附录A　部分习题的解答

第1章

1.2　SLIP驱动程序执行 `spltty`(图1-13)，其优先级必须低于或等于 `splimp`，且高于 `splnet`。因此，SLIP驱动程序由于中断而被阻塞。

第2章

2.1　`M_EXT`标志是mbuf自身的一个属性，而不是mbuf中保存的数据报的属性。

2.2　调用者请求大于100字节(`MHLEN`)的连续空间。

2.3　不可行，因为多个mbuf都可指向簇(2.9节)。此外，簇中也没有用于后向指针的空间(习题2.4)。

2.4　在`<sys/mbuf.h>`定义的宏`MCLALLOC`和`MCLFREE`中，我们看到引用计数器是一个名为`mclrefcnt`的数组。它在内核初始化时被分配，代码文件为 `machdep.c`。

第3章

3.3　采用很大的交互式队列，并不符合建立队列的目的，新的交互式流量跟在原有流量之后，会造成附加时延。

3.4　因为`sl_softc`结构都是全局变量，内核初始化时都被置为 0。

3.5

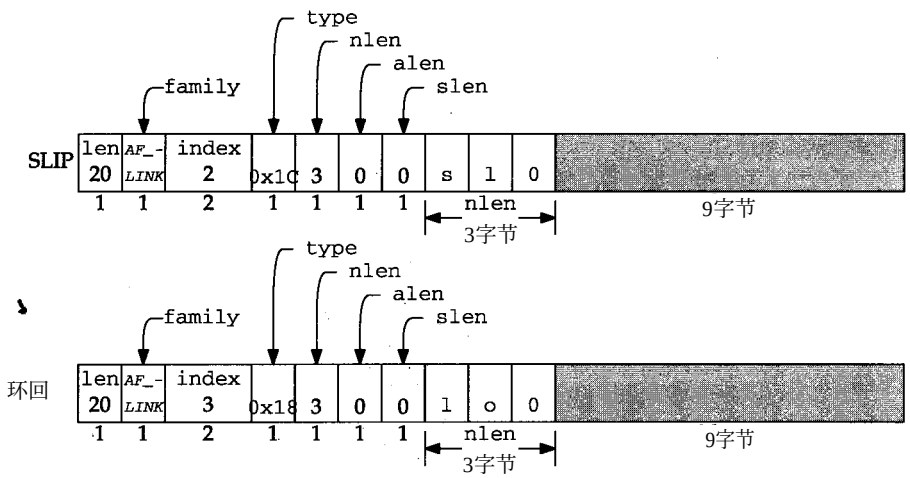

图　A-1

第4章

4.1　`leread`必须查看数据报，确认把数据报提交给 BPF之后，是否需要将其丢弃。因为

BPF开关会造成接口处于一种混杂模式，数据报的目的地有可能是以太网中的其他主机，BPF处理完毕后，必须将其丢弃。

如果接口没有加开关，则必须在ether_input中完成这一测试。

4.2 如果测试反过来，广播标志永远不会置位。

如果第二个if前没有else，所有广播分组都会带上多播标志。

第5章

5.1 环回接口不需要输入函数，因为它接收的所有分组都直接来自于 looutput，后者实际完成了输入功能。

5.2 堆栈分配快于动态存储器分配。对 BPF处理，性能是首要考虑的因素，因为对每个进入数据报都会执行该代码。

5.5 缓存溢出的第一个字节被丢弃， SC_ERROR置位， slinput重设簇指针，从缓存起始处开始收集字符。因为SC_ERROR置位， slinput收到SLIP END字符后，丢弃当前接收的数据帧。

5.6 如果检验和无效或者IP首部长度与实际数据报长度不匹配，数据报被丢弃。

5.7 因为ifp指向le_softc结构的第一个成员，

```
sc = (struct le_softc*) ifp;
```

sc初始化正确。

5.8 这是非常困难的。某些路由器在开始丢弃数据报时，可能会发送 ICMP源站抑制报文段，但Net/3实现中的UDP插口丢弃这些报文段(图23-30)。应用程序可以使用与TCP所采用的相同技术：根据确认的数据报估算往返时间，确认可用的带宽和时延。

第6章

6.1 IP子网出现之前(RFC 950 [Mogul和Postel 1985])，IP地址的网络和主机部分都以字节为界。in_addr结构的定义如下：

```
struct in_addr {
        union {
                struct { u_char s_b1, s_b2, s_b3, s_b4; } S_un_b;
                struct { u_short s_w1, s_w2; } S_un_w;
                u_long S_addr;
        } S_un;
#define s_addr  S_un.S_addr         /* should be used for all code */
#define s_host  S_un.S_un_b.s_b2    /* OBSOLETE: host on imp */
#define s_net   S_un.S_un_b.s_b1    /* OBSOLETE: network */
#define s_imp   S_un.S_un_w.s_w2    /* OBSOLETE: imp */
#define s_impno S_un.S_un_b.s_b4    /* OBSOLETE: imp # */
#define s_lh    S_un.S_un_b.s_b3    /* OBSOLETE: logical host */
};
```

图 A-2

Internet地址读写单位既可以是 8 bit字节，也可以是 16 bit单字，或32 bit双字。宏s_host、s_net、s_imp等等，它们的名字明确地反应出早期TCP/IP网络的结构。子网和超网概念的引入，淘汰了这种以字节和单字区分的做法。

6.2 返回指向结构sl_softc[0]的指针。

6.3 接口输出函数，如ether_output，只有一个指向接口ifnet结构的指针，而没有指向ifaddr的指针。在arpcom结构(最后一次为接口设定的IP地址)中使用IP地址可以避免从ifaddr地址链表中寻找所需地址。

6.4 只有超级用户进程才能创建原始IP插口。通过UDP插口，任何用户进程能够查看接口配置，但内核仍拥有超级用户特权，能够修改接口地址。

6.5 有3个函数循环处理网络掩码，一次处理一个字节。它们是ifa_ifwithnet、ifaof_ifpforaddr和rt_maskedcopy。较短的网络掩码能够提高这些函数的性能。

6.6 与远端系统建立Telnet连接。Net/2系统不应该转交这些数据报，而其他系统不会接受到达环回接口之外的非环回接口的环回数据报。

第7章

7.1 下列调用返回指向inetsw[6]的指针：

```
pffindproto(PF_INET, 0, SOCK_RAW);
```

第8章

8.1 可能不会。系统不可能响应任意的广播报文，因为没有可供响应的源地址。

8.4 因为数据报已经损坏，无法知道首部中的地址是否正确。

8.5 如果应用程序选取的源地址与指定的外出接口的地址不同，则无法发送到下一跳路由器。如果下一跳路由器发现数据报源地址与其到达的子网地址不符，则不会执行下一步的转发操作。这是尽量减少终端系统复杂性带来的后果，RFC 1122指出了这一问题。

8.6 新主机认为广播报文来自于某个没有划分子网的网络中的主机，并试图将数据报发回给源主机。网络接口开始广播ARP请求，向网络请求该广播地址，当然，这一请求永远不会收到响应。

8.7 减少TTL的操作出现在小于等于1的测试之后，是为了避免收到的TTL等于0，减1后将等于255，从而引起操作差错。

8.8 如果两个路由器彼此认为对方是某个数据报的下一跳路由器，则形成环路。除非该环路被打破，原始数据报在两个路由器间来回传递，并且每个路由器都向源主机发送ICMP重定向报文段，如果该主机与路由器处于同一个网络中。路由更新时，不同路由器中的路由表暂时存在的不一致现象，会造成这种环路。

原始报文段的TTL最终减为0，数据报被丢弃。这是TTL存在的一个主要原因。

8.9 不会检查4个以太网广播地址，因为它们不属于接收接口。但应检查有限的广播地址，说明带有SLIP链路的系统采用有限的广播地址，即使不知道对端地址，也能与对端通信。

8.10 只对数据报的第一个分片(分片偏移量等于0)生成ICMP差错报文。无论是主机字节序，还是网络字节序，0的表示都相同，因此无需转换。

第9章

9.1 RFC 1122建议如果数据报中的选项彼此冲突，处理方式由各实现代码自己决定。Net/3能正确处理第一个源路由选项，但因为它会更新数据报首部的ip_dst，第二条源路由处理将出现差错。

9.2 网络中的主机也可以用做到达网络其他主机的中继。如果目的主机不可直接到达，

源主机可在数据报中加入路由，首先到达中继主机，接着到达最终的目的主机。路由器不会丢弃数据报，因为目的地址指向中继主机，后者将处理路由并把数据报转发给最终目的主机。目的主机把路由反转，同样利用中继主机转发响应。

9.3 采用与前一个习题同样的原则。我们选取一个能够同时与源主机和目的主机通信的中继路由器，并构造源路由，穿过中继路由器到达目的地址。中继路由器必须与目的地址处于同一个网络，通信中无需默认路由。

9.4 如果源路由是仅有的IP选项，NOP选项使得所有IP地址以4字节边界对齐，从而能够优化存储器中的地址读取操作。这种对齐技术也适用于多个 IP选项，如果每个IP选项都通过NOP填充，保证按4字边界对齐。

9.5 不应混淆非标准时间值和标准时间值，最大的标准时间值等于86 399 399(24×60×60×1000 - 1)，需要28 bit才能表示。由于时间值有32 bit，从而避免了高位比特的混淆问题。

9.6 源路由选项代码在处理过程中可能会改变 `ip_dst`。保存目的地址，从保证时间戳处理使用原始目的地址。

第10章

10.2 重装后，只有第一个分片的选项上交给运输层协议。

10.3 因为数据长度(204 + 20)大于208(图2-16)。

图10-11中的`m_pullup`把头40字节复制到一个单独的mbuf中，如图2-18所示。

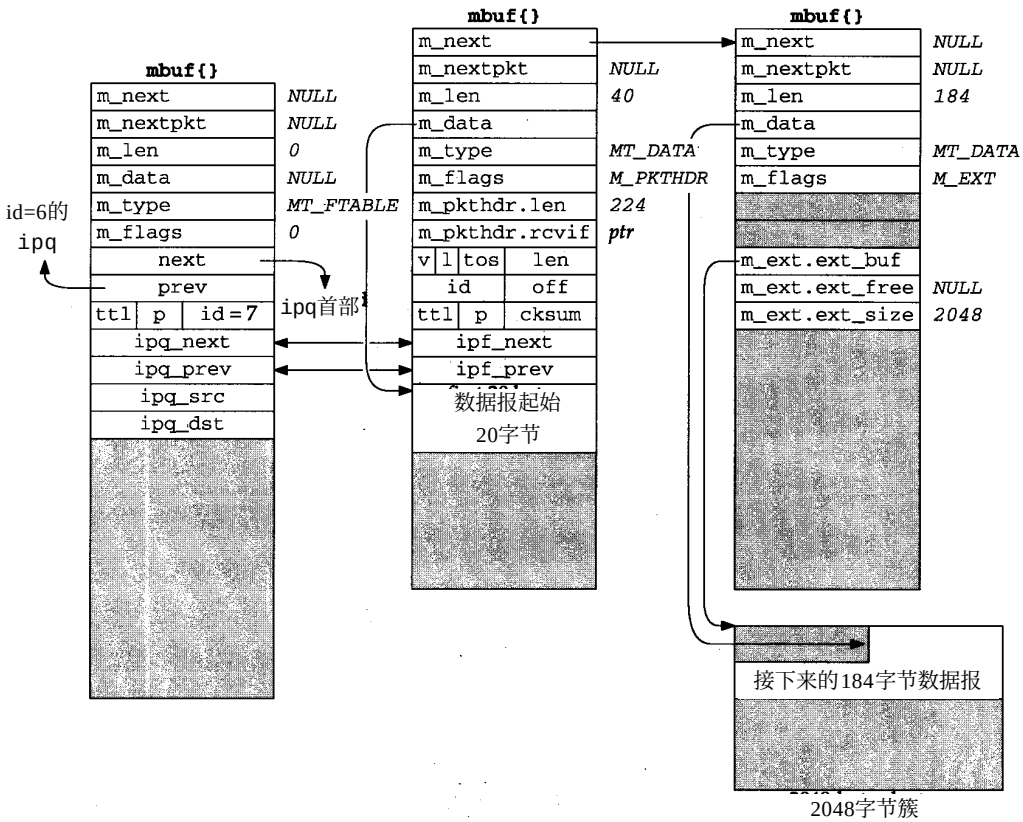

图 A-3

10.5 平均每个数据报收到的分片数等于

$$\frac{72786-349}{16\ 557} = 4.4$$

平均每个输出数据报新建的平均分片数等于

$$\frac{796\ 084}{2\ 6\ 0484} = 3.1$$

10.6 图10-11中，数据报最初被做为分片处理。当ip_off左移时，保留的比特位被丢弃。得到的数据报被视为分片或一个完整的数据报，取决于 MF和分片偏移量的值。

第11章

11.1 输出响应使用收到请求的接口的源地址。主机可能无法辩识 0.0.0.0是一个有效的广播地址，因此，有可能忽略请求。推荐的广播地址等于 255.255.255.255。

11.2 假定主机发送了一个链路层的广播数据报，其源 IP地址是另一台主机的地址，且数据报有差错，如内容差错的选项。所有主机都能接收并检测出差错，因为这是一个链路层的广播报文，而且选项的处理先于最终目的地的检测。许多发现差错的主机会向数据报的源IP地址发送ICMP报文，即使原数据报属于链路层广播。另一台主机将收到大量假的ICMP差错。这就是为什么不允许为链路层广播而发送 ICMP差错报文。

11.3 第一个例子中，这种重定向报文不会诱骗主机向另一个子网中的某个主机发送报文段。这台主机可能被误认为是路由器，但它确实记录收到的流量。 RFC 1009规定路由器只能向位于同一个子网的其他路由器发送重定向报文。即使主机忽略了这些要求把数据报转发到另一个子网的报文段，但如果报文段发送者与主机处于同一个子网中，它们就会被接受。第二个例子，为了防止出现上述现象，要求主机只接受它(错误地)选定的原始路由器的重定向报文，即假定这个错误的路由器是管理员指定的默认路由器。

11.4 通过向rip_input传递报文段，进程级的守护程序能够正确响应，一些依赖于这种行为的老系统能够继续得到支持。

11.5 ICMP差错只针对IP数据报的第一个分片。因为第一个分片的偏移量值必等于 0，字段的字节表示顺序是无关紧要的。

11.6 如果收到ICMP请求的接口还未配置IP地址，则ia将为空，且不生成响应。

11.7 Net/3处理与时间戳响应一起到达的数据。

11.10 高位比特被保留，并必须设为0。如果它必须被发送，则icmp_error将丢弃数据报。

11.11 返回值被丢弃，因为 icmp_send不返回差错。更重要的是， ICMP报文处理过程中生成的差错将被丢弃，以避免进入死循环，不断生成差错报文。

第12章

12.1 以太网中， IP广播地址255.255.255.255转换为以太网的广播地址 ff:ff:ff:ff:ff:ff，网络中的所有以太网接口都会接收这样的数据帧。没有运行 IP软件的系统必须主动接

收并丢弃这种广播报文。

数据报需发送给多播组 224.0.0.1中的所有主机，转换后的以太网多播地址为01:00:5e:00:00:01，只有明确要求其接口接收IP多播报文的系统才会收到它。没有运行IP或者在链路层不兼容的系统不会收到这些报文段，因为以太网接口的硬件已直接丢弃了这些报文段。

12.2 一种替代方案是通过文本名规定接口，如同ifreq结构和ioctl命令存取接口信息采取的方式一样。ip_setmoptions和ip_getmoptions可以调用ifunit，取代INADDR_TO_IFP，寻找指向接口ifnet结构的指针。

12.3 多播组高位4 bit通常为1110，因此，只有5个有意义的比特被匹配函数丢弃。

12.4 完整的ip_moptions结构必须能放入单个mbuf中，从而限制结构最大只能等于108字节(记住20字节的mbuf首部)。IP_MAX_MEMBERSHIPS可以大一些，但必须小于等于25(4+1+1+2+(4×25)=108)。

12.5 数据报重复，在IP输入队列中有两份复制的数据报。多播应用程序必须能识别并丢弃重复的数据报。

12.6

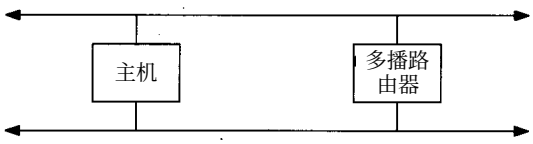

图 A-4

12.8 应用进程可以创建第二个插口，并通过第二个插口请求IP_MAX_MEMBERSHIPS。

12.9 为mbuf首部的m_flags成员变量定义一个新的mbuf标志M_LOCAL。ip_output处理环回数据报时置位该标志，从而取代检验和。如果该标置位，ipintr就跳过检验和验证。SunOS 5.X提供完成此功能的选项(ip_local_cksum，卷1的531页)。

12.10 存在$2^{23}-1$(8 388 607)个独立的以太网IP多播地址。记住保留的IP组224.0.0.0。

12.11 这个假设正确，因为in_addmulti拒绝所有新增请求，如果接口没有调用ioctl函数，说明如果if_ioctl为空，则永不会调用in_delmulti。

12.12 mbuf永远不会被释放，说明ip_getmoptions包含了一个存储器泄露。ip_getmoptions由ip_ctloutput调用，调用语句如下：

```
ip_getmoptions(IP_ADD_MEMBERSHIP, 0, mp)
```

会引发ip_getmoptions中的一个差错。

第13章

13.1 要求环回接口响应ICMP请求是没有必要的，因为本地主机是环回网络中唯一的系统，它已经知道自己的成员状态。

13.2 max_linkhdr + sizeof(struct ip) + IGMP_MINLEN=16+20+8=44<100

13.3 报告成员状态时出现随机延迟的主要原因是为了最大限度地减少出现在多播网络中的报告数(理想情况下应等于1)。一个点到点网络只包括两个接口，因此，无需延迟

以减少响应的数量。一个接口(假定是一个多播路由器)发出请求，另一个接口响应。另一个原因是避免过多的成员状态报告淹没接口的输出队列。大量 IGMP成员状态报文可能会超出输出队列关于数据报和字节的限制。例如，在 SLIP驱动程序中，如果输出队列已满或设备过忙，就会丢弃队列中所有等待的数据报 (图5-16)。

第14章

14.1 5个，分别对应网络A~E。

14.2 `grplst_member`只被`ip_mforward`调用，但在协议处理过程中，`ip_mforward`又将被`ipintr`或者`ip_output`调用，`ip_output`可以由插口层间接调用。缓存是一个共享数据区，在更新时必须加以保护。 `add_lgrp`和`del_lgrp`在更新成员列表时，通过`splx`保护此共享数据结构。

14.3 `SIOCDELMULTI`命令只影响以太网接口的多播列表，不改变 IP多播组列表，因此，接口仍然保留为组中成员。只要依旧是接口 IP组列表中的一员，接口将继续接收属于该组的多播数据报。

14.4 只有虚接口才能成为多播树的父接口。如果分组在隧道上接收，那么对应的物理接口不可能成为父接口，`ip_mforward`丢弃分组。

第15章

15.1 插口可以在分支上共享，或通过UNIX域插口传给应用进程([Stevens])。

15.2 `accept`返回后，结构的`sa_len`成员大于缓存大小。对固定长度的 Internet地址而言，这不是问题，但它有可能用于可变长度的地址，例如 OSI协议支持的地址格式。

15.4 只有`so_qlen`不等于0时，才会调用`soqremque`。如果`soqremque`返回一个空指针，说明插口队列代码必然出现了内核无法处理的问题。

15.5 复制的目的在于结构锁定时仍可调用 `bzero`清零，并可在 `splx`后接着调用`dom_dispose`和`sbrealse`，从而最大程度地减少了 CPU停留在`splimp`的时间，即网络中断被阻塞的时间。

15.6 宏`sbspace`返回0，从而`sbappendaddr`和`sbappendcontrol`函数(由UDP调用)将拒绝向队列添加新报文段。 TCP调用`sbappend`，后者假定调用者已事先检查过可用空间。即使`sbspace`返回0，TCP也会调用`sbappend`，但放入接收队列中的数据还不能提交给应用进程，因为`SS_CANTRCVMORE`标志阻止`read`系统调用返回任何数据。

第16章

16.1 如果给uio结构中的`uio_resid`赋值，它将成为一个大负数。 `sosend`拒绝带有`EINVAL`的报文段。

Net/2不检查负值，`sosend`起始处的注释说明了这个问题(图16-23)。

16.2 不。向簇中填充的字节数少于`MCLBYTES`只可能出现在报文段尾部，此时剩余的字节数小于`MCLBYTES`。此时，`resid`等于0，循环在394行`break`语句处终止，还未

到达测试条件spce>0。

16.5 应用进程阻塞，直到缓存解锁。本例中，只有在另一个进程检查缓存或向协议层传送数据时，缓存才会被锁定；而在应用进程等待缓存可用空间时不会加锁，后者有可能等待无限长的时间。

16.6 如果发送缓存包括许多mbuf，每个都包括若干字节的数据，那么当mbuf分配大块存储器时，sb_cc很可能大大低于sb_hiwat规定的限制。如果内核不限制每个缓存可拥有的mbuf的数量，应用进程就能轻易地造成存储器枯竭。

16.7 recvit分别由recvfrom和recvmsg调用。只有recvmsg处理控制信息。它把完整的msghdr结构，包括控制信息长度，复制给应用进程。至于地址信息，recvmsg把namelenp参数设为空，因为它可从msg_namelen中得到所需长度。当recvfrom调用recvit时，namelenp非空，因为函数需要从*namelenp中得到所需长度。

16.8 MSG_EOR由soreceive清除，因此，它不可能在M_EOR mbuf被处理前，被soreceive返回。

16.9 select检查描述符时，实际上存在一种竞争。如果某个选定事件发生在selscan查看描述符之后，但在select调用tsleep之前，该事件不会被发现，应用进程将保持睡眠状态，直到下一个选定事件发生。

第17章

17.1 简化在内核和应用进程间复制数据的代码。copyin和copyout可用于单个的mbuf，但需要uiomove处理多个mbuf。

17.2 代码工作正确，因为linger结构的第一个成员是所要求的整数标志。

第18章

18.1 做一个8行的表格，每行对应一种查找键、路由表键和路由表掩码中比特的组合方式：

行	1 查找键	2 路由表键	3 路由表掩码	1 & 3	2 = =4?	1 ^ 2	6 & 3
1	0	0	0	0	是	0	0=是
2	0	0	1	0	是	0	0=是
3	0	1	0	0	否	1	0=是
4	0	1	1	0	否	1	1=否
5	1	0	0	0	是	1	0=是
6	1	0	1	1	否	1	1=否
7	1	1	0	0	否	0	0=是
8	1	1	1	1	是	0	0=是

图 A-5

标注为“2 == 4？”和标注为“6 & 3”的两栏，值应相等。第一眼看上去，似乎并不完全相同，但我们可以略过第3行和第7行，因为这两行中路由表比特等于1，而

在路由表掩码中的对应比特也等于1。构建路由表时，键值与掩码逻辑与，保证掩码中的等于0每一比特位，键值中的对应比特位也等于0。

可以从另一个角度理解图18-40中的异或和逻辑与操作，异或结果等于1的条件是查找键比特不同于路由表键值中对应的比特位。之后的逻辑与操作忽略所有与掩码中等于0的比特相对应的比特。如果结果依然非零，则查找键与路由表键值不匹配。

18.2 rtentry结构的大小等于120字节，其中包括两个radix_node结构。每条记录还要求两个sockaddr_in结构(图18-28)，有152字节。总数约为3兆字节。

18.3 因为rn_b是一个短整数，假定短整数占16 bit，因此，每个键值最多有32767 bit(4095字节)。

第19章

19.1 图19-15中，如果重定向报文创建了新的路由，将置位RTF_DYNAMIC标志；如果重定向报文修改了现有路由的网关字段，则置位RTF_MODIFIED标志。如果重定向报文新建了一条路由，之后另一个重定向报文又修改了它，则两个标志都会置位。

19.2 在每个可通过默认路由到达的主机上创建一条主机路由。TCP能够对每个主机维护并更新路由矩阵(图27-3)。

19.3 每个rt_msghdr结构需要76字节。主机路由中包括还两个sockaddr_in结构(目的地和网关)，因此，报文段大小为108字节。每条ARP记录的报文段为112字节：一个sockaddr_in和一个sockaddr_dl。总长度等于(15×112+20×108)即3840字节。一条网络路由(非主机路由)还需要另外的8个字节存放网络掩码(数据大小等于116字节，而非108字节)，因此，如果20条路由全部为网络路由，总长度等于4000字节。

第20章

20.1 返回值放入报文段的rtm_errno成员变量中(图20-14)，同时也做为write的返回值(图20-22)。后者更可靠，因为前者可能会因为mbuf短缺，而丢弃响应报文段(图20-17)。

20.2 对SOCK_RAW型的插口，pffindproto函数(图7-20)将返回协议值等于0(通配)的记录，如果没有找到可匹配的记录。

第21章

21.1 它基于假定ifnet结构位于arpcom的开头，事实也是如此(图3-20)。

21.2 发送ICMP的回显请求不需要ARP，因为目的地址是广播地址。但ICMP的回显响应一般都是点对点的，因此，发送者必须通过ARP确定目的以太网地址。本地主机收到ARP请求时，in_arpinput应答并为另一主机创建一条记录。

21.3 如果创建了一条新的ARP记录，图19-8中的rtrequest从源记录中复制rt_gateway值，本例中为sockaddr_dl结构。图21-1中，我们看到该记录的sdl_alen值等于0。

21.4 Net/3中，如果arpresolve的调用者提供了指向路由表表项的指针，则不会再调用arplookup，通过rt_gateway指针可得到所需的以太网地址(假定它还未超

时)。这样可以避免通常意义上的任何类型的查询。第 22章中，我们将看到 TCP和UDP在自己的协议控制块中保存指向路由表的指针， TCP不再需要搜索路由表 (连接的目的IP地址不会变化)，在目的地址不变时UDP也不需这样做。

21.5 如果ARP记录不完整，则它在记录创建后 0~5分钟超时。arpresolve发送ARP请求时，令rt_expire等于当前时间。下一次执行 arpresolve时，如果记录还没有解析，则删除它。

21.6 ether_output返回EHOSTUNREACH，而非EHOSTDOWN，从而ip_forward将发送ICMP主机不可达差错报文。

21.7 图21-28中，为140.252.13.35创建记录时，值等于当前时间。它不会改变。140.252.13.33和140.252.13.34记录的值复制自 140.252.13.32，因为rtrequest根据140.252.13.32复制前两条记录。之后， arpresolve发送ARP请求时，把这两条记录的值更新为当前时间，最后由 in_arpinput将其更新为收到ARP响应的时间加上20分钟。

21.8 修改图21-19开始处的arplookup，第二个参数永远等于1(创建标志)。

21.9 在下一秒的后半秒发送第一个数据报。因此，第一个和第二个数据报都会导致发送ARP请求，间隔约为500 ms，因为内核的time.tv_sec变量在这两个数据报发送时的值不同。

21.10 每个待发送的数据报都是一个 mbuf链，m_nextpkt指针指向每个链的第一个mbuf，用于构成等待传输的mbuf链表。

第22章

22.1 无限循环等待某个端口变为可用，假定允许应用进程打开足够多的描述符，绑定所有临时端口。

22.2 极少有服务器支持此选项。 [Cheswick和Bellovin 1994]提到为什么它可用于实现防火墙系统。

22.4 udb结构初始化为0，因此， udb.inp_lport从0开始。第一次调用 ip_pcbbin时，它增加为1，因为小于1024，所以被设定为1024。

22.5 一般情况下，调用者把地址族(sa_family)设为AF_INET，但我们在图22-20的注释中看到，最好不进行关于地址族的测试。调用者设定长度变量 (sa_len)，但我们在图 15-20中看到，函数 sockargs将其做为 bind的第3个参数，对于sockaddr_in结构，应等于16，通常都使用C的sizeof操作符。

本地IP地址(sin_addr)可以指明为通配地址或某个本地 IP地址。本地端口号(sin_port)，可以等于0(告诉内核选择一个临时端口)或非0，如果应用进程希望指明端口号。通常情况下， TCP或UDP服务器指明一个通配 IP地址，端口号等于0。

22.6 应用进程可以bind一个本地广播地址，因为ifa_ifwithaddr(图22-22)的调用成功。它被用做在该插口上发送的 IP数据报的源地址。 C.2节中指出， RFC 1122不允许这种做法。但试图绑定 255.255.255.255时会失败，因为ifa_ifwithaddr不接受该地址。

第23章

23.1 sosend把用户数据放入单个的mbuf中，如果其长度小于等于100字节；放入两个mbuf中，如果长度小于等于207字节；否则，放入多个mbuf中，每个都带有一个簇。此外，如果长度小于100字节，sosend调用MH_ALIGN，希望能在mbuf起始处为协议首部保留空间。因为udp_output调用M_PREPEND，下述5种情况都是可能的：(1)如果用户数据长度小于等于72字节，一个mbuf就可以存放IP首部、UDP首部和数据；(2)如果长度位于73字节和100字节之间，sosend为用户数据分配一个mbuf，M_PERPEND为IP和TCP首部再分配一个mbuf；(3)如果长度位于101字节和207字节之间，sosend为用户数据分配两个mbuf，M_PREPEND为IP和TCP首部再分配一个mbuf；(4)如果长度位于208字节和MCLBYTES之间，sosend为用户数据分配一个带簇的mbuf，M_PERPEND为IP和TCP首部再分配一个mbuf；(5)如果长度超出，则sosend分配足够多的mbuf和簇，以存放数据(最大数据长度65507字节，需分配64个带1024字节簇的mbuf)，M_PERPEND为IP和TCP首部再分配一个mbuf。

23.2 IP选项提交给ip_output，后者调用ip_insertoptions在输出IP数据报中插入IP选项。它接着分配一个新的mbuf，存放带有IP选项的IP首部，如果第一个mbuf指向一个簇(UDP输出不可能出现这种情况)，或者第一个mbuf中没有足够的剩余空间存放新增选项。上个习题中给出的第一种情况中，选项大小将决定ip_insertoptions是否分配另一个mbuf：如果用户数据长度小于100-28-optlen(IP选项占用的字节数)，说明mbuf足够存放IP首部、IP选项、UDP首部和数据。第2、3、4和5种情况中，第一个mbuf都由M_PREPEND分配，只存放IP和UDP首部。M_PREPEND调用M_PREPEND，接着调用MH_ALIGN，把28字节的首部移到mbuf尾部，因此，第一个mbuf中必定有空间存放最大为40字节的IP选项。

23.3 不。函数in_pcbconnect只有在应用程序调用connect，或者在一个未连接的UDP插口上发送第一个数据报时，才会被调用。因为本地地址是通配地址，本地端口号等于0，所以in_pcbconnect给本地端口号赋一个临时端口(通过调用in_pcbbind)，并根据到达目的地的路由设定本地地址。

23.4 处理器优选级仍为splnet不变，没有还原为初始值，这是代码的一个差错。

23.5 不。in_pcbconnect不允许与等于0的端口建立连接。即使应用进程没有直接调用connect，也会间接地执行connect，因此，in_pcbconnect总会被调用。

23.6 应用程序必须调用ioctl，命令为SIOCGIFCONF，返回所有已配置的IP接口信息。之后，在ioctl返回的所有IP地址和广播地址中寻找接收数据报中的目的地址(也可不用ioctl，19.14节中介绍的sysctl系统调用也能够返回所有配置接口的信息)。

23.7 recvit释放带有控制信息的mbuf。

23.8 为了断开一个已建立连接的UDP插口，调用connect，传递一个无效的地址参数，如0.0.0.0，端口号等于0。因为插口已经建立了连接，soconnect调用sodisconnect，后者调用udp_usrreq，发送PRU_DISCONNECT请求，令远端地址等于0.0.0.0，远端端口号等于0。这样，接下来调用sendto时可以指明目的地

址。由于指明地址无效，sodisconnect发送的PRU_CONNECT请求失败。实际上，我们不希望connect成功，只是要执行PRU_DISCONNECT请求，而且通过connect来执行这一请求的做法是唯一可行的方案，因为插口API没有提供disconnect函数。

手册中关于connect(2)的描述通常包括下述说明："可通过把数据报插口连接到一个无效地址，如空地址，来断开其当前连接。"但没有明确指出调用connect时，如果传送的地址无效，会返回一个差错。"空地址"的含义也易造成混淆，它指IP地址0.0.0.0，而非bind的第二个参数的空指针。

23.9 因为in_pcbbind能够建立UDP插口与远端IP地址间的临时连接，情况与应用进程调用connect类似：如果某接口的目的IP地址与该接口的广播地址对应，则从该接口发送数据报。

23.10 服务器必须设定IP_RECVDSTADDR插口选项，并调用recvmsg从客户请求中获取目的IP地址。为了成为响应报文段中的源地址，必须将其绑定在插口上。由于一个插口只能bind一次，服务器每次响应时都必须创建新的插口。

23.11 注意，ip_output(图8-22)中，IP不修改调用者传递的DF比特。需要定义新的插口选项，促使udp_output在把数据报传递给IP之前，设定DF比特。

23.12 不。它只被udp_input使用，且应为该函数的局部变量。

第24章

24.1 状态为ESTABLISHED的连接总数为126 820。除以发送和接收的总字节数，得到每个方向上的平均字节数，约为30 000字节。

24.2 tcp_output中，保存IP和TCP首部的mbuf还有空间容纳链路层首部(max_linkhdr)。试图通过bcopy把IP和TCP首部原型复制到mbuf中是行不通的，因为有可能会把40字节的首部分散在两个mbuf中。尽管40字节的首部必须放入单个mbuf中，但链路层首部不存在这样的限制。不过这样做会降低性能，因为后续处理不得不为链路层首部再次分配mbuf。

24.3 在作者的bsdi系统中，计数器等于16，其中15个是标准系统守护程序(Telnet、Rlogin、FTP，等等)。而vangogh.cs.berkeley.edu系统，一个约有20个用户的中等规模的多用户系统，计数器约为60。对于大型的带有150个用户的多用户系统(world.std.com)，则有417个TCP端点和809个UDP端点。

第25章

25.1 图24-5中，2 592 000秒(30天)中出现了531 285次延迟ACK，平均每5秒钟有一次延迟ACK，或者说每25次调用tcp_fasttimo，才会有一次延迟ACK。这说明在代码检查所有TCP控制块，判定延迟ACK标志是否置位时，96%(25次中有24次)的时间都未置位。对于习题24.3中给出的大型的多用户系统，意味着需查看超过400个的TCP控制块，每秒钟查询5次。

另一种解决方案是，在需要延迟ACK时，设定全局标志。只有当全局标志置位时，才检查控制块列表。或者为需要延迟ACK的控制块单独建立并维护一个列表。例如，

图13-14中的变量igmp_timers_are_running。

25.2 这样使得变量tcp_keepintvl绑定在运行中的内核上，下次调用tcp_slowtimo时，内核可以改变tcp_maxidle的值。

25.3 t_idle中保存的实际上是从最后一次接收或发送报文段后算起的时间。因为TCP的输出必须被对端确认，与收到数据报文段相同，收到ACK也将清零t_idle。

25.4 图A-6给出代码的一种可能的重写方式。

```
case TCPT_2MSL:
    if (tp->t_state == TCPS_TIME_WAIT)
        tp = tcp_close(tp);
    else {
        if (tp->t_idle <= tcp_maxidle)
            tp->t_timer[TCPT_2MSL] = tcp_keepintvl;
        else
            tp = tcp_close(tp);
    }
    break;
```

图 A-6

25.5 如果收到了重复的ACK，t_idle等于150，但被复位为0。FIN_WAIT_2时钟超时后，t_idle将等于1048 (1198 - 150)，因此，定时器被设定为150个滴答。定时器再次超时，t_idle应等于1198+150，导致连接被关闭。重复ACK令时间延长，直到连接被关闭。

25.6 第一次连接探测报文段将在1小时后发送。应用进程设定该选项时，实际只置位了socket结构中的SO_KEEPALIVE选项。由于设定了该选项，定时器将于1小时后超时，图25-15中的代码将发送第一次连接探测报文段。

25.7 tcp_rttdflt用于为每条TCP连接初始化RTT估计器的值。如果需要，主机可通过更改全局变量，改变默认设置。如果通过#define将其定义为常量，则只有通过重新编译内核文件才能改变默认值。

第26章

26.1 事实上，TCP并没有刻意计算从连接上最后一次发送报文段算起的时间，因为连接上的计时器t_idle一直在起作用。

26.2 图25-26中，snd_nxt被设定为snd_una，len等于0。

26.3 如果运行Net/3系统，但对端主机却无法处理某个新选项(例如，对端拒绝建立连接，即使被要求忽略无法辨识的选项)。遇到这种情况时，通过在内核中改变这个全局变量的值，可以禁止某一个或两个选项。

26.4 时间戳选项能够在每次收到对新数据的ACK时，更新RTT估计器值。因此，使用时间戳选项后，RTT估计器值将被更新16次，是不使用该选项时更新次数的两倍。注意，在时刻217.944时，收到了对6145的ACK，RTT估计器值被更新，但这个新的计算值并不准确——或者是在时刻3.740发送的携带5633~6144字节的数据段，或者是收到的对6145的ACK，在网络中延迟了200秒。

26.5 这种存储器引用时，无法确保2字节的MSS值能够正确地对齐。

26.6 (该解决方案来自于 Dave Borman)一个数据段能够携带的 TCP数据量的最大值为65495字节，即65535减去IP和TCP首部的最小值(40)。因此，紧急数据偏移量可取值范围中有39个值是无意义的：65496~65535，包括65535。无论何时，只要发送方得到一个超过65495的紧急数据偏移量，则将其替换为 65535，并置位URG标志。从而迫使接收方进入紧急模式，并告知接收方紧急数据偏移量所指向的数据尚未被发送。发送方将持续发送紧急数据偏移量等于 65535、且URG标志置位的数据报文段，直到紧急数据偏移量小于等于65495，说明真正的紧急数据偏移量的开始。

26.7 我们提到，数据段的传输是可靠的 (重传机制)，而ACK则有可能丢失。RST报文段的传输同样也是不可靠的。如果连接上收到了一个假报文段 (例如，不属于本连接的报文段，或者一个不属于任何连接的报文段)，则传送RST报文段。如果RST报文段被ip_output丢弃，当对端重传导致发送 RST报文段的数据报文段时，将再次生成RST报文段。

26.8 应用程序执行了8次写入1024字节的操作。头4次调用sosend时，tcp_output被调用，报文段被发送。因为这4个报文段都包含了发送缓存中最后一个字节的数据，每个报文段的PSH标志都置位(图26-25)。第二个缓存装满后，应用进程进行下一次写操作，调用sosend时被挂起。收到对端通告窗口大小等于 0的ACK后，丢弃发送缓存中已被确认的4096字节的数据，应用进程被唤醒，又连续执行了 4次写操作，发送缓存再次被填满。但只有当接收方通告窗口大小不等于 0时，才能继续发送数据。条件满足时，接下来的 4个报文段被发送，但只有最后一个报文段的 PSH标志置位，因为前3个报文段并未清空发送缓存。

26.9 如果正在发送的报文段不属于任何连接，传给 tcp_respond的tp参数可以是空指针。代码只有在指针为空时，才会查看 tp，并代之以默认值。

26.10 tcp_output通常调用MGETHDR，分配一个仅能容纳IP和TCP首部的mbuf，参见图26-25和图26-26。在新的mbuf的前部，代码只预留了链路层首部(max_linkhdr)大小的空间。如果使用了 IP选项，而且选项的大小超过了max_linkhdr，ip_insertoptions会自动分配另一个mbuf。但如果IP选项的大小小于等于max_linkhdr，则ip_insertoptions也会占用mbuf首部的空间，从而导致ether_output仅为链路层首部分配另一个mbuf(假定以太网输出)。
为了避免多余的mbuf，图26-25和图26-26中的代码，可以在报文段中携带IP选项时调用MH_ALIGN。

26.11 约有80行代码，假定采用了RFC 1323中的时间戳选项，且报文段被计时。
宏MGETHDR调用了宏MALLOC，后者可能调用函数malloc。函数m_copy也会被调用，但一个完整大小的报文段可能需要一个簇，因此，不复制 mbuf，而是保存一个对簇的引用。m_copy中调用MGET，可能会导致对malloc的调用。函数bcopy复制模板，而in_cksum计算TCP的检验和。

26.12 调用writev没有区别，因为处理逻辑由 sosend实现。因为数据大小等于 150字节，小于MINCLSIZE (208)，所以为头100个字节分配了一个 mbuf。并且因为协议支持数据的分段，PRU_SEND请求被发送。接着为剩余的 50字节再分配一个mbuf，并发送相应的PRU_SEND请求(对于PR_ATOMIC协议，如UDP，writev只

生成一条“记录”，即只发送一个PRU_SEND请求。)
如果两个缓存的长度分别等于200和300，总长度超过了MINCLSIZE，则分配一个mbuf簇，且只发送一次PRU_SEND请求。TCP只生成一个500字节的报文段。

第27章

27.1 表中前6行记录的差错，都是由于接收报文段或者定时器超时引起的异步差错。通过在so_error中保存非零的差错代码，应用进程能够在下一次读/写操作中收到差错信息。但如果调用来自tcp_disconnect，说明应用进程调用了close，或者应用进程终止时系统自动关闭其所拥有的描述符。无论是哪一种情况，描述符被关闭，应用进程不可能再通过读/写操作来获取差错代码。此外，因为应用进程必须明确设定插口选项，强迫RST置位，此时返回一个差错代码并不能向应用进程提供有用的信息。

27.2 假定它是32 bit的u_long，最大值小于4298秒(1.2小时)。

27.3 路由表中的统计数据由tcp_close更新，但只有当连接进入CLOSED状态时，它才会被调用。因为FTP客户终止向对端发送数据(执行主动关闭)，本地连接端点进入TIME_WAIT状态。必须经过2MSL后，路由表统计值才会被更新。

第28章

28.1 0、1、2和3。

28.2 34.9Mb/s。对于更高的速率，连接两端需要更大的缓存。

28.3 通常，tcp_dooption不知道两个时间戳值是否按32 bit边界对齐。图28-4中的代码，在指定情况下，能够确认时间戳值按32 bit边界对齐，从而避免调用bcopy。

28.4 图28-4中实现“选项预测”代码，只能处理系统推荐的格式。如果连接对端未采用系统推荐的格式，会导致为每个接收到的报文段调用tcp_dooptions，降低了处理速度。

28.5 如果在每次创建插口时，而非每次连接建立时，调用tcp_template，则系统中的每个监听服务器都会拥有一个tcp_template，而该结构可能永远不会被使用。

28.6 时间戳时钟频率应该在1 b/ms和1 b/s之间(Net/3采用了2 b/s)。如果采用最高的时钟频率1 b/ms，32 bit的时间戳将在2^{31}/(24×60×60×1000)天，即24.8天后发生符号位回绕。

28.7 如果频率为每500 ms 1 bit，32 bit的时间戳将在2^{31}/(24×60×60×2)天，即12 427天，约34年后才会出现符号位回绕。

28.8 对RST报文段的处理应优先于时间戳，而且，RST报文段中最好不携带时间戳选项(图26-24中的tcp_input代码确保了这一点)。

28.9 因为客户端状态为ESTABLISHED，处理将在图28-24的代码处结束。todrop等于1，因为rcv_nxt在收到第一个SYN时已递增过。SYN标志被清除(因为这是一个重复报文段)，ti_seq递增，todrop减为0。因为todrop和ti_len都等于0，执行图28-25起始处的if语句，并跳过下一个if语句，直接调用m_adj。但下一章中介绍tcp_input后续代码时，将谈到在某些情况下不会调用tcp_output，本题即是一例。因此，客户端会不响应重复的SYN/ACK。服务器端超时后，再次发送SYN/ACK

(图28-17中介绍了某个被动打开的插口收到 SYN时，定时器的设置)，这个重发的SYN/ACK报文段同样被忽略。我们现在讨论的其实是图28-25代码中的另一个差错，图28-30中给出的代码同样也纠正了这一差错。

28.10 客户发出的SYN到达服务器，并被交给处于 TIME_WAIT状态的插口。图28-24中的代码关闭SYN标志，图28-25中的代码跳转至dropafterack，丢弃该报文段，但生成一个 ACK，确认字段等于 rcv_nxt(图26-27)。它被称作“再同步(resynchronization) ACK”报文段，因为其目的是告诉对端本地希望接收的下一序号。客户端收到此ACK后(客户处于SYN_SENT状态)，发现它的确认字段所携带的序号不等于自己期待得到的序号后 (图28-18)，向服务器发送RST报文段。RST报文段的ACK标志被清除，且序号等于再同步 ACK报文段中确认字段携带的序号(图29-28)。服务器收到此RST报文段后，其TIME_WAIT状态提前终止，相应插口被关闭(图28-36)。客户端6秒钟后超时，重传SYN报文段。假定监听服务器进程在服务器主机上运转正常，新的连接将建立。由于TIME_WAIT状态的这种防护作用，新连接建立时，下一个 SYN报文段携带的序号既可以高于前一连接上最后收到的序号(图28-28中的测试)，也可以低于该序号。

第29章

29.1 假定RTT等于2秒钟。服务器被动打开，客户端在时刻 0主动打开。服务器在时刻 1收到客户发出的SYN，并作出响应，发送自己的SYN和对客户SYN的ACK。客户端在时刻2收到报务器的响应报文段，图 28-20中的代码调用soisconnected(唤醒客户进程)，完成主动打开过程，并向服务器发送 ACK响应。服务器在时刻3收到客户的ACK，图29-2中的代码完成服务器端的被动打开过程，控制返回给服务器进程。一般情况下，客户进程比服务器进程提早 1/2 RTT时间得到控制。

29.2 假定SYN的序号等于1000，50字节数据的序号等于1001~1050。tcp_input处理此SYN报文段时，首先执行图28-15中的起始case语句，令rcv_nxt等于1001，接着跳到step6。图29-22中的代码调用cp_reass，把数据放入插口的重组队列中。但数据还不能放入插口的接收缓存 (图27-23)，因此， rcv_nxt还是等于 1001。在调用tcp_output生成ACK响应时，rcv_nxt (1001)被放入ACK报文段的确认字段。也说是说，SYN被确认，但与之同时到达的50字节的数据没有被确认。因此，客户端不得不重发50字节的数据，所以，在完成主动打开的SYN报文段中携带数据是没有意义的。

29.3 客户端的ACK/FIN报文段到达时，服务器处于 SYN_RCVD状态，因此，图 29-2中的tcp_input代码将结束对 ACK的处理。连接转移到 ESTABLISHED状态，tcp_reass把已在重组队列中的数据放入接收缓存， rcv_nxt递增为1051。tcp_input继续执行，图29-24中的代码负责处理FIN标志，此时TF_ACKNOW标志置位，rcv_nxt等于1052。socantrcvmove设定插口的状态，使服务器在读取50字节的数据之后，得到“文件结束”指示。服务器的插口也转移到CLOSE_WAIT状态。调用tcp_output，确认客户端的 FIN(因为rcv_nxt等于1052)。假定服务器进程在收到“文件结束”指示后，关闭其插口，服务器也将向

客户发送FIN，并等待回应。

在这个例子中，这了从客户端向服务器传送50字节的数据，双方需3个来回，共发送6个报文段。为了减少所需的报文段数，应采用“用于交易的TCP扩展[Braden 1994]”。

29.4 收到服务器响应时，客户插口处于 SYN_SENT状态。图28-20中的代码处理该报文段，连接转移到ESTABLISHED状态，控制跳转到step6，由图29-22中的代码继续处理数据。TCP_REASS把数据添加到插口的接收缓存，并递增 rcv_nxt。之后，图29-24中的代码开始处理FIN，再次递增rcv_nxt，连接转移到CLOSE_WAIT状态。在调用tcp_output时，rcv_nxt同时确认了SYN、50字节的数据和FIN。随后的客户进程首先读取50字节的数据，接着是“文件结束”指示，并可能关闭其插口。客户端连接进入LAST_ACK状态，向服务器发送FIN报文段，并等待其响应报文段。

29.5 问题出在图24-16中的tcp_outflags[TCPS_CLOSING]。它设定了TH_FIN标志，而状态变迁图(图24-15)并未规定FIN应被重传。解决问题的方法是，从该状态的tcp_outflags中除去TH_FIN标志。这个问题没有什么危害——只不过多交换两个报文段——而且同时关闭或者在关闭后紧接着自连接的情况是非常罕见的。

29.6 没有。系统调用write返回OK，只说明数据已复制到插口的缓存中。在数据得到对端确认时，Net/3不再通知应用进程。如果需要得到此类信息，应设计并实现应用级的确认机制。

29.7 RFC 1323的时间戳选项，造成“首部压缩”失效。因为只要时间戳变化，即 TCP选项发生了改变，报文段发送时就不会被压缩。窗口大小选项无效，因为 TCP首部中值的长度仍为16 bit。

29.8 IP 中ID字段的取值来自一个全局变量，只要发送一个 IP数据报，该变量递增一次。这种方式导致在同一 TCP连接上两个连续TCP报文段间的ID差值大于1的可能性大大增加。一旦ID差值大于1，图29-34中的Δ*ipid*字段将被发送，增大了压缩首部的大小。一个更好的解决方案是，TCP自己维护一个计数器，用于ID的赋值。

第30章

30.2 是的，仍会发送RST报文段。应用进程终止的处理中包括关闭它打开的所有描述符。同一个函数(soclose)最终会被调用，无论是应用进程明确地关闭了插口描述符，还是隐含地进行了关闭(首先被终止)。

30.3 不。这个常量只有在监听插口设定 SO_LINGER选项，且延迟时间等于0时，才会被用到。正常情况下，插口选项的这种设定方式会导致在连接关闭时发送 RST报文段(图30-12)，但图30-2中对于接收连接请求的监听插口，把该值从 0改为120(滴答)。

30.4 如果这是第一次使用默认路由，则为两次；否则为一次。当创建插口时，in_pcballoc将Internet PCB置为0，从而将PCB结构中的route结构设为0。发送第一个报文段(SYN)时，tcp_output调用ip_output。因为ro_rt指针为空，因此向ro_dst填充IP数据报的目的地址，并调用 rtalloc。在该连接的PCB中，route结构的ro_rt变量中保存默认路由。当ip_output调用ether_output时，后者检查路由表中的rt_gwroute变量是否为空。如果是，则调用 rtalloc1。假

定路由没有改变，该连接每次调用tcp_output时，都会使用保存的ro_rt指针，以避免多余的路由表查询。

第31章

31.1 因为在bpf_wakeup调用唤醒任何沉睡进程之前，catchpacket肯定会结束。

31.2 打开BPF设备的应用进程可能调用fork，导致多个应用进程都有权访问同一个BPF设备。

31.3 只有支持BPF的设备才会出现在BPF接口表(bpf_iflist)中，因此，如果无法找到指定接口，bpf_setif将返回ENXIO。

第32章

32.1 在第一个例子中等于0，第二个例子中等于255。这些值都是RFC 1700 [Reynolds和Postel 1994]中的保留值，不应出现在数据报中。也就是说，如果某个插口创建时的协议号设定为IPPROTO_RAW，则必须设定其IP_HDRINCL插口选项，且写入到该插口的数据报必须拥有一个有效的协议值。

32.2 因为IP协议值255是保留值，不会出现在网络中传送的数据报中。但这又是一个非零的协议值，rip_input的3项测试中的第一项测试将忽略所有协议值不等于255的数据报。因此，应用进程无法在该插口上收到任何数据报。

32.3 即使该协议值是一个保留值，不会出现在网络中传送的数据报中，但rip_input的3项测试中的第一项测试保证此类型的插口能够接收任何协议类型的数据报。如果应用进程调用了connect或者bind，或者两者都调用，对于此种原始插口而言，对输入的唯一限制是IP报的源地址和目的地址。

32.4 因为ip_protox数组(图7-22)保存了有关内核所能支持的协议类型的信息，只有在该协议既没有相关的原始监听插口，而且指针inetsw[ip_protox[ip->ip_p]].pr_input等于rip_input时，才会生成ICMP差错报告。

32.5 两种情况下，应用进程都必须自己构造IP首部，以及其后的内容(UDP报文段，TCP报文段或任何其他的报文段)。对于原始IP插口，输出时同样调用sendto，通过Internet插口地址结构指明目的IP地址。调用ip_output，并依据给定的目的IP地址执行正常的IP选路。

BPF要求应用进程提供完整的数据链路层首部，例如以太网首部。输出时，需调用write，因为无法指明目的地址。数据分组被直接交给接口输出函数，跳过ip_output函数(图31-20)。应用进程通过BIOCSETIFioctl(图31-16)选择外出接口。因为未执行IP选路，数据帧只能发给是直接相连的网络上的另一个主机(除非应用进程重复IP选路函数，并将数据帧发给直接相联网络上的某个路由器，由路由器根据目的IP地址完成转发)。

32.6 原始IP插口只能接收具有内核不处理的协议类型的数据报，例如，应用进程无法在原始插口上接收TCP报文段或UDP报文段。

BPF能够接收到达指定接口的所有数据帧，无论它们是否是IP数据报。BIOCPROMISCioctl使接口处于一种混杂状态，甚至能够接收不是发给本主机的数据报。