Subject: [RFC PATCH 0/31] An introduction and A path for merging network namespace work
Posted by ebiederm on Thu, 25 Jan 2007 18:55:04 GMT
View Forum Message <> Reply to Message

The idea of a network namespace is fundamentally quite simple. We create a mechanism that from the users perspective allows creation of separate instances of the network stack. When combined with mechanism like chroot this results in a much more complete isolation. When seen in the context of application migration this allows for taking your IP address and other global identifiers with you.

What does this mean in the context of the networking stack? The basic idea is to tag processes with a network namespace that is used when they create new sockets or otherwise initiate a new fresh communication with the networking stack. The idea is to tag all sockets with a network namespace they will always be in and all operations on them will be relative to. The idea is to tag all network devices with a network namespace they are a member of, but may be changed during the lifetime of a device.

Mostly a network namespace at it's most basic level is about names. It is about creating a view of the networking stack where you can name the network devices that are members anything you want. Likewise for iptables rules and all of the rest of the state. It is a lot like creating a new directory in a filesystem. The underlying data structures don't really change just the users view of those data structures, and we continue to have a single network stack.

My goal today is that even if we can't agree on a specific set of patches that we come to an agreement on roughly what those patches should accomplish, and what process we should go through to get them merged.

For implementing a network namespace the core problem is that there is a lot of networking code, and it is continually evolving. This means that the task of implementing a network namespace is not a small one, a lot of code must be read, touched and updated, while hoping someone doesn't change something important before you get your changes in. To do this sanely means we need an incremental path to our goal, that allows small pieces to be reviewed and merged as they are ready.

The path I am recommending today is to first lay down some basic infrastructure. Then one layer at a time modify the existing code to handle multiple simultaneous network namespaces but to modify each component of that layer to refuse to operate in the context of anything but the initial network namespace, thus preventing code that has not yet been updated with situations it does not know how to deal with.

Eventually this will get down to the real meat of the problem and practical things like ipv4 sockets will work.

This should allow for a network stack that compiles, builds and works at each step of the way. Not too far into the process support for multiple network namespaces that works should be available with the limitation that except for the initial network namespace all of the rest will look like a kernel with most parts of the networking stack compiled out, but within those parts that are present it should be fully useable.

To make my thinking clear I have provided a initial patchset, that makes quite a bit of progress especially in laying the ground work. My goal is to have the question does this basic path make sense?

To that end I have omitted posting some of the prerequisite cleanup and infrastructure patches (like my sysctl work), that are just noise in this context, and I have failed to rebase my patchset against Dave Miller's latest networking tree. Those are important details but they are not important to this conversation.

If my basic path and the basic patches look like they are heading in the right direction we can start moving towards what needs to happen to ensure a review of the patches, and what we need to do to start merging them. If the basic path does not appear reasonable well that would be good to know as well.

There are essentially two different approaches to modify networking code to handle multiple network namesspaces. Either all of the global variables can be replicated once for each network namespace and we

build up parallel namespace specific data structures. Or the data
elements in the data structure are tagged, with what namespace they
belong to and we filter them. It depends on the context which
is most appropriate and easier. As a general rule large hash tables
call for filtering and a small global variable set calls for simply
having multiple instances of the data structure.

The biggest intrusion I expect to see in the logic of the networking
stack is initialization and tear down. As we need to initialize
and clean up all of those per network namespace variables when
we create and destroy and network namespace.

A git tree with all of my patches against 2.6.20-rc5 is available at:
git://git.kernel.org/pub/scm/linux/kernel/git/ebiederm/linux-2.6-netns.git

In addition to what I have posted here and all of it's prerequisites
the tree includes further patches that get the basics of ipv4 and
iptables working. So people who are interested actually have
something more or less useful to play with.

At a big practical level what I don't yet see is how exactly the
inifiniband/rdma network subsystem fits into network namespaces yet.
Not at the ipoib layer but at the native layer. I think I want the
ability to say each pkey of each IB device can potentially be in
a different namespace or possibly each different queue pair. Suggestions
are welcome. I don't quite have my head wrapped around that the user
space API there yet.

I suppose on the infiniband/rdma side I should dig up all interactions
with user space and simply fail if that user is not in the initial
network namespace as a start. At the very least this is necessary
given how many calls the connection manager makes into the IP stack.

Eric

_____