Subject: [PATCH 15/16] net: Implement network device movement between namespaces
Posted by ebiederm on Sat, 08 Sep 2007 21:38:46 GMT
View Forum Message <> Reply to Message

This patch introduces NETIF_F_NETNS_LOCAL a flag to indicate
a network device is local to a single network namespace and
should never be moved. Useful for pseudo devices that we
need an instance in each network namespace (like the loopback
device) and for any device we find that cannot handle multiple
network namespaces so we may trap them in the initial network
namespace.

This patch introduces the function dev_change_net_namespace
a function used to move a network device from one network
namespace to another. To the network device nothing
special appears to happen, to the components of the network
stack it appears as if the network device was unregistered
in the network namespace it is in, and a new device
was registered in the network namespace the device
was moved to.

This patch sets up a namespace device destructor that
upon the exit of a network namespace moves all of the
movable network devices to the initial network namespace
so they are not lost.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/net/loopback.c | 3 +-
 include/linux/netdevice.h | 3 +
 net/core/dev.c | 189 ++---
 3 files changed, 184 insertions(+), 11 deletions(-)

diff --git a/drivers/net/loopback.c b/drivers/net/loopback.c
index 5106c23..e399f7b 100644
--- a/drivers/net/loopback.c
+++ b/drivers/net/loopback.c
@@ -222,7 +222,8 @@ struct net_device loopback_dev = {
       | NETIF_F_TSO
 #endif
       | NETIF_F_NO_CSUM | NETIF_F_HIGHDMA
-      | NETIF_F_LLTX,
+ | NETIF_F_LLTX
+ | NETIF_F_NETNS_LOCAL,
   .ethtool_ops = &loopback_ethtool_ops,
 };

```
diff --git a/include/linux/netdevice.h b/include/linux/netdevice.h
index ec90d1a..d33d897 100644
--- a/include/linux/netdevice.h
+++ b/include/linux/netdevice.h
@@ -435,6 +435,7 @@ struct net_device
 #define NETIF_F_VLAN_CHALLENGED 1024 /* Device cannot handle VLAN packets */
 #define NETIF_F_GSO  2048 /* Enable software GSO. */
 #define NETIF_F_LLTX  4096 /* LockLess TX */
+#define NETIF_F_NETNS_LOCAL 8192 /* Does not change network namespaces */
 #define NETIF_F_MULTI_QUEUE 16384 /* Has multiple TX/RX queues */
 #define NETIF_F_LRO  32768 /* large receive offload */

@@ -1002,6 +1003,8 @@ extern int  dev_ethtool(struct net *net, struct ifreq *);
 extern unsigned  dev_get_flags(const struct net_device *);
 extern int  dev_change_flags(struct net_device *, unsigned);
 extern int  dev_change_name(struct net_device *, char *);
+extern int  dev_change_net_namespace(struct net_device *,
+       struct net *, const char *);
 extern int  dev_set_mtu(struct net_device *, int);
 extern int  dev_set_mac_address(struct net_device *,
         struct sockaddr *);
diff --git a/net/core/dev.c b/net/core/dev.c
index 53cdb64..d82ec5a 100644
--- a/net/core/dev.c
+++ b/net/core/dev.c
@@ -208,6 +208,34 @@ static inline struct hlist_head *dev_index_hash(struct net *net, int
ifindex)
  return &net->dev_index_head[ifindex & ((1 << NETDEV_HASHBITS) - 1)];
 }

+/* Device list insertion */
+static int list_netdevice(struct net_device *dev)
+{
+ struct net *net = dev->nd_net;
+
+ ASSERT_RTNL();
+
+ write_lock_bh(&dev_base_lock);
+ list_add_tail(&dev->dev_list, &net->dev_base_head);
+ hlist_add_head(&dev->name_hlist, dev_name_hash(net, dev->name));
+ hlist_add_head(&dev->index_hlist, dev_index_hash(net, dev->ifindex));
+ write_unlock_bh(&dev_base_lock);
+ return 0;
+}
+
+/* Device list removal */
+static void unlist_netdevice(struct net_device *dev)
+{
```

```
+ ASSERT_RTNL();
+
+ /* Unlink dev from the device chain */
+ write_lock_bh(&dev_base_lock);
+ list_del(&dev->dev_list);
+ hlist_del(&dev->name_hlist);
+ hlist_del(&dev->index_hlist);
+ write_unlock_bh(&dev_base_lock);
+}
+
 /*
  * Our notifier list
  */
@@ -3553,12 +3581,8 @@ int register_netdevice(struct net_device *dev)
  set_bit(__LINK_STATE_PRESENT, &dev->state);

  dev_init_scheduler(dev);
- write_lock_bh(&dev_base_lock);
- list_add_tail(&dev->dev_list, &net->dev_base_head);
- hlist_add_head(&dev->name_hlist, head);
- hlist_add_head(&dev->index_hlist, dev_index_hash(net, dev->ifindex));
  dev_hold(dev);
- write_unlock_bh(&dev_base_lock);
+ list_netdevice(dev);

  /* Notify protocols, that a new device appeared. */
  ret = raw_notifier_call_chain(&netdev_chain, NETDEV_REGISTER, dev);
@@ -3865,11 +3889,7 @@ void unregister_netdevice(struct net_device *dev)
   dev_close(dev);

  /* And unlink it from device chain. */
- write_lock_bh(&dev_base_lock);
- list_del(&dev->dev_list);
- hlist_del(&dev->name_hlist);
- hlist_del(&dev->index_hlist);
- write_unlock_bh(&dev_base_lock);
+ unlist_netdevice(dev);

  dev->reg_state = NETREG_UNREGISTERING;

@@ -3927,6 +3947,122 @@ void unregister_netdev(struct net_device *dev)

 EXPORT_SYMBOL(unregister_netdev);

+/**
+ * dev_change_net_namespace - move device to different nethost namespace
+ * @dev: device
+ * @net: network namespace
```

```
+ * @pat: If not NULL name pattern to try if the current device name
+ *       is already taken in the destination network namespace.
+ *
+ * This function shuts down a device interface and moves it
+ * to a new network namespace. On success 0 is returned, on
+ * a failure a netagive errno code is returned.
+ *
+ * Callers must hold the rtnl semaphore.
+ */
+
+int dev_change_net_namespace(struct net_device *dev, struct net *net, const char *pat)
+{
+ char buf[IFNAMSIZ];
+ const char *destname;
+ int err;
+
+ ASSERT_RTNL();
+
+ /* Don't allow namespace local devices to be moved. */
+ err = -EINVAL;
+ if (dev->features & NETIF_F_NETNS_LOCAL)
+  goto out;
+
+ /* Ensure the device has been registrered */
+ err = -EINVAL;
+ if (dev->reg_state != NETREG_REGISTERED)
+  goto out;
+
+ /* Get out if there is nothing todo */
+ err = 0;
+ if (dev->nd_net == net)
+  goto out;
+
+ /* Pick the destination device name, and ensure
+  * we can use it in the destination network namespace.
+  */
+ err = -EEXIST;
+ destname = dev->name;
+ if (__dev_get_by_name(net, destname)) {
+  /* We get here if we can't use the current device name */
+  if (!pat)
+   goto out;
+  if (!dev_valid_name(pat))
+   goto out;
+  if (strchr(pat, '%')) {
+   if (__dev_alloc_name(net, pat, buf) < 0)
+    goto out;
+   destname = buf;
```

```
+  } else
+   destname = pat;
+  if (__dev_get_by_name(net, destname))
+   goto out;
+ }
+
+ /*
+  * And now a mini version of register_netdevice unregister_netdevice.
+  */
+
+ /* If device is running close it first. */
+ if (dev->flags & IFF_UP)
+  dev_close(dev);
+
+ /* And unlink it from device chain */
+ err = -ENODEV;
+ unlist_netdevice(dev);
+
+ synchronize_net();
+
+ /* Shutdown queueing discipline. */
+ dev_shutdown(dev);
+
+ /* Notify protocols, that we are about to destroy
+    this device. They should clean all the things.
+ */
+ call_netdevice_notifiers(NETDEV_UNREGISTER, dev);
+
+ /*
+  * Flush the unicast and multicast chains
+  */
+ dev_addr_discard(dev);
+
+ /* Actually switch the network namespace */
+ dev->nd_net = net;
+
+ /* Assign the new device name */
+ if (destname != dev->name)
+  strcpy(dev->name, destname);
+
+ /* If there is an ifindex conflict assign a new one */
+ if (__dev_get_by_index(net, dev->ifindex)) {
+  int iflink = (dev->iflink == dev->ifindex);
+  dev->ifindex = dev_new_index(net);
+  if (iflink)
+   dev->iflink = dev->ifindex;
+ }
+
```

```
+ /* Fixup sysfs */
+ err = device_rename(&dev->dev, dev->name);
+ BUG_ON(err);
+
+ /* Add the device back in the hashes */
+ list_netdevice(dev);
+
+ /* Notify protocols, that a new device appeared. */
+ call_netdevice_notifiers(NETDEV_REGISTER, dev);
+
+ synchronize_net();
+ err = 0;
+out:
+ return err;
+}
+
 static int dev_cpu_callback(struct notifier_block *nfb,
        unsigned long action,
        void *ocpu)
@@ -4159,6 +4295,36 @@ static struct pernet_operations netdev_net_ops = {
  .exit = netdev_exit,
 };

+static void default_device_exit(struct net *net)
+{
+ struct net_device *dev, *next;
+ /*
+  * Push all migratable of the network devices back to the
+  * initial network namespace
+  */
+ rtnl_lock();
+ for_each_netdev_safe(net, dev, next) {
+  int err;
+
+  /* Ignore unmoveable devices (i.e. loopback) */
+  if (dev->features & NETIF_F_NETNS_LOCAL)
+   continue;
+
+  /* Push remaing network devices to init_net */
+  err = dev_change_net_namespace(dev, &init_net, "dev%d");
+  if (err) {
+   printk(KERN_WARNING "%s: failed to move %s to init_net: %d\n",
+    __func__, dev->name, err);
+   unregister_netdevice(dev);
+  }
+ }
+ rtnl_unlock();
+}
```

```
+
+static struct pernet_operations default_device_ops = {
+ .exit = default_device_exit,
+};
+
 /*
  * Initialize the DEV module. At boot time this walks the device list and
  * unhooks any devices that fail to initialise (normally hardware not
@@ -4189,6 +4355,9 @@ static int __init net_dev_init(void)
  if (register_pernet_subsys(&netdev_net_ops))
   goto out;

+ if (register_pernet_device(&default_device_ops))
+  goto out;
+
  /*
   * Initialise the packet receive queues.
   */
--
1.5.3.rc6.17.g1911
```