Subject: [PATCH 9/12] L2 network namespace: device to pass packets between namespaces
Posted by Mishin Dmitry on Wed, 06 Dec 2006 22:29:13 GMT
View Forum Message <> Reply to Message

A simple device to pass packets between a namespace and its child.

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

```
---
 drivers/net/Makefile     |    3
 drivers/net/veth.c       |  321 ++++++++++++++++++++++++++++++++++++++++++++++
 net/core/net_namespace.c |    1
 3 files changed, 325 insertions(+)

--- linux-2.6.19-rc6-mm2.orig/drivers/net/Makefile
+++ linux-2.6.19-rc6-mm2/drivers/net/Makefile
@@ -124,6 +124,9 @@ obj-$(CONFIG_SLIP) += slip.o
 obj-$(CONFIG_SLHC) += slhc.o

 obj-$(CONFIG_DUMMY) += dummy.o
+ifeq ($(CONFIG_NET_NS),y)
+obj-m += veth.o
+endif
 obj-$(CONFIG_IFB) += ifb.o
 obj-$(CONFIG_DE600) += de600.o
 obj-$(CONFIG_DE620) += de620.o
--- /dev/null
+++ linux-2.6.19-rc6-mm2/drivers/net/veth.c
@@ -0,0 +1,321 @@
+/*
+ * Copyright (C) 2006  SWsoft
+ *
+ * Written by Andrey Savochkin <saw@sw.ru>,
+ * reusing code by Andrey Mirkin <amirkin@sw.ru>.
+ */
+#include <linux/list.h>
+#include <linux/spinlock.h>
+#include <linux/ctype.h>
+#include <asm/semaphore.h>
+#include <linux/netdevice.h>
+#include <linux/etherdevice.h>
+#include <linux/proc_fs.h>
+#include <linux/seq_file.h>
+#include <net/dst.h>
+#include <net/xfrm.h>
+
+struct veth_struct
```

```
+{
+ struct net_device *pair;
+ struct net_device_stats stats;
+};
+
+#define veth_from_netdev(dev) ((struct veth_struct *)(netdev_priv(dev)))
+
+/* ---------------------------------------------------------------- *
+ *
+ * Device functions
+ *
+ * ---------------------------------------------------------------- */
+
+static struct net_device_stats *get_stats(struct net_device *dev);
+static int veth_xmit(struct sk_buff *skb, struct net_device *dev)
+{
+ struct net_device_stats *stats;
+ struct veth_struct *entry;
+ struct net_device *rcv;
+ struct net_namespace *orig_net_ns;
+ int length;
+
+ stats = get_stats(dev);
+ entry = veth_from_netdev(dev);
+ rcv = entry->pair;
+
+ if (!(rcv->flags & IFF_UP))
+  /* Target namespace does not want to receive packets */
+  goto outf;
+
+ dst_release(skb->dst);
+ skb->dst = NULL;
+ secpath_reset(skb);
+ skb_orphan(skb);
+ nf_reset(skb);
+
+ orig_net_ns = push_net_ns(rcv->net_ns);
+ skb->dev = rcv;
+ skb->pkt_type = PACKET_HOST;
+ skb->protocol = eth_type_trans(skb, rcv);
+
+ length = skb->len;
+ stats->tx_bytes += length;
+ stats->tx_packets++;
+ stats = get_stats(rcv);
+ stats->rx_bytes += length;
+ stats->rx_packets++;
+
```

```
+ netif_rx(skb);
+ pop_net_ns(orig_net_ns);
+ return 0;
+
+outf:
+ stats->tx_dropped++;
+ kfree_skb(skb);
+ return 0;
+}
+
+static int veth_open(struct net_device *dev)
+{
+ return 0;
+}
+
+static int veth_close(struct net_device *dev)
+{
+ return 0;
+}
+
+static void veth_destructor(struct net_device *dev)
+{
+ free_netdev(dev);
+}
+
+static struct net_device_stats *get_stats(struct net_device *dev)
+{
+ return &veth_from_netdev(dev)->stats;
+}
+
+int veth_init_dev(struct net_device *dev)
+{
+ dev->hard_start_xmit = veth_xmit;
+ dev->open = veth_open;
+ dev->stop = veth_close;
+ dev->destructor = veth_destructor;
+ dev->get_stats = get_stats;
+
+ ether_setup(dev);
+
+ dev->tx_queue_len = 0;
+ return 0;
+}
+
+static void veth_setup(struct net_device *dev)
+{
+ dev->init = veth_init_dev;
+}
```

```
+
+static inline int is_veth_dev(struct net_device *dev)
+{
+ return dev->init == veth_init_dev;
+}
+
+/* ---------------------------------------------------------------- *
+ *
+ * Management interface
+ *
+ * ---------------------------------------------------------------- */
+
+struct net_device *veth_dev_alloc(char *name, char *addr)
+{
+ struct net_device *dev;
+
+ dev = alloc_netdev(sizeof(struct veth_struct), name, veth_setup);
+ if (dev != NULL) {
+  memcpy(dev->dev_addr, addr, ETH_ALEN);
+  dev->addr_len = ETH_ALEN;
+ }
+ return dev;
+}
+
+int veth_entry_add(char *parent_name, char *parent_addr,
+ struct net_namespace *parent_ns, char *child_name, char *child_addr,
+       struct net_namespace *child_ns)
+{
+ struct net_device *parent_dev, *child_dev;
+ int err;
+
+ err = -ENOMEM;
+ if ((parent_dev = veth_dev_alloc(parent_name, parent_addr)) == NULL)
+  goto out_alocp;
+ if ((child_dev = veth_dev_alloc(child_name, child_addr)) == NULL)
+  goto out_alocc;
+ veth_from_netdev(parent_dev)->pair = child_dev;
+ veth_from_netdev(child_dev)->pair = parent_dev;
+
+ /*
+  * About serialization, see comments to veth_pair_del().
+  */
+ rtnl_lock();
+ /* refcounts should be already upped, so, just put old ones */
+ put_net_ns(parent_dev->net_ns);
+ parent_dev->net_ns = parent_ns;
+ if ((err = register_netdevice(parent_dev)))
+  goto out_regp;
```

```
+
+ put_net_ns(child_dev->net_ns);
+ child_dev->net_ns = child_ns;
+ if ((err = register_netdevice(child_dev)))
+  goto out_regc;
+ rtnl_unlock();
+ return 0;
+
+out_regc:
+ unregister_netdevice(parent_dev);
+ rtnl_unlock();
+ free_netdev(child_dev);
+ return err;
+
+out_regp:
+ rtnl_unlock();
+ free_netdev(child_dev);
+out_alocc:
+ free_netdev(parent_dev);
+out_alocp:
+ return err;
+}
+
+static void veth_pair_del(struct net_device *parent_dev)
+{
+ struct net_device *child_dev;
+ struct net_namespace *parent_ns, *child_ns;
+
+ child_dev = veth_from_netdev(parent_dev)->pair;
+ get_net_ns(child_dev->net_ns);
+ child_ns = child_dev->net_ns;
+
+ dev_close(child_dev);
+ synchronize_net();
+ /*
+  * Now child_dev does not send or receives anything.
+  * This means child_dev->hard_start_xmit is not called anymore.
+  */
+ unregister_netdevice(parent_dev);
+ /*
+  * At this point child_dev has dead pointer to parent_dev.
+  * But this pointer is not dereferenced.
+  */
+ parent_ns = push_net_ns(child_ns);
+ unregister_netdevice(child_dev);
+ pop_net_ns(parent_ns);
+
+ put_net_ns(child_ns);
```

```
+}
+
+int veth_entry_del(char *parent_name)
+{
+ struct net_device *dev;
+
+ if ((dev = dev_get_by_name(parent_name)) == NULL)
+  return -ENODEV;
+
+ rtnl_lock();
+ veth_pair_del(dev);
+ dev_put(dev);
+ rtnl_unlock();
+
+ return 0;
+}
+
+void veth_entry_del_all(void)
+{
+ struct net_device **p, *dev;
+
+ rtnl_lock();
+ for (p = &dev_base; (dev = *p) != NULL; ) {
+  if (!is_veth_dev(dev)) {
+   p = &dev->next;
+   continue;
+  }
+
+  dev_hold(dev);
+  veth_pair_del(dev);
+  dev_put(dev);
+ }
+ rtnl_unlock();
+}
+
+/* ------------------------------------------------------------------ *
+ *
+ * Information in proc
+ *
+ * ------------------------------------------------------------------ */
+
+#ifdef CONFIG_PROC_FS
+
+#define ADDR_FMT "%02x:%02x:%02x:%02x:%02x:%02x"
+#define ADDR(x) (x)[0],(x)[1],(x)[2],(x)[3],(x)[4],(x)[5]
+#define ADDR_HDR "%-17s"
+
+static int veth_proc_show(struct seq_file *m,
```

```
+  struct net_device *dev, void *data)
+{
+ struct net_device *pair;
+
+ if (dev == SEQ_START_TOKEN) {
+  seq_puts(m, "Version: 1.0\n");
+  seq_printf(m, "%-*s " ADDR_HDR " %-*s " ADDR_HDR "\n",
+    IFNAMSIZ, "Name", "Address",
+    IFNAMSIZ, "PeerName", "PeerAddress");
+  return 0;
+ }
+
+ if (!is_veth_dev(dev))
+  return 0;
+
+ pair = veth_from_netdev(dev)->pair;
+ seq_printf(m, "%-*s " ADDR_FMT " %-*s " ADDR_FMT "\n",
+   IFNAMSIZ, dev->name, ADDR(dev->dev_addr),
+   IFNAMSIZ, pair->name, ADDR(pair->dev_addr));
+ return 0;
+}
+
+static int veth_proc_create(void)
+{
+ return netdev_proc_create("veth_list", &veth_proc_show, NULL,
+   THIS_MODULE);
+}
+
+static void veth_proc_remove(void)
+{
+ netdev_proc_remove("net/veth_list");
+}
+
+#else
+
+static inline int veth_proc_create(void) { return 0; }
+static inline void veth_proc_remove(void) { }
+
+#endif
+
+/* ---------------------------------------------------------------- *
+ *
+ * Module initialization
+ *
+ * ---------------------------------------------------------------- */
+
+int __init veth_init(void)
+{
```

```
+ veth_proc_create();
+ return 0;
+}
+
+void __exit veth_exit(void)
+{
+ veth_proc_remove();
+ veth_entry_del_all();
+}
+
+module_init(veth_init)
+module_exit(veth_exit)
+
+MODULE_DESCRIPTION("Virtual Ethernet Device");
+MODULE_LICENSE("GPL v2");
--- linux-2.6.19-rc6-mm2.orig/net/core/net_namespace.c
+++ linux-2.6.19-rc6-mm2/net/core/net_namespace.c
@@ -106,5 +106,6 @@ void free_net_ns(struct kref *kref)
  ip_fib_struct_cleanup();
  kfree(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 9/12] L2 network namespace: device to pass packets between namespaces
Posted by Daniel Lezcano on Tue, 12 Dec 2006 10:45:42 GMT
View Forum Message <> Reply to Message

> +
> +MODULE_DESCRIPTION("Virtual Ethernet Device");
> +MODULE_LICENSE("GPL v2");
> --- linux-2.6.19-rc6-mm2.orig/net/core/net_namespace.c
> +++ linux-2.6.19-rc6-mm2/net/core/net_namespace.c
> @@ -106,5 +106,6 @@ void free_net_ns(struct kref *kref)
> ip_fib_struct_cleanup();
> kfree(ns);
> }
> +EXPORT_SYMBOL(free_net_ns);

Why do you need to export this function ? It should be only used by the put_net_ns, no ?

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

Subject: Re: [PATCH 9/12] L2 network namespace: device to pass packets between namespaces
Posted by Mishin Dmitry on Tue, 12 Dec 2006 10:48:49 GMT
View Forum Message <> Reply to Message

On Tuesday 12 December 2006 13:45, Daniel Lezcano wrote:
> > +
> > +MODULE_DESCRIPTION("Virtual Ethernet Device");
> > +MODULE_LICENSE("GPL v2");
> > --- linux-2.6.19-rc6-mm2.orig/net/core/net_namespace.c
> > +++ linux-2.6.19-rc6-mm2/net/core/net_namespace.c
> > @@ -106,5 +106,6 @@ void free_net_ns(struct kref *kref)
> > ip_fib_struct_cleanup();
> > kfree(ns);
> > }
> > +EXPORT_SYMBOL(free_net_ns);
>
> Why do you need to export this function ? It should be only used by the
> put_net_ns, no ?
Yes, and put_net_ns is used in veth module.

--
Thanks,
Dmitry.

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