

附录B 编写T/TCP应用程序

在第一部分，我们介绍了T/TCP的两大好处：

- 1) 避免了TCP的三次握手。
- 2) 减少了连接持续时间短于MSL时处于TIME_WAIT状态的时间。

如果一个TCP连接两端的主机支持T/TCP，那么第2条好处是所有的TCP应用程序都能感受到的，不需要修改程序。

然而，为了避免三次握手，应用程序中对connect和write函数的调用要改写为调用sendto和sendmsg。为了把FIN标志与数据组合在一起，应用程序必须在最后一次调用send、sendto或sendmsg函数时指定MSG_EOF标志，同时不再调用shutdown。我们在第1章介绍TCP和T/TCP的客户和服务程序时说明了这些差别。

为了使可移植性最好，我们要求在编写应用程序时充分利用T/TCP，其条件是：

- 1) 将要执行编译的主机支持T/TCP，并且
- 2) 应用程序要编译成支持T/TCP。

如果运行程序的主机支持T/TCP，那么第2个条件也是在运行时要确定的，因为有时会在操作系统的某个版本上编译程序，而在另一个版本上运行。

如果在<sys/socket.h>头文件中定义了MSG_EOF标志，那就是说要执行程序编译的主机支持T/TCP。这会在C预处理器的#ifdef语句中用到。

```
#ifdef MSG_EOF
    /* 主机支持 T/TCP */
#else
    /* 主机不支持 T/TCP */
#endif
```

第2个条件要求应用程序采用隐式打开（用sendto或sendmsg指定目标地址，不调用connect），但要考虑在主机不支持T/TCP时处理连接失败。在不支持T/TCP的主机上，当采用面向连接的插口但没有连接上时，所有的输出函数都会返回ENOTCONN(卷2的图16-34)。这一点对于伯克利版系统和SVR4插口库都适用。举个例子，如果应用程序在调用sendto时接收到错误指示，那它就改为调用connect。

TCP或T/TCP的客户和服务程序

我们可以在下面的程序中实现这些思想，这些程序只是对第1章的T/TCP与TCP的客户和服务程序作了简单修改。与第1章中的C语言程序一样，这里也不对程序作详细介绍，同样假设读者已经对插口编程有一定的了解。第一个程序如图B-1所示，是客户的main函数。

8-13 用服务器的IP地址和端口号填入Internet插口地址结构，这两个参数来自命令行。

15-17 用函数send_request向服务器发送请求。如果一切正常，则这个函数返回插口描述符；否则返回一个负数，表示错误。第3个变量(1)告诉函数要在发送完请求以后再发送一个

结束标志。

18-19 函数read_stream与图1-6中的同名函数一样。

```

1 #include    "cliserv.h"
2 int
3 main(int argc, char *argv[])
4 {
5     struct sockaddr_in serv;
6     char    request[REQUEST], reply[REPLY];
7     int     sockfd, n;
8
9     if (argc != 3)
10        err_quit("usage: client <IP address of server> <port#>");
11
12    memset(&serv, 0, sizeof(serv));
13    serv.sin_family = AF_INET;
14    serv.sin_addr.s_addr = inet_addr(argv[1]);
15    serv.sin_port = htons(atoi(argv[2]));
16
17    /* form request[] ... */
18
19    if ((sockfd = send_request(request, REQUEST, 1,
20                               (SA) &serv, sizeof(serv))) < 0)
21        err_sys("send_request error %d", sockfd);
22
23    if ((n = read_stream(sockfd, reply, REPLY)) < 0)
24        err_sys("read error");
25
26    /* process "n" bytes of reply[] ... */
27
28    exit(0);
29 }

```

图B-1 T/TCP或TCP客户的main 函数

函数send_request如图B-2所示。

```

1 #include    "cliserv.h"
2 #include    <errno.h>
3 #include    <netinet/tcp.h>
4
5 /* Send a transaction request to a server, using T/TCP if possible,
6  * else TCP.  Returns < 0 on error, else nonnegative socket descriptor. */
7
8 int
9 send_request(const void *request, size_t nbytes, int sendeof,
10             const SA servptr, int servsize)
11 {
12     int     sockfd, n;
13
14     if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
15         return (-1);
16
17     #ifdef MSG_EOF
18         /* T/TCP is supported on compiling host */
19
20         n = 1;
21         if (setsockopt(sockfd, IPPROTO_TCP, TCP_NOPUSH,
22                        (char *) &n, sizeof(n)) < 0) {
23             if (errno == ENOPROTOOPT)

```

图B-2 send_request 函数：用T/TCP或TCP发送请求

```

18         goto doconnect;
19     return (-2);
20 }
21 if (sendto(sockfd, request, nbytes, sendeof ? MSG_EOF : 0,
22         servptr, servsize) != nbytes) {
23     if (errno == ENOTCONN)
24         goto doconnect;
25     return (-3);
26 }
27 return (sockfd);          /* success */

28 doconnect:                /* run-time host does not support T/TCP */
29 #endif

30 /*
31  * Must include following code even if compiling host supports
32  * T/TCP, in case run-time host does not support T/TCP.
33  */

34 if (connect(sockfd, servptr, servsize) < 0)
35     return (-4);
36 if (write(sockfd, request, nbytes) != nbytes)
37     return (-5);
38 if (sendeof && shutdown(sockfd, 1) < 0)
39     return (-6);

40 return (sockfd);          /* success */
41 }

```

— sendrequest.c

图B-2 (续)

1. 试试T/TCP的sendto

13-29 如果执行编译的主机支持 T/TCP，这段程序就会执行。我们在 3.6节讨论过插口选项 TCP_NOPUSH。如果运行该程序的主机不支持 T/TCP，则对 setsockopt 函数的调用将返回 ENOPROTOOPT，程序将转移到前面的分支，执行常规的 TCP调用 connect。如果函数要求的第3个变量为非0值，则发出请求后还会发出结束标志。

2. 发出正常的TCP调用

30-40 这些是常规的TCP程序：connect、write和可选的shutdown。

服务器的main函数如图B-3所示，几乎没有改变。

```

1 #include    "cliserv.h"
2 int
3 main(int argc, char *argv[])
4 {
5     /* T/TCP or TCP server */
6     struct sockaddr_in serv, cli;
7     char    request[REQUEST], reply[REPLY];
8     int     listenfd, sockfd, n, clien;

9     if (argc != 2)
10         err_quit("usage: server <port#>");

11     if ((listenfd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
12         err_sys("socket error");

13     memset(&serv, 0, sizeof(serv));

```

— server.c

图B-3 服务器的main 函数

```
13 serv.sin_family = AF_INET;
14 serv.sin_addr.s_addr = htonl(INADDR_ANY);
15 serv.sin_port = htons(atoi(argv[1]));

16 if (bind(listenfd, (SA) &serv, sizeof(serv)) < 0)
17     err_sys("bind error");

18 if (listen(listenfd, SOMAXCONN) < 0)
19     err_sys("listen error");

20 for (;;) {
21     clilen = sizeof(cli);
22     if ((sockfd = accept(listenfd, (SA) &cli, &clilen)) < 0)
23         err_sys("accept error");

24     if ((n = read_stream(sockfd, request, REQUEST)) < 0)
25         err_sys("read error");

26     /* process "n" bytes of request[] and create reply[] ... */

27 #ifndef MSG_EOF
28 #define MSG_EOF 0          /* send() with flags=0 identical to write() */
29 #endif

30     if (send(sockfd, reply, REPLY, MSG_EOF) != REPLY)
31         err_sys("send error");

32     close(sockfd);
33 }
34 }
```

server.c

图B-3 (续)

27-31 唯一的修改是这里总是调用 send(图1-7中调用了 write), 但如果主机不支持 T/TCP, 就让第4个变量的值为 0。即使编译时主机是支持 T/TCP的, 到运行时也可能主机并不支持 T/TCP(因此运行时的内核不一定能够理解编译时的 MSG_EOF值), 因此, 在伯克利版内核中的 sosend并不会对它所不理解的标志作出反映。