第7章 T/TCP实现：协议控制块

7.1 概述

对于T/TCP而言，协议控制块 PCB函数(卷2的第22章)需要作一些小的修改。函数in_pcbconnect(卷2第22.2节)现在要分为两部分：一个名为 in_pcbladdr的内部例程，用于分配本地接口地址；另一个为 in_pcbconnect函数，完成原来的功能 (它要调用in_pcbladdr)。

我们把这两部分功能分开的原因是因为，当同一连接 (即相同的插口对)的前一次操作还处在TIME_WAIT状态时，T/TCP就可以发布下一个connect了。如果先前一次连接的持续时间少于MSL，并且两端都使用了CC选项，那么处于TIME_WAIT状态的连接就关闭，允许建立新的连接。如果我们没有做上述修改，并且 T/TCP使用了未修改的in_pcbconnect，当遇到现有PCB处于TIME_WAIT状态时，应用程序就会收到“地址已被使用”这样的出错消息。

不仅在发布TCP的connect时要调用in_pcbconnect，并且在新的TCP连接请求到达时、发布UDP connect以及发布UDP sendto时都要调用该函数。图7-1总结了Net/3中修改之前的调用关系。

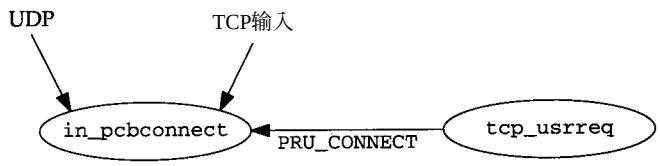

图7-1 Net/3中调用in_pcbconnect 的小结

在TCP输入和UDP中，对in_pcbconnect的调用是一样的，但是处理 TCP connect (PRU_CONNECT请求)时就要调用一个新的函数tcp_connect(图12-2和图12-3)，该函数又调用新的函数in_pcbladdr。另外，当T/TCP客户采用sendto或sendmsg隐式打开连接时，所产生的PRU_SEND或PRU_SEND_EOF请求也将调用tcp_connect。我们在图7-2中给出了这种新的调用方案。

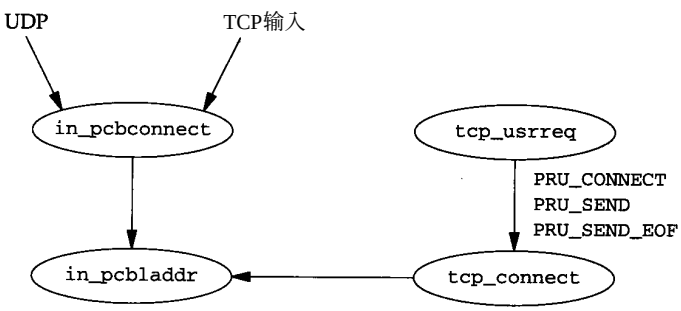

图7-2 in_pcbconnect 和in_pcbladdr 的新安排

7.2 in_pcbladdr函数

in_pcbladdr函数的第一部分如图7-3所示。这一部分仅仅给出了变量定义和头两行代码，它与卷2第590页的第138~139行完全相同。

in_pcb.c

```
136 int
137 in_pcbladdr(inp, nam, plocal_sin)
138 struct inpcb *inp;
139 struct mbuf *nam;
140 struct sockaddr_in **plocal_sin;
141 {
142     struct in_ifaddr *ia;
143     struct sockaddr_in *ifaddr;
144     struct sockaddr_in *sin = mtod(nam, struct sockaddr_in *);

145     if (nam->m_len != sizeof(*sin))
146         return (EINVAL);
```

in_pcb.c

图7-3 in_pcbladdr 函数：第一部分

136-140 头两个变量与in_pcbconnect中是一样的，第三个变量是一个指针的指针，用于返回本地地址。

这个函数的其余部分与卷2中图22-25和图22-26完全相同，与该卷图22-27的大部分也相同。卷2的图22-27中最后两行，即第593页，则用图7-4中的代码代替。

in_pcb.c

```
232         /*
233          * Don't call in_pcblookup here; return interface in
234          * plocal_sin and exit to caller, who will do the lookup.
235          */
236         *plocal_sin = &ia->ia_addr;

237     }
238     return (0);
239 }
```

in_pcb.c

图7-4 in_pcbladdr 函数：最后一部分

232-236 如果调用进程给定了通配符作为本地地址，指向 sockaddr_in结构的一个指针就会通过第三个变量返回。

基本上，in_pcbladdr所做的全部操作是进行差错检查，目标地址为 0.0.0.0或255.255.255.255这些特殊情况的处理，接着进行本地IP地址的分配(如果调用进程还没有分配IP地址)。connect所需要的其他处理操作都在in_pcbconnect中实现。

7.3 in_pcbconnect函数

图7-5中给出了in_pcbconnect函数。这个函数调用了上一节所介绍的in_pcbladdr，然后接下来就是卷2中图22-28中的代码。

1. 分配本地地址

255-259 如果调用进程还未将一个IP地址绑定到其插口，则调用in_pcbladdr函数计算出本地IP地址，然后通过ifaddr指针返回。

2. 验证插口对的唯一性

260-266 in_pcblookup验证插口对是唯一的。在TCP客户端调用connect(当客户端尚

未将一个本地端口或本地地址绑定到一个插口时)的一般情况下，本地端口号为 0，in_pcblookup就总是返回0，因为端口0是不会与任何一个现有的PCB匹配上的。

3. 如果还没有绑定，则绑定本地地址和本地端口

267-271 如果还没有本地地址和本地端口绑定到插口上，in_pcbbind要对这两者都进行分配。如果只是还没有本地地址绑定到插口，本地端口号已经为非零，则 in_pcbladdr返回的本地地址记录在PCB中。在本地端口号还是0时是不可能将一个本地地址绑定上去的，因为调用in_pcbbind函数绑定本地地址的同时会给插口分配一个临时使用的端口号。

272-273 外部地址和外部端口(in_pcbconnect的变量)记录在PCB中。

in_pcb.c

```
247 int
248 in_pcbconnect(inp, nam)
249 struct inpcb *inp;
250 struct mbuf *nam;
251 {
252     struct sockaddr_in *ifaddr;
253     struct sockaddr_in *sin = mtod(nam, struct sockaddr_in *);
254     int     error;

255     /*
256      * Call inner function to assign local interface address.
257      */
258     if (error = in_pcbladdr(inp, nam, &ifaddr))
259         return (error);

260     if (in_pcblookup(inp->inp_head,
261                      sin->sin_addr,
262                      sin->sin_port,
263                  inp->inp_laddr.s_addr ? inp->inp_laddr : ifaddr->sin_addr,
264                      inp->inp_lport,
265                      0))
266         return (EADDRINUSE);
267     if (inp->inp_laddr.s_addr == INADDR_ANY) {
268         if (inp->inp_lport == 0)
269             (void) in_pcbbind(inp, (struct mbuf *) 0);
270         inp->inp_laddr = ifaddr->sin_addr;
271     }
272     inp->inp_faddr = sin->sin_addr;
273     inp->inp_fport = sin->sin_port;
274     return (0);
275 }
```

in_pcb.c

图7-5 in_pcbconnect 函数

7.4 小结

T/TCP所作的修改是从in_pcbconnect函数中移去计算本地地址的所有代码，创建一个名为in_pcbladdr的新函数来完成这项任务。in_pcbconnect调用该函数，然后完成正常的连接处理过程。这将使处理 T/TCP客户连接请求 (或者用connect显式建连，或者用sendto隐式地建连)时可以调用in_pcbladdr来计算本地地址。T/TCP客户的端处理则是复制了图7-5中的处理步骤，但即使前一次连接尚处于 TIME_WAIT状态，T/TCP也还是允许处理同一连接的后续请求。常规的 TCP是不允许这种情况发生的；这时图 7-5的in_pcbconnect将返回EADDRINUSE。