = (struct sl_softc *) tp->t_sc;
539     if (sc == NULL)
540         return;
541     if (c & TTY_ERRORMASK || ((tp->t_state & TS_CARR_ON) == 0 &&
542                               (tp->t_cflag & CLOCAL) == 0)) {
543         sc->sc_flags |= SC_ERROR;
544         return;
545     }
546     c &= TTY_CHARMASK;

547     ++sc->sc_if.if_ibytes;

548     switch (c) {

549     case TRANS_FRAME_ESCAPE:
550         if (sc->sc_escape)
551             c = FRAME_ESCAPE;
552         break;
553     case TRANS_FRAME_END:
554         if (sc->sc_escape)
555             c = FRAME_END;
556         break;

557     case FRAME_ESCAPE:
558         sc->sc_escape = 1;
559         return;

560     case FRAME_END:

                /* FRAME_END code (Figure 5.13) */

636     }
637     if (sc->sc_mp < sc->sc_ep) {
638         *sc->sc_mp++ = c;
639         sc->sc_escape = 0;
640         return;
641     }
642     /* can't put lower; would miss an extra frame */
643     sc->sc_flags |= SC_ERROR;

644   error:
645     sc->sc_if.if_ierrors++;
646   newpack:
647     sc->sc_mp = sc->sc_buf = sc->sc_ep - SLMAX;
648     sc->sc_escape = 0;
649 }
```

if_sl.c

图5-12 函数slinput

546-636 slinput丢弃c中的控制比特，并用TTY_CHARMASK来屏蔽掉，更新接口上接收字节数的计数，同时跳过接收到的字符：

- 如果c是一个转义的ESC字符，并且前一字符为ESC，则slinput用一个ESC字符替代c。
- 如果c是一个转义的END字符，并且前一字符为ESC，则slinput用一个END字符代替c。
- 如果c是SLIP ESC字符，则将sc_escape置位，并且slinput立即返回(即，ESC字符被丢弃)。
- 如果c是SLIP END字符，则将分组放到IP输入队列。处理SLIP帧结束字符的代码显示在图5-13中。

if_sl.c

```
560     case FRAME_END:
561         if (sc->sc_flags & SC_ERROR) {
562             sc->sc_flags &= ~SC_ERROR;
563             goto newpack;
564         }
565         len = sc->sc_mp - sc->sc_buf;
566         if (len < 3)
567             /* less than min length packet - ignore */
568             goto newpack;

569         if (sc->sc_bpf) {
570             /*
571              * Save the compressed header, so we
572              * can tack it on later.  Note that we
573              * will end up copying garbage in some
574              * cases but this is okay.  We remember
575              * where the buffer started so we can
576              * compute the new header length.
577              */
578             bcopy(sc->sc_buf, chdr, CHDR_LEN);
579         }
580         if ((c = (*sc->sc_buf & 0xf0)) != (IPVERSION << 4)) {
581             if (c & 0x80)
582                 c = TYPE_COMPRESSED_TCP;
583             else if (c == TYPE_UNCOMPRESSED_TCP)
584                 *sc->sc_buf &= 0x4f;     /* XXX */
585             /*
586              * We've got something that's not an IP packet.
587              * If compression is enabled, try to decompress it.
588              * Otherwise, if auto-enable compression is on and
589              * it's a reasonable packet, decompress it and then
590              * enable compression.  Otherwise, drop it.
591              */
592             if (sc->sc_if.if_flags & SC_COMPRESS) {
593                 len = sl_uncompress_tcp(&sc->sc_buf, len,
594                                         (u_int) c, &sc->sc_comp);
595                 if (len <= 0)
596                     goto error;
597             } else if ((sc->sc_if.if_flags & SC_AUTOCOMP) &&
598                        c == TYPE_UNCOMPRESSED_TCP && len >= 40) {
599                 len = sl_uncompress_tcp(&sc->sc_buf, len,
600                                         (u_int) c, &sc->sc_comp);
601                 if (len <= 0)
602                     goto error;
603                 sc->sc_if.if_flags |= SC_COMPRESS;
604             } else
```

图5-13 函数slinput：帧结束处理

```
605                 goto error;
606         }
607         if (sc->sc_bpf) {
608             /*
609              * Put the SLIP pseudo-"link header" in place.
610              * We couldn't do this any earlier since
611              * decompression probably moved the buffer
612              * pointer.  Then, invoke BPF.
613              */
614             u_char *hp = sc->sc_buf - SLIP_HDRLEN;

615             hp[SLX_DIR] = SLIPDIR_IN;
616             bcopy(chdr, &hp[SLX_CHDR], CHDR_LEN);
617             bpf_tap(sc->sc_bpf, hp, len + SLIP_HDRLEN);
618         }
619         m = sl_btom(sc, len);
620         if (m == NULL)
621             goto error;

622         sc->sc_if.if_ipackets++;
623         sc->sc_if.if_lastchange = time;
624         s = splimp();
625         if (IF_QFULL(&ipintrq)) {
626             IF_DROP(&ipintrq);
627             sc->sc_if.if_ierrors++;
628             sc->sc_if.if_iqdrops++;
629             m_freem(m);
630         } else {
631             IF_ENQUEUE(&ipintrq, m);
632             schednetisr(NETISR_IP);
633         }
634         splx(s);
635         goto newpack;
```

if_sl.c

图5-13 (续)

通过这个switch语句的普通控制流会落到switch外(这里没有default情况)。大多数字节是数据，并且不与这4种情况中的任何一种匹配。前两个case的控制也会落到这个switch外。

637-649 如果控制落到switch外，接收的字符为IP分组中的一部分。这个字符被存储到簇中(如果还有空间)，指针增加，sc_escape被清除，并且slinput返回。

如果簇满，字符被丢弃，并且slinput设置SC_ERROR。如果簇满或在处理帧结束时检测到一个差错，则控制跳到error。程序在newpack为一个新的分组重设簇指针，sc_escape被清除，并且slinput返回。

图5-13显示了图5-12中跳过的FRAME_END代码。

560-579 如果SC_ERROR被设置，同时正在接收分组或如果分组长度小于3字节(记住，分组可能被压缩)，则slinput立即丢弃此输入SLIP分组。

如果SLIP接口带有BPF，slinput在chdr数组中保存这个首部的一个备份(可能被压缩)。

580-606 通过检查分组的第一个字节，slinput判断它是一个未压缩的IP分组，还是一个压缩的TCP分段，或者一个未压缩的TCP分段。类型存放在c中，并且类型信息从数据的第一个字节中移去(29.13节)。如果分组以压缩形式出现，并且允许压缩，sl_uncompress_tcp对分组进行解压缩。如果禁止压缩，自动允许压缩被设置，并且如果分组足够大，则仍然调

用sl_uncompress_tcp。如果是一个压缩的TCP分组，则设置压缩标志。

若分组不被识别，slinput跳到error，丢弃此分组。29.13节详细讨论了首部压缩技术。现在簇中包含一个完整的未压缩分组。

607-618 SLIP解压缩分组后，首部和数据传给BPF。图5-14显示了slinput构造的缓存格式。

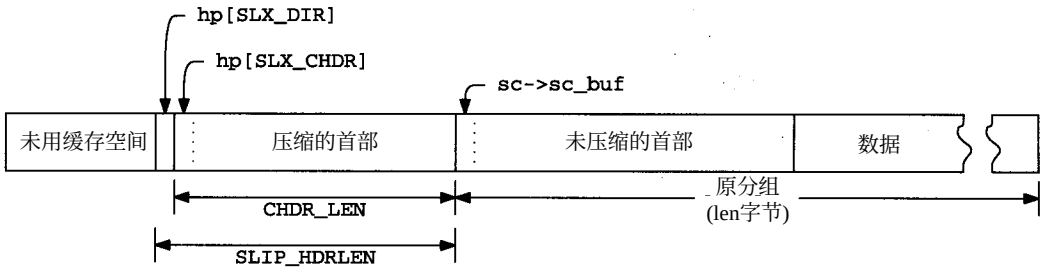

图5-14 BPF格式的SLIP分组

BPF首部的第一个字节是分组方向的编码，在此例中是输入(SLIPDIR_IN)。接下来的15字节包含压缩的首部。整个分组被传给bpf_tap。

619-635 sl_btom将簇转换为一个mbuf链表。如果分组足够小，能放到一个单独的mbuf中，sl_btom就将分组从簇复制到一个新分配的mbuf的分组首部；否则sl_btom将这个簇连接到一个mbuf，并为这个接口分配一个新簇。这样比从一个簇复制到另一个簇要快。我们在本书中不显示sl_btom的代码。

因为在SLIP接口上只能传输IP分组，slinput不必选择协议队列(如以太网驱动程序所做)。分组在ipintrq中排队，一个IP软件中断被调度，并且slinput跳到newpack，更新簇的分组指针，并清除sc_escape。

如果分组不能在ipintrq上排队，SLIP驱动程序增加if_ierrors，而在这种情况下，以太网或环回驱动程序都不增加这个统计量。

即使在spltty调用slinput，访问IP输入队列必须用splimp保护。回忆图1-14，一个splimp中断能抢占spltty进程。

5.3.4 SLIP输出处理：sloutput

如所有的网络接口，当一个网络层协议调用接口的if_output函数时，开始处理输出。对于以太网驱动程序，此函数是ether_output。而对于SLIP，此函数是sloutput(图5-15)。

259-289 sloutput的4个参数为：ifp，指向SLIP ifnet结构(在此例中是一个sl_softc结构)的指针；m，指向排队等待输出的分组的指针；dst，分组下一跳的目标地址；rtp，指向一个路由表项的指针。sloutput未用第4个参数，但却是要求的，因为sloutput必须匹配在ifnet结构中的if_output函数原型。

sloutput确认dst是一个IP地址，接口被连接到一个TTY设备，并且这个TTY设备是正在运行的(即有载波信号，或应忽略它)。如果任何检测失败，则返回差错。

290-291 SLIP为输出分组维护两个队列。默认选择标准队列if_snd。

292-295 如果输出分组包含一个ICMP报文，并且接口的SC_NOICMP被置位，则丢弃此分组。这防止一个SLIP链路被一个恶意用户发送的无关ICMP分组(例如ECHO分组)所淹没(第11章)。

if_sl.c

```
259 int
260 sloutput(ifp, m, dst, rtp)
261 struct ifnet *ifp;
262 struct mbuf *m;
263 struct sockaddr *dst;
264 struct rtentry *rtp;
265 {
266     struct sl_softc *sc = &sl_softc[ifp->if_unit];
267     struct ip *ip;
268     struct ifqueue *ifq;
269     int     s;
270     /*
271      * Cannot happen (see slioctl).  Someday we will extend
272      * the line protocol to support other address families.
273      */
274     if (dst->sa_family != AF_INET) {
275         printf("sl%d: af%d not supported\n", sc->sc_if.if_unit,
276                dst->sa_family);
277         m_freem(m);
278         sc->sc_if.if_noproto++;
279         return (EAFNOSUPPORT);
280     }
281     if (sc->sc_ttyp == NULL) {
282         m_freem(m);
283         return (ENETDOWN);      /* sort of */
284     }
285     if ((sc->sc_ttyp->t_state & TS_CARR_ON) == 0 &&
286         (sc->sc_ttyp->t_cflag & CLOCAL) == 0) {
287         m_freem(m);
288         return (EHOSTUNREACH);
289     }
290     ifq = &sc->sc_if.if_snd;
291     ip = mtod(m, struct ip *);
292     if (sc->sc_if.if_flags & SC_NOICMP && ip->ip_p == IPPROTO_ICMP) {
293         m_freem(m);
294         return (ENETRESET);     /* XXX ? */
295     }
296     if (ip->ip_tos & IPTOS_LOWDELAY)
297         ifq = &sc->sc_fastq;
298     s = splimp();
299     if (IF_QFULL(ifq)) {
300         IF_DROP(ifq);
301         m_freem(m);
302         splx(s);
303         sc->sc_if.if_oerrors++;
304         return (ENOBUFS);
305     }
306     IF_ENQUEUE(ifq, m);
307     sc->sc_if.if_lastchange = time;
308     if (sc->sc_ttyp->t_outq.c_cc == 0)
309         slstart(sc->sc_ttyp);
310     splx(s);
311     return (0);
312 }
```

if_sl.c

图5-15 函数sloutput

差错码ENETRESET指示分组因决策而被丢弃 (相对于网络故障)。我们在第11章会看到除

了在本地产生一个ICMP报文外，此差错简单地被忽略，在这种情况下，一个差错返回给发送此报文的进程。

Net/2在这种情况返回一个0。对于一个诊断工具，如ping或traceroute，会出现这种情况：好像这个分组消失了，因为输出操作会报告成功完成。

通常，ICMP报文可以被丢弃。对于正确的操作，它们并不必要，但丢弃它们会造成更多的麻烦，可能导致不佳的路由决定和较差的性能，并且会浪费网络资源。

296-297 如果在输出分组的TOS字段指明低时延服务(IPTOS_LOWDELAY)，则输出队列改为sc_fastq。

RFC 1700和RFC 1349 [Almquist 1992]规定了标准协议的TOS设置。为Telnet、Rlogin、FTP(控制)、TFTP、SMTP(命令阶段)和DNS(UDP查询)指明了低时延服务。更多细节见卷1的3.2节。

在以前的BSD版本中，ip_tos不由应用程序设置。SLIP驱动程序通过检查在IP分组中的传输首部来实现TOS排队。如果发现FTP(命令)、Telnet或Rlogin端口的TCP分组，分组就如指明了IPTOS_LOWDELAY一样被排队。很多路由器仍然这样，因为很多这些交互服务的实现仍然不设置ip_tos。

298-312 现在分组被放到所选择的队列中，接口统计被更新，并且 (如果TTY输出队列为空) sloutput调用slstart来发起对此分组的传输。

如果接口队列满，则SLIP增加if_oerrors；而对于ether_output，则不是这样做的。

不像以太网输出函数(ether_output)，sloutput不为输出分组构造一个数据链路首部。因为在SLIP网络上的另一系统在串行链路的另一端，所以不需要硬件地址或一个协议 (如ARP)在IP地址和硬件地址间进行转换。协议标识符 (如以太网类型字段)也是多余的，因为一个SLIP链路仅承载IP分组。

5.3.5 slstart函数

除了被sloutput调用外，当TTY取完它的输出队列并要传输更多的字节时， TTY设备调用slstart。TTY子系统通过一个 clist结构管理它的队列。在图 5-8中，输出 clist t_outq显示在slstart下面和设备的t_oproc函数的上面。slstart把字节添加到队列中，而t_oproc将队列取完并传输这些字节。

函数slstart显示在图5-16中。

318-358 当slstart函数被调用时，tp指向设备的tty结构。slstart的主体由一个for循环构成。如果输出队列t_outq不空，slstart调用设备的输出函数t_oproc，此函数传输设备所能接收的字节数。如果 TTY输出队列中剩余的字节超过 100字节(SLIP_HIWAT)，则slstart返回而不是将另一分组的字节添加到队列中。当传输完所有字节，输出设备产生一个中断，并且当输出列表为空时， TTY子系统调用slstart。

如果TTY输出队列为空，则一个分组从sc_fastq中退队，或者，若sc_fastq为空，则从if_snd队列中退队，这样在其他分组前传输所有交互的分组。

没有标准的SNMP变量来统计根据TOS字段排队的分组。在353行的XXX注释表示SLIP驱动程序在if_omcasts中统计低时延分组数，而不是多播分组数。

359-383 如果SLIP接口带有BPF，slstart在任何首部压缩前为输出分组产生一个备份。这个备份存储在bpfbuf数组的栈中。

384-388 如果允许压缩，并且分组包含一个TCP报文段，则sloutput调用sl_compress_tcp来压缩这个分组。得到的分组类型被返回，并与IP首部的第一个字节(29.13节)进行逻辑或运算。

389-398 压缩的首部现在复制到BPF首部，并且方向标记为SLIPDIR_OUT。完整的BPF分组传给bpf_tap。

483-484 如果for循环终止，则slstart返回。

if_sl.c

```
318 void
319 slstart(tp)
320 struct tty *tp;
321 {
322     struct sl_softc *sc = (struct sl_softc *) tp->t_sc;
323     struct mbuf *m;
324     u_char *cp;
325     struct ip *ip;
326     int     s;
327     struct mbuf *m2;
328     u_char  bpfbuf[SLMTU + SLIP_HDRLEN];
329     int     len;
330     extern int cfreecount;

331     for (;;) {
332         /*
333          * If there is more in the output queue, just send it now.
334          * We are being called in lieu of ttstart and must do what
335          * it would.
336          */
337         if (tp->t_outq.c_cc != 0) {
338             (*tp->t_oproc) (tp);
339             if (tp->t_outq.c_cc > SLIP_HIWAT)
340                 return;
341         }
342         /*
343          * This happens briefly when the line shuts down.
344          */
345         if (sc == NULL)
346             return;

347         /*
348          * Get a packet and send it to the interface.
349          */
350         s = splimp();
351         IF_DEQUEUE(&sc->sc_fastq, m);
352         if (m)
353             sc->sc_if.if_omcasts++;     /* XXX */
354         else
355             IF_DEQUEUE(&sc->sc_if.if_snd, m);
356         splx(s);
357         if (m == NULL)
358             return;
```

图5-16 函数slstart：分组退队

```
359         /*
360          * We do the header compression here rather than in sloutput
361          * because the packets will be out of order if we are using TOS
362          * queueing, and the connection id compression will get
363          * munged when this happens.
364          */
365         if (sc->sc_bpf) {
366             /*
367              * We need to save the TCP/IP header before it's
368              * compressed.  To avoid complicated code, we just
369              * copy the entire packet into a stack buffer (since
370              * this is a serial line, packets should be short
371              * and/or the copy should be negligible cost compared
372              * to the packet transmission time).
373              */
374             struct mbuf *m1 = m;
375             u_char *cp = bpfbuf + SLIP_HDRLEN;
376             len = 0;
377             do {
378                 int     mlen = m1->m_len;
379                 bcopy(mtod(m1, caddr_t), cp, mlen);
380                 cp += mlen;
381                 len += mlen;
382             } while (m1 = m1->m_next);
383         }
384         if ((ip = mtod(m, struct ip *))->ip_p == IPPROTO_TCP) {
385             if (sc->sc_if.if_flags & SC_COMPRESS)
386                 *mtod(m, u_char *) |= sl_compress_tcp(m, ip,
387                                                       &sc->sc_comp, 1);
388         }
389         if (sc->sc_bpf) {
390             /*
391              * Put the SLIP pseudo-"link header" in place.  The
392              * compressed header is now at the beginning of the
393              * mbuf.
394              */
395             bpfbuf[SLX_DIR] = SLIPDIR_OUT;
396             bpfbuf(mtod(m, caddr_t), &bpfbuf[SLX_CHDR], CHDR_LEN);
397             bpf_tap(sc->sc_bpf, bpfbuf, len + SLIP_HDRLEN);
398         }

                        /* packet output code */

483     }
484 }
```

if_sl.c

图5-16 (续)

slstart的下一部分(图5-17)在系统存储器容量不足时丢弃分组，并且采用一种简单的技术来丢弃由于串行线上的噪声产生的数据。这些代码在图 5-16中忽略了。

399-409 如果系统缺少clist结构，则分组被丢弃，并且作为一个冲突被统计。通过不断地循环而不是返回，slstart快速地丢弃所有剩余的排队输出的分组。由于设备仍然有太多字节为输出排队，每次迭代都要丢弃一个分组。高层协议必须检测丢失的分组并重传它们。

410-418 如果TTY输出队列为空，则通信线路可能有一段时间空闲，并且接收方在另一端

可能接收了线路噪声产生的无关数据。 slstart在输出队列中放置一个额外的 SLIP END字符。一个长度为0的帧或一个由线路噪声产生的帧应该被接收方 SLIP接口或IP协议丢弃。

if_sl.c

```
399         sc->sc_if.if_lastchange = time;
400         /*
401          * If system is getting low on clists, just flush our
402          * output queue (if the stuff was important, it'll get
403          * retransmitted).
404          */
405         if (cfreecount < CLISTRESERVE + SLMTU) {
406             m_freem(m);
407             sc->sc_if.if_collisions++;
408             continue;
409         }
410         /*
411          * The extra FRAME_END will start up a new packet, and thus
412          * will flush any accumulated garbage.  We do this whenever
413          * the line may have been idle for some time.
414          */
415         if (tp->t_outq.c_cc == 0) {
416             ++sc->sc_if.if_obytes;
417             (void) putc(FRAME_END, &tp->t_outq);
418         }
```

if_sl.c

图5-17 函数slstart ：资源缺乏和线路噪声

图5-18说明了这个丢弃线路噪声的技术，它来源于由 Phil Karn撰写的RFC 1055。在图5-18中，传输第二个帧结束符 (END)，因为线路空闲了一段时间。由噪声产生的无效帧和这个END字节被接收系统丢弃。

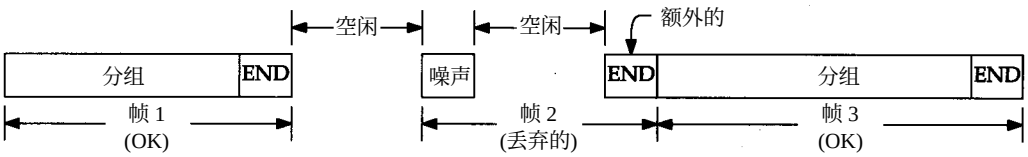

图5-18 Karn的丢弃SLIP线路噪声的方法

在图5-19中，线路上没有噪声并且0长度帧被接收系统丢弃。

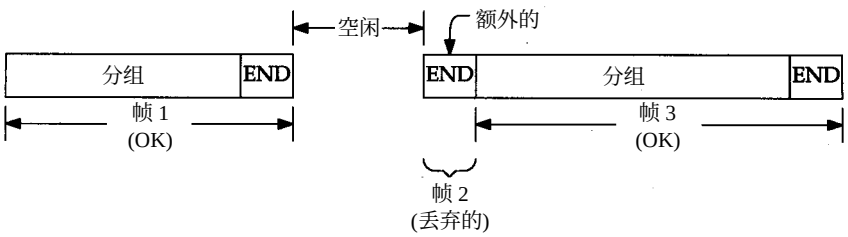

图5-19 无噪声的Karn方法

slstart的下一部分(图5-20)将数据从一个mbuf传给TTY设备的输出队列。

419-467 在这部分的外部while循环对链表中的每个mbuf执行一次。中间的while循环将数据从每个mbuf传给输出设备。内部的while循环不断递增cp，直到它找到一个END或ESC字符。b_to_q传输bp到cp之间的数据。END和ESC字符被转义，并且两次通过调用putc放

入队列。中间的循环直到 mbuf的所有字节都传给 TTY设备输出队列才停止。图 5-21说明了对包含了一个SLIP END字符和一个SLIP ESC字符的mbuf的处理。

bp标记用b_to_q传输的mbuf的第一部分的开始， cp标记这个部分的结束。 ep标记这个mbuf中数据的结束位置。

if_sl.c

```
419         while (m) {
420             u_char *ep;

421             cp = mtod(m, u_char *);
422             ep = cp + m->m_len;
423             while (cp < ep) {
424                 /*
425                  * Find out how many bytes in the string we can
426                  * handle without doing something special.
427                  */
428                 u_char *bp = cp;

429                 while (cp < ep) {
430                     switch (*cp++) {
431                     case FRAME_ESCAPE:
432                     case FRAME_END:
433                         --cp;
434                         goto out;
435                     }
436                 }
437               out:
438                 if (cp > bp) {
439                     /*
440                      * Put n characters at once
441                      * into the tty output queue.
442                      */
443                     if (b_to_q((char *) bp, cp - bp,
444                                &tp->t_outq))
445                         break;
446                     sc->sc_if.if_obytes += cp - bp;
447                 }
448                 /*
449                  * If there are characters left in the mbuf,
450                  * the first one must be special..
451                  * Put it out in a different form.
452                  */
453                 if (cp < ep) {
454                     if (putc(FRAME_ESCAPE, &tp->t_outq))
455                         break;
456                     if (putc(*cp++ == FRAME_ESCAPE ?
457                              TRANS_FRAME_ESCAPE : TRANS_FRAME_END,
458                              &tp->t_outq)) {
459                         (void) unputc(&tp->t_outq);
460                         break;
461                     }
462                     sc->sc_if.if_obytes += 2;
463                 }
464             }
465             MFREE(m, m2);
466             m = m2;
467         }
```

if_sl.c

图5-20 函数slstart ：传输分组

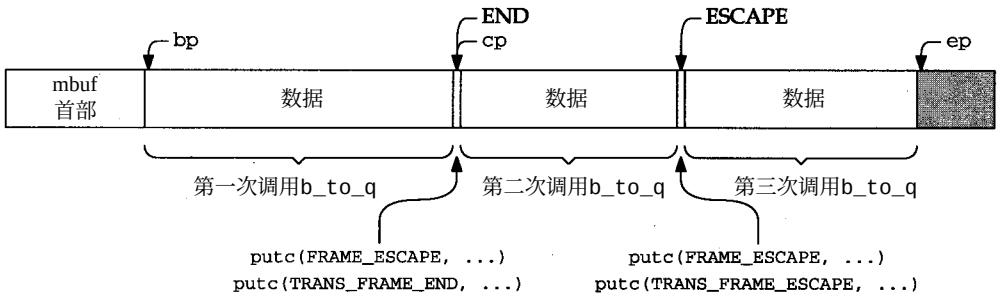

图5-21 单个mbuf的SLIP传输

如果b_to_q或putc失败(即，数据不能在TTY设备排队)，则break导致slstart退出内部while循环。这种失败表示内核 clist资源用完。在每个 mbuf被复制到TTY设备后，或者当一个差错发生时， mbuf被释放， m增加到链表的下一个 mbuf，并且外部while循环继续执行直到链表中所有mbuf被处理。

图5-22显示了slstart完成输出帧的处理。

if_sl.c

```
468         if (putc(FRAME_END, &tp->t_outq)) {
469             /*
470              * Not enough room.  Remove a char to make room
471              * and end the packet normally.
472              * If you get many collisions (more than one or two
473              * a day) you probably do not have enough clists
474              * and you should increase "nclist" in param.c.
475              */
476             (void) unputc(&tp->t_outq);
477             (void) putc(FRAME_END, &tp->t_outq);
478             sc->sc_if.if_collisions++;
479         } else {
480             ++sc->sc_if.if_obytes;
481             sc->sc_if.if_opackets++;
482         }
```

if_sl.c

图5-22 函数slstart ：帧结束处理

468-482 当外部while循环处理完对输出队列中的字节排队时，控制到达这段代码。驱动程序发送一个SLIP END字符，来终止这个帧。

如果这些字节在排队时发生差错，则输出帧无效，并会因为“无效的检验和”或“无效的长度”被接收系统检测出来。

无论这个帧是不是因为一个差错而终止，如果 END字符没有填充到输出队列中，队列的最后一个字符就要被丢弃，并且 slstart将使这个帧结束。这保证传输了一个 END字符。这个无效帧在目标站被丢弃。

5.3.6 SLIP分组丢失

SLIP接口提供了一个尽最大努力服务的好例子。如果 TTY超载，则SLIP丢弃分组；在分组开始传输后，如果资源不可用，则它截断分组，并且为了检测和丢弃线路噪声插入无关的空分组。对以上的每一种情况都不产生差错报文。 SLIP依靠IP层和运输层来检测损坏的和丢失的分组。

在一个路由器上从一个高速接口例如以太网，发送帧到一个低速的 SLIP线路上。如果发送方不能意识到瓶颈并相应调节数据速率，则会有大比例的分组被丢弃。在 25.11节我们会看到TCP是如何检测并对此响应的。应用程序使用一个无流量控制的协议，如 UDP，必须自己识别和响应这种情况(习题5.8)。

5.3.7 SLIP性能考虑

一个SLIP帧的MTU(SLMTU)、clist高水位标记(high-water mark)(SLIP_HIWAT)和SLIP的TOS排队策略都是用来设计交互通信的低速串行链，使得固有的时延最小。

1) 一个小的MTU能够改进交互数据的时延 (如敲键和回显)，但有损批量数据传输的吞吐量。一个大的MTU能改进批量数据的吞吐量，但增加了交互时延。 SLIP链路的另一个问题是键入一个字符就要有 40字节的开销来写入 TCP首部和IP首部的信息，这就增加了通信的时延。

   解决办法是挑选一个足够大的 MTU来提供好的交互响应时间和适当的批量数据吞吐量，并压缩TCP/IP首部来减小每个分组的负荷。 RFC 1144 [Jacobson 1990a]描述了一个压缩方案和时间计算，它为一个典型的 9600 b/s异步SLIP链路选择了一个数值为 296的MTU。我们在29.13节讨论压缩的SLIP(CSLIP)。卷1的2.10节和7.2节总结了这种定时考虑，并说明了在SLIP链路上的时延。

2) 如果有太多的字节缓存在clist中(因为SLIP_HIWAT设置得太高)，TOS排队会受到阻碍，因为新的交互式通信等在大量缓存数据的后面。如果 SLIP一次传给TTY驱动程序一个字节(因为SLIP_HIWAT设置得太低)，设备为每个字节调用 slstart，并在每个字节传输后线路空闲一段时间。把SLIP_HIWAT设置为100可使在设备排队的数据量最小化，并且减小了TTY子系统调用slstart的频率，大约每100字符必须调用slstart一次。

3) 如前所述， SLIP驱动程序提供了TOS排队，其策略是先从 sc_fastq队列中发送交互式通信数据，然后在标准接口队列 if_snd中发送其他的通信数据。

5.3.8 slclose函数

为了完整性，我们显示函数 slclose。当slattach程序关闭SLIP的TTY设备，并且中断对远程系统的连接时，调用它。

if_sl.c

```
210 void
211 slclose(tp)
212 struct tty *tp;
213 {
214     struct sl_softc *sc;
215     int     s;

216     ttywflush(tp);
217     s = splimp();               /* actually, max(spltty, splnet) */
218     tp->t_line = 0;
219     sc = (struct sl_softc *) tp->t_sc;
220     if (sc != NULL) {
221         if_down(&sc->sc_if);
222         sc->sc_ttyp = NULL;
223         tp->t_sc = NULL;
```

图5-23 函数slclose

```
224             MCLFREE((caddr_t) (sc->sc_ep - SLBUFSIZE));
225             sc->sc_ep = 0;
226             sc->sc_mp = 0;
227             sc->sc_buf = 0;
228         }
229         splx(s);
230 }
```

if_sl.c

图5-23 (续)

210-230 tp指向要关闭的TTY设备。slclose清除任何残留在串行设备中的数据，中断TTY和网络处理，并且将TTY复位到默认的线路规程。如果TTY设备被连接到一个SLIP接口，则关闭这个接口，在这两个结构间的链接被切断，与此接口关联的mbuf簇被释放，并且指向现在被丢弃的簇的指针被复位。最后，splx重新允许TTY中断和网络中断。

5.3.9 sltioctl函数

回忆一下，SLIP在内核中有两种作用：

- 作为一个网络接口；
- 作为一个TTY线路规程。

图5-7显示了slioctl处理通过一个插口描述符发送给一个SLIP接口的ioctl命令。在4.4节中，我们显示了ifioctl是如何调用slioctl的。我们会看到一个处理ioctl命令的相似模型，并且在后面的章节中会讨论到。

图5-7还表示了sltioctl处理发送给与一个SLIP网络接口关联的TTY设备的ioctl命令。这个被sltioctl识别的命令显示在图5-24中。

命 令	参 数	函 数	说 明
SLIOCGUNIT	int *	sltioctl	返回与TTY设备关联的接口联合

图5-24 sltioctl 命令

函数sltioctl显示在图5-25中。

```
236 int
237 sltioctl(tp, cmd, data, flag)
238 struct tty *tp;
239 int      cmd;
240 caddr_t data;
241 int      flag;
242 {
243     struct sl_softc *sc = (struct sl_softc *) tp->t_sc;

244     switch (cmd) {
245     case SLIOCGUNIT:
246         *(int *) data = sc->sc_if.if_unit;
247         break;

248     default:
249         return (-1);
250     }
251     return (0);
252 }
```

if_sl.c

图5-25 函数sltioctl

236-252 tty结构的t_sc指针指向关联的sl_softc结构。这个SLIP接口的设备号从if_unit被复制到*data，它最后返回给进程(17.5节)。

当系统被初始化时，slattach初始化if_unit，并且当slattach程序为此TTY设备选择SLIP线路规程时，slopen初始化t_sc。因为一个TTY设备和一个SLIP sl_softc结构间的关系是在运行时建立的，一个进程能通过SLIOCGUNIT命令发现所选择的接口结构。

5.4 环回接口

任何发送给环回接口(图5-26)的分组立即排入输入队列。接口完全用软件实现。

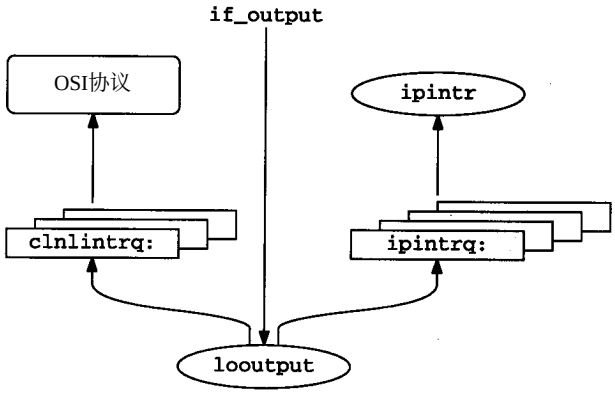

图5-26 环回设备驱动程序

环回接口的if_output指向的函数looutput，将输出分组放置到分组的目的地址指明的协议的输入队列中。

我们已经看到当设备被设置为IFF_SIMPLEX时，ether_output会调用looutput来排队一个输出广播分组。在第 12章中，我们会看到多播分组也可能以这种方式环回。looutput显示在图5-27中。

if_loop.c

```
57 int
58 looutput(ifp, m, dst, rt)
59 struct ifnet *ifp;
60 struct mbuf *m;
61 struct sockaddr *dst;
62 struct rtentry *rt;
63 {
64     int     s, isr;
65     struct ifqueue *ifq = 0;

66     if ((m->m_flags & M_PKTHDR) == 0)
67         panic("looutput no HDR");
68     ifp->if_lastchange = time;
69     if (loif.if_bpf) {
70         /*
71          * We need to prepend the address family as
72          * a four byte field.  Cons up a dummy header
73          * to pacify bpf.  This is safe because bpf
74          * will only read from the mbuf (i.e., it won't
75          * try to free it or keep a pointer a to it).
76          */
```

图5-27 函数looutput

```
 77          struct mbuf m0;
 78          u_int   af = dst->sa_family;

 79          m0.m_next = m;
 80          m0.m_len = 4;
 81          m0.m_data = (char *) &af;

 82          bpf_mtap(loif.if_bpf, &m0);
 83      }
 84      m->m_pkthdr.rcvif = ifp;

 85      if (rt && rt->rt_flags & (RTF_REJECT | RTF_BLACKHOLE)) {
 86          m_freem(m);
 87          return (rt->rt_flags & RTF_BLACKHOLE ? 0 :
 88                  rt->rt_flags & RTF_HOST ? EHOSTUNREACH : ENETUNREACH);
 89      }
 90      ifp->if_opackets++;
 91      ifp->if_obytes += m->m_pkthdr.len;
 92      switch (dst->sa_family) {
 93      case AF_INET:
 94          ifq = &ipintrq;
 95          isr = NETISR_IP;
 96          break;

 97      case AF_ISO:
 98          ifq = &clnlintrq;
 99          isr = NETISR_ISO;
100          break;

101     default:
102         printf("lo%d: can't handle af%d\n", ifp->if_unit,
103                dst->sa_family);
104         m_freem(m);
105         return (EAFNOSUPPORT);
106     }
107     s = splimp();
108     if (IF_QFULL(ifq)) {
109         IF_DROP(ifq);
110         m_freem(m);
111         splx(s);
112         return (ENOBUFS);
113     }
114     IF_ENQUEUE(ifq, m);
115     schednetisr(isr);
116     ifp->if_ipackets++;
117     ifp->if_ibytes += m->m_pkthdr.len;
118     splx(s);
119     return (0);
120 }
```

if_loop.c

图5-27 (续)

57-66 looutput的参数同ether_output一样，因为都是通过它们的ifnet结构中的if_output指针直接调用的。ifp，指向输出接口的ifnet结构的指针；m，要发送的分组；dst，分组的目的地址；rt，路由信息。如果链表中的第一个mbuf不包含一个分组，looutput调用panic。

图5-28所示的是一个BPF环回分组的逻辑格式。

69-83 驱动程序在堆栈上的m0中构造BPF环回分组，并且把m0连接到包含原始分组的mbuf

链表中。注意m0的声明不同往常。它是一个mbuf，而不是一个mbuf指针。m0的m_data指向af，它也分配在这个堆栈中。图5-29显示了这种安排。

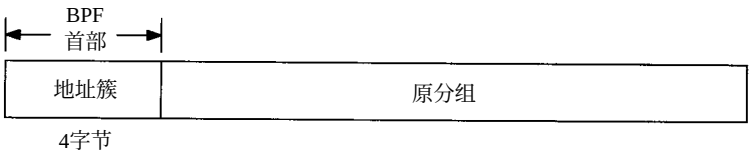

图5-28 BPF环回分组：逻辑格式

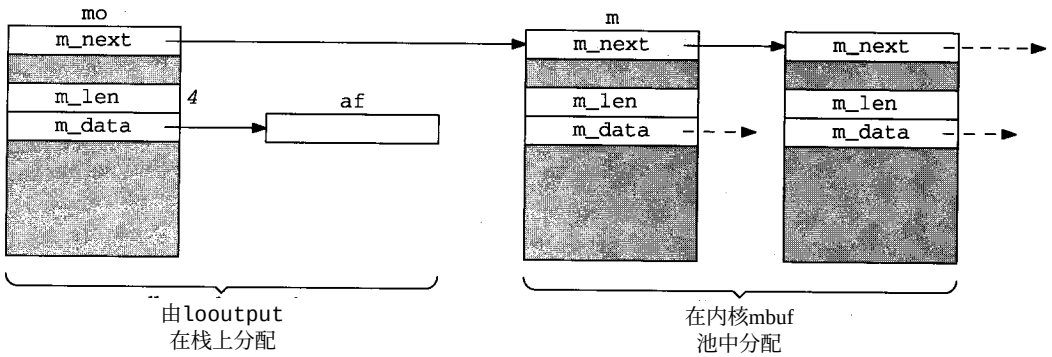

图5-29 BPF环回分组：mbuf格式

looutput将目的地址族复制到af，并且将新mbuf链表传递给bpf_mtap，去处理这个分组。与bpf_tap相比，它在一个单独的连续缓存中接收这个分组而不是在一个 mbuf链表中。如图中注释所示，BPF从来不释放一个链表中的mbuf，因此将m0 (它指向栈中的一个mbuf)传给bpf_mtap是安全的。

84-89 looutput剩下的代码包含input对此分组的处理。虽然这是一个输出函数，但分组被环回到输入。首先，m->m_pkthdr.rcvif设置为指向接收接口。如果调用方提供一个路由项，looutput检查是否它指示此分组应该被拒绝 (RTF_REJECT)或直接被丢弃(RTF_BLACKHOLE)。通过丢弃mbuf并返回0来实现一个黑洞。从调用者看来就好像分组已经被传输了。要拒绝一个分组，如果路由是一个主机，则 looutput返回EHOSTUNREACH；如果路由是一个网络则返回ENETUNREACH。

各种RTF_*xxx*标志在图18-25中描述。

90-120 然后looutput通过检查分组目的地址中的sa_family来选择合适的协议输入队列和软件中断。接着把识别的分组进行排队，并用 schednetisr来调度一个软件中断。

5.5 小结

我们讨论了两个剩下的接口，它们在书中多次引用：sl0，一个SLIP接口；lo0，标准的环回接口。

我们显示了在SLIP接口和SLIP线路规程之间的关系，讨论了 SLIP封装方法，并且讨论了TOS处理交互式通信和SLIP驱动程序的其他性能考虑。

我们显示了环回接口是如何按目的地址分用输出分组及将分组放到相应的输入队列中去。

习题

5.1 为什么环回接口没有输入函数？

5.2 你认为为什么图5-27中的m0要分配在堆栈中？

5.3 分析一个19 200 b/s的串行线的SLIP特性。对于这个线路，SLIP MTU应该改变吗？

5.4 导出一个根据串行线速率选择SLIP MTU的公式。

5.5 如果一个分组对于SLIP输入缓存太大，会发生什么情况？

5.6 一个`slinput`的早期版本，当一个分组在输入缓存溢出时，不将`SC_ERROR`置位。在这种情况下如何检测这种差错？

5.7 在图4-31中`le`被下标为`ifp->if_unit`的`le_softc`数组项初始化。你能想出另一种初始化`le`的方法吗？

5.8 当分组因为网络瓶颈被丢弃时，一个UDP应用程序如何知道？