
Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.

Posted by [paulmck](#) on Fri, 11 Apr 2008 16:27:52 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Fri, Apr 11, 2008 at 07:45:06PM +0400, Pavel Emelyanov wrote:

> Paul E. McKenney wrote:

> > On Fri, Apr 11, 2008 at 11:55:59AM +0400, Pavel Emelyanov wrote:

> > > Paul E. McKenney wrote:

> > > > On Thu, Apr 10, 2008 at 07:06:24PM +0400, Pavel Emelyanov wrote:

> > > > > This is the first step in making tuntap devices work in net

> > > > namespaces. The structure mentioned is pointed by generic

> > > > net pointer with tun_net_id id, and tun driver fills one on

> > > > its load. It will contain only the tun devices list.

> > > >

> > > > So declare this structure and introduce net init and exit hooks.

> > > OK, I have to ask... What prevents someone else from invoking

> > > net_generic() concurrently with a call to tun_exit_net(), potentially

> > > obtaining a pointer to the structure that tun_exit_net() is about

> > > to kfree()?

> > It's the same as if the tun_net was directly pointed by the struct

> > net. Nobody can grant, that the pointer got by you from the struct

> > net is not going to become free, unless you provide this security

> > by yourself.

> >

> > So tun_net acquires some lock before calling net_generic(), and that

> > same lock is held when calling tun_exit_net()? Or is there but a

>

> No.

>

> > single tun_net task, so that it will never call tun_net_exit()

> > at the same time that it calls net_generic() for the tun_net pointer?

>

> tun_net_exit is called only when a struct net is no longer referenced

> and is going to be kfree-ed itself, so it's impossible (or BUGy by its

> own) that someone still has a pointer on this net.

>

> Providing the struct net is alive (!), the net->gen array is alive (or

> is scheduled for kfree after RCU grace period). Thus, if your code

> holds the net and uses the net_generic() call, then it will get alive

> net->gen array and alive tun_net pointer.

>

> Next, what happens after net_generic() completes and leaves the RCU-read

> section? Simple - the struct net is (should be) still referenced, so the

> tun_net_exit cannot yet be called and thus the tun_net pointer obtained

> earlier is alive. Unlike the (possibly) former instance of the net_generic

> array, but nobody references this one in my code (and should not do so,

> hm... I think I'll add this rule to the comments).

>

> >> But if you call net_generic to get some pointer other than tun_net,
> >> then you're fine (due to RCU), providing you play the same rules with
> >> the pointer you're getting.
> >
> > Agreed, RCU protects the net_generic structure, but not the structures
> > pointed to by that structure.
>
> They are protected by struct net reference counting.

Ah, OK, got it! Thank you for the tutorial!

> >> Maybe I'm missing something in your question, can you provide some
> >> testcase, that you suspect may cause an OOPS?
> >
> > Just trying to understand what prevents one task from calling
> > net_generic() to pick up the tun_net pointer at the same time some other
> > task calls tun_net_exit().
>
> If this task dereferences a "held" struct net, then should be OK. If
> this task does not, this will OOPs in any case.
>
> Consider the struct net to look like
>
> struct net {
> ...
> void *ptrs[N];
> }
>
> and the net_generic to be just
>
> static inline void net_generic(struct net *net, int id)
> {
> BUG_ON(id >= N);
> return net->ptrs[id - 1];
> }
>
> That's the same to what I propose, except for the ptrs array is on the
> RCU protected memory.

So RCU is protecting -only- the net_generic structure that net_generic()
is traversing, and the structure returned by net_generic() is protected
by a reference counter in the upper-level struct net.

If this is the approach, I am happy. ;-)

Thanx, Paul

Containers mailing list

