

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_pcbldaddr`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_pcbldaddr`）。

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。

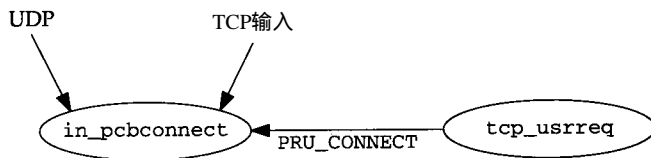


图7-1 Net/3中调用 `in_pcbconnect` 的小结

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_pcbldaddr`。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。

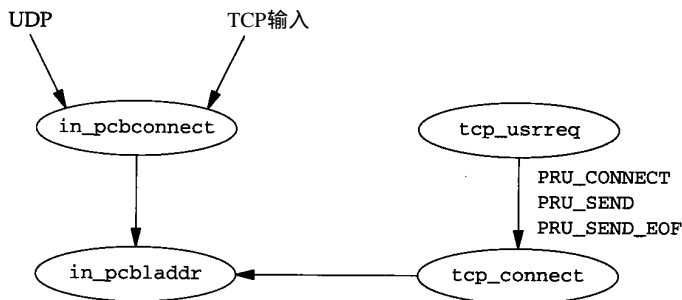


图7-2 `in_pcbconnect` 和 `in_pcbldaddr` 的新安排

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。

```

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
146     if (nam->m_len != sizeof(*sin))
147         return (EINVAL);
148 }
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

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。

```

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 }

```

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。

```

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     /*
257      * Call inner function to assign local interface address.
258      */
259     if (error = in_pcbladdr(inp, nam, &ifaddr))
260         return (error);
261
262     if (in_pcblookup(inp->inp_head,
263                     sin->sin_addr,
264                     sin->sin_port,
265                     inp->inp_laddr.s_addr ? inp->inp_laddr : ifaddr->sin_addr,
266                     inp->inp_lport,
267                     0))
268         return (EADDRINUSE);
269     if (inp->inp_laddr.s_addr == INADDR_ANY) {
270         if (inp->inp_lport == 0)
271             (void) in_pcbbind(inp, (struct mbuf *) 0);
272         inp->inp_laddr = ifaddr->sin_addr;
273     }
274     inp->inp_faddr = sin->sin_addr;
275     inp->inp_fport = sin->sin_port;
276     return (0);
277 }

```

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`。