

---

Subject: Re: [PATCH 03/16] net: Basic network namespace infrastructure.  
Posted by [ebiederm](#) on Sun, 09 Sep 2007 10:04:45 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

"Paul E. McKenney" <paulmck@linux.vnet.ibm.com> writes:

> On Sat, Sep 08, 2007 at 03:15:34PM -0600, Eric W. Biederman wrote:  
>>  
>> This is the basic infrastructure needed to support network  
>> namespaces. This infrastructure is:  
>> - Registration functions to support initializing per network  
>> namespace data when a network namespaces is created or destroyed.  
>>  
>> - struct net. The network namespace data structure.  
>> This structure will grow as variables are made per network  
>> namespace but this is the minimal starting point.  
>>  
>> - Functions to grab a reference to the network namespace.  
>> I provide both get/put functions that keep a network namespace  
>> from being freed. And hold/release functions serve as weak references  
>> and will warn if their count is not zero when the data structure  
>> is freed. Useful for dealing with more complicated data structures  
>> like the ipv4 route cache.  
>>  
>> - A list of all of the network namespaces so we can iterate over them.  
>>  
>> - A slab for the network namespace data structure allowing leaks  
>> to be spotted.  
>  
> If I understand this correctly, the only way to get to a namespace is  
> via get\_net\_ns\_by\_pid(), which contains the rcu\_read\_lock() that matches  
> the rcu\_barrier() below.

Not quite. That is the convoluted case for getting a namespace someone else is using. current->nsproxy->net\_ns works and should require no locking to read (only the current process may modify it) and does hold a reference to the network namespace. Similarly for sock->sk\_net.

> So, is the get\_net() in sock\_copy() in this patch adding a reference to  
> an element that is guaranteed to already have at least one reference?

Yes.

> If not, how are we preventing sock\_copy() from running concurrently with  
> cleanup\_net()? Ah, I see -- in sock\_copy() we are getting a reference  
> to the new struct sock that no one else can get a reference to, so OK.  
> Ditto for the get\_net() in sk\_alloc().

> But I still don't understand what is protecting the get\_net() in  
> dev\_seq\_open(). Is there an existing reference?

Sort of. The directories under /proc/net are created when create a network namespace and they are destroyed when the network namespace is removed. And those directories remember which network namespace they are for and that is what dev\_seq\_open is referencing.

So the tricky case what happens if we open a directory under /proc/net as we are cleaning up a network namespace.

> If so, how do we know  
> that it won't be removed just as we are trying to add our reference  
> (while at the same time cleanup\_net() is running)? Ditto for the other  
> \_open() operations in the same patch. And for netlink\_seq\_open().  
>  
> Enlightenment?

Good spotting. It looks like you have found a legitimate race. Grr. I thought I had a reference to the network namespace there. I need to step back and think about this a bit, and see if I can come up with a legitimate idiom.

I know the network namespace exists and I have not finished cleanup\_net because I can still get to the /proc entries.

I know I cannot use get\_net for the reference in in /proc because otherwise I could not release the network namespace unless I was to unmount the filesystem, which is not a desirable property.

I think I can change the idiom to:

```
struct net *maybe_get_net(struct net *net)
{
    if (!atomic_inc_not_zero(&net->count))
        net = NULL;
    return net;
}
```

Which would make dev\_seq\_open be:

```
static int dev_seq_open(struct inode *inode, struct file *file)
{
    struct seq_file *seq;
    int res;
    res = seq_open(file, &dev_seq_ops);
    if (!res) {
        seq = file->private_data;
```

```
seq->private = maybe_get_net(PROC_NET(inode));
if (!seq->private) {
    res = -ENOENT;
    seq_release(inode, file);
}
}
return res;
}
```

I'm still asking myself if I need any kind of locking to ensure struct net does not go away in the mean time, if so rcu\_read\_lock() should be sufficient.

I will read through the generic proc code very carefully after I have slept and see if there is what I the code above is sufficient, and if so update the patchset.

Eric

---

Containers mailing list  
Containers@lists.linux-foundation.org  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---