Linux 下 Socket 编程的端口问题

(Bind(): Address already in use)

转载请注明 原文： http://blog.csdn.net/xl_xunzhao/archive/2008/10/23/3130037.aspx

最近在开发一个 Linux 下的聊天软件，每次修改了源代码并再次编译运行时，常遇到下面的地使用错误：

Cann't bind server socket !

: Address already in use

虽然用 Ctrl＋C 强制结束了进程，但错误依然存在，用 netstat -an |grep 5120 和 ps aux |grep 5120 都还能看到刚才用 Ctrl+C "强制结束" 了的进程，端口还是使用中，只好每次用 kill 结束进程，很是麻烦。昨天晚上无意间浏览到 IBM 网站上的一篇题为《Linux 套接字编程中的 5 个隐患》的文章，恍然大悟，今天试了一下，果然解决问题，在此表示感谢，也希望更多的 coder 看到这篇文章，避免出错。

主要代码为：

```
1.  int on;
2.  on = 1;
4.  ret = setsockopt( sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on) );
```

现在我每次用 Ctrl＋C 强制结束进程后，用 netstat 和 ps 都还能看到端口在使用中，但运行程序不会出现 "Address already in use" 的错误了，实现了端口的重用。

以下是原文中的第三个隐患－－地址使用错误

地址使用错误（EADDRINUSE）

您可以使用 bind API 函数来绑定一个地址（一个接口和一个端口）到一个套接字端点。可以在服务器设置中使用这个函数，以便限制可能有连接到来的接口。也可以在客户端设置中使用这个函数，以便限制应当供出去的连接所使用的接口。bind 最常见的用法是关联端口号和服务器，并使用通配符地址（INADDR_ANY），它允许任何接口为到来的连接所使用。

bind 普遍遭遇的问题是试图绑定一个已经在使用的端口。该陷阱是也许没有活动的套接字存在，但仍然禁止绑定端口（bind 返回 EADDRINUSE），它由 TCP 套接字状态 TIME_WAIT 引起。该状态在套接字关闭后约保留 2 到 4 分钟。在 TIME_WAIT 状态退出之后，套接字被删除，该地址才能被重新绑定而不出问题。

等待 TIME_WAIT 结束可能是令人恼火的一件事，特别是如果您正在开发一个套接字服务器，就需要停止服务器来做一些改动，然后重启。幸运的是，有方法可以避开 TIME_WAIT 状态。可以给套接字应用 SO_REUSEADDR 套接字选项，以便端口可以马上重用。

考虑清单 3 的例子。在绑定地址之前，我以 SO_REUSEADDR 选项调用 setsockopt。为了允许地址重用，我设置整型参数（on）为 1 （不然，可以设为 0 来禁止地址重用）。

使用 SO_REUSEADDR 套接字选项避免地址使用错误

```
101.  int sock, ret, on;
102.  struct sockaddr_in servaddr;
103.  /* Create a new stream (TCP) socket */
104.  sock = socket( AF_INET, SOCK_STREAM, 0 ):
105.  /* Enable address reuse */
106.  on = 1;
107.  ret = setsockopt( sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on) );
108.  /* Allow connections to port 8080 from any available interface */
109.  memset( &servaddr, 0, sizeof(servaddr) );
110.  servaddr.sin_family = AF_INET;
111.  servaddr.sin_addr.s_addr = htonl( INADDR_ANY );
112.  servaddr.sin_port = htons( 45000 );
113.  /* Bind to the address (interface/port) */
114.  ret = bind( sock, (struct sockaddr *)&servaddr, sizeof(servaddr) );
```

在应用了 SO_REUSEADDR 选项之后，bind API 函数将允许地址的立即重用。

附录：man setsockopt

参考资料：

Linux 套接字编程中的 5 个隐患

http://www.ibm.com/developerworks/cn/linux/l-sockpit/

本文引用地址：http://blog.csdn.net/xl_xunzhao/archive/2008/10/23/3130037.aspx

附录：

man setsockopt 摘自 Open Group

原文：http://www.opengroup.org/onlinepubs/7990989775/xns/setsockopt.html

NAME

setsockopt - set the socket options

SYNOPSIS

```
#include <sys/socket.h>
int setsockopt(int socket, int level, int option_name, const void
               *option_value, socklen_t option_len);
```

DESCRIPTION

The *setsockopt()* function sets the option specified by the *option_name* argument, at the protocol level specified by the *level* argument, to the value pointed to by the *option_value* argument for the socket associated with the file descriptor specified by the *socket* argument.

The *level* argument specifies the protocol level at which the option resides. To set options at the socket level, specify the *level* argument as SOL_SOCKET. To set options at other levels, supply the appropriate protocol number for the protocol controlling the option. For example, to indicate that an option will be interpreted by the TCP (Transport Control Protocol), set *level* to the protocol number of TCP, as defined in the <**netinet/in.h**> header, or as determined by using *getprotobyname()*.

The *option_name* argument specifies a single option to set. The *option_name* argument and any specified options are passed uninterpreted to the appropriate protocol module for interpretations. The <**sys/socket.h**> header defines the socket level options. The options are as follows:

SO_DEBUG

> Turns on recording of debugging information. This option enables or disables debugging in the underlying protocol modules. This option takes an **int** value. This is a boolean option.

SO_BROADCAST

> Permits sending of broadcast messages, if this is supported by the protocol. This option takes an **int** value. This is a boolean option.

SO_REUSEADDR

> Specifies that the rules used in validating addresses supplied to *bind()* should allow reuse of local addresses, if this is supported by the protocol. This option takes an **int** value. This is a boolean option.

SO_KEEPALIVE

> Keeps connections active by enabling the periodic transmission of messages, if this is supported by the protocol. This option takes an **int** value. If the connected socket fails to respond to these messages, the connection is broken and processes writing to that socket are notified with a SIGPIPE signal. This is a boolean option.

SO_LINGER

> Lingers on a *close()* if data is present. This option controls the action taken when unsent messages queue on a socket and *close()* is performed. If SO_LINGER is set, the system blocks the process

during *close()* until it can transmit the data or until the time expires. If SO_LINGER is not specified, and *close()* is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option takes a **linger** structure, as defined in the <**sys/socket.h**> header, to specify the state of the option and linger interval.

SO_OOBINLINE

Leaves received out-of-band data (data marked urgent) in line. This option takes an **int** value. This is a boolean option.

SO_SNDBUF

Sets send buffer size. This option takes an **int** value.

SO_RCVBUF

Sets receive buffer size. This option takes an **int** value.

SO_DONTROUTE

Requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an **int** value. This is a boolean option.

SO_RCVLOWAT

Sets the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different than that returned, e.g. out of band data). This option takes an **int** value. Note that not all implementations allow this option to be set.

SO_RCVTIMEO

Sets the timeout value that specifies the maximum amount of time an input function waits until it completes. It accepts a **timeval** structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without

receiving additional data, it returns with a partial count or *errno* set to [EAGAIN] or [EWOULDBLOCK] if no data were received. The default for this option is zero, which indicates that a receive operation will not time out. This option takes a **timeval** structure. Note that not all implementations allow this option to be set.

SO_SNDLOWAT

Sets the minimum number of bytes to process for socket output operations. Non-blocking output operations will process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option takes an **int** value. Note that not all implementations allow this option to be set.

SO_SNDTIMEO

Sets the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it returns with a partial count or with errno set to [EAGAIN] ore [EWOULDBLOCK] if no data were sent. The default for this option is zero, which indicates that a send operation will not time out. This option stores a **timeval** structure. Note that not all implementations allow this option to be set.

For boolean options, 0 indicates that the option is disabled and 1 indicates that the option is enabled.

Options at other protocol levels vary in format and name.

RETURN VALUE

Upon successful completion, *setsockopt()* returns 0. Otherwise, -1 is returned and *errno* is set to indicate the error.

ERRORS

The *setsockopt()* function will fail if:

[EBADF]The *socket* argument is not a valid file descriptor.

[EDOM]The send and receive timeout values are too big to fit into the timeout fields in the socket structure.

[EFAULT]The *option_value* parameter can not be accessed or written.

[EINVAL]The specified option is invalid at the specified socket level or the socket has been shut down.

[EISCONN]The socket is already connected, and a specified option can not be set while the socket is connected.

[ENOPROTOOPT]The option is not supported by the protocol.

[ENOTSOCK]The *socket* argument does not refer to a socket.

The *setsockopt()* function may fail if:

[ENOMEM]There was insufficient memory available for the operation to complete.

[ENOBUFS]Insufficient resources are available in the system to complete the call.

[ENOSR]There were insufficient STREAMS resources available for the operation to complete.

APPLICATION USAGE

The *setsockopt()* function provides an application program with the means to control socket behaviour. An application program can use *setsockopt()* to allocate buffer space, control timeouts, or permit socket data broadcasts. The *<sys/socket.h>* header defines the socket-level options available to *setsockopt()*.

Options may exist at multiple protocol levels. The SO_ options are always present at the uppermost socket level.

SEE ALSO

bind(), *endprotoent()*, *getsockopt()*, *socket()*, *<sys/socket.h>*.