任务信息

TCP 支持 SO_REUSEADDR选项 #9

Open Azure-stars opened this issue last week · 0 comments

Azure-stars commented last week • edited Contributor

`SO_REUSEADDR` 选项有以下几个作用：

1. **允许端口重用**：当一个 TCP 连接关闭后，连接的端口会进入 `TIME_WAIT` 状态，此时端口虽然没有被使用，但仍然不能立即重新绑定。启用 `SO_REUSEADDR` 选项后，可以在该端口仍处于 `TIME_WAIT` 状态时，重新绑定该端口。
2. **允许多个套接字绑定到同一个端口**：在某些多播协议的应用中，多个套接字可以绑定到同一个端口，使用 `SO_REUSEADDR` 可以实现这一点。
3. **解决服务器重启绑定端口的问题**：在服务器程序重启时，如果上一次运行的服务器占用了某个端口且该端口尚未完全释放，启用 `SO_REUSEADDR` 选项可以让新启动的服务器成功绑定到同一个端口。

之前在 UDP 上支持了 `SO_REUSEADDR`，见该 PR：https://github.com/Arceos-monolithic/Starry/pull/37/files，可以用作 TCP 支持 `SO_REUSEADDR` 参考。

测试方式：可以根据第一和二个点设计测例，将多个套接字绑定到同一个端口，或者重用一个端口。

修改代码

1. linux_syscall_api

```
diff --git a/src/syscall_net/socket.rs b/src/syscall_net/socket.rs
index 75879c6..adc27e6 100644
--- a/src/syscall_net/socket.rs
+++ b/src/syscall_net/socket.rs
@@ -470,7 +470,7 @@ impl Socket {
     }
     fn get_reuse_addr(&self) -> bool {
         match &self.inner {
-            SocketInner::Tcp(s) => unimplemented!("get_reuse_addr on other socket"),
+            SocketInner::Tcp(s) => s.is_reuse_addr(),
             SocketInner::Udp(s) => s.is_reuse_addr(),
         }
     }
@@ -495,7 +495,7 @@ impl Socket {

     fn set_reuse_addr(&self, flag: bool) {
         match &self.inner {
-            SocketInner::Tcp(s) => (),
+            SocketInner::Tcp(s) => s.set_reuse_addr(flag),
             SocketInner::Udp(s) => s.set_reuse_addr(flag),
         }
     }
```

2. axnet

6 src/smoltcp_impl/mod.rs

```
@@ -130,6 +130,12 @@ impl<'a> SocketSetWrapper<'a> {
130 130             let mut sockets = self.0.lock();
131 131             for item in sockets.iter_mut() {
132 132                 match item.1 {
133     +                 Socket::Tcp(s) => {
134     +                     let local_addr = s.get_bound_endpoint();
135     +                     if local_addr.addr == Some(addr) {
136     +                         return Err(AxError::AddrInUse);
137     +                     }
138     +                 }
133 139                   Socket::Udp(s) => {
134 140                       if s.endpoint().addr == Some(addr) {
135 141                           return Err(AxError::AddrInUse);
```

30 src/smoltcp_impl/tcp.rs

```
@@ -43,6 +43,7 @@ pub struct TcpSocket {
43 43      local_addr: UnsafeCell<IpEndpoint>,
44 44      peer_addr: UnsafeCell<IpEndpoint>,
45 45      nonblock: AtomicBool,
46     +   reuse_addr: AtomicBool,
46 47  }
47 48
48 49  unsafe impl Sync for TcpSocket {}
@@ -56,6 +57,7 @@ impl TcpSocket {
56 57              local_addr: UnsafeCell::new(UNSPECIFIED_ENDPOINT),
57 58              peer_addr: UnsafeCell::new(UNSPECIFIED_ENDPOINT),
58 59              nonblock: AtomicBool::new(false),
60     +           reuse_addr: AtomicBool::new(false),
59 61          }
60 62      }
61 63
@@ -71,6 +73,7 @@ impl TcpSocket {
71 73              local_addr: UnsafeCell::new(local_addr),
72 74              peer_addr: UnsafeCell::new(peer_addr),
73 75              nonblock: AtomicBool::new(false),
76     +           reuse_addr: AtomicBool::new(false),
74 77          }
75 78      }
76 79
@@ -118,6 +121,22 @@ impl TcpSocket {
```

```
124 +     ///Returns whether this socket is in reuse address mode.
125 +     #[inline]
126 +     pub fn is_reuse_addr(&self) -> bool {
127 +         self.reuse_addr.load(Ordering::Acquire)
128 +     }
129 +
130 +     /// Moves this TCP socket into or out of reuse address mode.
131 +     ///
132 +     /// When a socket is bound, the `SO_REUSEADDR` option allows multiple sockets to be bound to the
133 +     /// same address if they are bound to different local addresses. This option must be set before
134 +     /// calling `bind`.
135 +     #[inline]
136 +     pub fn set_reuse_addr(&self, reuse_addr: bool) {
137 +         self.reuse_addr.store(reuse_addr, Ordering::Release);
138 +     }
139 +
121 140      /// To get the address pair of the socket.
122 141      ///
123 142      /// Returns the local and remote endpoint pair.
    ...  @@ -235,6 +254,17 @@ impl TcpSocket {
235 254                  }
236 255                  self.local_addr.get().write(from_core_sockaddr(local_addr));
237 256              }
257 +             let local_endpoint = from_core_sockaddr(local_addr);
258 +             let bound_endpoint = self.bound_endpoint()?;
259 +             let handle = unsafe { self.handle.get().read() }
260 +                 .unwrap_or_else(|| SOCKET_SET.add(SocketSetWrapper::new_tcp_socket()));
261 +             SOCKET_SET.with_socket_mut::<tcp::Socket, _, _>(handle, |socket| {
262 +                 socket.set_bound_endpoint(bound_endpoint);
263 +             });
264 +
265 +             if !self.is_reuse_addr() {
266 +                 SOCKET_SET.bind_check(local_endpoint.addr, local_endpoint.port)?;
267 +             }
238 268              Ok(())
239 269          })
240 270          .unwrap_or_else(|_| ax_err!(InvalidInput, "socket bind() failed: already bound"))
    ...
```

3. smoltcp

15 ■■■■■ src/socket/tcp.rs

Expand all

```
@@ -429,6 +429,8 @@ pub struct Socket<'a> {
429 429     /// Address passed to listen(). Listen address is set when listen() is called and
430 430     /// used every time the socket is reset back to the LISTEN state.
431 431     listen_endpoint: IpListenEndpoint,
    432 +   /// Address passed to bind(). Record the binding address of the socket.
    433 +   bound_endpoint: IpListenEndpoint,
432 434     /// Current 4-tuple (local and remote endpoints).
433 435     tuple: Option<Tuple>,
434 436     /// The sequence number corresponding to the beginning of the transmit buffer.
@@ -521,6 +523,7 @@ impl<'a> Socket<'a> {
521 523             keep_alive: None,
522 524             hop_limit: None,
523 525             listen_endpoint: IpListenEndpoint::default(),
    526 +           bound_endpoint: IpListenEndpoint::default(),
524 527             tuple: None,
525 528             local_seq_no: TcpSeqNumber::default(),
526 529             remote_seq_no: TcpSeqNumber::default(),
@@ -770,6 +773,18 @@ impl<'a> Socket<'a> {
770 773         Some(self.tuple?.remote)
771 774     }
772 775
    776 +   /// get bound endpoint.
    777 +   #[inline]
    778 +   pub fn get_bound_endpoint(&self) -> IpListenEndpoint {
    779 +       self.bound_endpoint
    780 +   }
    781 +
    782 +   /// set bound endpoint.
    783 +   #[inline]
    784 +   pub fn set_bound_endpoint(&mut self, bound_endpoint: IpListenEndpoint) {
    785 +       self.bound_endpoint = bound_endpoint
    786 +   }
    787 +
773 788     /// Return the connection state, in terms of the TCP state machine.
774 789     #[inline]
775 790     pub fn state(&self) -> State {
```

测试

暂时可行的tcp测例

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>

#define PORT 12345
#define BUF_SIZE 1024

int main() {
    int sockfd;
    struct sockaddr_in servaddr;
    char buffer[BUF_SIZE];
    const int opt = 1;

    // 创建TCP套接字
```

```
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    // 设置SO_REUSEADDR选项
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
        perror("setsockopt failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // 第一次绑定
    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
{
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    printf("TCP server bound to port %d.\n", PORT);

    // 开始监听
    if (listen(sockfd, 5) < 0) {
        perror("listen failed");
        exit(EXIT_FAILURE);
    }

    while (1) {
        int connfd;
        struct sockaddr_in cliaddr;
        socklen_t len = sizeof(cliaddr);

        // 等待客户端连接
        connfd = accept(sockfd, (struct sockaddr *)&cliaddr, &len);
        if (connfd < 0) {
            perror("accept failed");
            exit(EXIT_FAILURE);
        }

        // 接收数据
        int n = read(connfd, buffer, sizeof(buffer));
        buffer[n] = '\0';
        printf("Message from client: %s\n", buffer);

        // 发送回应
        write(connfd, buffer, strlen(buffer));
        printf("Message sent to client.\n");

        // 关闭连接
        close(connfd);
    }
```

```
    close(sockfd);
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>

#define PORT 12345
#define BUF_SIZE 1024

int main() {
    int sockfd;
    struct sockaddr_in servaddr;
    char buffer[BUF_SIZE];
    const int opt = 1;

    // 创建TCP套接字
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    // 设置SO_REUSEADDR选项
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
        perror("setsockopt failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // 第一次绑定
    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
{
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    printf("TCP server bound to port %d.\n", PORT);

    // 开始监听
```

```
    if (listen(sockfd, 5) < 0) {
        perror("listen failed");
        exit(EXIT_FAILURE);
    }

    while (1) {
        int connfd;
        struct sockaddr_in cliaddr;
        socklen_t len = sizeof(cliaddr);

        // 等待客户端连接
        connfd = accept(sockfd, (struct sockaddr *)&cliaddr, &len);
        if (connfd < 0) {
            perror("accept failed");
            exit(EXIT_FAILURE);
        }

        // 接收数据
        int n = read(connfd, buffer, sizeof(buffer));
        buffer[n] = '\0';
        printf("Message from client: %s\n", buffer);

        // 发送回应
        write(connfd, buffer, strlen(buffer));
        printf("Message sent to client.\n");

        // 关闭连接
        close(connfd);
    }

    close(sockfd);
    return 0;

}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>

int main() {
    int server_sock, client_sock;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_len;
    char buffer[1024];

    // Create TCP socket
    server_sock = socket(AF_INET, SOCK_STREAM, 0);
```

```
    if (server_sock < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set SO_REUSEADDR option
    int reuse = 1;
    if (setsockopt(server_sock, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse))
< 0) {
        perror("Set SO_REUSEADDR failed");
        exit(EXIT_FAILURE);
    }

    // Prepare server address
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(12345); // Port number

    // Bind socket to address
    if (bind(server_sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_sock, 5) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    printf("TCP server listening on port 12345 with SO_REUSEADDR enabled\n");

    // Accept incoming connections
    client_addr_len = sizeof(client_addr);
    client_sock = accept(server_sock, (struct sockaddr *)&client_addr,
&client_addr_len);
    if (client_sock < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    // Receive data from client
    memset(buffer, 0, sizeof(buffer));
    ssize_t bytes_received = recv(client_sock, buffer, sizeof(buffer), 0);
    if (bytes_received < 0) {
        perror("Receive failed");
        exit(EXIT_FAILURE);
    }

    printf("Received message from client: %s\n", buffer);

    // Close sockets
    close(client_sock);
```

```
    close(server_sock);

    return 0;

}
```

tcp最终测例

1. 测试允许端口重用

当一个 TCP 连接关闭后，连接的端口会进入 TIME_WAIT 状态，此时端口虽然没有被使用，但仍然不能立即重新绑定。启用 SO_REUSEADDR 选项后，可以在该端口仍处于 TIME_WAIT 状态时，重新绑定该端口

server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <stdbool.h>

#define PORT 12345
#define BUF_SIZE 1024

void error(const char *msg) {
    perror(msg);
    exit(1);
}

void test_reuseaddr(bool is_reuse) {
    int sock1, sock2;
    int enable = 1;
    struct sockaddr_in addr;
    char buffer[BUF_SIZE];


    sock1 = socket(AF_INET, SOCK_STREAM, 0);
    setsockopt(sock1, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(enable));
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
    addr.sin_port = htons(PORT);
    if (bind(sock1, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
```

```
        error("Bind failed for socket 1");
    }
    printf("Socket 1 bound successfully to port %d\n", PORT);

    listen(sock1, 5);
    int connfd;
    struct sockaddr_in cliaddr;
    socklen_t len = sizeof(cliaddr);
    connfd = accept(sock1, (struct sockaddr *)&cliaddr, &len);

    sleep(1);
    close(sock1);
    printf("Socket 1 closed\n");

    sock2 = socket(AF_INET, SOCK_STREAM, 0);
    if (is_reuse) {
        setsockopt(sock2, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(enable));
    }
    if (bind(sock2, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        error("Bind failed for socket 2");
    }
    printf("Socket 2 bound successfully to port %d\n", PORT);
    close(sock2);

}

int main() {
    bool is_reuse = true;   // 同时测试true和false两种情况在Starry中显示比较混乱，因此手动设置打开或关闭SO_REUSEADDR
    test_reuseaddr(is_reuse);
    return 0;
}
```

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define PORT 12345
#define SERVER_IP "127.0.0.1"

void error(const char *msg) {
    perror(msg);
    exit(0);
}
```

```
int main() {
    int sockfd, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR connecting");

    printf("Connected to server on %s:%d\n", SERVER_IP, PORT);
    close(sockfd);
    return 0;

}
```

2. 允许多个套接字绑定到同一个端口

在某些多播协议的应用中，多个套接字可以绑定到同一个端口，使用 `SO_REUSEADDR` 可以实现这一点。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <stdbool.h>

#define PORT 12345

void error(const char *msg) {
    perror(msg);
    exit(1);
}

void test_multi_bind(bool is_reuse) {
    int sock1, sock2;
    int enable = 1;
    struct sockaddr_in addr;

    sock1 = socket(AF_INET, SOCK_STREAM, 0);
    setsockopt(sock1, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(enable));
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
    addr.sin_port = htons(PORT);
    if (bind(sock1, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        error("Bind failed");
```

```
    }

    sock2 = socket(AF_INET, SOCK_STREAM, 0);
    if (is_reuse) {
        setsockopt(sock2, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(enable));
    }
    if (bind(sock2, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
        error("ERROR on binding second socket");
    }
    printf("Both sockets bound successfully to port %d\n", PORT);
    close(sock1);
    close(sock2);

}

int main() {
    bool is_reuse = true;       // 同时测试true和false两种情况在Starry中显示比较混乱，因此手动设置打开或关闭SO_REUSEADDR
    test_multi_bind(is_reuse);
    return 0;
}
```

```
/ # ./tcp1 &
/ #          9: file=./tcp1 [0];  generating link map
         9:       dynamic: 0x0000000000004d40  base: 0x0000000000001000   size: 0x0000000000004018
         9:         entry: 0x0000000000002280  phdr: 0x0000000000001040  phnum:                 13
         9:
         9:
         9:     file=libc.so.6 [0];  needed by ./tcp1 [0]
         9:     file=libc.so.6 [0];  generating link map
         9:       dynamic: 0x0000000000208940  base: 0x0000000000006000   size: 0x0000000000211d90
         9:         entry: 0x0000000000030390  phdr: 0x0000000000006040  phnum:                 14
         9:
         9:
         9:     calling init: /lib64/ld-linux-x86-64.so.2
         9:
         9:
         9:     calling init: /lib/libc.so.6
         9:
         9:
         9:     initialize program: ./tcp1
         9:
         9:
         9:     transferring control: ./tcp1
         9:
[ 16.645883 0:10 axnet::smoltcp_impl::tcp:272] bind tcp

/ # busybox telnet 127.0.0.1 12345
Connected to 127.0.0.1
ad
[ 33.335265 0:10 axnet::smoltcp_impl::tcp:272] bind tcp
         9:
         9:     calling fini:  [0]
         9:
         9:
         9:     calling fini: /lib/libc.so.6 [0]
         9:
         9:
         9:     calling fini: /lib64/ld-linux-x86-64.so.2 [0]
         9:
Socket 1 bound successfully to port 12345
Message from client: ad

Socket 1 closed
Socket 2 bound successfully to port 12345
```

```
/ # ./tcp1 &
/ #          9: file=./tcp1 [0];  generating link map
         9:       dynamic: 0x0000000000004d40  base: 0x0000000000001000   size: 0x0000000000004018
         9:         entry: 0x0000000000002280  phdr: 0x0000000000001040  phnum:                 13
         9:
         9:
         9:     file=libc.so.6 [0];  needed by ./tcp1 [0]
         9:     file=libc.so.6 [0];  generating link map
         9:       dynamic: 0x0000000000208940  base: 0x0000000000006000   size: 0x0000000000211d90
         9:         entry: 0x0000000000030390  phdr: 0x0000000000006040  phnum:                 14
         9:
         9:
         9:     calling init: /lib64/ld-linux-x86-64.so.2
         9:
         9:
         9:     calling init: /lib/libc.so.6
         9:
         9:
         9:     initialize program: ./tcp1
         9:
         9:
         9:     transferring control: ./tcp1
         9:

/ # busybox telnet 127.0.0.1 12345
Connected to 127.0.0.1
555
Bind failed for socket 2: Operation not permitted
         9:
         9:     calling fini:  [0]
         9:
         9:
         9:     calling fini: /lib/libc.so.6 [0]
         9:
         9:
         9:     calling fini: /lib64/ld-linux-x86-64.so.2 [0]
         9:
Socket 1 bound successfully to port 12345
Message from client: 555

Socket 1 closed
```

```
/ # ./tcp_noreuseaddr
        9:     file=./tcp_noreuseaddr [0];  generating link map
        9:       dynamic: 0x0000000000004d78  base: 0x0000000000001000   size: 0x0000000000004018
        9:         entry: 0x00000000000021a0  phdr: 0x0000000000001040  phnum:                 13
        9:
        9:
        9:     file=libc.so.6 [0];  needed by ./tcp_noreuseaddr [0]
        9:     file=libc.so.6 [0];  generating link map
        9:       dynamic: 0x000000000020a940  base: 0x0000000000008000   size: 0x0000000000211d90
        9:         entry: 0x0000000000032390  phdr: 0x0000000000008040  phnum:                 14
        9:
        9:
        9:     calling init: /lib64/ld-linux-x86-64.so.2
        9:
        9:
        9:     calling init: /lib/libc.so.6
        9:
        9:
        9:     initialize program: ./tcp_noreuseaddr
        9:
        9:
        9:     transferring control: ./tcp_noreuseaddr
        9:
[248.640808 0:10 linux_syscall_api::syscall_net::imp:71] [bind()] binding socket 4 to Endpoint { addr: Ipv4(Address([127, 0, 0, 1])), port: 12345 }
[248.641335 0:10 axnet::smoltcp_impl::tcp:273] bind tcp
[248.711336 0:10 linux_syscall_api::syscall_net::imp:71] [bind()] binding socket 5 to Endpoint { addr: Ipv4(Address([127, 0, 0, 1])), port: 12345 }
[248.711867 0:10 axnet::smoltcp_impl::tcp:273] bind tcp
[248.712071 0:10 axnet::smoltcp_impl::tcp:275] Check tcp if the address is already in use
[248.712395 0:10 axnet::smoltcp_impl:131] bind_check
[248.712620 0:10 axnet::smoltcp_impl:132] sockets = SocketSet { sockets: Owned([SocketStorage { inner: Some(Item { meta: Meta { handle: SocketHandle(0
[253.376849 0:10 axnet::smoltcp_impl:134] loop
[253.377069 0:10 axnet::smoltcp_impl:135] item.1 = Tcp(Socket { state: Closed, timer: Idle { keep_alive_at: None }, rtte: RttEstimator { rtt: 300, dev
[254.390923 0:10 axnet::smoltcp_impl:138] Some(addr) = Some(Ipv4(Address([127, 0, 0, 1])))
[254.391255 0:10 axnet::smoltcp_impl:140] local_addr = ListenEndpoint { addr: Some(Ipv4(Address([127, 0, 0, 1]))), port: 12345 }
[254.391752 0:10 axnet::smoltcp_impl:142] equal
ERROR on binding second socket: Operation not permitted
        9:
        9:     calling fini:  [0]
        9:
        9:
        9:     calling fini: /lib/libc.so.6 [0]
        9:
        9:
        9:     calling fini: /lib64/ld-linux-x86-64.so.2 [0]
        9:
Listening on port 12345
```