Subject: [PATCH net-2.6.25][NETNS]: Fix race between put_net() and netlink_kernel_create().
Posted by Pavel Emelianov on Thu, 24 Jan 2008 13:15:13 GMT
View Forum Message <> Reply to Message

The comment about "race free view of the set of network
namespaces" was a bit hasty. Look (there even can be only
one CPU, as discovered by Alexey Dobriyan and Denis Lunev):

```
put_net()
  if (atomic_dec_and_test(&net->refcnt))
    /* true */
      __put_net(net);
        queue_work(...);

/*
 * note: the net now has refcnt 0, but still in
 * the global list of net namespaces
 */

== re-schedule ==

register_pernet_subsys(&some_ops);
  register_pernet_operations(&some_ops);
    (*some_ops)->init(net);
      /*
       * we call netlink_kernel_create() here
       * in some places
       */
      netlink_kernel_create();
        sk_alloc();
          get_net(net); /* refcnt = 1 */
        /*
         * now we drop the net refcount not to
         * block the net namespace exit in the
         * future (or this can be done on the
         * error path)
         */
        put_net(sk->sk_net);
            if (atomic_dec_and_test(&...))
                /*
                 * true. BOOOM! The net is
                 * scheduled for release twice
                 */
```

When thinking on this problem, I decided, that getting and
putting the net in init callback is wrong. If some init
callback needs to have a refcount-less reference on the struct

net, _it_ has to be careful himself, rather than relying on
the infrastructure to handle this correctly.

In case of netlink_kernel_create(), the problem is that the
sk_alloc() gets the given namespace, but passing the info
that we don't want to get it inside this call is too heavy.

Instead, I propose to crate the socket inside an init_net
namespace and then re-attach it to the desired one right
after the socket is created.

After doing this, we also have to be careful on error paths
not to drop the reference on the namespace, we didn't get
the one on.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Acked-by: Denis Lunev <den@openvz.org>

```
diff --git a/net/netlink/af_netlink.c b/net/netlink/af_netlink.c
index 6b178e1..ff9fb6b 100644
--- a/net/netlink/af_netlink.c
+++ b/net/netlink/af_netlink.c
@@ -1344,6 +1344,22 @@ static void netlink_data_ready(struct sock *sk, int len)
  * queueing.
  */

+static void __netlink_release(struct sock *sk)
+{
+ /*
+  * Last sock_put should drop referrence to sk->sk_net. It has already
+  * been dropped in netlink_kernel_create. Taking referrence to stopping
+  * namespace is not an option.
+  * Take referrence to a socket to remove it from netlink lookup table
+  * _alive_ and after that destroy it in the context of init_net.
+  */
+
+ sock_hold(sk);
+ sock_release(sk->sk_socket);
+ sk->sk_net = get_net(&init_net);
+ sock_put(sk);
+}
+
 struct sock *
 netlink_kernel_create(struct net *net, int unit, unsigned int groups,
        void (*input)(struct sk_buff *skb),
@@ -1362,8 +1378,18 @@ netlink_kernel_create(struct net *net, int unit, unsigned int groups,
```

```
  if (sock_create_lite(PF_NETLINK, SOCK_DGRAM, unit, &sock))
   return NULL;

- if (__netlink_create(net, sock, cb_mutex, unit) < 0)
-  goto out_sock_release;
+ /*
+  * We have to just have a reference on the net from sk, but don't
+  * get_net it. Besides, we cannot get and then put the net here.
+  * So we create one inside init_net and the move it to net.
+  */
+
+ if (__netlink_create(&init_net, sock, cb_mutex, unit) < 0)
+  goto out_sock_release_nosk;
+
+ sk = sock->sk;
+ put_net(sk->sk_net);
+ sk->sk_net = net;

  if (groups < 32)
   groups = 32;
@@ -1372,7 +1398,6 @@ netlink_kernel_create(struct net *net, int unit, unsigned int groups,
  if (!listeners)
   goto out_sock_release;

- sk = sock->sk;
  sk->sk_data_ready = netlink_data_ready;
  if (input)
   nlk_sk(sk)->netlink_rcv = input;
@@ -1395,14 +1420,14 @@ netlink_kernel_create(struct net *net, int unit, unsigned int groups,
   nl_table[unit].registered++;
  }
  netlink_table_ungrab();
-
- /* Do not hold an extra referrence to a namespace as this socket is
-  * internal to a namespace and does not prevent it to stop. */
- put_net(net);
  return sk;

 out_sock_release:
  kfree(listeners);
+ __netlink_release(sk);
+ return NULL;
+
+out_sock_release_nosk:
  sock_release(sock);
  return NULL;
 }
@@ -1415,18 +1440,7 @@ netlink_kernel_release(struct sock *sk)
```

```
  if (sk == NULL || sk->sk_socket == NULL)
   return;

- /*
-  * Last sock_put should drop referrence to sk->sk_net. It has already
-  * been dropped in netlink_kernel_create. Taking referrence to stopping
-  * namespace is not an option.
-  * Take referrence to a socket to remove it from netlink lookup table
-  * _alive_ and after that destroy it in the context of init_net.
-  */
- sock_hold(sk);
- sock_release(sk->sk_socket);
-
- sk->sk_net = get_net(&init_net);
- sock_put(sk);
+ __netlink_release(sk);
 }
 EXPORT_SYMBOL(netlink_kernel_release);
```