

15 | 怎么老是出现“地址已经被使用”？

2019-09-04 盛延敏

网络编程实战

[进入课程 >](#)



讲述：冯永吉

时长 09:45 大小 8.94M



你好，我是盛延敏，这里是网络编程实战的第 15 讲，欢迎回来。


上一讲我们讲到 UDP 也可以像 TCP 一样，使用 connect 方法，以快速获取异步错误的信息。在今天的內容里，我们将讨论服务器端程序重启时，地址被占用的原因和解决方法。

我们已经知道，网络编程中，服务器程序需要绑定本地地址和一个端口，然后就监听在这个地址和端口上，等待客户端连接的到来。在实战中，你可能会经常碰到一个问题，当服务器端程序重启之后，总是碰到“Address in use”的报错信息，服务器程序不能很快地重启。那么这个问题是如何产生的？我们又该如何避免呢？

今天我们就来讲一讲这个“地址已经被使用”的问题。

从例子开始

为了引入讨论，我们从之前讲过的一个 TCP 服务器端程序开始说起：

 复制代码


```
1 static int count;
2
3 static void sig_int(int signo) {
4     printf("\nreceived %d datagrams\n", count);
5     exit(0);
6 }
7
8 int main(int argc, char **argv) {
9     int listenfd;
10    listenfd = socket(AF_INET, SOCK_STREAM, 0);
11
12    struct sockaddr_in server_addr;
13    bzero(&server_addr, sizeof(server_addr));
14    server_addr.sin_family = AF_INET;
15    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
16    server_addr.sin_port = htons(SERV_PORT);
17
18    int rt1 = bind(listenfd, (struct sockaddr *) &server_addr, sizeof(server_addr));
19    if (rt1 < 0) {
20        error(1, errno, "bind failed ");
21    }
22
23    int rt2 = listen(listenfd, LISTENQ);
24    if (rt2 < 0) {
25        error(1, errno, "listen failed ");
26    }
27
28    signal(SIGPIPE, SIG_IGN);
29
30    int connfd;
31    struct sockaddr_in client_addr;
32    socklen_t client_len = sizeof(client_addr);
33
34    if ((connfd = accept(listenfd, (struct sockaddr *) &client_addr, &client_len)) < 0)
35        error(1, errno, "bind failed ");
36    }
37
38    char message[MAXLINE];
39    count = 0;
40
41    for (;;) {
42        int n = read(connfd, message, MAXLINE);
43        if (n < 0) {
44            error(1, errno, "error read");
45        } else if (n == 0) {
```

```
46         error(1, 0, "client closed \n");
47     }
48     message[n] = 0;
49     printf("received %d bytes: %s\n", n, message);
50     count++;
51 }
52 }
```

这个服务器端程序绑定到一个本地端口，使用的是通配地址 ANY，当连接建立之后，从该连接中读取输入的字符流。


启动服务器，之后我们使用 Telnet 登录这个服务器，并在屏幕上输入一些字符，例如：
network, good。

和我们期望的一样，服务器端打印出 Telnet 客户端的输入。在 Telnet 端关闭连接之后，服务器端接收到 EOF，也顺利地关闭了连接。服务器端也可以很快重启，等待新的连接到来。

 复制代码


```
1  $./addressused
2  received 9 bytes: network
3  received 6 bytes: good
4  client closed
5  $./addressused
```

接下来，我们改变一下连接的关闭顺序。和前面的过程一样，先启动服务器，再使用 Telnet 作为客户端登录到服务器，在屏幕上输入一些字符。注意接下来的不同，我不会在 Telnet 端关闭连接，而是直接使用 Ctrl+C 的方式在服务器端关闭连接。

 复制代码

```
1  $telnet 127.0.0.1 9527
2  network
3  bad
4  Connection closed by foreign host.
```

我们看到，连接已经被关闭，Telnet 客户端也感知连接关闭并退出了。接下来，我们尝试重启服务器端程序。你会发现，这个时候服务端程序重启失败，报错信息为：**bind failed: Address already in use.**

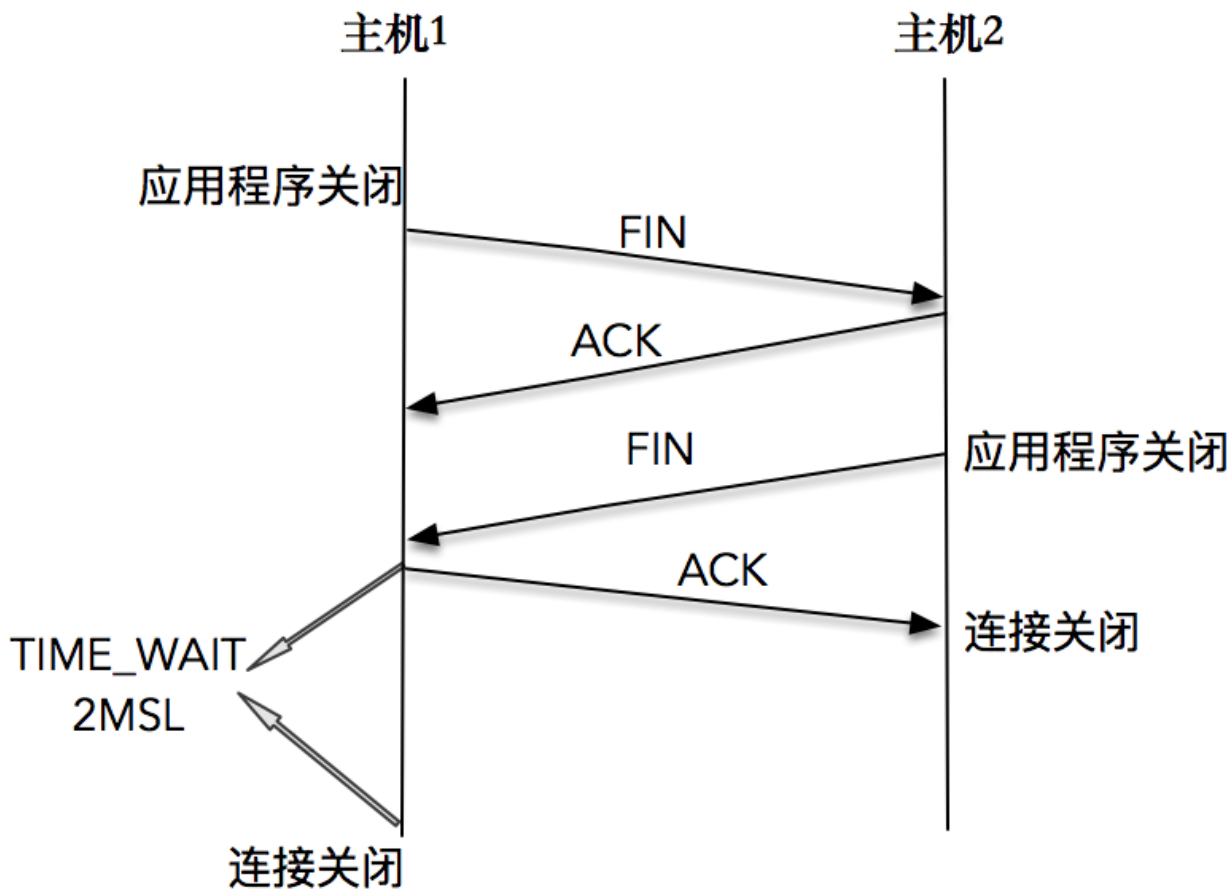
 复制代码

```
1  $./addressused
2  received 9 bytes: network
3  received 6 bytes: good
4  client closed
5  $./addressused
6  bind faied: Address already in use(98)
```

复习 TIME_WAIT

那么，这个错误到底是怎么发生的呢？

还记得第 10 篇文章里提到的 TIME_WAIT 么？当连接的一方主动关闭连接，在接收到对端的 FIN 报文之后，主动关闭连接的一方会在 TIME_WAIT 这个状态里停留一段时间，这个时间大约为 2MSL。如果你对此有点淡忘，没有关系，我在下面放了一张图，希望能唤起你的记忆。



如果我们此时使用 netstat 去查看服务器程序所在主机的 TIME_WAIT 的状态连接，你会发现有一个服务器程序生成的 TCP 连接，当前正处于 TIME_WAIT 状态。这里 9527 是本地监听端口，36650 是 telnet 客户端端口。当然了，Telnet 客户端端口每次也会不尽相同。

```

vagrant@ubuntu-xenial-01:~/shared/Code/network/yolanda/build/bin$ netstat -alepn | grep TIME_WAIT
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp        0      0 127.0.0.1:10252      127.0.0.1:50646      TIME_WAIT   0
tcp        0      0 127.0.0.1:45456      127.0.0.1:2379       TIME_WAIT   0
tcp        0      0 127.0.0.1:45348      127.0.0.1:2379       TIME_WAIT   0
tcp        0      0 127.0.0.1:10252      127.0.0.1:50592      TIME_WAIT   0
tcp        0      0 10.244.0.1:37814     10.244.0.100:8080    TIME_WAIT   0
tcp        0      0 127.0.0.1:10252      127.0.0.1:50702      TIME_WAIT   0
tcp        0      0 127.0.0.1:10251      127.0.0.1:54222      TIME_WAIT   0
tcp        0      0 127.0.0.1:10251      127.0.0.1:54114      TIME_WAIT   0
tcp        0      0 127.0.0.1:10251      127.0.0.1:54276      TIME_WAIT   0
tcp        0      0 127.0.0.1:10252      127.0.0.1:50756      TIME_WAIT   0
tcp        0      0 127.0.0.1:45240      127.0.0.1:2379       TIME_WAIT   0
tcp        0      0 127.0.0.1:10251      127.0.0.1:54060      TIME_WAIT   0
tcp        0      0 127.0.0.1:10252      127.0.0.1:50810      TIME_WAIT   0
tcp        0      0 127.0.0.1:54168      127.0.0.1:10251      TIME_WAIT   0
tcp        0      0 127.0.0.1:45294      127.0.0.1:2379       TIME_WAIT   0
tcp        0      0 127.0.0.1:45184      127.0.0.1:2379       TIME_WAIT   0
tcp        0      0 127.0.0.1:45402      127.0.0.1:2379       TIME_WAIT   0
tcp        0      0 10.244.0.1:49532     10.244.0.99:8080     TIME_WAIT   0
tcp        0      0 127.0.0.1:10251      127.0.0.1:54004      TIME_WAIT   0
tcp        0      0 127.0.0.1:9527       127.0.0.1:36650      TIME_WAIT   0
tcp        0      0 10.244.0.1:37922     10.244.0.100:8080    TIME_WAIT   0
tcp        0      0 127.0.0.1:10252      127.0.0.1:50864      TIME_WAIT   0
  
```

通过服务器端发起的关闭连接操作，引起了一个已有的 TCP 连接处于 TIME_WAIT 状态，正是这个 TIME_WAIT 的连接，使得服务器重启时，继续绑定在 127.0.0.1 地址和 9527 端口上的操作，返回了 **Address already in use** 的错误。

重用套接字选项

我们知道，一个 TCP 连接是通过四元组（源地址、源端口、目的地址、目的端口）来唯一确定的，如果每次 Telnet 客户端使用的本地端口都不同，就不会和已有的四元组冲突，也就不会有 TIME_WAIT 的新旧连接化身冲突的问题。


事实上，即使在很小的概率下，客户端 Telnet 使用了相同的端口，从而造成了新连接和旧连接的四元组相同，在现代 Linux 操作系统下，也不会有什么大的问题，原因是现代 Linux 操作系统对此进行了一些优化。

第一种优化是新连接 SYN 告知的初始序列号，一定比 TIME_WAIT 老连接的末序列号大，这样通过序列号就可以区别出新老连接。

第二种优化是开启了 tcp_timestamps，使得新连接的时间戳比老连接的时间戳大，这样通过时间戳也可以区别出新老连接。

在这样的优化之下，一个 TIME_WAIT 的 TCP 连接可以忽略掉旧连接，重新被新的连接所使用。

这就是重用套接字选项，通过给套接字配置可重用属性，告诉操作系统内核，这样的 TCP 连接完全可以复用 TIME_WAIT 状态的连接。代码片段已经放在文章中：


 复制代码

```
1 int on = 1;
2 setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
```

SO_REUSEADDR 套接字选项，允许启动绑定在一个端口，即使之前存在一个和该端口一样的连接。前面的例子已经表明，在默认情况下，服务器端历经创建 socket、bind 和 listen 重启时，如果试图绑定到一个现有连接上的端口，bind 操作会失败，但是如果我们

在创建 socket 和 bind 之间，使用上面的代码片段设置 SO_REUSEADDR 套接字选项，情况就会不同。


下面我们对原来的服务器端代码进行升级，升级的部分主要在 11-12 行，在 bind 监听套接字之前，调用 setsockopt 方法，设置重用套接字选项：

 复制代码

```
1  int main(int argc, char **argv) {
2      int listenfd;
3      listenfd = socket(AF_INET, SOCK_STREAM, 0);
4
5      struct sockaddr_in server_addr;
6      bzero(&server_addr, sizeof(server_addr));
7      server_addr.sin_family = AF_INET;
8      server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
9      server_addr.sin_port = htons(SERV_PORT);
10
11     int on = 1;
12     setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
13
14     int rt1 = bind(listenfd, (struct sockaddr *) &server_addr, sizeof(server_addr));
15     if (rt1 < 0) {
16         error(1, errno, "bind failed ");
17     }
18
19     int rt2 = listen(listenfd, LISTENQ);
20     if (rt2 < 0) {
21         error(1, errno, "listen failed ");
22     }
23
24     signal(SIGPIPE, SIG_IGN);
25
26     int connfd;
27     struct sockaddr_in client_addr;
28     socklen_t client_len = sizeof(client_addr);
29
30     if ((connfd = accept(listenfd, (struct sockaddr *) &client_addr, &client_len)) < 0)
31         error(1, errno, "bind failed ");
32 }
33
34 char message[MAXLINE];
35 count = 0;
36
37 for (;;) {
38     int n = read(connfd, message, MAXLINE);
39     if (n < 0) {
40         error(1, errno, "error read");
41     } else if (n == 0) {
```

```
42         error(1, 0, "client closed \n");
43     }
44     message[n] = 0;
45     printf("received %d bytes: %s\n", n, message);
46     count++;
47 }
48 }
```

重新编译过后，重复上面那个例子，先启动服务器，再使用 Telnet 作为客户端登录到服务器，在屏幕上输入一些字符，使用 Ctrl+C 的方式在服务器端关闭连接。马上尝试重启服务器，这个时候我们发现，服务器正常启动，没有出现 **Address already in use** 的错误。这说明我们的修改已经起作用。

 复制代码

```
1  $./addressused2
2  received 9 bytes: network
3  received 6 bytes: good
4  client closed
5  $./addressused2
```

SO_REUSEADDR 套接字选项还有一个作用，那就是本机服务器如果有多个地址，可以在不同地址上使用相同的端口提供服务。

比如，一台服务器有 192.168.1.101 和 10.10.2.102 两个地址，我们可以在这台机器上启动三个不同的 HTTP 服务，第一个以本地通配地址 ANY 和端口 80 启动；第二个以 192.168.101 和端口 80 启动；第三个以 10.10.2.102 和端口 80 启动。

这样目的地址为 192.168.101，目的端口为 80 的连接请求会被发往第二个服务；目的地址为 10.10.2.102，目的端口为 80 的连接请求会被发往第三个服务；目的端口为 80 的所有其他连接请求被发往第一个服务。

我们必须给这三个服务设置 SO_REUSEADDR 套接字选项，否则第二个和第三个服务调用 bind 绑定到 80 端口时会出错。


最佳实践

这里的最佳实践可以总结成一句话：服务器端程序，都应该设置 `SO_REUSEADDR` 套接字选项，以便服务端程序可以在极短时间内复用同一个端口启动。

有些人可能觉得这不是安全的。其实，单独重用一個套接字不会有任何问题。我在前面已经讲过，TCP 连接是通过四元组唯一区分的，只要客户端不使用相同的源端口，连接服务器是没有问题的，即使使用了相同的端口，根据序列号或者时间戳，也是可以区分出新旧连接的。

而且，TCP 的机制绝对不允许在相同的地址和端口上绑定不同的服务器，即使我们设置 `SO_REUSEADDR` 套接字选项，也不可能在 ANY 通配符地址下和端口 9527 上重复启动两个服务器实例。如果我们启动第二个服务器实例，不出所料会得到 **Address already in use** 的报错，即使当前还没有任何一条有效 TCP 连接产生。

比如下面就是第二次运行服务器端程序的报错信息：

 复制代码

```
1  $./addressused2
2  bind faied: Address already in use(98)
```

你可能还记得[第 10 讲](#)中，我们提到过一个叫做 `tcp_tw_reuse` 的内核配置选项，这里又提到了 `SO_REUSEADDR` 套接字选择，你会不会觉得两个有点混淆呢？

其实，这两个东西一点关系也没有。

`tcp_tw_reuse` 是内核选项，主要用在连接的发起方。`TIME_WAIT` 状态的连接创建时间超过 1 秒后，新的连接才可以被复用，注意，这里是连接的发起方；

`SO_REUSEADDR` 是用户态的选项，`SO_REUSEADDR` 选项用来告诉操作系统内核，如果端口已被占用，但是 TCP 连接状态位于 `TIME_WAIT`，可以重用端口。如果端口忙，而 TCP 处于其他状态，重用端口时依旧得到 “Address already in use” 的错误信息。注意，这里一般都是连接的服务方。

总结

今天我们分析了“Address already in use”产生的原因和解决方法。你只要记住一句话，**在所有 TCP 服务器程序中，调用 bind 之前请设置 SO_REUSEADDR 套接字选项**。这不会产生危害，相反，它会帮助我们在很短时间内重启服务端程序，而这一点恰恰是很多场景所需要的。

思考题

跟往常一样，给大家布置两道思考题：

第一道，之前我们看到的例子，都是对 TCP 套接字设置 SO_REUSEADDR 套接字选项，你知道吗，我们也可以对 UDP 设置 SO_REUSEADDR 套接字选项。那么问题来了，对 UDP 来说，设置 SO_REUSEADDR 套接字选项有哪些场景和好处呢？

第二道，在服务器端程序中，设置 SO_REUSEADDR 套接字选项时，需要在 bind 函数之前对监听字进行设置，想一想，为什么不是对已连接的套接字进行设置呢？

欢迎你在评论区写下你的思考，我会和你一起讨论交流，也欢迎把这篇文章分享给你的朋友或者同事，一起交流一下。



网络编程实战

从底层到实战，深度解析网络编程

盛延敏

前大众点评云平台首席架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

上一篇 14 | UDP也可以是“已连接”？

下一篇 16 | 如何理解TCP的“流”？

精选留言 (6)

写留言



刘晓林

2019-09-07

关于tcp_tw_reuse和SO_REUSEADDR的区别，可以概括为：tcp_tw_reuse是为了缩短time_wait的时间，避免出现大量的time_wait链接而占用系统资源，解决的是accept后的问题；SO_REUSEADDR是为了解决time_wait状态带来的端口占用问题，以及支持同一个port对应多个ip，解决的是bind时的的问题。

展开



1



传说中的成大大

2019-09-04

我竟然是沙发

第一问: 百度出来的 针对udp是允许完全的重复的捆绑 就是是udp允许把ip地址绑定到多个套接口上,大概是为了在同一机器上运行多个多播程序的情况下,具体的实例却想不出来
2. 因为我觉得bind函数时告诉内核我要监听这个ip地址和端口是在内核层的事情, 如果bind过后再进行设置套接字选项的话虽然是在应用层对套接字进行了修改,但是没告诉内核,...

展开



1



石将从

2019-09-07

怎么用telnet连接，求老师回答

展开



不动声色满心澎湃

2019-09-04

老师 有个疑问想问下：如果我的服务器是双网卡。一个192.168.1.220 一个是192.68.1.221 然后我让220和端口8010 处于time_wait状态，这个时候再用221和8010去启动一个程序，那会报addr in use吗



不动声色满心澎湃



2019-09-04

老师， 希望可以多讲一点。 感觉听了很爽 但是觉得不够 哈哈哈



nil

2019-09-04

学生时代写网络编程作业，调试的时候经常有遇到这个问题，然后通过每次改变端口号绕过这个问题。想想当时遇到问题一知半解，也不知道去寻找根本原因，哈哈哈，估计心思都在完成作业上，而根本不是想要掌握这个技术底层的原理

展开 ▼

