
Subject: [PATCH (resubmit)] Fix inet_diag.ko register vs rcv race
Posted by [Pavel Emelianov](#) on Thu, 29 Nov 2007 13:01:25 GMT
[View Forum Message](#) <> [Reply to Message](#)

The following race is possible when one cpu unregisters the handler while other one is trying to receive a message and call this one:

```
CPU1:                                CPU2:
inet_diag_rcv()                      inet_diag_unregister()
mutex_lock(&inet_diag_mutex);
netlink_rcv_skb(skb, &inet_diag_rcv_msg);
if (inet_diag_table[nlh->nmsg_type] ==
    NULL) /* false handler is still registered */
...
netlink_dump_start(idiagnl, skb, nlh,
    inet_diag_dump, NULL);
cb = kzalloc(sizeof(*cb), GFP_KERNEL);
/* sleep here freeing memory
 * or preempt
 * or sleep later on nlk->cb_mutex
 */
...
spin_lock(&inet_diag_register_lock);
inet_diag_table[type] = NULL;
spin_unlock(&inet_diag_register_lock);
synchronize_rcu();
/* CPU1 is sleeping - RCU quiescent
 * state is passed
 */
return;
/* inet_diag_dump is finally called: */
inet_diag_dump()
handler = inet_diag_table[cb->nlh->nmsg_type];
BUG_ON(handler == NULL);
/* OOPS! While we slept the unregister has set
 * handler to NULL :(
 */
```

Grep showed, that the register/unregister functions are called from init/fini module callbacks for tcp_/dccp_diag, so it's OK to use the inet_diag_mutex to synchronize manipulations with the inet_diag_table and the access to it.

Besides, as Herbert pointed out, asynchronous dumps should hold this mutex as well, and thus, we provide the mutex as cb_mutex one.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```

diff --git a/net/ipv4/inet_diag.c b/net/ipv4/inet_diag.c
index b017073..6b3fffb 100644
--- a/net/ipv4/inet_diag.c
+++ b/net/ipv4/inet_diag.c
@@ -853,8 +853,6 @@ static void inet_diag_rcv(struct sk_buff *skb)
     mutex_unlock(&inet_diag_mutex);
 }

-static DEFINE_SPINLOCK(inet_diag_register_lock);
-
int inet_diag_register(const struct inet_diag_handler *h)
{
    const __u16 type = h->idiag_type;
@@ -863,13 +861,13 @@ int inet_diag_register(const struct inet_diag_handler *h)
    if (type >= INET_DIAG_GETSOCK_MAX)
        goto out;

- spin_lock(&inet_diag_register_lock);
+ mutex_lock(&inet_diag_mutex);
    err = -EEXIST;
    if (inet_diag_table[type] == NULL) {
        inet_diag_table[type] = h;
        err = 0;
    }
- spin_unlock(&inet_diag_register_lock);
+ mutex_unlock(&inet_diag_mutex);
out:
    return err;
}
@@ -882,11 +880,9 @@ void inet_diag_unregister(const struct inet_diag_handler *h)
    if (type >= INET_DIAG_GETSOCK_MAX)
        return;

- spin_lock(&inet_diag_register_lock);
+ mutex_lock(&inet_diag_mutex);
    inet_diag_table[type] = NULL;
- spin_unlock(&inet_diag_register_lock);
-
- synchronize_rcu();
+ mutex_unlock(&inet_diag_mutex);
}
EXPORT_SYMBOL_GPL(inet_diag_unregister);

@@ -901,7 +897,7 @@ static int __init inet_diag_init(void)
    goto out;

    idiaogl = netlink_kernel_create(&init_net, NETLINK_INET_DIAG, 0,

```

```
-    inet_diag_rcv, NULL, THIS_MODULE);  
+    inet_diag_rcv, &inet_diag_mutex, THIS_MODULE);  
    if (idiagnl == NULL)  
        goto out_free_table;  
    err = 0;
```
