Subject: [PATCH 10/12] L2 network namespace: playing with pass-through device
Posted by Mishin Dmitry on Wed, 06 Dec 2006 22:29:39 GMT
View Forum Message <> Reply to Message

Temporary code to debug and play with pass-through device.
Create device pair by
 modprobe veth
	echo 'add veth1 0:1:2:3:4:1 eth0 0:1:2:3:4:2' >/proc/net/veth_ctl
and your shell will appear into a new namespace with `eth0' device.
Configure device in this namespace
	ip l s eth0 up
	ip a a 1.2.3.4/24 dev eth0
and in the root namespace
	ip l s veth1 up
	ip a a 1.2.3.1/24 dev veth1
to establish a communication channel between root namespace and the newly
created one.

Code is done by Andrey Savochkin and ported by me over Cedric'c patchset

Signed-off-by: Dmitry Mishin <dim@openvz.org>

```
---
 drivers/net/veth.c           | 121 +++++++++++++++++++++++++++++++++++++++++++
 fs/proc/array.c              |   8 ++
 include/linux/net_namespace.h |  10 +++
 kernel/nsproxy.c             |   2
 net/core/net_namespace.c     |  28 +++++++++
 5 files changed, 168 insertions(+), 1 deletion(-)

--- linux-2.6.19-rc6-mm2.orig/drivers/net/veth.c
+++ linux-2.6.19-rc6-mm2/drivers/net/veth.c
@@ -12,6 +12,7 @@
 #include <linux/etherdevice.h>
 #include <linux/proc_fs.h>
 #include <linux/seq_file.h>
+#include <linux/syscalls.h>
 #include <net/dst.h>
 #include <net/xfrm.h>

@@ -245,6 +246,123 @@ void veth_entry_del_all(void)

 /* ------------------------------------------------------------------- *
  *
+ * Temporary interface to create veth devices
+ *
+ * ------------------------------------------------------------------- */
+
```

```
+#ifdef CONFIG_PROC_FS
+
+static int veth_debug_open(struct inode *inode, struct file *file)
+{
+ return 0;
+}
+
+static char *parse_addr(char *s, char *addr)
+{
+ int i, v;
+
+ for (i = 0; i < ETH_ALEN; i++) {
+  if (!isxdigit(*s))
+   return NULL;
+  *addr = 0;
+  v = isdigit(*s) ? *s - '0' : toupper(*s) - 'A' + 10;
+  s++;
+  if (isxdigit(*s)) {
+   *addr += v << 16;
+   v = isdigit(*s) ? *s - '0' : toupper(*s) - 'A' + 10;
+   s++;
+  }
+  *addr++ += v;
+  if (i < ETH_ALEN - 1 && ispunct(*s))
+   s++;
+ }
+ return s;
+}
+
+static ssize_t veth_debug_write(struct file *file, const char __user *user_buf,
+  size_t size, loff_t *ppos)
+{
+ char buf[128], *s, *parent_name, *child_name;
+ char parent_addr[ETH_ALEN], child_addr[ETH_ALEN];
+ struct net_namespace *parent_ns, *child_ns;
+ int err;
+
+ s = buf;
+ err = -EINVAL;
+ if (size >= sizeof(buf))
+  goto out;
+ err = -EFAULT;
+ if (copy_from_user(buf, user_buf, size))
+  goto out;
+ buf[size] = 0;
+
+ err = -EBADRQC;
+ if (!strncmp(buf, "add ", 4)) {
```

```
+  parent_name = buf + 4;
+  if ((s = strchr(parent_name, ' ')) == NULL)
+   goto out;
+  *s = 0;
+  if ((s = parse_addr(s + 1, parent_addr)) == NULL)
+   goto out;
+  if (!*s)
+   goto out;
+  child_name = s + 1;
+  if ((s = strchr(child_name, ' ')) == NULL)
+   goto out;
+  *s = 0;
+  if ((s = parse_addr(s + 1, child_addr)) == NULL)
+   goto out;
+
+  get_net_ns(current_net_ns);
+  parent_ns = current_net_ns;
+  if (*s == ' ') {
+   unsigned int id;
+   id = simple_strtoul(s + 1, &s, 0);
+   err = sys_bind_ns(id, NS_ALL);
+  } else
+   err = sys_unshare_ns(NS_NET);
+  if (err)
+   goto out;
+  /* after bind_ns() or unshare_ns() namespace is changed */
+  get_net_ns(current_net_ns);
+  child_ns = current_net_ns;
+  err = veth_entry_add(parent_name, parent_addr, parent_ns,
+   child_name, child_addr, child_ns);
+  if (err) {
+   put_net_ns(child_ns);
+   put_net_ns(parent_ns);
+  } else
+   err = size;
+ }
+out:
+ return err;
+}
+
+static struct file_operations veth_debug_ops = {
+ .open = &veth_debug_open,
+ .write = &veth_debug_write,
+};
+
+static int veth_debug_create(void)
+{
+ proc_net_fops_create("veth_ctl", 0200, &veth_debug_ops);
```

```
+ return 0;
+}
+
+static void veth_debug_remove(void)
+{
+ proc_net_remove("veth_ctl");
+}
+
+#else
+
+static int veth_debug_create(void) { return -1; }
+static void veth_debug_remove(void) { }
+
+#endif
+
+/* ---------------------------------------------------------------- *
+ *
  * Information in proc
  *
  * ---------------------------------------------------------------- */
@@ -304,12 +422,15 @@ static inline void veth_proc_remove(void

 int __init veth_init(void)
 {
+ if (veth_debug_create())
+  return -EINVAL;
  veth_proc_create();
  return 0;
 }

 void __exit veth_exit(void)
 {
+ veth_debug_remove();
  veth_proc_remove();
  veth_entry_del_all();
 }
--- linux-2.6.19-rc6-mm2.orig/fs/proc/array.c
+++ linux-2.6.19-rc6-mm2/fs/proc/array.c
@@ -72,6 +72,7 @@
 #include <linux/highmem.h>
 #include <linux/file.h>
 #include <linux/times.h>
+#include <linux/net_namespace.h>
 #include <linux/cpuset.h>
 #include <linux/rcupdate.h>
 #include <linux/delayacct.h>
@@ -198,6 +199,13 @@ static inline char * task_state(struct t
  put_group_info(group_info);
```

```
  buffer += sprintf(buffer, "\n");
+
+#ifdef CONFIG_NET_NS
+ if (p == current)
+  buffer += sprintf(buffer, "NetContext: %u\n",
+    p->nsproxy->net_ns->id);
+#endif
+
  return buffer;
 }

--- linux-2.6.19-rc6-mm2.orig/include/linux/net_namespace.h
+++ linux-2.6.19-rc6-mm2/include/linux/net_namespace.h
@@ -24,6 +24,9 @@ struct net_namespace {
  int   fib4_trie_last_dflt;
 #endif
  unsigned int  hash;
+ struct net_namespace *parent;
+ struct list_head child_list, sibling_list;
+ unsigned int  id;
 };

 extern struct net_namespace init_net_ns;
@@ -69,6 +72,8 @@ static inline void pop_net_ns(struct net

 #define net_ns_hash(ns)   ((ns)->hash)

+extern struct net_namespace *find_net_ns(unsigned int id);
+
 #else /* CONFIG_NET_NS */

 #define INIT_NET_NS(net_ns)
@@ -110,6 +115,11 @@ static inline void pop_net_ns(struct net

 #define net_ns_hash(ns)   (0)

+static inline struct net_namespace *find_net_ns(unsigned int id)
+{
+ return NULL;
+}
+
 #endif /* !CONFIG_NET_NS */

 #endif /* _LINUX_NET_NAMESPACE_H */
--- linux-2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ linux-2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -430,6 +430,7 @@ unlock:
```

```
  put_nsproxy(ns);
  return ret;
 }
+EXPORT_SYMBOL(sys_bind_ns);

 /*
  * sys_unshare_ns - unshare one or more of namespaces which were
@@ -573,6 +574,7 @@ bad_unshare_ns_cleanup_fs:
 bad_unshare_ns_out:
   return err;
 }
+EXPORT_SYMBOL(sys_unshare_ns);

 static int __init nshash_init(void)
 {
--- linux-2.6.19-rc6-mm2.orig/net/core/net_namespace.c
+++ linux-2.6.19-rc6-mm2/net/core/net_namespace.c
@@ -13,6 +13,8 @@
 #include <linux/netdevice.h>
 #include <net/ip_fib.h>

+static spinlock_t net_ns_list_lock = SPIN_LOCK_UNLOCKED;
+
 struct net_namespace init_net_ns = {
   .kref = {
    .refcount = ATOMIC_INIT(2),
@@ -22,6 +24,8 @@ struct net_namespace init_net_ns = {
   .dev_tail_p = &init_net_ns.dev_base_p,
   .loopback_dev_p = NULL,
   .pcpu_lstats_p = NULL,
+ .child_list = LIST_HEAD_INIT(init_net_ns.child_list),
+ .sibling_list = LIST_HEAD_INIT(init_net_ns.sibling_list),
 };

 #ifdef CONFIG_NET_NS
@@ -44,6 +48,12 @@ static struct net_namespace *clone_net_n
   ns->dev_base_p = NULL;
   ns->dev_tail_p = &ns->dev_base_p;
   ns->hash = net_random();
+ INIT_LIST_HEAD(&ns->child_list);
+ spin_lock_irq(&net_ns_list_lock);
+ get_net_ns(old_ns);
+ ns->parent = old_ns;
+ list_add_tail(&ns->sibling_list, &old_ns->child_list);
+ spin_unlock_irq(&net_ns_list_lock);

 #ifdef CONFIG_IP_MULTIPLE_TABLES
   INIT_LIST_HEAD(&ns->fib_rules_ops_list);
```

```
@@ -52,6 +62,8 @@ static struct net_namespace *clone_net_n
   goto out_fib4;
  if (loopback_init(ns))
   goto out_loopback;
+ printk(KERN_DEBUG "NET_NS: created new netcontext %p for %s "
+  "(pid=%d)\n", ns, current->comm, current->tgid);
  return ns;

 out_loopback:
@@ -95,8 +107,20 @@ int copy_net_ns(int flags, struct task_s
 void free_net_ns(struct kref *kref)
 {
  struct net_namespace *ns;
+ unsigned long flags;

+ /* taking lock after atomic_dec_and_test is racy */
+ spin_lock_irqsave(&net_ns_list_lock, flags);
  ns = container_of(kref, struct net_namespace, kref);
+ if (atomic_read(&ns->kref.refcount) ||
+     list_empty(&ns->sibling_list)) {
+  spin_unlock_irqrestore(&net_ns_list_lock, flags);
+  return;
+ }
+ list_del_init(&ns->sibling_list);
+ spin_unlock_irqrestore(&net_ns_list_lock, flags);
+ put_net_ns(ns->parent);
+
  unregister_netdev(ns->loopback_dev_p);
  if (ns->dev_base_p != NULL) {
   printk("NET_NS: BUG: namespace %p has devices! ref %d\n",
@@ -104,8 +128,10 @@ void free_net_ns(struct kref *kref)
   return;
  }
  ip_fib_struct_cleanup();
+ printk(KERN_DEBUG "NET_NS: net namespace %p (%u) destroyed\n",
+   ns, ns->id);
  kfree(ns);
 }
+/* because of put_net_ns() */
 EXPORT_SYMBOL(free_net_ns);
-
 #endif /* CONFIG_NET_NS */
```

Subject: Re: [PATCH 10/12] L2 network namespace: playing with pass-through device
Posted by Cedric Le Goater on Mon, 11 Dec 2006 15:35:14 GMT
View Forum Message <> Reply to Message

> void free_net_ns(struct kref *kref)
> {
> struct net_namespace *ns;
> + unsigned long flags;
>
> + /* taking lock after atomic_dec_and_test is racy */
> + spin_lock_irqsave(&net_ns_list_lock, flags);
> ns = container_of(kref, struct net_namespace, kref);
> + if (atomic_read(&ns->kref.refcount) ||
> + list_empty(&ns->sibling_list)) {
> + spin_unlock_irqrestore(&net_ns_list_lock, flags);
> + return;

what about the cleanup ? is it skipped ?

> + }
> + list_del_init(&ns->sibling_list);
> + spin_unlock_irqrestore(&net_ns_list_lock, flags);
> + put_net_ns(ns->parent);
> +
> unregister_netdev(ns->loopback_dev_p);
> if (ns->dev_base_p != NULL) {
> printk("NET_NS: BUG: namespace %p has devices! ref %d\n",
> @@ -104,8 +128,10 @@ void free_net_ns(struct kref *kref)
> return;
> }
> ip_fib_struct_cleanup();
> + printk(KERN_DEBUG "NET_NS: net namespace %p (%u) destroyed\n",
> + ns, ns->id);
> kfree(ns);
> }
> +/* because of put_net_ns() */
> EXPORT_SYMBOL(free_net_ns);
> -
> #endif /* CONFIG_NET_NS */

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [PATCH 10/12] L2 network namespace: playing with pass-through device

Dmitry Mishin wrote:
> On Wednesday 13 December 2006 23:27, Daniel Lezcano wrote:
>> Vlad Yasevich wrote:
>>> Hi Daniel
>>>
>>> Daniel Lezcano wrote:
>>>> Herbert Poetzl wrote:
>>>>> On Tue, Dec 12, 2006 at 04:50:50PM +0100, Daniel Lezcano wrote:
>>>>>> Dmitry Mishin wrote:
>>>>>>> On Tuesday 12 December 2006 17:19, Daniel Lezcano wrote:
>>>>>>>> Dmitry Mishin wrote:
>>>>>>>>
>>>>>>>>>>>>>>>> Why do yo need to have a child list and sibling list ?
>>>>>>>>>>>>>>>> Because of the level2<->level3 hierarchy, for example.
>>>>>>>>>>>>>>>> This hierarchy doesn't exist with ns->parent ? Do you have an example
>>>>>>>>>>>>>>>> when the hierarchy should be used ? I mean when we need to browse from
>>>>>>>>>>>>>>>> l2 -> l3 ?
>>>>>>>>>>>>>>>> For example, to check that new ifaddr is already used by child l3 namespace.
>>>>>>>>>>>>>>> The devinet isolation does already do that, you can not add a new ifaddr
>>>>>>>>>>>>>>> if it already exists. Do you have another example ?
>>>>>>>>>>>>>> Could devinet isolation provide ifaddrs list with namespaces?
>>>>>>>>>>>>>> What will be with child namespaces if you decide to destroy parent namespace?
>>>>>>>>>>>>>> If we decide to destroy them, than how we could get their list?
>>>>>>>>>>>>>> It is a question of flexibility and easy management.
>>>>>>>>>>>>>> Why do you want to remove this code?
>>>>>>>>>>>> I don't want to especially remove this code, I just want to understand
>>>>>>>>>>>> what it does and why. If it appears to be useless, let's remove it, if
>>>>>>>>>>>> it appears to be useful, let's keep it.
>>>>>>>>>>>>
>>>>>>>>>>>> By the way, what is the meaning on destroying the namespaces directly,
>>>>>>>>>>>> is it not the kref mechanism which needs to do that ? For example, if
>>>>>>>>>>>> you create a l2 namespace and after you create l3 namespaces. You want
>>>>>>>>>>>> to destroy the l2 namespace, the l2 namespace should stay "zombie" until
>>>>>>>>>>>> all the l3 namespaces exit. If you need to wipe out all the namespaces,
>>>>>>>>>>>> you should destroy all the related namespaces' ressources, like killing
>>>>>>>>>>>> all processes inside it. The namespaces will "put" their respective kref
>>>>>>>>>>>> and will trigger the freeing of the ressources.
>>>>>>>>>> networking (mostly sockets) will probably require
>>>>>>>>>> some mechanism to 'zap' them, ignoring the defined
>>>>>>>>>> timeouts. otherwise the spaces could hang around
>>>>>>>>>> for quite a while waiting for some response, which
>>>>>>>>>> might never come ...
>>>>>>>> Yes, exact. We will need a specific socket cleanup by namespace in order
>>>>>>>> to do network migration. This is the only case I see to 'zap' the sockets.
>>>>>>>> The sockets should never be flushed in other cases. For example, you
>>>>>>>> launch an application into a network namespace, it sends 10MB to a peer

>>>> and exits. The network namespace should stay "alive" until all orphans
>>>> sockets have flushed their buffers to the peer. This behavior is
>>>> perfectly handled by the kref mechanism because sock_release will "put"
>>>> the network namespace and that will trigger the network namespace
>>>> destruction.
>>>>
>>> Are you saying that you can't see the reason to be able to shutdown/destroy a
>>> given container. What if it's misbehaving or has been compromised???
>>>
>>> I would think an administrator, should be able to shutdown/destroy a
>>> given container or namespace from above or outside of such container/namespace
>>> if it's warranted. If this case, if we destroy an L2 namespace, L3 children
>>> should probably be cleaned up as well.
>> Yes, I agree, you should want to destroy a specific container. IMHO,
>> freeing it directly is not the right way, you should destroy the
>> namespaces' ressources and the namespace will be released. For example,
>> you create a l3 container and into it you spawn 10 processes and each of
>> them creates 10 connections, roughly you have 10 + 100 reference to the
>> namespace. You want to destroy this container for any reason, you kill
>> the 10 processes and you destroy the 100 connections => the ref count
>> reach zero and the namespace is released.
>> In the case of the l2 namespace having l3 childs, I think it is up to
>> the administrator to know what he does. In other words, he should kill
>> all l3 namespaces and after kill the l2 parent.
> So, you suppose, that administrator remembers which l3 namespaces are over
> this l2 namespace.

I guess it remembers which IP address has been assigned to the
namespaces, what name he gave to them, how many they are, what are the
virtual hostname, etc ... The administrator can be a human or a
container system manager.

> List of childs gives an ability to track such relations easily.
> If administrator decides to kill l2, he could decide is it force kill, when all
> underlied l3 namespaces and their processes should be killed, or soft kill,
> where only l2 refcount decremented.

If you want the list of childs to be used to browse the namespace
hierarchy in order to destroy them, what about the pid namespace,
ipc_namespace, utsname namespace, ... ? You should have the same list
for each namespace's ressources, no ? Perhaps having a child list into
the network namespace is not the right place ...

>> Imagine you have a shell and you are into the foo directory, in another
>> shell you remove the foo directory, the shell in the foo directory will
>> not be pushed outside of the directory. The foo dir will stay in a
>> "transient" state until the shell exits or change directory, I think
>> that should be same with the namespaces or better to say "containers".

> It is unacceptable in some cases. As Eric already said, network connections
> could take refcount for a long time and you will wait container stop indefinitely,
> because somebody spoofs your namespace. It is necessary to have an ability of
> force kill.

Don't me wrong, I don't say we should not have a force kill for a
container, I am just saying the ressources must be killed for the
container and that will refdown the ressources to reach the namespaces
destruction.

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [PATCH 10/12] L2 network namespace: playing with pass-through device
Posted by Mishin Dmitry on Thu, 14 Dec 2006 11:17:44 GMT
View Forum Message <> Reply to Message

On Thursday 14 December 2006 13:48, Daniel Lezcano wrote:
> Dmitry Mishin wrote:
> > On Wednesday 13 December 2006 23:27, Daniel Lezcano wrote:
> >> Vlad Yasevich wrote:
> >>> Hi Daniel
> >>>
> >>> Daniel Lezcano wrote:
> >>>> Herbert Poetzl wrote:
> >>>>> On Tue, Dec 12, 2006 at 04:50:50PM +0100, Daniel Lezcano wrote:
> >>>>>> Dmitry Mishin wrote:
> >>>>>>> On Tuesday 12 December 2006 17:19, Daniel Lezcano wrote:
> >>>>>>>> Dmitry Mishin wrote:
> >>>>>>>>
> >>>>>>>>>>>>>>>> Why do yo need to have a child list and sibling list ?
> >>>>>>>>>>>>>>>> Because of the level2<->level3 hierarchy, for example.
> >>>>>>>>>>>>>>>> This hierarchy doesn't exist with ns->parent ? Do you have an example
> >>>>>>>>>>>>>>>> when the hierarchy should be used ? I mean when we need to browse from
> >>>>>>>>>>>>>>>> l2 -> l3 ?
> >>>>>>>>>>>>>>>> For example, to check that new ifaddr is already used by child l3 namespace.
> >>>>>>>>>>>>>>>> The devinet isolation does already do that, you can not add a new ifaddr
> >>>>>>>>>>>>>>>> if it already exists. Do you have another example ?
> >>>>>>>>>>>>>>> Could devinet isolation provide ifaddrs list with namespaces?
> >>>>>>>>>>>>>>> What will be with child namespaces if you decide to destroy parent namespace?
> >>>>>>>>>>>>>>> If we decide to destroy them, than how we could get their list?
> >>>>>>>>>>>>>>> It is a question of flexibility and easy management.
> >>>>>>>>>>>>>>> Why do you want to remove this code?

> >>>>>> I don't want to especially remove this code, I just want to understand
> >>>>>> what it does and why. If it appears to be useless, let's remove it, if
> >>>>>> it appears to be useful, let's keep it.
> >>>>>>
> >>>>>> By the way, what is the meaning on destroying the namespaces directly,
> >>>>>> is it not the kref mechanism which needs to do that ? For example, if
> >>>>>> you create a l2 namespace and after you create l3 namespaces. You want
> >>>>>> to destroy the l2 namespace, the l2 namespace should stay "zombie" until
> >>>>>> all the l3 namespaces exit. If you need to wipe out all the namespaces,
> >>>>>> you should destroy all the related namespaces' ressources, like killing
> >>>>>> all processes inside it. The namespaces will "put" their respective kref
> >>>>>> and will trigger the freeing of the ressources.
> >>>>> networking (mostly sockets) will probably require
> >>>>> some mechanism to 'zap' them, ignoring the defined
> >>>>> timeouts. otherwise the spaces could hang around
> >>>>> for quite a while waiting for some response, which
> >>>>> might never come ...
> >>>> Yes, exact. We will need a specific socket cleanup by namespace in order
> >>>> to do network migration. This is the only case I see to 'zap' the sockets.
> >>>> The sockets should never be flushed in other cases. For example, you
> >>>> launch an application into a network namespace, it sends 10MB to a peer
> >>>> and exits. The network namespace should stay "alive" until all orphans
> >>>> sockets have flushed their buffers to the peer. This behavior is
> >>>> perfectly handled by the kref mechanism because sock_release will "put"
> >>>> the network namespace and that will trigger the network namespace
> >>>> destruction.
> >>>>
> >>> Are you saying that you can't see the reason to be able to shutdown/destroy a
> >>> given container. What if it's misbehaving or has been compromised???
> >>>
> >>> I would think an administrator, should be able to shutdown/destroy a
> >>> given container or namespace from above or outside of such container/namespace
> >>> if it's warranted. If this case, if we destroy an L2 namespace, L3 children
> >>> should probably be cleaned up as well.
> >> Yes, I agree, you should want to destroy a specific container. IMHO,
> >> freeing it directly is not the right way, you should destroy the
> >> namespaces' ressources and the namespace will be released. For example,
> >> you create a l3 container and into it you spawn 10 processes and each of
> >> them creates 10 connections, roughly you have 10 + 100 reference to the
> >> namespace. You want to destroy this container for any reason, you kill
> >> the 10 processes and you destroy the 100 connections => the ref count
> >> reach zero and the namespace is released.
> >> In the case of the l2 namespace having l3 childs, I think it is up to
> >> the administrator to know what he does. In other words, he should kill
> >> all l3 namespaces and after kill the l2 parent.
> > So, you suppose, that administrator remembers which l3 namespaces are over
> > this l2 namespace.
>

> I guess it remembers which IP address has been assigned to the
> namespaces, what name he gave to them, how many they are, what are the
> virtual hostname, etc ... The administrator can be a human or a
> container system manager.
>
> > List of childs gives an ability to track such relations easily.
> > If administrator decides to kill l2, he could decide is it force kill, when all
> > underlied l3 namespaces and their processes should be killed, or soft kill,
> > where only l2 refcount decremented.
>
> If you want the list of childs to be used to browse the namespace
> hierarchy in order to destroy them, what about the pid namespace,
> ipc_namespace, utsname namespace, ... ? You should have the same list
> for each namespace's ressources, no ? Perhaps having a child list into
> the network namespace is not the right place ...
L3 namespace is dependant on L2 resources. This is not a true for above
namespaces, which are independent. That's why hierarchy is required.

>
> >> Imagine you have a shell and you are into the foo directory, in another
> >> shell you remove the foo directory, the shell in the foo directory will
> >> not be pushed outside of the directory. The foo dir will stay in a
> >> "transient" state until the shell exits or change directory, I think
> >> that should be same with the namespaces or better to say "containers".
> > It is unacceptable in some cases. As Eric already said, network connections
> > could take refcount for a long time and you will wait container stop indefinitely,
> > because somebody spoofs your namespace. It is necessary to have an ability of
> > force kill.
>
> Don't me wrong, I don't say we should not have a force kill for a
> container, I am just saying the ressources must be killed for the
> container and that will refdown the ressources to reach the namespaces
> destruction.
The easiest way to kill such resources is to know which namespace resources
should be killed.

You propose to do the same from userspace. Userspace tools should know, which
namespaces are underlied, which resources are used by them, etc. I vote, that
easier and more nature to do this from kernel space. Just because there is no
need to design different user-space handles for force flush different resources
instead of one to flush them all.

--
Thanks,
Dmitry.

_____
Containers mailing list
Containers@lists.osdl.org