Subject: [patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by Andrey Savochkin on Mon, 26 Jun 2006 09:49:45 GMT
View Forum Message <> Reply to Message

Cleanup of dev_base list use, with the aim to make device list per-namespace.
In almost every occasion, use of dev_base variable and dev->next pointer
could be easily replaced by for_each_netdev loop.
A few most complicated places were converted to using
first_netdev()/next_netdev().

Signed-off-by: Andrey Savochkin <saw@swsoft.com>

```
 arch/s390/appldata/appldata_net_sum.c |    2
 arch/sparc64/solaris/ioctl.c          |    2
 drivers/block/aoe/aoecmd.c            |    8 ++-
 drivers/net/wireless/strip.c          |    4 -
 drivers/parisc/led.c                  |    2
 include/linux/netdevice.h             |   28 ++++++++++--
 net/8021q/vlan.c                      |    4 -
 net/8021q/vlanproc.c                  |   10 ++--
 net/bridge/br_if.c                    |    4 -
 net/bridge/br_ioctl.c                 |    4 +
 net/bridge/br_netlink.c               |    3 -
 net/core/dev.c                        |   70 ++++++++++++++++++----------------
 net/core/dev_mcast.c                  |    4 -
 net/core/rtnetlink.c                  |   18 ++++----
 net/decnet/af_decnet.c                |   11 +++--
 net/decnet/dn_dev.c                   |   17 ++++----
 net/decnet/dn_fib.c                   |    2
 net/decnet/dn_route.c                 |   12 ++---
 net/ipv4/devinet.c                    |   15 ++++---
 net/ipv4/igmp.c                       |   25 +++++++-----
 net/ipv6/addrconf.c                   |   28 ++++++++-----
 net/ipv6/anycast.c                    |   22 ++++++----
 net/ipv6/mcast.c                      |   20 +++++---
 net/llc/llc_core.c                    |    7 ++-
 net/netrom/nr_route.c                 |    4 -
 net/rose/rose_route.c                 |    8 ++-
 net/sched/sch_api.c                   |    8 ++-
 net/sctp/protocol.c                   |    2
 net/tipc/eth_media.c                  |   12 +++--
 29 files changed, 200 insertions, 156 deletions

--- ./arch/s390/appldata/appldata_net_sum.c.vedevbase Mon Mar 20 08:53:29 2006
+++ ./arch/s390/appldata/appldata_net_sum.c Thu Jun 22 12:03:07 2006
@@ -108,7 +108,7 @@ static void appldata_get_net_sum_data(vo
  tx_dropped = 0;
  collisions = 0;
```

```
  read_lock(&dev_base_lock);
- for (dev = dev_base; dev != NULL; dev = dev->next) {
+ for_each_netdev(dev) {
   if (dev->get_stats == NULL) {
    continue;
   }
--- ./arch/sparc64/solaris/ioctl.c.vedevbase Mon Mar 20 08:53:29 2006
+++ ./arch/sparc64/solaris/ioctl.c Thu Jun 22 12:03:07 2006
@@ -686,7 +686,7 @@ static inline int solaris_i(unsigned int
    int i = 0;

    read_lock_bh(&dev_base_lock);
-   for (d = dev_base; d; d = d->next) i++;
+   for_each_netdev(d) i++;
    read_unlock_bh(&dev_base_lock);

    if (put_user (i, (int __user *)A(arg)))
--- ./drivers/block/aoe/aoecmd.c.vedevbase Wed Jun 21 18:50:28 2006
+++ ./drivers/block/aoe/aoecmd.c Thu Jun 22 12:03:07 2006
@@ -204,14 +204,17 @@ aoecmd_cfg_pkts(ushort aoemajor, unsigne
  sl = sl_tail = NULL;

  read_lock(&dev_base_lock);
- for (ifp = dev_base; ifp; dev_put(ifp), ifp = ifp->next) {
+ for_each_netdev(dev) {
   dev_hold(ifp);
-  if (!is_aoe_netif(ifp))
+  if (!is_aoe_netif(ifp)) {
+   dev_put(ifp);
    continue;
+  }

   skb = new_skb(ifp, sizeof *h + sizeof *ch);
   if (skb == NULL) {
    printk(KERN_INFO "aoe: aoecmd_cfg: skb alloc failure\n");
+   dev_put(ifp);
    continue;
   }
   if (sl_tail == NULL)
@@ -229,6 +232,7 @@ aoecmd_cfg_pkts(ushort aoemajor, unsigne

   skb->next = sl;
   sl = skb;
+  dev_put(ifp);
  }
  read_unlock(&dev_base_lock);

--- ./drivers/net/wireless/strip.c.vedevbase Wed Jun 21 18:50:43 2006
```

```
+++ ./drivers/net/wireless/strip.c Thu Jun 22 12:03:07 2006
@@ -1970,8 +1970,7 @@ static struct net_device *get_strip_dev(
         sizeof(zero_address))) {
   struct net_device *dev;
   read_lock_bh(&dev_base_lock);
-  dev = dev_base;
-  while (dev) {
+  for_each_netdev(dev) {
    if (dev->type == strip_info->dev->type &&
        !memcmp(dev->dev_addr,
         &strip_info->true_dev_addr,
@@ -1982,7 +1981,6 @@ static struct net_device *get_strip_dev(
     read_unlock_bh(&dev_base_lock);
     return (dev);
    }
-   dev = dev->next;
   }
   read_unlock_bh(&dev_base_lock);
  }
--- ./drivers/parisc/led.c.vedevbase Wed Jun 21 18:52:58 2006
+++ ./drivers/parisc/led.c Thu Jun 22 12:03:07 2006
@@ -368,7 +368,7 @@ static __inline__ int led_get_net_activi
   * for reading should be OK */
  read_lock(&dev_base_lock);
  rcu_read_lock();
- for (dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
      struct net_device_stats *stats;
      struct in_device *in_dev = __in_dev_get_rcu(dev);
      if (!in_dev || !in_dev->ifa_list)
--- ./include/linux/netdevice.h.vedevbase Wed Jun 21 18:53:17 2006
+++ ./include/linux/netdevice.h Thu Jun 22 18:57:50 2006
@@ -289,8 +289,8 @@ struct net_device

  unsigned long  state;

- struct net_device *next;
-
+ struct list_head dev_list;
+
  /* The device initialization function. Called only once. */
  int   (*init)(struct net_device *dev);

@@ -543,9 +543,27 @@ struct packet_type {
 #include <linux/interrupt.h>
 #include <linux/notifier.h>

-extern struct net_device  loopback_dev;  /* The loopback */
```

```
-extern struct net_device  *dev_base;  /* All devices */
-extern rwlock_t    dev_base_lock;  /* Device list lock */
+extern struct net_device loopback_dev;  /* The loopback */
+extern struct list_head  dev_base_head;  /* All devices */
+extern rwlock_t    dev_base_lock;  /* Device list lock */
+
+#define for_each_netdev(p) list_for_each_entry(p, &dev_base_head, dev_list)
+
+/* DO NOT USE first_netdev/next_netdev, use loop defined above */
+#define first_netdev()  ({ \
+     list_empty(&dev_base_head) ? NULL : \
+      list_entry(dev_base_head.next, \
+       struct net_device, \
+       dev_list); \
+     })
+#define next_netdev(dev) ({ \
+     struct list_head *__next; \
+     __next = (dev)->dev_list.next; \
+     __next == &dev_base_head ? NULL : \
+      list_entry(__next, \
+       struct net_device, \
+       dev_list); \
+     })

 extern int    netdev_boot_setup_check(struct net_device *dev);
 extern unsigned long  netdev_boot_base(const char *prefix, int unit);
--- ./net/8021q/vlan.c.vedevbase Wed Jun 21 18:51:08 2006
+++ ./net/8021q/vlan.c Thu Jun 22 12:03:07 2006
@@ -121,8 +121,8 @@ static void __exit vlan_cleanup_devices(
  struct net_device *dev, *nxt;

  rtnl_lock();
- for (dev = dev_base; dev; dev = nxt) {
-  nxt = dev->next;
+ for (dev = first_netdev(); dev; dev = nxt) {
+  nxt = next_netdev(dev);
   if (dev->priv_flags & IFF_802_1Q_VLAN) {
    unregister_vlan_dev(VLAN_DEV_INFO(dev)->real_dev,
          VLAN_DEV_INFO(dev)->vlan_id);
--- ./net/8021q/vlanproc.c.vedevbase Mon Mar 20 08:53:29 2006
+++ ./net/8021q/vlanproc.c Thu Jun 22 12:03:07 2006
@@ -242,7 +242,7 @@ int vlan_proc_rem_dev(struct net_device
 static struct net_device *vlan_skip(struct net_device *dev)
 {
  while (dev && !(dev->priv_flags & IFF_802_1Q_VLAN))
-  dev = dev->next;
+  dev = next_netdev(dev);
```

```
  return dev;
 }
@@ -258,8 +258,8 @@ static void *vlan_seq_start(struct seq_f
  if (*pos == 0)
   return SEQ_START_TOKEN;

- for (dev = vlan_skip(dev_base); dev && i < *pos;
-      dev = vlan_skip(dev->next), ++i);
+ for (dev = vlan_skip(first_netdev()); dev && i < *pos;
+      dev = vlan_skip(next_netdev(dev)), ++i);

  return  (i == *pos) ? dev : NULL;
 }
@@ -269,8 +269,8 @@ static void *vlan_seq_next(struct seq_fi
  ++*pos;

  return vlan_skip((v == SEQ_START_TOKEN)
-       ? dev_base
-       : ((struct net_device *)v)->next);
+       ? first_netdev()
+       : next_netdev((struct net_device *)v));
 }

 static void vlan_seq_stop(struct seq_file *seq, void *v)
--- ./net/bridge/br_if.c.vedevbase Wed Jun 21 18:53:18 2006
+++ ./net/bridge/br_if.c Thu Jun 22 12:03:07 2006
@@ -468,8 +468,8 @@ void __exit br_cleanup_bridges(void)
  struct net_device *dev, *nxt;

  rtnl_lock();
- for (dev = dev_base; dev; dev = nxt) {
-  nxt = dev->next;
+ for (dev = first_netdev(); dev; dev = nxt) {
+  nxt = next_netdev(dev);
   if (dev->priv_flags & IFF_EBRIDGE)
    del_br(dev->priv);
  }
--- ./net/bridge/br_ioctl.c.vedevbase Mon Mar 20 08:53:29 2006
+++ ./net/bridge/br_ioctl.c Thu Jun 22 12:03:07 2006
@@ -27,7 +27,9 @@ static int get_bridge_ifindices(int *ind
  struct net_device *dev;
  int i = 0;

- for (dev = dev_base; dev && i < num; dev = dev->next) {
+ for_each_netdev(dev) {
+  if (i >= num)
+   break;
   if (dev->priv_flags & IFF_EBRIDGE)
```

```
    indices[i++] = dev->ifindex;
  }
--- ./net/bridge/br_netlink.c.vedevbase Wed Jun 21 18:53:18 2006
+++ ./net/bridge/br_netlink.c Thu Jun 22 12:03:07 2006
@@ -109,7 +109,8 @@ static int br_dump_ifinfo(struct sk_buff
  int err = 0;

  read_lock(&dev_base_lock);
- for (dev = dev_base, idx = 0; dev; dev = dev->next) {
+ idx = 0;
+ for_each_netdev(dev) {
    struct net_bridge_port *p = dev->br_port;

    /* not a bridge port */
--- ./net/core/dev.c.vedevbase Wed Jun 21 18:53:18 2006
+++ ./net/core/dev.c Thu Jun 22 17:40:13 2006
@@ -174,13 +174,12 @@ static spinlock_t net_dma_event_lock;
  * unregister_netdevice(), which must be called with the rtnl
  * semaphore held.
  */
-struct net_device *dev_base;
-static struct net_device **dev_tail = &dev_base;
 DEFINE_RWLOCK(dev_base_lock);
-
-EXPORT_SYMBOL(dev_base);
 EXPORT_SYMBOL(dev_base_lock);

+LIST_HEAD(dev_base_head);
+EXPORT_SYMBOL(dev_base_head);
+
 #define NETDEV_HASHBITS 8
 static struct hlist_head dev_name_head[1<<NETDEV_HASHBITS];
 static struct hlist_head dev_index_head[1<<NETDEV_HASHBITS];
@@ -575,7 +574,7 @@ struct net_device *dev_getbyhwaddr(unsig

  ASSERT_RTNL();

- for (dev = dev_base; dev; dev = dev->next)
+ for_each_netdev(dev)
    if (dev->type == type &&
        !memcmp(dev->dev_addr, ha, dev->addr_len))
      break;
@@ -589,7 +588,7 @@ struct net_device *dev_getfirstbyhwtype(
  struct net_device *dev;

  rtnl_lock();
- for (dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
```

```
   if (dev->type == type) {
    dev_hold(dev);
    break;
@@ -617,7 +616,7 @@ struct net_device * dev_get_by_flags(uns
  struct net_device *dev;

  read_lock(&dev_base_lock);
- for (dev = dev_base; dev != NULL; dev = dev->next) {
+ for_each_netdev(dev) {
   if (((dev->flags ^ if_flags) & mask) == 0) {
    dev_hold(dev);
    break;
@@ -680,7 +679,7 @@ int dev_alloc_name(struct net_device *de
   if (!inuse)
    return -ENOMEM;

-  for (d = dev_base; d; d = d->next) {
+  for_each_netdev(d) {
    if (!sscanf(d->name, name, &i))
     continue;
    if (i < 0 || i >= max_netdevices)
@@ -966,7 +965,7 @@ int register_netdevice_notifier(struct n
  rtnl_lock();
  err = raw_notifier_chain_register(&netdev_chain, nb);
  if (!err) {
-  for (dev = dev_base; dev; dev = dev->next) {
+  for_each_netdev(dev) {
    nb->notifier_call(nb, NETDEV_REGISTER, dev);

    if (dev->flags & IFF_UP)
@@ -1903,7 +1902,7 @@ static int dev_ifconf(char __user *arg)
   */

  total = 0;
- for (dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
   for (i = 0; i < NPROTO; i++) {
    if (gifconf_list[i]) {
     int done;
@@ -1935,26 +1934,25 @@ static int dev_ifconf(char __user *arg)
  * This is invoked by the /proc filesystem handler to display a device
  * in detail.
  */
-static __inline__ struct net_device *dev_get_idx(loff_t pos)
-{
- struct net_device *dev;
- loff_t i;
-
```

```
- for (i = 0, dev = dev_base; dev && i < pos; ++i, dev = dev->next);
-
- return i == pos ? dev : NULL;
-}
-
 void *dev_seq_start(struct seq_file *seq, loff_t *pos)
 {
+ struct net_device *dev;
+ loff_t off = 1;
  read_lock(&dev_base_lock);
- return *pos ? dev_get_idx(*pos - 1) : SEQ_START_TOKEN;
+ if (!*pos)
+  return SEQ_START_TOKEN;
+ for_each_netdev(dev) {
+  if (off++ == *pos)
+   return dev;
+ }
+ return NULL;
 }

 void *dev_seq_next(struct seq_file *seq, void *v, loff_t *pos)
 {
+ struct net_device *dev = v;
  ++*pos;
- return v == SEQ_START_TOKEN ? dev_base : ((struct net_device *)v)->next;
+ return v == SEQ_START_TOKEN ? first_netdev() : next_netdev(dev);
 }

 void dev_seq_stop(struct seq_file *seq, void *v)
@@ -2837,11 +2835,9 @@ int register_netdevice(struct net_device

  set_bit(__LINK_STATE_PRESENT, &dev->state);

- dev->next = NULL;
  dev_init_scheduler(dev);
  write_lock_bh(&dev_base_lock);
- *dev_tail = dev;
- dev_tail = &dev->next;
+ list_add_tail(&dev->dev_list, &dev_base_head);
  hlist_add_head(&dev->name_hlist, head);
  hlist_add_head(&dev->index_hlist, dev_index_hash(dev->ifindex));
  dev_hold(dev);
@@ -3119,8 +3115,6 @@ void synchronize_net(void)

 int unregister_netdevice(struct net_device *dev)
 {
- struct net_device *d, **dp;
-
```

```
  BUG_ON(dev_boot_phase);
  ASSERT_RTNL();

@@ -3138,23 +3132,11 @@ int unregister_netdevice(struct net_devi
    dev_close(dev);

  /* And unlink it from device chain. */
- for (dp = &dev_base; (d = *dp) != NULL; dp = &d->next) {
-  if (d == dev) {
-   write_lock_bh(&dev_base_lock);
-   hlist_del(&dev->name_hlist);
-   hlist_del(&dev->index_hlist);
-   if (dev_tail == &dev->next)
-    dev_tail = dp;
-   *dp = d->next;
-   write_unlock_bh(&dev_base_lock);
-   break;
-  }
- }
- if (!d) {
-  printk(KERN_ERR "unregister net_device: '%s' not found\n",
-         dev->name);
-  return -ENODEV;
- }
+ 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);

  dev->reg_state = NETREG_UNREGISTERING;

--- ./net/core/dev_mcast.c.vedevbase Wed Jun 21 18:53:18 2006
+++ ./net/core/dev_mcast.c Thu Jun 22 12:03:08 2006
@@ -225,7 +225,7 @@ static void *dev_mc_seq_start(struct seq
  loff_t off = 0;

  read_lock(&dev_base_lock);
- for (dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
   if (off++ == *pos)
    return dev;
  }
@@ -236,7 +236,7 @@ static void *dev_mc_seq_next(struct seq_
 {
  struct net_device *dev = v;
  ++*pos;
- return dev->next;
```

```
+ return next_netdev(dev);
 }

 static void dev_mc_seq_stop(struct seq_file *seq, void *v)
--- ./net/core/rtnetlink.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/core/rtnetlink.c Thu Jun 22 12:03:08 2006
@@ -320,14 +320,16 @@ static int rtnetlink_dump_ifinfo(struct
  struct net_device *dev;

  read_lock(&dev_base_lock);
- for (dev=dev_base, idx=0; dev; dev = dev->next, idx++) {
-  if (idx < s_idx)
-   continue;
-  if (rtnetlink_fill_ifinfo(skb, dev, RTM_NEWLINK,
-       NETLINK_CB(cb->skb).pid,
-       cb->nlh->nlmsg_seq, 0,
-       NLM_F_MULTI) <= 0)
-   break;
+ idx = 0;
+ for_each_netdev(dev) {
+  if (idx >= s_idx) {
+   if (rtnetlink_fill_ifinfo(skb, dev, RTM_NEWLINK,
+       NETLINK_CB(cb->skb).pid,
+       cb->nlh->nlmsg_seq, 0,
+       NLM_F_MULTI) <= 0)
+    break;
+  }
+  idx++;
  }
  read_unlock(&dev_base_lock);
  cb->args[0] = idx;
--- ./net/decnet/af_decnet.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/decnet/af_decnet.c Thu Jun 22 12:03:08 2006
@@ -721,7 +721,7 @@ static int dn_bind(struct socket *sock,
  struct sock *sk = sock->sk;
  struct dn_scp *scp = DN_SK(sk);
  struct sockaddr_dn *saddr = (struct sockaddr_dn *)uaddr;
- struct net_device *dev;
+ struct net_device *pdev, *dev;
  int rv;

  if (addr_len != sizeof(struct sockaddr_dn))
@@ -745,12 +745,15 @@ static int dn_bind(struct socket *sock,

  if (!(saddr->sdn_flags & SDF_WILD)) {
   if (dn_ntohs(saddr->sdn_nodeaddrl)) {
+   dev = NULL;
    read_lock(&dev_base_lock);
```

```
-   for(dev = dev_base; dev; dev = dev->next) {
-    if (!dev->dn_ptr)
+   for_each_netdev(pdev) {
+    if (!pdev->dn_ptr)
      continue;
-    if (dn_dev_islocal(dev, dn_saddr2dn(saddr)))
+    if (dn_dev_islocal(pdev, dn_saddr2dn(saddr))) {
+     dev = pdev;
      break;
+    }
    }
    read_unlock(&dev_base_lock);
   if (dev == NULL)
--- ./net/decnet/dn_dev.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/decnet/dn_dev.c Thu Jun 22 12:03:08 2006
@@ -776,13 +776,14 @@ static int dn_dev_dump_ifaddr(struct sk_
  s_idx = cb->args[0];
  s_dn_idx = dn_idx = cb->args[1];
  read_lock(&dev_base_lock);
- for(dev = dev_base, idx = 0; dev; dev = dev->next, idx++) {
+ idx = 0;
+ for_each_netdev(dev) {
   if (idx < s_idx)
-   continue;
+   goto cont;
   if (idx > s_idx)
    s_dn_idx = 0;
   if ((dn_db = dev->dn_ptr) == NULL)
-   continue;
+   goto cont;

   for(ifa = dn_db->ifa_list, dn_idx = 0; ifa; ifa = ifa->ifa_next, dn_idx++) {
    if (dn_idx < s_dn_idx)
@@ -795,6 +796,8 @@ static int dn_dev_dump_ifaddr(struct sk_
            NLM_F_MULTI) <= 0)
     goto done;
   }
+cont:
+  idx++;
  }
 done:
  read_unlock(&dev_base_lock);
@@ -1265,7 +1268,7 @@ void dn_dev_devices_off(void)
  struct net_device *dev;

  rtnl_lock();
- for(dev = dev_base; dev; dev = dev->next)
+ for_each_netdev(dev)
```

```
   dn_dev_down(dev);
  rtnl_unlock();

@@ -1276,7 +1279,7 @@ void dn_dev_devices_on(void)
   struct net_device *dev;

   rtnl_lock();
- for(dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
    if (dev->flags & IFF_UP)
     dn_dev_up(dev);
   }
@@ -1297,7 +1300,7 @@ int unregister_dnaddr_notifier(struct no
  static inline struct net_device *dn_dev_get_next(struct seq_file *seq, struct net_device *dev)
  {
   do {
-   dev = dev->next;
+   dev = next_netdev(dev);
   } while(dev && !dev->dn_ptr);

   return dev;
@@ -1307,7 +1310,7 @@ static struct net_device *dn_dev_get_idx
  {
   struct net_device *dev;

- dev = dev_base;
+ dev = first_netdev();
   if (dev && !dev->dn_ptr)
    dev = dn_dev_get_next(seq, dev);
   if (pos) {
--- ./net/decnet/dn_fib.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/decnet/dn_fib.c Thu Jun 22 12:03:08 2006
@@ -631,7 +631,7 @@ static void dn_fib_del_ifaddr(struct dn_

   /* Scan device list */
   read_lock(&dev_base_lock);
- for(dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
    dn_db = dev->dn_ptr;
    if (dn_db == NULL)
     continue;
--- ./net/decnet/dn_route.c.vedevbase Wed Jun 21 18:53:19 2006
+++ ./net/decnet/dn_route.c Thu Jun 22 12:03:08 2006
@@ -923,16 +923,16 @@ static int dn_route_output_slow(struct d
     goto out;
    }
    read_lock(&dev_base_lock);
-   for(dev_out = dev_base; dev_out; dev_out = dev_out->next) {
```

```
+  for_each_netdev(dev_out) {
    if (!dev_out->dn_ptr)
     continue;
-   if (dn_dev_islocal(dev_out, oldflp->fld_src))
-    break;
+   if (dn_dev_islocal(dev_out, oldflp->fld_src)) {
+    dev_hold(dev_out);
+    goto source_ok;
+   }
   }
   read_unlock(&dev_base_lock);
-  if (dev_out == NULL)
-   goto out;
-  dev_hold(dev_out);
+  goto out;
 source_ok:
   ;
  }
--- ./net/ipv4/devinet.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/ipv4/devinet.c Thu Jun 22 12:03:08 2006
@@ -842,7 +842,7 @@ no_in_dev:
   */
  read_lock(&dev_base_lock);
  rcu_read_lock();
- for (dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
   if ((in_dev = __in_dev_get_rcu(dev)) == NULL)
    continue;

@@ -921,7 +921,7 @@ u32 inet_confirm_addr(const struct net_d

  read_lock(&dev_base_lock);
  rcu_read_lock();
- for (dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
   if ((in_dev = __in_dev_get_rcu(dev))) {
    addr = confirm_addr_indev(in_dev, dst, local, scope);
    if (addr)
@@ -1095,17 +1095,18 @@ static int inet_dump_ifaddr(struct sk_bu
  struct in_ifaddr *ifa;
  int s_ip_idx, s_idx = cb->args[0];

+ idx = 0;
  s_ip_idx = ip_idx = cb->args[1];
  read_lock(&dev_base_lock);
- for (dev = dev_base, idx = 0; dev; dev = dev->next, idx++) {
+ for_each_netdev(dev) {
   if (idx < s_idx)
```

```
-   continue;
+   goto cont;
    if (idx > s_idx)
     s_ip_idx = 0;
    rcu_read_lock();
    if ((in_dev = __in_dev_get_rcu(dev)) == NULL) {
     rcu_read_unlock();
-   continue;
+   goto cont;
    }

    for (ifa = in_dev->ifa_list, ip_idx = 0; ifa;
@@ -1120,6 +1121,8 @@ static int inet_dump_ifaddr(struct sk_bu
     }
    }
    rcu_read_unlock();
+cont:
+  idx++;
  }

 done:
@@ -1171,7 +1174,7 @@ void inet_forward_change(void)
  ipv4_devconf_dflt.forwarding = on;

  read_lock(&dev_base_lock);
- for (dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
    struct in_device *in_dev;
    rcu_read_lock();
    in_dev = __in_dev_get_rcu(dev);
--- ./net/ipv4/igmp.c.vedevbase Wed Jun 21 18:53:19 2006
+++ ./net/ipv4/igmp.c Thu Jun 22 12:03:08 2006
@@ -2255,19 +2255,21 @@ struct igmp_mc_iter_state {

 static inline struct ip_mc_list *igmp_mc_get_first(struct seq_file *seq)
 {
+ struct net_device *dev;
  struct ip_mc_list *im = NULL;
  struct igmp_mc_iter_state *state = igmp_mc_seq_private(seq);

- for (state->dev = dev_base, state->in_dev = NULL;
-      state->dev;
-      state->dev = state->dev->next) {
+ state->dev = NULL;
+ state->in_dev = NULL;
+ for_each_netdev(dev) {
    struct in_device *in_dev;
-  in_dev = in_dev_get(state->dev);
```

```
+  in_dev = in_dev_get(dev);
   if (!in_dev)
    continue;
   read_lock(&in_dev->mc_list_lock);
   im = in_dev->mc_list;
   if (im) {
+   state->dev = dev;
    state->in_dev = in_dev;
    break;
   }
@@ -2286,7 +2288,7 @@ static struct ip_mc_list *igmp_mc_get_ne
    read_unlock(&state->in_dev->mc_list_lock);
    in_dev_put(state->in_dev);
   }
-  state->dev = state->dev->next;
+  state->dev = next_netdev(state->dev);
   if (!state->dev) {
    state->in_dev = NULL;
    break;
@@ -2417,15 +2419,17 @@ struct igmp_mcf_iter_state {

 static inline struct ip_sf_list *igmp_mcf_get_first(struct seq_file *seq)
 {
+ struct net_device *dev;
  struct ip_sf_list *psf = NULL;
  struct ip_mc_list *im = NULL;
  struct igmp_mcf_iter_state *state = igmp_mcf_seq_private(seq);

- for (state->dev = dev_base, state->idev = NULL, state->im = NULL;
-      state->dev;
-      state->dev = state->dev->next) {
+ state->dev = NULL;
+ state->im = NULL;
+ state->idev = NULL;
+ for_each_netdev(dev) {
   struct in_device *idev;
-  idev = in_dev_get(state->dev);
+  idev = in_dev_get(dev);
   if (unlikely(idev == NULL))
    continue;
   read_lock(&idev->mc_list_lock);
@@ -2434,6 +2438,7 @@ static inline struct ip_sf_list *igmp_mc
    spin_lock_bh(&im->lock);
    psf = im->sources;
    if (likely(psf != NULL)) {
+    state->dev = dev;
     state->im = im;
     state->idev = idev;
```

```
     break;
@@ -2459,7 +2464,7 @@ static struct ip_sf_list *igmp_mcf_get_n
     read_unlock(&state->idev->mc_list_lock);
     in_dev_put(state->idev);
    }
-   state->dev = state->dev->next;
+   state->dev = next_netdev(state->dev);
    if (!state->dev) {
     state->idev = NULL;
     goto out;
--- ./net/ipv6/addrconf.c.vedevbase Wed Jun 21 18:53:20 2006
+++ ./net/ipv6/addrconf.c Thu Jun 22 12:03:08 2006
@@ -470,7 +470,7 @@ static void addrconf_forward_change(void
  struct inet6_dev *idev;

  read_lock(&dev_base_lock);
- for (dev=dev_base; dev; dev=dev->next) {
+ for_each_netdev(dev) {
   read_lock(&addrconf_lock);
   idev = __in6_dev_get(dev);
   if (idev) {
@@ -889,7 +889,7 @@ int ipv6_dev_get_saddr(struct net_device
  read_lock(&dev_base_lock);
  read_lock(&addrconf_lock);

- for (dev = dev_base; dev; dev=dev->next) {
+ for_each_netdev(dev) {
   struct inet6_dev *idev;
   struct inet6_ifaddr *ifa;

@@ -1971,7 +1971,7 @@ static void sit_add_v4_addrs(struct inet
   return;
  }

-       for (dev = dev_base; dev != NULL; dev = dev->next) {
+ for_each_netdev(dev) {
   struct in_device * in_dev = __in_dev_get_rtnl(dev);
   if (in_dev && (dev->flags & IFF_UP)) {
    struct in_ifaddr * ifa;
@@ -2120,7 +2120,7 @@ static void ip6_tnl_add_linklocal(struct
    return;
  }
  /* then try to inherit it from any device */
- for (link_dev = dev_base; link_dev; link_dev = link_dev->next) {
+ for_each_netdev(link_dev) {
   if (!ipv6_inherit_linklocal(idev, link_dev))
    return;
  }
```

```
@@ -3005,18 +3005,19 @@ static int inet6_dump_addr(struct sk_buf
  struct ifmcaddr6 *ifmca;
  struct ifacaddr6 *ifaca;

+ idx = 0;
  s_idx = cb->args[0];
  s_ip_idx = ip_idx = cb->args[1];
  read_lock(&dev_base_lock);

- for (dev = dev_base, idx = 0; dev; dev = dev->next, idx++) {
+ for_each_netdev(dev) {
   if (idx < s_idx)
-   continue;
+   goto cont;
   if (idx > s_idx)
    s_ip_idx = 0;
   ip_idx = 0;
   if ((idev = in6_dev_get(dev)) == NULL)
-   continue;
+   goto cont;
   read_lock_bh(&idev->lock);
   switch (type) {
   case UNICAST_ADDR:
@@ -3063,6 +3064,8 @@ static int inet6_dump_addr(struct sk_buf
   }
   read_unlock_bh(&idev->lock);
   in6_dev_put(idev);
+cont:
+  idx++;
  }
 done:
  if (err <= 0) {
@@ -3230,17 +3233,20 @@ static int inet6_dump_ifinfo(struct sk_b
  struct net_device *dev;
  struct inet6_dev *idev;

+ idx = 0;
  read_lock(&dev_base_lock);
- for (dev=dev_base, idx=0; dev; dev = dev->next, idx++) {
+ for_each_netdev(dev) {
   if (idx < s_idx)
-   continue;
+   goto cont;
   if ((idev = in6_dev_get(dev)) == NULL)
-   continue;
+   goto cont;
   err = inet6_fill_ifinfo(skb, idev, NETLINK_CB(cb->skb).pid,
     cb->nlh->nlmsg_seq, RTM_NEWLINK, NLM_F_MULTI);
```

```
   in6_dev_put(idev);
   if (err <= 0)
    break;
+cont:
+  idx++;
  }
  read_unlock(&dev_base_lock);
  cb->args[0] = idx;
@@ -3864,7 +3870,7 @@ void __exit addrconf_cleanup(void)
   * clean dev list.
   */

- for (dev=dev_base; dev; dev=dev->next) {
+ for_each_netdev(dev) {
   if ((idev = __in6_dev_get(dev)) == NULL)
    continue;
   addrconf_ifdown(dev, 1);
--- ./net/ipv6/anycast.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/ipv6/anycast.c Thu Jun 22 12:03:08 2006
@@ -428,11 +428,13 @@ int ipv6_chk_acast_addr(struct net_devic
  if (dev)
   return ipv6_chk_acast_dev(dev, addr);
  read_lock(&dev_base_lock);
- for (dev=dev_base; dev; dev=dev->next)
-  if (ipv6_chk_acast_dev(dev, addr))
-   break;
+ for_each_netdev(dev)
+  if (ipv6_chk_acast_dev(dev, addr)) {
+   read_unlock(&dev_base_lock);
+   return 1;
+  }
  read_unlock(&dev_base_lock);
- return dev != 0;
+ return 0;
 }


@@ -446,19 +448,21 @@ struct ac6_iter_state {

 static inline struct ifacaddr6 *ac6_get_first(struct seq_file *seq)
 {
+ struct net_device *dev;
  struct ifacaddr6 *im = NULL;
  struct ac6_iter_state *state = ac6_seq_private(seq);

- for (state->dev = dev_base, state->idev = NULL;
-      state->dev;
-      state->dev = state->dev->next) {
```

```
+ state->dev = NULL;
+ state->idev = NULL;
+ for_each_netdev(dev) {
   struct inet6_dev *idev;
-  idev = in6_dev_get(state->dev);
+  idev = in6_dev_get(dev);
   if (!idev)
    continue;
   read_lock_bh(&idev->lock);
   im = idev->ac_list;
   if (im) {
+   state->dev = dev;
    state->idev = idev;
    break;
   }
@@ -477,7 +481,7 @@ static struct ifacaddr6 *ac6_get_next(st
    read_unlock_bh(&state->idev->lock);
    in6_dev_put(state->idev);
   }
-  state->dev = state->dev->next;
+  state->dev = next_netdev(state->dev);
   if (!state->dev) {
    state->idev = NULL;
    break;
--- ./net/ipv6/mcast.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/ipv6/mcast.c Thu Jun 22 12:03:08 2006
@@ -2326,9 +2326,8 @@ static inline struct ifmcaddr6 *igmp6_mc
  struct ifmcaddr6 *im = NULL;
  struct igmp6_mc_iter_state *state = igmp6_mc_seq_private(seq);

- for (state->dev = dev_base, state->idev = NULL;
-      state->dev;
-      state->dev = state->dev->next) {
+ state->idev = NULL;
+ for_each_netdev(state->dev) {
   struct inet6_dev *idev;
   idev = in6_dev_get(state->dev);
   if (!idev)
@@ -2355,7 +2354,7 @@ static struct ifmcaddr6 *igmp6_mc_get_ne
    read_unlock_bh(&state->idev->lock);
    in6_dev_put(state->idev);
   }
-  state->dev = state->dev->next;
+  state->dev = next_netdev(state->dev);
   if (!state->dev) {
    state->idev = NULL;
    break;
@@ -2466,15 +2465,17 @@ struct igmp6_mcf_iter_state {
```

```
 static inline struct ip6_sf_list *igmp6_mcf_get_first(struct seq_file *seq)
 {
+ struct net_device *dev;
  struct ip6_sf_list *psf = NULL;
  struct ifmcaddr6 *im = NULL;
  struct igmp6_mcf_iter_state *state = igmp6_mcf_seq_private(seq);

- for (state->dev = dev_base, state->idev = NULL, state->im = NULL;
-      state->dev;
-      state->dev = state->dev->next) {
+ state->dev = NULL;
+ state->im = NULL;
+ state->idev = NULL;
+ for_each_netdev(dev) {
   struct inet6_dev *idev;
-  idev = in6_dev_get(state->dev);
+  idev = in6_dev_get(dev);
   if (unlikely(idev == NULL))
    continue;
   read_lock_bh(&idev->lock);
@@ -2483,6 +2484,7 @@ static inline struct ip6_sf_list *igmp6_
    spin_lock_bh(&im->mca_lock);
    psf = im->mca_sources;
    if (likely(psf != NULL)) {
+    state->dev = dev;
     state->im = im;
     state->idev = idev;
     break;
@@ -2508,7 +2510,7 @@ static struct ip6_sf_list *igmp6_mcf_get
     read_unlock_bh(&state->idev->lock);
     in6_dev_put(state->idev);
    }
-   state->dev = state->dev->next;
+   state->dev = next_netdev(state->dev);
    if (!state->dev) {
     state->idev = NULL;
     goto out;
--- ./net/llc/llc_core.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/llc/llc_core.c Thu Jun 22 12:03:08 2006
@@ -161,8 +161,11 @@ static struct packet_type llc_tr_packet_

 static int __init llc_init(void)
 {
- if (dev_base->next)
-  memcpy(llc_station_mac_sa, dev_base->next->dev_addr, ETH_ALEN);
+ struct net_device *dev;
+
```

```
+ dev = next_netdev(first_netdev());
+ if (dev)
+  memcpy(llc_station_mac_sa, dev->dev_addr, ETH_ALEN);
  else
   memset(llc_station_mac_sa, 0, ETH_ALEN);
  dev_add_pack(&llc_packet_type);
--- ./net/netrom/nr_route.c.vedevbase Mon Mar 20 08:53:29 2006
+++ ./net/netrom/nr_route.c Thu Jun 22 12:03:08 2006
@@ -595,7 +595,7 @@ struct net_device *nr_dev_first(void)
  struct net_device *dev, *first = NULL;

  read_lock(&dev_base_lock);
- for (dev = dev_base; dev != NULL; dev = dev->next) {
+ for_each_netdev(dev) {
   if ((dev->flags & IFF_UP) && dev->type == ARPHRD_NETROM)
    if (first == NULL || strncmp(dev->name, first->name, 3) < 0)
     first = dev;
@@ -615,7 +615,7 @@ struct net_device *nr_dev_get(ax25_addre
  struct net_device *dev;

  read_lock(&dev_base_lock);
- for (dev = dev_base; dev != NULL; dev = dev->next) {
+ for_each_netdev(dev) {
   if ((dev->flags & IFF_UP) && dev->type == ARPHRD_NETROM && ax25cmp(addr,
(ax25_address *)dev->dev_addr) == 0) {
    dev_hold(dev);
    goto out;
--- ./net/rose/rose_route.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/rose/rose_route.c Thu Jun 22 12:03:08 2006
@@ -600,7 +600,7 @@ struct net_device *rose_dev_first(void)
  struct net_device *dev, *first = NULL;

  read_lock(&dev_base_lock);
- for (dev = dev_base; dev != NULL; dev = dev->next) {
+ for_each_netdev(dev) {
   if ((dev->flags & IFF_UP) && dev->type == ARPHRD_ROSE)
    if (first == NULL || strncmp(dev->name, first->name, 3) < 0)
     first = dev;
@@ -618,12 +618,13 @@ struct net_device *rose_dev_get(rose_add
  struct net_device *dev;

  read_lock(&dev_base_lock);
- for (dev = dev_base; dev != NULL; dev = dev->next) {
+ for_each_netdev(dev) {
   if ((dev->flags & IFF_UP) && dev->type == ARPHRD_ROSE && rosecmp(addr, (rose_address
*)dev->dev_addr) == 0) {
    dev_hold(dev);
    goto out;
```

```
   }
  }
+ dev = NULL;
 out:
  read_unlock(&dev_base_lock);
  return dev;
@@ -634,10 +635,11 @@ static int rose_dev_exists(rose_address
  struct net_device *dev;

  read_lock(&dev_base_lock);
- for (dev = dev_base; dev != NULL; dev = dev->next) {
+ for_each_netdev(dev) {
   if ((dev->flags & IFF_UP) && dev->type == ARPHRD_ROSE && rosecmp(addr, (rose_address
*)dev->dev_addr) == 0)
    goto out;
  }
+ dev = NULL;
 out:
  read_unlock(&dev_base_lock);
  return dev != NULL;
--- ./net/sched/sch_api.c.vedevbase Mon Mar 20 08:53:29 2006
+++ ./net/sched/sch_api.c Thu Jun 22 12:03:08 2006
@@ -830,12 +830,15 @@ static int tc_dump_qdisc(struct sk_buff
  struct net_device *dev;
  struct Qdisc *q;

+ idx = 0;
  s_idx = cb->args[0];
  s_q_idx = q_idx = cb->args[1];
  read_lock(&dev_base_lock);
- for (dev=dev_base, idx=0; dev; dev = dev->next, idx++) {
-  if (idx < s_idx)
+ for_each_netdev(dev) {
+  if (idx < s_idx) {
+   idx++;
    continue;
+  }
   if (idx > s_idx)
    s_q_idx = 0;
   read_lock_bh(&qdisc_tree_lock);
@@ -853,6 +856,7 @@ static int tc_dump_qdisc(struct sk_buff
    q_idx++;
   }
   read_unlock_bh(&qdisc_tree_lock);
+  idx++;
  }

 done:
```

```
--- ./net/sctp/protocol.c.vedevbase Wed Jun 21 18:53:23 2006
+++ ./net/sctp/protocol.c Thu Jun 22 12:03:08 2006
@@ -177,7 +177,7 @@ static void __sctp_get_local_addr_list(v
  struct sctp_af *af;

  read_lock(&dev_base_lock);
- for (dev = dev_base; dev; dev = dev->next) {
+ for_each_netdev(dev) {
   __list_for_each(pos, &sctp_address_families) {
    af = list_entry(pos, struct sctp_af, list);
    af->copy_addrlist(&sctp_local_addr_list, dev);
--- ./net/tipc/eth_media.c.vedevbase Wed Jun 21 18:51:09 2006
+++ ./net/tipc/eth_media.c Thu Jun 22 12:03:08 2006
@@ -118,17 +118,19 @@ static int recv_msg(struct sk_buff *buf,

 static int enable_bearer(struct tipc_bearer *tb_ptr)
 {
- struct net_device *dev = dev_base;
+ struct net_device *pdev, *dev;
  struct eth_bearer *eb_ptr = &eth_bearers[0];
  struct eth_bearer *stop = &eth_bearers[MAX_ETH_BEARERS];
  char *driver_name = strchr((const char *)tb_ptr->name, ':') + 1;

  /* Find device with specified name */

- while (dev && dev->name &&
-        (memcmp(dev->name, driver_name, strlen(dev->name)))) {
-  dev = dev->next;
- }
+ dev = NULL;
+ for_each_netdev(pdev)
+  if (!memcmp(pdev->name, driver_name, strlen(pdev->name))) {
+   dev = pdev;
+   break;
+  }
  if (!dev)
   return -ENODEV;
```

Subject: [patch 2/4] Network namespaces: cleanup of dev_base list use
Posted by Andrey Savochkin on Mon, 26 Jun 2006 09:52:50 GMT
View Forum Message <> Reply to Message

CONFIG_NET_NS and net_namespace structure are introduced.
List of network devices is made per-namespace.
Each namespace gets its own loopback device.

Task's net_namespace pointer is not incorporated into nsproxy structure,

since current namespace changes temporarily for processing of packets
in softirq.

Signed-off-by: Andrey Savochkin <saw@swsoft.com>
---
 drivers/net/loopback.c | 70 ++++++++++++--------
 include/linux/init_task.h | 9 ++
 include/linux/net_ns.h | 88 ++++++++++++++++++++++++
 include/linux/netdevice.h | 20 ++++-
 include/linux/nsproxy.h | 3
 include/linux/sched.h | 3
 kernel/nsproxy.c | 14 +++
 net/Kconfig | 7 +
 net/core/dev.c | 162 ++-
 net/core/net-sysfs.c | 24 ++++++
 net/ipv4/devinet.c | 2
 net/ipv6/addrconf.c | 2
 net/ipv6/route.c | 3
 13 files changed, 371 insertions, 36 deletions

--- ./drivers/net/loopback.c.venshd Wed Jun 21 18:50:39 2006
+++ ./drivers/net/loopback.c Fri Jun 23 11:48:09 2006
@@ -196,42 +196,56 @@ static struct ethtool_ops loopback_ethto
  .set_tso = ethtool_op_set_tso,
 };

-struct net_device loopback_dev = {
- .name   = "lo",
- .mtu   = (16 * 1024) + 20 + 20 + 12,
- .hard_start_xmit = loopback_xmit,
- .hard_header = eth_header,
- .hard_header_cache = eth_header_cache,
- .header_cache_update = eth_header_cache_update,
- .hard_header_len = ETH_HLEN, /* 14 */
- .addr_len = ETH_ALEN, /* 6 */
- .tx_queue_len = 0,
- .type   = ARPHRD_LOOPBACK, /* 0x0001*/
- .rebuild_header = eth_rebuild_header,
- .flags   = IFF_LOOPBACK,
- .features   = NETIF_F_SG | NETIF_F_FRAGLIST
+struct net_device loopback_dev_static;
+EXPORT_SYMBOL(loopback_dev_static);
+
+void loopback_dev_dtor(struct net_device *dev)
+{
+ if (dev->priv) {
+ kfree(dev->priv);
+ dev->priv = NULL;

```
+ }
+ free_netdev(dev);
+}
+
+void loopback_dev_ctor(struct net_device *dev)
+{
+ struct net_device_stats *stats;
+
+ memset(dev, 0, sizeof(*dev));
+ strcpy(dev->name, "lo");
+ dev->mtu  = (16 * 1024) + 20 + 20 + 12;
+ dev->hard_start_xmit = loopback_xmit;
+ dev->hard_header = eth_header;
+ dev->hard_header_cache = eth_header_cache;
+ dev->header_cache_update = eth_header_cache_update;
+ dev->hard_header_len = ETH_HLEN; /* 14 */
+ dev->addr_len  = ETH_ALEN; /* 6 */
+ dev->tx_queue_len = 0;
+ dev->type  = ARPHRD_LOOPBACK; /* 0x0001*/
+ dev->rebuild_header = eth_rebuild_header;
+ dev->flags  = IFF_LOOPBACK;
+ dev->features   = NETIF_F_SG | NETIF_F_FRAGLIST
 #ifdef LOOPBACK_TSO
       | NETIF_F_TSO
 #endif
       | NETIF_F_NO_CSUM | NETIF_F_HIGHDMA
-      | NETIF_F_LLTX,
- .ethtool_ops  = &loopback_ethtool_ops,
-};
-
-/* Setup and register the loopback device. */
-int __init loopback_init(void)
-{
- struct net_device_stats *stats;
+      | NETIF_F_LLTX
+      | NETIF_F_NSOK;
+ dev->ethtool_ops = &loopback_ethtool_ops;

  /* Can survive without statistics */
  stats = kmalloc(sizeof(struct net_device_stats), GFP_KERNEL);
  if (stats) {
   memset(stats, 0, sizeof(struct net_device_stats));
-  loopback_dev.priv = stats;
-  loopback_dev.get_stats = &get_stats;
+  dev->priv = stats;
+  dev->get_stats = &get_stats;
  }
-
```

```
- return register_netdev(&loopback_dev);
-};
+}

-EXPORT_SYMBOL(loopback_dev);
+/* Setup and register the loopback device. */
+int __init loopback_init(void)
+{
+ loopback_dev_ctor(&loopback_dev_static);
+ return register_netdev(&loopback_dev_static);
+};
--- ./include/linux/init_task.h.venshd Wed Jun 21 18:53:16 2006
+++ ./include/linux/init_task.h Fri Jun 23 11:48:09 2006
@@ -87,6 +87,14 @@ extern struct nsproxy init_nsproxy;

 extern struct group_info init_groups;

+#ifdef CONFIG_NET_NS
+extern struct net_namespace init_net_ns;
+#define INIT_NET_NS \
+ .net_context = &init_net_ns,
+#else
+#define INIT_NET_NS
+#endif
+
 /*
  *  INIT_TASK is used to set up the first task table, touch at
  * your own risk!. Base=0, limit=0x1fffff (=2MB)
@@ -129,6 +137,7 @@ extern struct group_info init_groups;
  .signal  = &init_signals,    \
  .sighand = &init_sighand,    \
  .nsproxy = &init_nsproxy,    \
+ INIT_NET_NS       \
  .pending = {      \
   .list = LIST_HEAD_INIT(tsk.pending.list),  \
   .signal = {{0}}},     \
--- ./include/linux/net_ns.h.venshd Thu Jun 22 12:10:13 2006
+++ ./include/linux/net_ns.h Fri Jun 23 11:49:42 2006
@@ -0,0 +1,88 @@
+/*
+ * Copyright (C) 2006  SWsoft
+ */
+#ifndef __LINUX_NET_NS__
+#define __LINUX_NET_NS__
+
+#ifdef CONFIG_NET_NS
+
+#include <asm/atomic.h>
```

```
+#include <linux/list.h>
+#include <linux/workqueue.h>
+
+struct net_namespace {
+ atomic_t  active_ref, use_ref;
+ struct list_head dev_base;
+ struct net_device *loopback;
+ unsigned int  hash;
+ struct execute_work destroy_work;
+};
+
+static inline struct net_namespace *get_net_ns(struct net_namespace *ns)
+{
+ atomic_inc(&ns->active_ref);
+ return ns;
+}
+
+extern void net_ns_stop(struct net_namespace *ns);
+static inline void put_net_ns(struct net_namespace *ns)
+{
+ if (atomic_dec_and_test(&ns->active_ref))
+  net_ns_stop(ns);
+}
+
+static inline struct net_namespace *ref_net_ns(struct net_namespace *ns)
+{
+ atomic_inc(&ns->use_ref);
+ return ns;
+}
+
+extern void net_ns_free(struct net_namespace *ns);
+static inline void unref_net_ns(struct net_namespace *ns)
+{
+ if (atomic_dec_and_test(&ns->use_ref))
+  net_ns_free(ns);
+}
+
+extern struct net_namespace init_net_ns;
+#define current_net_ns  (current->net_context)
+
+#define push_net_ns(to, orig) do { \
+     task_t *__cur; \
+     __cur = current; \
+     orig = __cur->net_context; \
+     __cur->net_context = ref_net_ns(to); \
+    } while (0)
+#define pop_net_ns(orig) do { \
+     task_t *__cur; \
```

```
+     struct net_namespace *__cur_ns; \
+     __cur = current; \
+     __cur_ns = __cur->net_context; \
+     __cur->net_context = orig; \
+     unref_net_ns(__cur_ns); \
+    } while (0)
+#define switch_net_ns(to) do { \
+     task_t *__cur; \
+     struct net_namespace *__cur_ns; \
+     __cur = current; \
+     __cur_ns = __cur->net_context; \
+     __cur->net_context = ref_net_ns(to); \
+     unref_net_ns(__cur_ns); \
+    } while (0)
+
+#define net_ns_same(target, context) ((target) == (context))
+
+#else /* CONFIG_NET_NS */
+
+struct net_namespace;
+
+#define get_net_ns(x)  NULL
+#define put_net_ns(x)  ((void)0)
+
+#define current_net_ns  NULL
+
+#define net_ns_same(target, context) 1
+
+#endif /* CONFIG_NET_NS */
+
+#endif /* __LINUX_NET_NS__ */
--- ./include/linux/netdevice.h.venshd Thu Jun 22 18:57:50 2006
+++ ./include/linux/netdevice.h Fri Jun 23 11:48:15 2006
@@ -311,6 +311,7 @@ struct net_device
 #define NETIF_F_TSO  2048 /* Can offload TCP/IP segmentation */
 #define NETIF_F_LLTX  4096 /* LockLess TX */
 #define NETIF_F_UFO             8192    /* Can offload UDP Large Send*/
+#define NETIF_F_NSOK  16384 /* OK for namespaces */

 #define NETIF_F_GEN_CSUM (NETIF_F_NO_CSUM | NETIF_F_HW_CSUM)
 #define NETIF_F_ALL_CSUM (NETIF_F_IP_CSUM | NETIF_F_GEN_CSUM)
@@ -366,6 +367,10 @@ struct net_device
  int   promiscuity;
  int   allmulti;

+#ifdef CONFIG_NET_NS
+ struct net_namespace *net_ns;
+#endif
```

```
+

  /* Protocol specific pointers */

@@ -542,17 +547,26 @@ struct packet_type {

 #include <linux/interrupt.h>
 #include <linux/notifier.h>
+#include <linux/net_ns.h>

-extern struct net_device loopback_dev;  /* The loopback */
+extern struct net_device loopback_dev_static;
+#ifndef CONFIG_NET_NS
+#define loopback_dev  loopback_dev_static /* The loopback */
 extern struct list_head  dev_base_head;  /* All devices */
+#else
+#define loopback_dev  (*current_net_ns->loopback)
+#define dev_base_head  (current_net_ns->dev_base)
+#endif
 extern rwlock_t   dev_base_lock;  /* Device list lock */

 #define for_each_netdev(p) list_for_each_entry(p, &dev_base_head, dev_list)

 /* DO NOT USE first_netdev/next_netdev, use loop defined above */
 #define first_netdev()  ({ \
-     list_empty(&dev_base_head) ? NULL : \
-      list_entry(dev_base_head.next, \
+     struct list_head *__base; \
+      __base = &dev_base_head; \
+     list_empty(__base) ? NULL : \
+      list_entry(__base->next, \
        struct net_device, \
        dev_list); \
      })
--- ./include/linux/nsproxy.h.venshd Wed Jun 21 18:53:17 2006
+++ ./include/linux/nsproxy.h Fri Jun 23 11:48:15 2006
@@ -33,6 +33,7 @@ struct nsproxy *dup_namespaces(struct ns
 int copy_namespaces(int flags, struct task_struct *tsk);
 void get_task_namespaces(struct task_struct *tsk);
 void free_nsproxy(struct nsproxy *ns);
+void release_net_context(struct task_struct *tsk);

 static inline void put_nsproxy(struct nsproxy *ns)
 {
@@ -48,5 +49,7 @@ static inline void exit_task_namespaces(
   put_nsproxy(ns);
   p->nsproxy = NULL;
  }
```

```
+ release_net_context(p);
 }
+
 #endif
--- ./include/linux/sched.h.venshd Wed Jun 21 18:53:17 2006
+++ ./include/linux/sched.h Fri Jun 23 11:48:15 2006
@@ -887,6 +887,9 @@ struct task_struct {
  struct files_struct *files;
 /* namespaces */
  struct nsproxy *nsproxy;
+#ifdef CONFIG_NET_NS
+ struct net_namespace *net_context;
+#endif
 /* signal handlers */
  struct signal_struct *signal;
  struct sighand_struct *sighand;
--- ./kernel/nsproxy.c.venshd Wed Jun 21 18:53:17 2006
+++ ./kernel/nsproxy.c Fri Jun 23 11:48:15 2006
@@ -16,6 +16,7 @@
 #include <linux/module.h>
 #include <linux/version.h>
 #include <linux/nsproxy.h>
+#include <linux/net_ns.h>
 #include <linux/namespace.h>
 #include <linux/utsname.h>

@@ -84,6 +85,7 @@ int copy_namespaces(int flags, struct ta
   return 0;

  get_nsproxy(old_ns);
+ (void) get_net_ns(tsk->net_context); /* for pointer copied by memcpy */

  if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
   return 0;
@@ -134,3 +136,15 @@ void free_nsproxy(struct nsproxy *ns)
    put_ipc_ns(ns->ipc_ns);
   kfree(ns);
 }
+
+void release_net_context(struct task_struct *tsk)
+{
+#ifdef CONFIG_NET_NS
+ struct net_namespace *net_ns;
+
+ net_ns = tsk->net_context;
+ /* do not get refcounter here, nobody can put it later */
+ tsk->net_context = &init_net_ns;
+ put_net_ns(net_ns);
```

```
+#endif
+}
--- ./net/Kconfig.venshd Wed Jun 21 18:53:22 2006
+++ ./net/Kconfig Fri Jun 23 11:48:15 2006
@@ -66,6 +66,13 @@ source "net/ipv6/Kconfig"

 endif # if INET

+config NET_NS
+ bool "Network Namespaces"
+ help
+   This option enables multiple independent network namespaces,
+   each having own network devices, IP addresses, routes, and so on.
+   If unsure, answer N.
+
 config NETWORK_SECMARK
  bool "Security Marking"
  help
--- ./net/core/dev.c.venshd Thu Jun 22 17:40:13 2006
+++ ./net/core/dev.c Fri Jun 23 11:48:15 2006
@@ -91,6 +91,7 @@
 #include <linux/if_ether.h>
 #include <linux/netdevice.h>
 #include <linux/etherdevice.h>
+#include <linux/net_ns.h>
 #include <linux/notifier.h>
 #include <linux/skbuff.h>
 #include <net/sock.h>
@@ -177,8 +178,10 @@ static spinlock_t net_dma_event_lock;
 DEFINE_RWLOCK(dev_base_lock);
 EXPORT_SYMBOL(dev_base_lock);

+#ifndef CONFIG_NET_NS
 LIST_HEAD(dev_base_head);
 EXPORT_SYMBOL(dev_base_head);
+#endif

 #define NETDEV_HASHBITS 8
 static struct hlist_head dev_name_head[1<<NETDEV_HASHBITS];
@@ -187,6 +190,9 @@ static struct hlist_head dev_index_head[
 static inline struct hlist_head *dev_name_hash(const char *name)
 {
  unsigned hash = full_name_hash(name, strnlen(name, IFNAMSIZ));
+#ifdef CONFIG_NET_NS
+ hash ^= current_net_ns->hash;
+#endif
  return &dev_name_head[hash & ((1<<NETDEV_HASHBITS)-1)];
 }
```

```
@@ -211,10 +217,12 @@ DEFINE_PER_CPU(struct softnet_data, soft
 extern int netdev_sysfs_init(void);
 extern int netdev_register_sysfs(struct net_device *);
 extern void netdev_unregister_sysfs(struct net_device *);
+extern int netdev_rename_sysfs(struct net_device *);
 #else
 #define netdev_sysfs_init()   (0)
 #define netdev_register_sysfs(dev) (0)
 #define netdev_unregister_sysfs(dev) do { } while(0)
+#define netdev_rename_sysfs(dev) (0)
 #endif


@@ -474,10 +482,13 @@ __setup("netdev=", netdev_boot_setup);
 struct net_device *__dev_get_by_name(const char *name)
 {
  struct hlist_node *p;
+ struct net_namespace *ns __attribute_used__ = current_net_ns;

  hlist_for_each(p, dev_name_hash(name)) {
   struct net_device *dev
    = hlist_entry(p, struct net_device, name_hlist);
+  if (!net_ns_same(dev->net_ns, ns))
+   continue;
   if (!strncmp(dev->name, name, IFNAMSIZ))
    return dev;
  }
@@ -740,7 +751,7 @@ int dev_change_name(struct net_device *d
  else
   strlcpy(dev->name, newname, IFNAMSIZ);

- err = class_device_rename(&dev->class_dev, dev->name);
+ err = netdev_rename_sysfs(dev);
  if (!err) {
   hlist_del(&dev->name_hlist);
   hlist_add_head(&dev->name_hlist, dev_name_hash(dev->name));
@@ -1531,7 +1542,14 @@ static void net_tx_action(struct softirq
    clear_bit(__LINK_STATE_SCHED, &dev->state);

    if (spin_trylock(&dev->queue_lock)) {
+#ifdef CONFIG_NET_NS
+    struct net_namespace *orig_net_ns;
+    push_net_ns(dev->net_ns, orig_net_ns);
+#endif
     qdisc_run(dev);
+#ifdef CONFIG_NET_NS
+    pop_net_ns(orig_net_ns);
```

```
+#endif
     spin_unlock(&dev->queue_lock);
    } else {
     netif_schedule(dev);
@@ -1618,6 +1636,7 @@ int netif_receive_skb(struct sk_buff *sk
 {
  struct packet_type *ptype, *pt_prev;
  struct net_device *orig_dev;
+ struct net_namespace *orig_net_ns __attribute_used__;
  int ret = NET_RX_DROP;
  unsigned short type;

@@ -1636,6 +1655,10 @@ int netif_receive_skb(struct sk_buff *sk
  if (!orig_dev)
   return NET_RX_DROP;

+#ifdef CONFIG_NET_NS
+ push_net_ns(skb->dev->net_ns, orig_net_ns);
+#endif
+
  __get_cpu_var(netdev_rx_stat).total++;

  skb->h.raw = skb->nh.raw = skb->data;
@@ -1706,6 +1729,9 @@ ncls:

 out:
  rcu_read_unlock();
+#ifdef CONFIG_NET_NS
+ pop_net_ns(orig_net_ns);
+#endif
  return ret;
 }

@@ -2732,6 +2758,7 @@ int register_netdevice(struct net_device
 {
  struct hlist_head *head;
  struct hlist_node *p;
+ struct net_namespace *ns __attribute_used__ = current_net_ns;
  int ret;

  BUG_ON(dev_boot_phase);
@@ -2749,9 +2776,19 @@ int register_netdevice(struct net_device
  spin_lock_init(&dev->ingress_lock);
 #endif

+#ifdef CONFIG_NET_NS
+ dev->net_ns = ref_net_ns(ns);
+ /*
```

```
+  * loopback device doesn't hold active reference: it doesn't prevent
+  * stopping of net_namespace
+  */
+ if (dev != ns->loopback)
+  get_net_ns(ns);
+#endif
+
  ret = alloc_divert_blk(dev);
  if (ret)
-  goto out;
+  goto out_divert;

  dev->iflink = -1;

@@ -2779,6 +2816,8 @@ int register_netdevice(struct net_device
  hlist_for_each(p, head) {
   struct net_device *d
    = hlist_entry(p, struct net_device, name_hlist);
+  if (!net_ns_same(d->net_ns, ns))
+   continue;
   if (!strncmp(d->name, dev->name, IFNAMSIZ)) {
    ret = -EEXIST;
     goto out_err;
@@ -2852,6 +2891,13 @@ out:
  return ret;
 out_err:
  free_divert_blk(dev);
+out_divert:
+#ifdef CONFIG_NET_NS
+ unref_net_ns(ns);
+ if (dev != ns->loopback)
+  put_net_ns(ns);
+ dev->net_ns = NULL;
+#endif
  goto out;
 }

@@ -2977,9 +3023,13 @@ static DEFINE_MUTEX(net_todo_run_mutex);
 void netdev_run_todo(void)
 {
  struct list_head list;
+ struct net_namespace *orig_net_ns __attribute_used__;

  /* Need to guard against multiple cpu's getting out of order. */
  mutex_lock(&net_todo_run_mutex);
+#ifdef CONFIG_NET_NS
+ push_net_ns(current_net_ns, orig_net_ns);
+#endif
```

```
  /* Not safe to do outside the semaphore.  We must not return
   * until all unregister events invoked by the local processor
@@ -3006,6 +3056,9 @@ void netdev_run_todo(void)
    continue;
   }

+#ifdef CONFIG_NET_NS
+  switch_net_ns(dev->net_ns);
+#endif
   netdev_unregister_sysfs(dev);
   dev->reg_state = NETREG_UNREGISTERED;

@@ -3025,6 +3078,9 @@ void netdev_run_todo(void)
  }

 out:
+#ifdef CONFIG_NET_NS
+ pop_net_ns(orig_net_ns);
+#endif
  mutex_unlock(&net_todo_run_mutex);
 }

@@ -3077,6 +3133,17 @@ EXPORT_SYMBOL(alloc_netdev);
  */
 void free_netdev(struct net_device *dev)
 {
+#ifdef CONFIG_NET_NS
+ struct net_namespace *ns;
+
+ ns = dev->net_ns;
+ if (ns != NULL) {
+  unref_net_ns(ns);
+  if (dev != ns->loopback)
+   put_net_ns(ns);
+  dev->net_ns = NULL;
+ }
+#endif
 #ifdef CONFIG_SYSFS
  /*  Compatibility with error handling in drivers */
  if (dev->reg_state == NETREG_UNINITIALIZED) {
@@ -3087,6 +3154,13 @@ void free_netdev(struct net_device *dev)
  BUG_ON(dev->reg_state != NETREG_UNREGISTERED);
  dev->reg_state = NETREG_RELEASED;

+#ifdef CONFIG_NET_NS
+ if (ns != NULL && ns != &init_net_ns) {
+  kfree((char *)dev - dev->padded);
```

```
+  return;
+ }
+#endif
+
  /* will free via class release */
  class_device_put(&dev->class_dev);
 #else
@@ -3323,6 +3397,90 @@ static int __init netdev_dma_register(vo
 static int __init netdev_dma_register(void) { return -ENODEV; }
 #endif /* CONFIG_NET_DMA */

+#ifdef CONFIG_NET_NS
+struct net_namespace init_net_ns = {
+ .active_ref = ATOMIC_INIT(2),
+     /* one for init_task->net_context,
+        one not to let init_net_ns go away */
+ .use_ref = ATOMIC_INIT(1), /* for active references */
+ .dev_base = LIST_HEAD_INIT(init_net_ns.dev_base),
+ .loopback = &loopback_dev_static,
+};
+
+extern void loopback_dev_ctor(struct net_device *dev);
+extern void loopback_dev_dtor(struct net_device *dev);
+int net_ns_start(void)
+{
+ struct net_namespace *ns, *orig_ns;
+ struct net_device *dev;
+ task_t *task;
+ int err;
+
+ err = -ENOMEM;
+ ns = kmalloc(sizeof(*ns), GFP_KERNEL);
+ if (ns == NULL)
+  goto out_ns;
+ dev = kmalloc(sizeof(*dev), GFP_KERNEL);
+ if (dev == NULL)
+  goto out_dev;
+ loopback_dev_ctor(dev);
+ dev->destructor = loopback_dev_dtor;
+
+ memset(ns, 0, sizeof(*ns));
+ atomic_set(&ns->active_ref, 1);
+ atomic_set(&ns->use_ref, 1);
+ INIT_LIST_HEAD(&ns->dev_base);
+ ns->hash = net_random();
+ ns->loopback = dev;
+
+ task = current;
```

```
+ orig_ns = task->net_context;
+ task->net_context = ns;
+ err = register_netdev(dev);
+ if (err)
+  goto out_register;
+ put_net_ns(orig_ns);
+ return 0;
+
+out_register:
+ dev->destructor(dev);
+ task->net_context = orig_ns;
+ BUG_ON(atomic_read(&ns->active_ref) != 1);
+out_dev:
+ kfree(ns);
+out_ns:
+ return err;
+}
+EXPORT_SYMBOL(net_ns_start);
+
+void net_ns_free(struct net_namespace *ns)
+{
+ kfree(ns);
+}
+EXPORT_SYMBOL(net_ns_free);
+
+/* destroy loopback device and protocol datastructures in process context */
+static void net_ns_destroy(void *data)
+{
+ struct net_namespace *ns, *orig_ns;
+
+ ns = data;
+ push_net_ns(ns, orig_ns);
+ unregister_netdev(ns->loopback);
+ BUG_ON(!list_empty(&ns->dev_base));
+ pop_net_ns(orig_ns);
+
+ /* drop (hopefully) final reference */
+ unref_net_ns(ns);
+}
+
+void net_ns_stop(struct net_namespace *ns)
+{
+ execute_in_process_context(net_ns_destroy, ns, &ns->destroy_work);
+}
+EXPORT_SYMBOL(net_ns_stop);
+#endif
+
 /*
```

```
  * Initialize the DEV module. At boot time this walks the device list and
  * unhooks any devices that fail to initialise (normally hardware not
--- ./net/core/net-sysfs.c.venshd Wed Jun 21 18:51:08 2006
+++ ./net/core/net-sysfs.c Fri Jun 23 11:48:15 2006
@@ -13,6 +13,7 @@
 #include <linux/config.h>
 #include <linux/kernel.h>
 #include <linux/netdevice.h>
+#include <linux/net_ns.h>
 #include <linux/if_arp.h>
 #include <net/sock.h>
 #include <linux/rtnetlink.h>
@@ -445,6 +446,12 @@ static struct class net_class = {

 void netdev_unregister_sysfs(struct net_device * net)
 {
+#ifdef CONFIG_NET_NS
+ if (current_net_ns != &init_net_ns)
+  /* not supported yet: sysfs virtualization is required */
+  return;
+#endif
+
  class_device_del(&(net->class_dev));
 }

@@ -454,6 +461,12 @@ int netdev_register_sysfs(struct net_dev
  struct class_device *class_dev = &(net->class_dev);
  struct attribute_group **groups = net->sysfs_groups;

+#ifdef CONFIG_NET_NS
+ if (current_net_ns != &init_net_ns)
+  /* not supported yet: sysfs virtualization is required */
+  return 0;
+#endif
+
  class_device_initialize(class_dev);
  class_dev->class = &net_class;
  class_dev->class_data = net;
@@ -474,6 +487,17 @@ int netdev_register_sysfs(struct net_dev
  return class_device_add(class_dev);
 }

+int netdev_rename_sysfs(struct net_device *dev)
+{
+#ifdef CONFIG_NET_NS
+ if (current_net_ns != &init_net_ns)
+  /* not supported yet: sysfs virtualization is required */
+  return 0;
```

```
+#endif
+
+ return class_device_rename(&dev->class_dev, dev->name);
+}
+
 int netdev_sysfs_init(void)
 {
  return class_register(&net_class);
--- ./net/ipv4/devinet.c.venshd Thu Jun 22 12:03:08 2006
+++ ./net/ipv4/devinet.c Fri Jun 23 11:48:15 2006
@@ -190,7 +190,7 @@ static void inetdev_destroy(struct in_de
  ASSERT_RTNL();

  dev = in_dev->dev;
- if (dev == &loopback_dev)
+ if (dev == &loopback_dev_static)
   return;

  in_dev->dead = 1;
--- ./net/ipv6/addrconf.c.venshd Thu Jun 22 12:03:08 2006
+++ ./net/ipv6/addrconf.c Fri Jun 23 11:48:15 2006
@@ -2277,7 +2277,7 @@ static int addrconf_ifdown(struct net_de

  ASSERT_RTNL();

- if (dev == &loopback_dev && how == 1)
+ if (dev == &loopback_dev_static && how == 1)
   how = 0;

  rt6_ifdown(dev);
--- ./net/ipv6/route.c.venshd Wed Jun 21 18:53:20 2006
+++ ./net/ipv6/route.c Fri Jun 23 11:48:15 2006
@@ -125,7 +125,7 @@ struct rt6_info ip6_null_entry = {
   .dst = {
    .__refcnt = ATOMIC_INIT(1),
    .__use  = 1,
-   .dev  = &loopback_dev,
+   /* .dev  = &loopback_dev, */
    .obsolete = -1,
    .error  = -ENETUNREACH,
    .metrics = { [RTAX_HOPLIMIT - 1] = 255, },
@@ -2268,6 +2268,7 @@ void __init ip6_route_init(void)
 #ifdef CONFIG_XFRM
  xfrm6_init();
 #endif
+ ip6_null_entry.u.dst.dev = &loopback_dev;
 }
```

void ip6_route_cleanup(void)

Subject: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Andrey Savochkin on Mon, 26 Jun 2006 09:54:27 GMT
View Forum Message <> Reply to Message

Structures related to IPv4 rounting (FIB and routing cache)
are made per-namespace.

Signed-off-by: Andrey Savochkin <saw@swsoft.com>
---
 include/linux/net_ns.h | 9 +++
 include/net/flow.h | 3 +
 include/net/ip_fib.h | 62 ++++++++++++++++++++-----
 net/core/dev.c | 7 ++
 net/ipv4/Kconfig | 4 -
 net/ipv4/fib_frontend.c | 87 ++++++++++++++++++++++++++++++------
 net/ipv4/fib_hash.c | 13 ++++-
 net/ipv4/fib_rules.c | 114 ++
 net/ipv4/fib_semantics.c | 104 ++++++++++++++++++++++++++++++++++++++-----
 net/ipv4/route.c | 26 ++++++++++
 10 files changed, 348 insertions, 81 deletions

--- ./include/linux/net_ns.h.vensrt Fri Jun 23 11:49:42 2006
+++ ./include/linux/net_ns.h Fri Jun 23 11:50:16 2006
@@ -14,7 +14,16 @@ struct net_namespace {
  atomic_t active_ref, use_ref;
  struct list_head dev_base;
  struct net_device *loopback;
+#ifndef CONFIG_IP_MULTIPLE_TABLES
+ struct fib_table *fib4_local_table, *fib4_main_table;
+#else
+ struct fib_table **fib4_tables;
+ struct hlist_head fib4_rules;
+#endif
+ struct hlist_head *fib4_hash, *fib4_laddrhash;
+ unsigned fib4_hash_size, fib4_info_cnt;
  unsigned int hash;
+ char   destroying;
  struct execute_work destroy_work;
 };

--- ./include/net/flow.h.vensrt Wed Jun 21 18:51:08 2006
+++ ./include/net/flow.h Fri Jun 23 11:50:16 2006
@@ -78,6 +78,9 @@ struct flowi {
 #define fl_icmp_type uli_u.icmpt.type
 #define fl_icmp_code uli_u.icmpt.code

```
 #define fl_ipsec_spi uli_u.spi
+#ifdef CONFIG_NET_NS
+ struct net_namespace *net_ns;
+#endif
 } __attribute__((__aligned__(BITS_PER_LONG/8)));

 #define FLOW_DIR_IN 0
--- ./include/net/ip_fib.h.vensrt Wed Jun 21 18:53:17 2006
+++ ./include/net/ip_fib.h Fri Jun 23 11:50:16 2006
@@ -18,6 +18,7 @@

 #include <net/flow.h>
 #include <linux/seq_file.h>
+#include <linux/net_ns.h>

 /* WARNING: The ordering of these elements must match ordering
  *          of RTA_* rtnetlink attribute numbers.
@@ -169,14 +170,21 @@ struct fib_table {

 #ifndef CONFIG_IP_MULTIPLE_TABLES

-extern struct fib_table *ip_fib_local_table;
-extern struct fib_table *ip_fib_main_table;
+#ifndef CONFIG_NET_NS
+extern struct fib_table *ip_fib_local_table_static;
+extern struct fib_table *ip_fib_main_table_static;
+#define ip_fib_local_table_ns()  ip_fib_local_table_static
+#define ip_fib_main_table_ns()  ip_fib_main_table_static
+#else
+#define ip_fib_local_table_ns()  (current_net_ns->fib4_local_table)
+#define ip_fib_main_table_ns()  (current_net_ns->fib4_main_table)
+#endif

 static inline struct fib_table *fib_get_table(int id)
 {
  if (id != RT_TABLE_LOCAL)
-  return ip_fib_main_table;
- return ip_fib_local_table;
+  return ip_fib_main_table_ns();
+ return ip_fib_local_table_ns();
 }

 static inline struct fib_table *fib_new_table(int id)
@@ -186,23 +194,36 @@ static inline struct fib_table *fib_new_

 static inline int fib_lookup(const struct flowi *flp, struct fib_result *res)
 {
- if (ip_fib_local_table->tb_lookup(ip_fib_local_table, flp, res) &&
```

```
-     ip_fib_main_table->tb_lookup(ip_fib_main_table, flp, res))
+ struct fib_table *tb;
+
+ tb = ip_fib_local_table_ns();
+ if (!tb->tb_lookup(tb, flp, res))
+  return 0;
+ tb = ip_fib_main_table_ns();
+ if (tb->tb_lookup(tb, flp, res))
   return -ENETUNREACH;
  return 0;
 }

 static inline void fib_select_default(const struct flowi *flp, struct fib_result *res)
 {
+ struct fib_table *tb;
+
+ tb = ip_fib_main_table_ns();
  if (FIB_RES_GW(*res) && FIB_RES_NH(*res).nh_scope == RT_SCOPE_LINK)
-  ip_fib_main_table->tb_select_default(ip_fib_main_table, flp, res);
+  tb->tb_select_default(main_table, flp, res);
 }

 #else /* CONFIG_IP_MULTIPLE_TABLES */
-#define ip_fib_local_table (fib_tables[RT_TABLE_LOCAL])
-#define ip_fib_main_table (fib_tables[RT_TABLE_MAIN])
+#define ip_fib_local_table_ns() (fib_tables_ns()[RT_TABLE_LOCAL])
+#define ip_fib_main_table_ns() (fib_tables_ns()[RT_TABLE_MAIN])

-extern struct fib_table * fib_tables[RT_TABLE_MAX+1];
+#ifndef CONFIG_NET_NS
+extern struct fib_table * fib_tables_static[RT_TABLE_MAX+1];
+#define fib_tables_ns() fib_tables_static
+#else
+#define fib_tables_ns() (current_net_ns->fib4_tables)
+#endif
 extern int fib_lookup(const struct flowi *flp, struct fib_result *res);
 extern struct fib_table *__fib_new_table(int id);
 extern void fib_rule_put(struct fib_rule *r);
@@ -212,7 +233,7 @@ static inline struct fib_table *fib_get_
  if (id == 0)
   id = RT_TABLE_MAIN;

- return fib_tables[id];
+ return fib_tables_ns()[id];
 }

 static inline struct fib_table *fib_new_table(int id)
@@ -220,7 +241,7 @@ static inline struct fib_table *fib_new_
```

```
  if (id == 0)
   id = RT_TABLE_MAIN;

- return fib_tables[id] ? : __fib_new_table(id);
+ return fib_tables_ns()[id] ? : __fib_new_table(id);
 }

 extern void fib_select_default(const struct flowi *flp, struct fib_result *res);
@@ -229,6 +250,10 @@ extern void fib_select_default(const str

 /* Exported by fib_frontend.c */
 extern void  ip_fib_init(void);
+#ifdef CONFIG_NET_NS
+extern int ip_fib_struct_init(void);
+extern void ip_fib_struct_fini(void);
+#endif
 extern int inet_rtm_delroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg);
 extern int inet_rtm_newroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg);
 extern int inet_rtm_getroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg);
@@ -246,9 +271,16 @@ extern int fib_sync_up(struct net_device
 extern int fib_convert_rtentry(int cmd, struct nlmsghdr *nl, struct rtmsg *rtm,
          struct kern_rta *rta, struct rtentry *r);
 extern u32  __fib_res_prefsrc(struct fib_result *res);
+#ifdef CONFIG_NET_NS
+extern void fib_hashtable_destroy(void);
+#endif

 /* Exported by fib_hash.c */
 extern struct fib_table *fib_hash_init(int id);
+#ifdef CONFIG_NET_NS
+extern void fib_hash_fini(struct fib_table *tb);
+extern void fib_hash_destroy_hash(void);
+#endif

 #ifdef CONFIG_IP_MULTIPLE_TABLES
 /* Exported by fib_rules.c */
@@ -259,7 +291,11 @@ extern int inet_dump_rules(struct sk_buf
 #ifdef CONFIG_NET_CLS_ROUTE
 extern u32 fib_rules_tclass(struct fib_result *res);
 #endif
-extern void fib_rules_init(void);
+extern int fib_rules_struct_init(void);
+extern void fib_rules_notif_init(void);
+#ifdef CONFIG_NET_NS
+extern void fib_rules_struct_fini(void);
+#endif
 #endif
```

```
 static inline void fib_combine_itag(u32 *itag, struct fib_result *res)
--- ./net/core/dev.c.vensrt Fri Jun 23 11:48:15 2006
+++ ./net/core/dev.c Fri Jun 23 11:50:16 2006
@@ -3398,6 +3398,8 @@ static int __init netdev_dma_register(vo
 #endif /* CONFIG_NET_DMA */

 #ifdef CONFIG_NET_NS
+#include <net/ip_fib.h>
+
 struct net_namespace init_net_ns = {
  .active_ref = ATOMIC_INIT(2),
      /* one for init_task->net_context,
@@ -3436,6 +3438,8 @@ int net_ns_start(void)
  task = current;
  orig_ns = task->net_context;
  task->net_context = ns;
+ if (ip_fib_struct_init())
+  goto out_fib4;
  err = register_netdev(dev);
  if (err)
   goto out_register;
@@ -3443,6 +3447,8 @@ int net_ns_start(void)
  return 0;

 out_register:
+ ip_fib_struct_fini();
+out_fib4:
  dev->destructor(dev);
  task->net_context = orig_ns;
  BUG_ON(atomic_read(&ns->active_ref) != 1);
@@ -3467,6 +3473,7 @@ static void net_ns_destroy(void *data)
  ns = data;
  push_net_ns(ns, orig_ns);
  unregister_netdev(ns->loopback);
+ ip_fib_struct_fini();
  BUG_ON(!list_empty(&ns->dev_base));
  pop_net_ns(orig_ns);

--- ./net/ipv4/Kconfig.vensrt Wed Jun 21 18:53:19 2006
+++ ./net/ipv4/Kconfig Fri Jun 23 11:50:16 2006
@@ -53,7 +53,7 @@ config IP_ADVANCED_ROUTER

 choice
  prompt "Choose IP: FIB lookup algorithm (choose FIB_HASH if unsure)"
- depends on IP_ADVANCED_ROUTER
+ depends on IP_ADVANCED_ROUTER && !NET_NS
  default ASK_IP_FIB_HASH
```

```
 config ASK_IP_FIB_HASH
@@ -83,7 +83,7 @@ config IP_FIB_TRIE
 endchoice

 config IP_FIB_HASH
- def_bool ASK_IP_FIB_HASH || !IP_ADVANCED_ROUTER
+ def_bool ASK_IP_FIB_HASH || !IP_ADVANCED_ROUTER || NET_NS

 config IP_MULTIPLE_TABLES
  bool "IP: policy routing"
--- ./net/ipv4/fib_frontend.c.vensrt Wed Jun 21 18:53:19 2006
+++ ./net/ipv4/fib_frontend.c Fri Jun 23 11:50:16 2006
@@ -53,14 +53,18 @@

 #define RT_TABLE_MIN RT_TABLE_MAIN

-struct fib_table *ip_fib_local_table;
-struct fib_table *ip_fib_main_table;
+#ifndef CONFIG_NET_NS
+struct fib_table *ip_fib_local_table_static;
+struct fib_table *ip_fib_main_table_static;
+#endif

 #else

 #define RT_TABLE_MIN 1

-struct fib_table *fib_tables[RT_TABLE_MAX+1];
+#ifndef CONFIG_NET_NS
+struct fib_table *fib_tables_static[RT_TABLE_MAX+1];
+#endif

 struct fib_table *__fib_new_table(int id)
 {
@@ -69,7 +73,7 @@ struct fib_table *__fib_new_table(int id
  tb = fib_hash_init(id);
  if (!tb)
   return NULL;
- fib_tables[id] = tb;
+ fib_tables_ns()[id] = tb;
  return tb;
 }

@@ -80,8 +84,8 @@ struct fib_table *__fib_new_table(int id
 static void fib_flush(void)
 {
  int flushed = 0;
-#ifdef CONFIG_IP_MULTIPLE_TABLES
```

```
  struct fib_table *tb;
+#ifdef CONFIG_IP_MULTIPLE_TABLES
  int id;

  for (id = RT_TABLE_MAX; id>0; id--) {
@@ -90,8 +94,10 @@ static void fib_flush(void)
   flushed += tb->tb_flush(tb);
  }
 #else /* CONFIG_IP_MULTIPLE_TABLES */
- flushed += ip_fib_main_table->tb_flush(ip_fib_main_table);
- flushed += ip_fib_local_table->tb_flush(ip_fib_local_table);
+ tb = ip_fib_main_table_ns();
+ flushed += tb->tb_flush(tb);
+ tb = ip_fib_local_table_ns();
+ flushed += tb->tb_flush(tb);
 #endif /* CONFIG_IP_MULTIPLE_TABLES */

  if (flushed)
@@ -106,14 +112,15 @@ struct net_device * ip_dev_find(u32 addr
 {
  struct flowi fl = { .nl_u = { .ip4_u = { .daddr = addr } } };
  struct fib_result res;
+ struct fib_table *tb;
  struct net_device *dev = NULL;

 #ifdef CONFIG_IP_MULTIPLE_TABLES
  res.r = NULL;
 #endif

- if (!ip_fib_local_table ||
-     ip_fib_local_table->tb_lookup(ip_fib_local_table, &fl, &res))
+ tb = ip_fib_local_table_ns();
+ if (!tb || tb->tb_lookup(tb, &fl, &res))
   return NULL;
  if (res.type != RTN_LOCAL)
   goto out;
@@ -130,6 +137,7 @@ unsigned inet_addr_type(u32 addr)
 {
  struct flowi  fl = { .nl_u = { .ip4_u = { .daddr = addr } } };
  struct fib_result res;
+ struct fib_table *tb;
  unsigned ret = RTN_BROADCAST;

  if (ZERONET(addr) || BADCLASS(addr))
@@ -141,10 +149,10 @@ unsigned inet_addr_type(u32 addr)
  res.r = NULL;
 #endif
```

```
- if (ip_fib_local_table) {
+ tb = ip_fib_local_table_ns();
+ if (tb) {
   ret = RTN_UNICAST;
-  if (!ip_fib_local_table->tb_lookup(ip_fib_local_table,
-         &fl, &res)) {
+  if (!tb->tb_lookup(tb, &fl, &res)) {
    ret = res.type;
    fib_res_put(&res);
   }
@@ -651,19 +659,66 @@ static struct notifier_block fib_netdev_
  .notifier_call =fib_netdev_event,
 };

-void __init ip_fib_init(void)
+int ip_fib_struct_init(void)
 {
 #ifndef CONFIG_IP_MULTIPLE_TABLES
- ip_fib_local_table = fib_hash_init(RT_TABLE_LOCAL);
- ip_fib_main_table  = fib_hash_init(RT_TABLE_MAIN);
+ ip_fib_local_table_ns() = fib_hash_init(RT_TABLE_LOCAL);
+ ip_fib_main_table_ns()  = fib_hash_init(RT_TABLE_MAIN);
+#else
+#ifndef CONFIG_NET_NS
+ return fib_rules_struct_init();
 #else
- fib_rules_init();
+ struct fib_table **tables;
+
+ tables = kmalloc((RT_TABLE_MAX+1) * sizeof(*tables), GFP_KERNEL);
+ if (tables == NULL)
+  return -ENOMEM;
+ memset(tables, 0, (RT_TABLE_MAX+1) * sizeof(*tables));
+ fib_tables_ns() = tables;
+ if (fib_rules_struct_init()) {
+  kfree(tables);
+  fib_tables_ns() = NULL;
+  return -ENOMEM;
+ }
 #endif
+#endif
+ return 0;
+}

+void __init ip_fib_init(void)
+{
+ ip_fib_struct_init();
+
```

```
+#ifdef CONFIG_IP_MULTIPLE_TABLES
+ fib_rules_notif_init();
+#endif
  register_netdevice_notifier(&fib_netdev_notifier);
  register_inetaddr_notifier(&fib_inetaddr_notifier);
  nl_fib_lookup_init();
 }

+#ifdef CONFIG_NET_NS
+void ip_fib_struct_fini(void)
+{
+ current_net_ns->destroying = 1;
+ rtnl_lock();
+#ifdef CONFIG_IP_MULTIPLE_TABLES
+ fib_rules_struct_fini();
+#endif
+ /*
+  * FIB should already be empty since there is no netdevice,
+  * but clear it anyway
+  */
+ fib_flush();
+ rt_cache_flush(0);
+#ifdef CONFIG_IP_MULTIPLE_TABLES
+ kfree(fib_tables_ns());
+ fib_tables_ns() = NULL;
+#endif
+ fib_hashtable_destroy();
+ rtnl_unlock();
+}
+#endif /* CONFIG_NET_NS */
+
 EXPORT_SYMBOL(inet_addr_type);
 EXPORT_SYMBOL(ip_dev_find);
--- ./net/ipv4/fib_hash.c.vensrt Mon Mar 20 08:53:29 2006
+++ ./net/ipv4/fib_hash.c Fri Jun 23 11:50:16 2006
@@ -629,6 +629,11 @@ static int fn_flush_list(struct fn_zone
  struct hlist_node *node, *n;
  struct fib_node *f;
  int found = 0;
+#ifndef CONFIG_NET_NS
+ const int destroy = 0;
+#else
+ const int destroy = current_net_ns->destroying;
+#endif

  hlist_for_each_entry_safe(f, node, n, head, fn_hash) {
   struct fib_alias *fa, *fa_node;
@@ -638,7 +643,9 @@ static int fn_flush_list(struct fn_zone
```

```
   list_for_each_entry_safe(fa, fa_node, &f->fn_alias, fa_list) {
    struct fib_info *fi = fa->fa_info;

-   if (fi && (fi->fib_flags&RTNH_F_DEAD)) {
+   if (fi == NULL)
+    continue;
+   if (destroy || (fi->fib_flags&RTNH_F_DEAD)) {
     write_lock_bh(&fib_hash_lock);
     list_del(&fa->fa_list);
     if (list_empty(&f->fn_alias)) {
@@ -819,7 +826,7 @@ struct fib_iter_state {
 static struct fib_alias *fib_get_first(struct seq_file *seq)
 {
  struct fib_iter_state *iter = seq->private;
- struct fn_hash *table = (struct fn_hash *) ip_fib_main_table->tb_data;
+ struct fn_hash *table = (struct fn_hash *) ip_fib_main_table_ns()->tb_data;

  iter->bucket    = 0;
  iter->hash_head = NULL;
@@ -958,7 +965,7 @@ static void *fib_seq_start(struct seq_fi
  void *v = NULL;

  read_lock(&fib_hash_lock);
- if (ip_fib_main_table)
+ if (ip_fib_main_table_ns())
   v = *pos ? fib_get_idx(seq, *pos - 1) : SEQ_START_TOKEN;
  return v;
 }
--- ./net/ipv4/fib_rules.c.vensrt Wed Jun 21 18:51:09 2006
+++ ./net/ipv4/fib_rules.c Fri Jun 23 11:50:16 2006
@@ -100,7 +100,12 @@ static struct fib_rule local_rule = {
  .r_action = RTN_UNICAST,
 };

-static struct hlist_head fib_rules;
+#ifndef CONFIG_NET_NS
+static struct hlist_head fib_rules_static;
+#define fib_rules_ns() (&fib_rules_static)
+#else
+#define fib_rules_ns() (&current_net_ns->fib4_rules)
+#endif

 /* writer func called from netlink -- rtnl_sem hold*/

@@ -110,11 +115,13 @@ int inet_rtm_delrule(struct sk_buff *skb
 {
  struct rtattr **rta = arg;
  struct rtmsg *rtm = NLMSG_DATA(nlh);
```

```
+ struct hlist_head *fib_rules;
  struct fib_rule *r;
  struct hlist_node *node;
  int err = -ESRCH;

- hlist_for_each_entry(r, node, &fib_rules, hlist) {
+ fib_rules = fib_rules_ns();
+ hlist_for_each_entry(r, node, fib_rules, hlist) {
   if ((!rta[RTA_SRC-1] || memcmp(RTA_DATA(rta[RTA_SRC-1]), &r->r_src, 4) == 0) &&
       rtm->rtm_src_len == r->r_src_len &&
       rtm->rtm_dst_len == r->r_dst_len &&
@@ -128,7 +135,7 @@ int inet_rtm_delrule(struct sk_buff *skb
       (!rta[RTA_IIF-1] || rtattr_strcmp(rta[RTA_IIF-1], r->r_ifname) == 0) &&
       (!rtm->rtm_table || (r && rtm->rtm_table == r->r_table))) {
    err = -EPERM;
-   if (r == &local_rule)
+   if (&r->hlist == fib_rules->first)
     break;

    hlist_del_rcu(&r->hlist);
@@ -147,9 +154,11 @@ int inet_rtm_delrule(struct sk_buff *skb
 static struct fib_table *fib_empty_table(void)
 {
  int id;
+ struct fib_table **tbs;

+ tbs = fib_tables_ns();
  for (id = 1; id <= RT_TABLE_MAX; id++)
-  if (fib_tables[id] == NULL)
+  if (tbs[id] == NULL)
    return __fib_new_table(id);
  return NULL;
 }
@@ -176,6 +185,7 @@ int inet_rtm_newrule(struct sk_buff *skb
 {
  struct rtattr **rta = arg;
  struct rtmsg *rtm = NLMSG_DATA(nlh);
+ struct hlist_head *fib_rules;
  struct fib_rule *r, *new_r, *last = NULL;
  struct hlist_node *node = NULL;
  unsigned char table_id;
@@ -234,7 +244,8 @@ int inet_rtm_newrule(struct sk_buff *skb
  if (rta[RTA_FLOW-1])
   memcpy(&new_r->r_tclassid, RTA_DATA(rta[RTA_FLOW-1]), 4);
 #endif
- r = container_of(fib_rules.first, struct fib_rule, hlist);
+ fib_rules = fib_rules_ns();
+ r = container_of(fib_rules->first, struct fib_rule, hlist);
```

```
  if (!new_r->r_preference) {
   if (r && r->hlist.next != NULL) {
@@ -244,7 +255,7 @@ int inet_rtm_newrule(struct sk_buff *skb
   }
  }

- hlist_for_each_entry(r, node, &fib_rules, hlist) {
+ hlist_for_each_entry(r, node, fib_rules, hlist) {
   if (r->r_preference > new_r->r_preference)
    break;
   last = r;
@@ -273,10 +284,12 @@ u32 fib_rules_tclass(struct fib_result *

 static void fib_rules_detach(struct net_device *dev)
 {
+ struct hlist_head *fib_rules;
  struct hlist_node *node;
  struct fib_rule *r;

- hlist_for_each_entry(r, node, &fib_rules, hlist) {
+ fib_rules = fib_rules_ns();
+ hlist_for_each_entry(r, node, fib_rules, hlist) {
   if (r->r_ifindex == dev->ifindex)
    r->r_ifindex = -1;

@@ -287,10 +300,12 @@ static void fib_rules_detach(struct net_

 static void fib_rules_attach(struct net_device *dev)
 {
+ struct hlist_head *fib_rules;
  struct hlist_node *node;
  struct fib_rule *r;

- hlist_for_each_entry(r, node, &fib_rules, hlist) {
+ fib_rules = fib_rules_ns();
+ hlist_for_each_entry(r, node, fib_rules, hlist) {
   if (r->r_ifindex == -1 && strcmp(dev->name, r->r_ifname) == 0)
    r->r_ifindex = dev->ifindex;
  }
@@ -299,6 +314,7 @@ static void fib_rules_attach(struct net_
 int fib_lookup(const struct flowi *flp, struct fib_result *res)
 {
  int err;
+ struct hlist_head *fib_rules;
  struct fib_rule *r, *policy;
  struct fib_table *tb;
  struct hlist_node *node;
```

```
@@ -311,7 +327,8 @@ FRprintk("Lookup: %u.%u.%u.%u <- %u.%u.%

  rcu_read_lock();

- hlist_for_each_entry_rcu(r, node, &fib_rules, hlist) {
+ fib_rules = fib_rules_ns();
+ hlist_for_each_entry_rcu(r, node, fib_rules, hlist) {
    if (((saddr^r->r_src) & r->r_srcmask) ||
        ((daddr^r->r_dst) & r->r_dstmask) ||
        (r->r_tos && r->r_tos != flp->fl4_tos) ||
@@ -453,11 +470,13 @@ int inet_dump_rules(struct sk_buff *skb,
 {
  int idx = 0;
  int s_idx = cb->args[0];
+ struct hlist_head *fib_rules;
  struct fib_rule *r;
  struct hlist_node *node;

  rcu_read_lock();
- hlist_for_each_entry(r, node, &fib_rules, hlist) {
+ fib_rules = fib_rules_ns();
+ hlist_for_each_entry(r, node, fib_rules, hlist) {

    if (idx < s_idx)
     continue;
@@ -473,11 +492,80 @@ int inet_dump_rules(struct sk_buff *skb,
  return skb->len;
 }

-void __init fib_rules_init(void)
+#ifndef CONFIG_NET_NS
+
+int fib_rules_struct_init(void)
 {
- INIT_HLIST_HEAD(&fib_rules);
- hlist_add_head(&local_rule.hlist, &fib_rules);
+ INIT_HLIST_HEAD(&fib_rules_static);
+ hlist_add_head(&local_rule.hlist, &fib_rules_static);
  hlist_add_after(&local_rule.hlist, &main_rule.hlist);
  hlist_add_after(&main_rule.hlist, &default_rule.hlist);
+ return 0;
+}
+
+#else
+
+static struct fib_rule *fib_rule_create(struct fib_rule *orig,
+  struct fib_rule *prev)
+{
```

```
+ struct fib_rule *p;
+
+ p = kmalloc(sizeof(*p), GFP_KERNEL);
+ if (p == NULL)
+  goto out;
+ memcpy(p, orig, sizeof(*p));
+ if (prev != NULL)
+  hlist_add_after(&prev->hlist, &p->hlist);
+ else
+  hlist_add_head(&p->hlist, fib_rules_ns());
+out:
+ return p;
+}
+
+int fib_rules_struct_init(void)
+{
+ struct hlist_head *fib_rules;
+ struct fib_rule *p, *q;
+
+ fib_rules = fib_rules_ns();
+ INIT_HLIST_HEAD(fib_rules);
+ p = fib_rule_create(&local_rule, NULL);
+ if (p == NULL)
+  goto out_rule;
+ q = fib_rule_create(&main_rule, p);
+ if (q == NULL)
+  goto out_rule;
+ p = q;
+ q = fib_rule_create(&default_rule, p);
+ if (q == NULL)
+  goto out_rule;
+ return 0;
+
+out_rule:
+ while (!hlist_empty(fib_rules)) {
+  p = hlist_entry(fib_rules->first, struct fib_rule, hlist);
+  hlist_del(&p->hlist);
+  kfree(p);
+ }
+ return -ENOMEM;
+}
+
+void fib_rules_struct_fini(void)
+{
+ struct fib_rule *r, *nxt;
+
+ for (r = hlist_entry(fib_rules_ns()->first, struct fib_rule, hlist);
+   r != NULL; r = nxt) {
```

```
+  nxt = hlist_entry(r->hlist.next, struct fib_rule, hlist);
+  hlist_del_rcu(&r->hlist);
+  r->r_dead = 1;
+  fib_rule_put(r);
+ }
+}
+
+#endif
+
+void __init fib_rules_notif_init(void)
+{
  register_netdevice_notifier(&fib_rules_notifier);
 }
--- ./net/ipv4/fib_semantics.c.vensrt Mon Mar 20 08:53:29 2006
+++ ./net/ipv4/fib_semantics.c Fri Jun 23 11:50:16 2006
@@ -31,6 +31,7 @@
 #include <linux/inet.h>
 #include <linux/inetdevice.h>
 #include <linux/netdevice.h>
+#include <linux/net_ns.h>
 #include <linux/if_arp.h>
 #include <linux/proc_fs.h>
 #include <linux/skbuff.h>
@@ -51,10 +52,21 @@
 #define FSprintk(a...)

 static DEFINE_RWLOCK(fib_info_lock);
-static struct hlist_head *fib_info_hash;
-static struct hlist_head *fib_info_laddrhash;
-static unsigned int fib_hash_size;
-static unsigned int fib_info_cnt;
+#ifndef CONFIG_NET_NS
+static struct hlist_head *fib_info_hash_static;
+static struct hlist_head *fib_info_laddrhash_static;
+static unsigned int fib_hash_size_static;
+static unsigned int fib_info_cnt_static;
+#define fib_info_hash(ns) fib_info_hash_static
+#define fib_info_laddrhash(ns) fib_info_laddrhash_static
+#define fib_hash_size(ns) fib_hash_size_static
+#define fib_info_cnt(ns) fib_info_cnt_static
+#else
+#define fib_info_hash(ns) ((ns)->fib4_hash)
+#define fib_info_laddrhash(ns) ((ns)->fib4_laddrhash)
+#define fib_hash_size(ns) ((ns)->fib4_hash_size)
+#define fib_info_cnt(ns) ((ns)->fib4_info_cnt)
+#endif

 #define DEVINDEX_HASHBITS 8
```

```
 #define DEVINDEX_HASHSIZE (1U << DEVINDEX_HASHBITS)
@@ -145,6 +157,8 @@ static const struct

 void free_fib_info(struct fib_info *fi)
 {
+ struct net_namespace *ns __attribute_used__ = current_net_ns;
+
  if (fi->fib_dead == 0) {
   printk("Freeing alive fib_info %p\n", fi);
   return;
@@ -154,7 +168,7 @@ void free_fib_info(struct fib_info *fi)
    dev_put(nh->nh_dev);
   nh->nh_dev = NULL;
  } endfor_nexthops(fi);
- fib_info_cnt--;
+ fib_info_cnt(ns)--;
  kfree(fi);
 }

@@ -197,9 +211,10 @@ static __inline__ int nh_comp(const stru
  return 0;
 }

-static inline unsigned int fib_info_hashfn(const struct fib_info *fi)
+static inline unsigned int fib_info_hashfn(const struct fib_info *fi,
+  struct net_namespace *ns)
 {
- unsigned int mask = (fib_hash_size - 1);
+ unsigned int mask = (fib_hash_size(ns) - 1);
  unsigned int val = fi->fib_nhs;

  val ^= fi->fib_protocol;
@@ -211,13 +226,14 @@ static inline unsigned int fib_info_hash

 static struct fib_info *fib_find_info(const struct fib_info *nfi)
 {
+ struct net_namespace *ns = current_net_ns;
  struct hlist_head *head;
  struct hlist_node *node;
  struct fib_info *fi;
  unsigned int hash;

- hash = fib_info_hashfn(nfi);
- head = &fib_info_hash[hash];
+ hash = fib_info_hashfn(nfi, ns);
+ head = &fib_info_hash(ns)[hash];

  hlist_for_each_entry(fi, node, head, fib_hash) {
```

```
   if (fi->fib_nhs != nfi->fib_nhs)
@@ -237,11 +253,15 @@ static struct fib_info *fib_find_info(co

 static inline unsigned int fib_devindex_hashfn(unsigned int val)
 {
- unsigned int mask = DEVINDEX_HASHSIZE - 1;
+ unsigned int r, mask = DEVINDEX_HASHSIZE - 1;

- return (val ^
+ r = val ^
   (val >> DEVINDEX_HASHBITS) ^
-  (val >> (DEVINDEX_HASHBITS * 2))) & mask;
+  (val >> (DEVINDEX_HASHBITS * 2));
+#ifdef CONFIG_NET_NS
+ r ^= current_net_ns->hash;
+#endif
+ return r & mask;
 }

 /* Check, that the gateway is already configured.
@@ -564,9 +584,9 @@ out:
  return 0;
 }

-static inline unsigned int fib_laddr_hashfn(u32 val)
+static inline unsigned int fib_laddr_hashfn(u32 val, struct net_namespace *ns)
 {
- unsigned int mask = (fib_hash_size - 1);
+ unsigned int mask = (fib_hash_size(ns) - 1);

  return (val ^ (val >> 7) ^ (val >> 14)) & mask;
 }
@@ -595,17 +615,18 @@ static void fib_hash_move(struct hlist_h
      struct hlist_head *new_laddrhash,
      unsigned int new_size)
 {
+ struct net_namespace *ns = current_net_ns;
  struct hlist_head *old_info_hash, *old_laddrhash;
- unsigned int old_size = fib_hash_size;
+ unsigned int old_size = fib_hash_size(ns);
  unsigned int i, bytes;

  write_lock(&fib_info_lock);
- old_info_hash = fib_info_hash;
- old_laddrhash = fib_info_laddrhash;
- fib_hash_size = new_size;
+ old_info_hash = fib_info_hash(ns);
+ old_laddrhash = fib_info_laddrhash(ns);
```

```
+ fib_hash_size(ns) = new_size;

  for (i = 0; i < old_size; i++) {
-  struct hlist_head *head = &fib_info_hash[i];
+  struct hlist_head *head = &old_info_hash[i];
   struct hlist_node *node, *n;
   struct fib_info *fi;

@@ -615,15 +636,15 @@ static void fib_hash_move(struct hlist_h

    hlist_del(&fi->fib_hash);

-   new_hash = fib_info_hashfn(fi);
+   new_hash = fib_info_hashfn(fi, ns);
    dest = &new_info_hash[new_hash];
    hlist_add_head(&fi->fib_hash, dest);
   }
  }
- fib_info_hash = new_info_hash;
+ fib_info_hash(ns) = new_info_hash;

  for (i = 0; i < old_size; i++) {
-  struct hlist_head *lhead = &fib_info_laddrhash[i];
+  struct hlist_head *lhead = &old_laddrhash[i];
   struct hlist_node *node, *n;
   struct fib_info *fi;

@@ -633,12 +654,12 @@ static void fib_hash_move(struct hlist_h

    hlist_del(&fi->fib_lhash);

-   new_hash = fib_laddr_hashfn(fi->fib_prefsrc);
+   new_hash = fib_laddr_hashfn(fi->fib_prefsrc, ns);
    ldest = &new_laddrhash[new_hash];
    hlist_add_head(&fi->fib_lhash, ldest);
   }
  }
- fib_info_laddrhash = new_laddrhash;
+ fib_info_laddrhash(ns) = new_laddrhash;

  write_unlock(&fib_info_lock);

@@ -647,11 +668,27 @@ static void fib_hash_move(struct hlist_h
  fib_hash_free(old_laddrhash, bytes);
 }

+#ifdef CONFIG_NET_NS
+void fib_hashtable_destroy(void)
```

```
+{
+ struct net_namespace *ns;
+ unsigned int bytes;
+
+ ns = current_net_ns;
+ bytes = ns->fib4_hash_size * sizeof(struct hlist_head *);
+ fib_hash_free(ns->fib4_hash, bytes);
+ ns->fib4_hash = NULL;
+ fib_hash_free(ns->fib4_laddrhash, bytes);
+ ns->fib4_laddrhash = NULL;
+}
+#endif
+
 struct fib_info *
 fib_create_info(const struct rtmsg *r, struct kern_rta *rta,
   const struct nlmsghdr *nlh, int *errp)
 {
  int err;
+ struct net_namespace *ns = current_net_ns;
  struct fib_info *fi = NULL;
  struct fib_info *ofi;
 #ifdef CONFIG_IP_ROUTE_MULTIPATH
@@ -685,8 +722,8 @@ fib_create_info(const struct rtmsg *r, s
 #endif

  err = -ENOBUFS;
- if (fib_info_cnt >= fib_hash_size) {
-  unsigned int new_size = fib_hash_size << 1;
+ if (fib_info_cnt(ns) >= fib_hash_size(ns)) {
+  unsigned int new_size = fib_hash_size(ns) << 1;
   struct hlist_head *new_info_hash;
   struct hlist_head *new_laddrhash;
   unsigned int bytes;
@@ -706,14 +743,14 @@ fib_create_info(const struct rtmsg *r, s
    fib_hash_move(new_info_hash, new_laddrhash, new_size);
   }

-  if (!fib_hash_size)
+  if (!fib_hash_size(ns))
    goto failure;
  }

  fi = kmalloc(sizeof(*fi)+nhs*sizeof(struct fib_nh), GFP_KERNEL);
  if (fi == NULL)
   goto failure;
- fib_info_cnt++;
+ fib_info_cnt(ns)++;
  memset(fi, 0, sizeof(*fi)+nhs*sizeof(struct fib_nh));
```

```
  fi->fib_protocol = r->rtm_protocol;
@@ -824,11 +861,11 @@ link_it:
   atomic_inc(&fi->fib_clntref);
   write_lock(&fib_info_lock);
   hlist_add_head(&fi->fib_hash,
-         &fib_info_hash[fib_info_hashfn(fi)]);
+          &fib_info_hash(ns)[fib_info_hashfn(fi, ns)]);
   if (fi->fib_prefsrc) {
    struct hlist_head *head;

-  head = &fib_info_laddrhash[fib_laddr_hashfn(fi->fib_prefsrc)];
+  head = &fib_info_laddrhash(ns)[fib_laddr_hashfn(fi->fib_prefsrc, ns)];
    hlist_add_head(&fi->fib_lhash, head);
   }
   change_nexthops(fi) {
@@ -1162,15 +1199,16 @@ fib_convert_rtentry(int cmd, struct nlms

 int fib_sync_down(u32 local, struct net_device *dev, int force)
 {
+ struct net_namespace *ns = current_net_ns;
   int ret = 0;
   int scope = RT_SCOPE_NOWHERE;

   if (force)
    scope = -1;

- if (local && fib_info_laddrhash) {
-  unsigned int hash = fib_laddr_hashfn(local);
-  struct hlist_head *head = &fib_info_laddrhash[hash];
+ if (local && fib_info_laddrhash(ns)) {
+  unsigned int hash = fib_laddr_hashfn(local, ns);
+  struct hlist_head *head = &fib_info_laddrhash(ns)[hash];
    struct hlist_node *node;
    struct fib_info *fi;

--- ./net/ipv4/route.c.vensrt Wed Jun 21 18:53:19 2006
+++ ./net/ipv4/route.c Fri Jun 23 11:50:16 2006
@@ -267,6 +267,7 @@ struct rt_cache_iter_state {
   int bucket;
 };

+static struct rtable *rt_cache_get_next(struct seq_file *seq, struct rtable *r);
 static struct rtable *rt_cache_get_first(struct seq_file *seq)
 {
   struct rtable *r = NULL;
@@ -279,21 +280,28 @@ static struct rtable *rt_cache_get_first
     break;
```

```
   rcu_read_unlock_bh();
  }
+ if (r && !net_ns_same(r->fl.net_ns, current_net_ns))
+  r = rt_cache_get_next(seq, r);
  return r;
 }

 static struct rtable *rt_cache_get_next(struct seq_file *seq, struct rtable *r)
 {
  struct rt_cache_iter_state *st = rcu_dereference(seq->private);
+ struct net_namespace *ns __attribute_used__ = current_net_ns;

+next:
  r = r->u.rt_next;
  while (!r) {
   rcu_read_unlock_bh();
   if (--st->bucket < 0)
-   break;
+   goto out;
   rcu_read_lock_bh();
   r = rt_hash_table[st->bucket].chain;
  }
+ if (!net_ns_same(r->fl.net_ns, ns))
+  goto next;
+out:
  return r;
 }

@@ -564,6 +572,7 @@ static inline u32 rt_score(struct rtable
 static inline int compare_keys(struct flowi *fl1, struct flowi *fl2)
 {
  return memcmp(&fl1->nl_u.ip4_u, &fl2->nl_u.ip4_u, sizeof(fl1->nl_u.ip4_u)) == 0 &&
+        net_ns_same(fl1->net_ns, fl2->net_ns) &&
         fl1->oif     == fl2->oif &&
         fl1->iif     == fl2->iif;
 }
@@ -1127,6 +1136,7 @@ void ip_rt_redirect(u32 old_gw, u32 dadd
  struct rtable *rth, **rthp;
  u32  skeys[2] = { saddr, 0 };
  int  ikeys[2] = { dev->ifindex, 0 };
+ struct net_namespace *ns __attribute_used__ = current_net_ns;

  if (!in_dev)
   return;
@@ -1158,6 +1168,7 @@ void ip_rt_redirect(u32 old_gw, u32 dadd

     if (rth->fl.fl4_dst != daddr ||
         rth->fl.fl4_src != skeys[i] ||
```

```
+        !net_ns_same(rth->fl.net_ns, ns) ||
         rth->fl.oif != ikeys[k] ||
         rth->fl.iif != 0) {
       rthp = &rth->u.rt_next;
@@ -1643,6 +1654,9 @@ static int ip_route_input_mc(struct sk_b
  dev_hold(rth->u.dst.dev);
  rth->idev = in_dev_get(rth->u.dst.dev);
  rth->fl.oif = 0;
+#ifdef CONFIG_NET_NS
+ rth->fl.net_ns = current_net_ns;
+#endif
  rth->rt_gateway = daddr;
  rth->rt_spec_dst= spec_dst;
  rth->rt_type = RTN_MULTICAST;
@@ -1786,6 +1800,9 @@ static inline int __mkroute_input(struct
  dev_hold(rth->u.dst.dev);
  rth->idev = in_dev_get(rth->u.dst.dev);
  rth->fl.oif  = 0;
+#ifdef CONFIG_NET_NS
+ rth->fl.net_ns = current_net_ns;
+#endif
  rth->rt_spec_dst= spec_dst;

  rth->u.dst.input = ip_forward;
@@ -2087,6 +2104,7 @@ int ip_route_input(struct sk_buff *skb,
  struct rtable * rth;
  unsigned hash;
  int iif = dev->ifindex;
+ struct net_namespace *ns __attribute_used__ = current_net_ns;

  tos &= IPTOS_RT_MASK;
  hash = rt_hash_code(daddr, saddr ^ (iif << 5));
@@ -2096,6 +2114,7 @@ int ip_route_input(struct sk_buff *skb,
       rth = rcu_dereference(rth->u.rt_next)) {
   if (rth->fl.fl4_dst == daddr &&
       rth->fl.fl4_src == saddr &&
+      net_ns_same(rth->fl.net_ns, ns) &&
       rth->fl.iif == iif &&
       rth->fl.oif == 0 &&
 #ifdef CONFIG_IP_ROUTE_FWMARK
@@ -2235,6 +2254,9 @@ static inline int __mkroute_output(struc
  rth->u.dst.dev = dev_out;
  dev_hold(dev_out);
  rth->idev = in_dev_get(dev_out);
+#ifdef CONFIG_NET_NS
+ rth->fl.net_ns = current_net_ns;
+#endif
  rth->rt_gateway = fl->fl4_dst;
```

```
  rth->rt_spec_dst= fl->fl4_src;

@@ -2560,6 +2582,7 @@ int __ip_route_output_key(struct rtable
 {
  unsigned hash;
  struct rtable *rth;
+ struct net_namespace *ns __attribute_used__ = current_net_ns;

  hash = rt_hash_code(flp->fl4_dst, flp->fl4_src ^ (flp->oif << 5));

@@ -2568,6 +2591,7 @@ int __ip_route_output_key(struct rtable
    rth = rcu_dereference(rth->u.rt_next)) {
    if (rth->fl.fl4_dst == flp->fl4_dst &&
        rth->fl.fl4_src == flp->fl4_src &&
+       net_ns_same(rth->fl.net_ns, ns) &&
        rth->fl.iif == 0 &&
        rth->fl.oif == flp->oif &&
 #ifdef CONFIG_IP_ROUTE_FWMARK
```

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Daniel Lezcano on Mon, 26 Jun 2006 14:56:46 GMT
View Forum Message <> Reply to Message

Andrey Savochkin wrote:
> Structures related to IPv4 rounting (FIB and routing cache)
> are made per-namespace.

How do you handle ICMP_REDIRECT ?

Subject: Re: [RFC][patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by ebiederm on Mon, 26 Jun 2006 15:13:52 GMT
View Forum Message <> Reply to Message

Andrey Savochkin <saw@swsoft.com> writes:

> Cleanup of dev_base list use, with the aim to make device list per-namespace.
> In almost every occasion, use of dev_base variable and dev->next pointer
> could be easily replaced by for_each_netdev loop.
> A few most complicated places were converted to using
> first_netdev()/next_netdev().

As a proof of concept patch this is ok.

As a real world patch this is much too big, which prevents review.
Plus it takes a few actions that are more than replace just

iterators through the device list.

In addition I suspect several if not all of these iterators
can be replaced with the an appropriate helper function.

The normal structure for a patch like this would be to
introduce the new helper function. for_each_netdev.
And then to replace all of the users while cc'ing the
maintainers of those drivers. With each different
driver being a different patch.

There is another topic for discussion in this patch as well.
How much of the context should be implicit and how much
should be explicit.

If the changes from netchannels had already been implemented, and all of
the network processing was happening in a process context then I would
trivially agree that implicit would be the way to go.

However short of always having code always execute in the proper
context I'm not comfortable with implicit parameters to functions.
Not that this the contents of this patch should address this but the
later patches should.

When I went through this, my patchset just added an explicit
continue if the devices was not in the appropriate namespace.
I actually prefer the multiple list implementation but at
the same time I think it is harder to get a clean implementation
out of it.

Eric

> --- ./drivers/block/aoe/aoecmd.c.vedevbase Wed Jun 21 18:50:28 2006
> +++ ./drivers/block/aoe/aoecmd.c Thu Jun 22 12:03:07 2006
> @@ -204,14 +204,17 @@ aoecmd_cfg_pkts(ushort aoemajor, unsigne
>   sl = sl_tail = NULL;
>
>   read_lock(&dev_base_lock);
> - for (ifp = dev_base; ifp; dev_put(ifp), ifp = ifp->next) {
> + for_each_netdev(dev) {
>   dev_hold(ifp);
> - if (!is_aoe_netif(ifp))
> + if (!is_aoe_netif(ifp)) {
> +   dev_put(ifp);
>   continue;
> + }
>

>   skb = new_skb(ifp, sizeof *h + sizeof *ch);
>   if (skb == NULL) {
>   printk(KERN_INFO "aoe: aoecmd_cfg: skb alloc
> failure\n");
> +   dev_put(ifp);
>   continue;
>   }
>   if (sl_tail == NULL)
> @@ -229,6 +232,7 @@ aoecmd_cfg_pkts(ushort aoemajor, unsigne
>
>   skb->next = sl;
>   sl = skb;
> + dev_put(ifp);
>   }
>   read_unlock(&dev_base_lock);

These hunks should use for_each_netdev(ifp);

> --- ./include/linux/netdevice.h.vedevbase Wed Jun 21 18:53:17 2006
> +++ ./include/linux/netdevice.h Thu Jun 22 18:57:50 2006
> @@ -289,8 +289,8 @@ struct net_device
>
>   unsigned long state;
>
> - struct net_device *next;
> -
> + struct list_head dev_list;
> +
>   /* The device initialization function. Called only once. */
>   int   (*init)(struct net_device *dev);
>
> @@ -543,9 +543,27 @@ struct packet_type {
> #include <linux/interrupt.h>
> #include <linux/notifier.h>
>
> -extern struct net_device loopback_dev; /* The loopback */
> -extern struct net_device *dev_base; /* All devices */
> -extern rwlock_t dev_base_lock; /* Device list lock */
> +extern struct net_device loopback_dev; /* The loopback */
> +extern struct list_head dev_base_head; /* All devices */
> +extern rwlock_t dev_base_lock; /* Device list lock */
> +

No need to change the loopback_dev and dev_base_lock here.

What is the advantage of changing the type of dev_base?
I can guess but there should be an explanation of it.

> +#define for_each_netdev(p) list_for_each_entry(p, &dev_base_head, dev_list)
> +
> +/* DO NOT USE first_netdev/next_netdev, use loop defined above */
> +#define first_netdev() ({ \
> +   list_empty(&dev_base_head) ? NULL : \
> + list_entry(dev_base_head.next, \
> +   struct net_device, \
> +   dev_list); \
> +   })
> +#define next_netdev(dev) ({ \
> +   struct list_head *__next; \
> +   __next = (dev)->dev_list.next; \
> +   __next == &dev_base_head ? NULL : \
> +   list_entry(__next, \
> +   struct net_device, \
> +   dev_list); \
> +   })
>
> extern int netdev_boot_setup_check(struct net_device *dev);
> extern unsigned long netdev_boot_base(const char *prefix, int unit);

> @@ -1903,7 +1902,7 @@ static int dev_ifconf(char __user *arg)
>   */
>
>   total = 0;
> - for (dev = dev_base; dev; dev = dev->next) {
> + for_each_netdev(dev) {
>   for (i = 0; i < NPROTO; i++) {
>   if (gifconf_list[i]) {
>   int done;
> @@ -1935,26 +1934,25 @@ static int dev_ifconf(char __user *arg)

Hmm. The proc code here appears to be more than non-trivial
restructuring. I'm not certain it is but it looks that way
which make review harder.

>   * This is invoked by the /proc filesystem handler to display a device
>   * in detail.
>   */
> -static __inline__ struct net_device *dev_get_idx(loff_t pos)
> -{
> - struct net_device *dev;
> - loff_t i;
> -
> - for (i = 0, dev = dev_base; dev && i < pos; ++i, dev = dev->next);
> -
> - return i == pos ? dev : NULL;

```
> -}
> -
>  void *dev_seq_start(struct seq_file *seq, loff_t *pos)
>  {
> + struct net_device *dev;
> + loff_t off = 1;
>   read_lock(&dev_base_lock);
> - return *pos ? dev_get_idx(*pos - 1) : SEQ_START_TOKEN;
> + if (!*pos)
> +  return SEQ_START_TOKEN;
> + for_each_netdev(dev) {
> +  if (off++ == *pos)
> +   return dev;
> + }
> + return NULL;
>  }
>
>  void *dev_seq_next(struct seq_file *seq, void *v, loff_t *pos)
>  {
> + struct net_device *dev = v;
>   ++*pos;
> - return v == SEQ_START_TOKEN ? dev_base : ((struct net_device *)v)->next;
> + return v == SEQ_START_TOKEN ? first_netdev() : next_netdev(dev);
>  }
>
>  void dev_seq_stop(struct seq_file *seq, void *v)
```

Subject: Re: [patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by Andrey Savochkin on Mon, 26 Jun 2006 15:42:05 GMT
View Forum Message <> Reply to Message

Hi Eric,

On Mon, Jun 26, 2006 at 09:13:52AM -0600, Eric W. Biederman wrote:
> Andrey Savochkin <saw@swsoft.com> writes:
>
> > Cleanup of dev_base list use, with the aim to make device list per-namespace.
> > In almost every occasion, use of dev_base variable and dev->next pointer
> > could be easily replaced by for_each_netdev loop.
> > A few most complicated places were converted to using
> > first_netdev()/next_netdev().
>
> As a proof of concept patch this is ok.
>
> As a real world patch this is much too big, which prevents review.
> Plus it takes a few actions that are more than replace just
> iterators through the device list.

dev_base list is historically not the cleanest part of Linux networking.
I've still spotted a place where the first device in dev_base list is assumed
to be loopback. In early days we had more, now only one place or two...

>
> In addition I suspect several if not all of these iterators
> can be replaced with the an appropriate helper function.
>
> The normal structure for a patch like this would be to
> introduce the new helper function. for_each_netdev.
> And then to replace all of the users while cc'ing the
> maintainers of those drivers. With each different
> driver being a different patch.
>
> There is another topic for discussion in this patch as well.
> How much of the context should be implicit and how much
> should be explicit.
>
> If the changes from netchannels had already been implemented, and all of
> the network processing was happening in a process context then I would
> trivially agree that implicit would be the way to go.

Why would we want all network processing happen in a process context?

>
> However short of always having code always execute in the proper
> context I'm not comfortable with implicit parameters to functions.
> Not that this the contents of this patch should address this but the
> later patches should.

We just have too many layers in networking code, and FIB/routing
illustrates it well.

>
> When I went through this, my patchset just added an explicit
> continue if the devices was not in the appropriate namespace.
> I actually prefer the multiple list implementation but at
> the same time I think it is harder to get a clean implementation
> out of it.

Certainly, dev_base list reorganization is not the crucial point in network
namespaces. But it has to be done some way or other.
If people vote for a single list with skipping devices from a wrong
namespace, it's fine with me, I can re-make this patch.

I personally prefer per-namespace device list since we have too many places
in the kernel where this list is walked in a linear fashion,

and with many namespaces this list may become quite long.

Regards

Andrey

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Andrey Savochkin on Mon, 26 Jun 2006 15:46:25 GMT
View Forum Message <> Reply to Message

On Mon, Jun 26, 2006 at 04:56:46PM +0200, Daniel Lezcano wrote:
> Andrey Savochkin wrote:
> > Structures related to IPv4 rounting (FIB and routing cache)
> > are made per-namespace.
>
> How do you handle ICMP_REDIRECT ?

Are you talking about routing cache entries created on incoming redirects?
Or outgoing redirects?

Andrey

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Daniel Lezcano on Mon, 26 Jun 2006 15:57:01 GMT
View Forum Message <> Reply to Message

Andrey Savochkin wrote:
> On Mon, Jun 26, 2006 at 04:56:46PM +0200, Daniel Lezcano wrote:
>
>>Andrey Savochkin wrote:
>>
>>>Structures related to IPv4 rounting (FIB and routing cache)
>>>are made per-namespace.
>>
>>How do you handle ICMP_REDIRECT ?
>
>
> Are you talking about routing cache entries created on incoming redirects?
> Or outgoing redirects?
>
> Andrey

incoming redirects

Subject: Re: [patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by ebiederm on Mon, 26 Jun 2006 16:26:23 GMT
View Forum Message <> Reply to Message

Andrey Savochkin <saw@swsoft.com> writes:

> Hi Eric,
>
> On Mon, Jun 26, 2006 at 09:13:52AM -0600, Eric W. Biederman wrote:
>> Andrey Savochkin <saw@swsoft.com> writes:
>>
>> > Cleanup of dev_base list use, with the aim to make device list
> per-namespace.
>> > In almost every occasion, use of dev_base variable and dev->next pointer
>> > could be easily replaced by for_each_netdev loop.
>> > A few most complicated places were converted to using
>> > first_netdev()/next_netdev().
>>
>> As a proof of concept patch this is ok.
>>
>> As a real world patch this is much too big, which prevents review.
>> Plus it takes a few actions that are more than replace just
>> iterators through the device list.
>
> dev_base list is historically not the cleanest part of Linux networking.
> I've still spotted a place where the first device in dev_base list is assumed
> to be loopback. In early days we had more, now only one place or two...

I agree. I'm just saying this should be several patches in a patchset
not just one big one.

>> In addition I suspect several if not all of these iterators
>> can be replaced with the an appropriate helper function.
>>
>> The normal structure for a patch like this would be to
>> introduce the new helper function. for_each_netdev.
>> And then to replace all of the users while cc'ing the
>> maintainers of those drivers. With each different
>> driver being a different patch.
>>
>> There is another topic for discussion in this patch as well.
>> How much of the context should be implicit and how much
>> should be explicit.
>>
>> If the changes from netchannels had already been implemented, and all of
>> the network processing was happening in a process context then I would
>> trivially agree that implicit would be the way to go.
>
> Why would we want all network processing happen in a process context?

The basic idea is that an interrupt comes in. A light weigh classifier
looks at the packet and throws it into the appropriate socket packet queue.

Beyond that everything can happen in the socket packet queue in process
context which reduces the number of locks you need, and increases cache
locality.

Van Jacobson's slides showed some impressive system load reductions by
doing that.

The increased locality aids the kind of work we are doing as well,
by meaning we don't have to guess.

It is a big enough problem that I don't think we want to gate on
that development but we need to be ready to take advantage of it when
it happens.

>> However short of always having code always execute in the proper
>> context I'm not comfortable with implicit parameters to functions.
>> Not that this the contents of this patch should address this but the
>> later patches should.
>
> We just have too many layers in networking code, and FIB/routing
> illustrates it well.

I don't follow this comment. How does a lot of layers affect
the choice of implicit or explicit parameters? If you are maintaining
a patch outside the kernel I could see how there could be a win for
touching the least amount of code possible but for merged code that
you only have to go through once I don't see how the number of layers
affects things.

As I recall for most of the FIB/routing code once you have removed
the global variable accesses and introduce namespace checks in
the hash table (because allocating hash tables at runtime isn't sane)
the rest of the code was agnostic about what was going on. So I think
you have touched everything that needs touching. So I don't see
a code size or complexity argument there.

I do agree that we do have a lot of code there.

>> When I went through this, my patchset just added an explicit
>> continue if the devices was not in the appropriate namespace.
>> I actually prefer the multiple list implementation but at
>> the same time I think it is harder to get a clean implementation
>> out of it.
>

> Certainly, dev_base list reorganization is not the crucial point in network
> namespaces. But it has to be done some way or other.
> If people vote for a single list with skipping devices from a wrong
> namespace, it's fine with me, I can re-make this patch.
>
> I personally prefer per-namespace device list since we have too many places
> in the kernel where this list is walked in a linear fashion,
> and with many namespaces this list may become quite long.

I completely agree that cleaning up the list is a good thing to do,
regardless of how we implement things. So that can easily be a pre cursor
to the work. My only practical question is if we can remove the list
walks be calling some of the generic helper functions.

I would say just separate out the list walk cleanup code and submit it.
The important point is that the patch needs to stand on it's as a
cleanup rather than depending on network namespaces as a justification.

Eric

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Andrey Savochkin on Mon, 26 Jun 2006 19:39:09 GMT
View Forum Message <> Reply to Message

On Mon, Jun 26, 2006 at 05:57:01PM +0200, Daniel Lezcano wrote:
> Andrey Savochkin wrote:
> > On Mon, Jun 26, 2006 at 04:56:46PM +0200, Daniel Lezcano wrote:
> >>
> >>How do you handle ICMP_REDIRECT ?
> >
> >
> > Are you talking about routing cache entries created on incoming redirects?
> > Or outgoing redirects?
> >
>
> incoming redirects

They are inserted into routing cache with the current namespace tag, in
the same way as input routing cache entries.

Andrey

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Herbert Poetzl on Mon, 26 Jun 2006 20:05:14 GMT
View Forum Message <> Reply to Message

On Mon, Jun 26, 2006 at 04:56:46PM +0200, Daniel Lezcano wrote:
> Andrey Savochkin wrote:
> >Structures related to IPv4 rounting (FIB and routing cache)
> >are made per-namespace.
>
> How do you handle ICMP_REDIRECT ?

and btw. how do you handle the beloved 'ping'
(i.e. ICMP_ECHO_REQUEST/REPLY for and from
guests?

best,
Herbert

Subject: Re: [patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by Andrey Savochkin on Mon, 26 Jun 2006 20:14:37 GMT
View Forum Message <> Reply to Message

Eric,

On Mon, Jun 26, 2006 at 10:26:23AM -0600, Eric W. Biederman wrote:
> Andrey Savochkin <saw@swsoft.com> writes:
>
> > On Mon, Jun 26, 2006 at 09:13:52AM -0600, Eric W. Biederman wrote:
> >>
> >> There is another topic for discussion in this patch as well.
> >> How much of the context should be implicit and how much
> >> should be explicit.
> >>
> >> If the changes from netchannels had already been implemented, and all of
> >> the network processing was happening in a process context then I would
> >> trivially agree that implicit would be the way to go.
> >
>
[snip]
> It is a big enough problem that I don't think we want to gate on
> that development but we need to be ready to take advantage of it when
> it happens.

Well, ok, implicit namespace reference will take advantage of it
if it happens.

>
> >> However short of always having code always execute in the proper
> >> context I'm not comfortable with implicit parameters to functions.
> >> Not that this the contents of this patch should address this but the
> >> later patches should.

> >
> > We just have too many layers in networking code, and FIB/routing
> > illustrates it well.
>
> I don't follow this comment. How does a lot of layers affect
> the choice of implicit or explicit parameters? If you are maintaining
> a patch outside the kernel I could see how there could be a win for
> touching the least amount of code possible but for merged code that
> you only have to go through once I don't see how the number of layers
> affects things.

I agree that implicit vs explicit parameters is a topic for discussion.
>From what you see from my patch, I vote for implicit ones in this case :)

I was talking about layers because they imply changing more code,
and usually imply adding more parameters to functions and passing these
additional parameters to next layers.
In "routing" code it goes from routing entry points, to routing cache, to
general FIB functions, to table-specific code (FIB hash).

These additional parameters bloat the code to some extent.
Sometimes it's possible to save here and there by fetching the parameter
(namespace pointer) indirectly from structures you already have at hand,
but it can't be done universally.

One of the properties of implicit argument which I especially like
is that both input and output paths are absolutely symmetric in how
the namespace pointer is extracted.

>
> As I recall for most of the FIB/routing code once you have removed
> the global variable accesses and introduce namespace checks in
> the hash table (because allocating hash tables at runtime isn't sane)
> the rest of the code was agnostic about what was going on. So I think
> you have touched everything that needs touching. So I don't see
> a code size or complexity argument there.

 Andrey

Subject: Re: [patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by ebiederm on Mon, 26 Jun 2006 21:02:49 GMT
View Forum Message <> Reply to Message

Andrey Savochkin <saw@swsoft.com> writes:

> Eric,
>

> On Mon, Jun 26, 2006 at 10:26:23AM -0600, Eric W. Biederman wrote:
>>
>>
> [snip]
>> It is a big enough problem that I don't think we want to gate on
>> that development but we need to be ready to take advantage of it when
>> it happens.
>
> Well, ok, implicit namespace reference will take advantage of it
> if it happens.

And if fact in that case we don't have to do anything special because
the process pointer will always be correct.

>> >> However short of always having code always execute in the proper
>> >> context I'm not comfortable with implicit parameters to functions.
>> >> Not that this the contents of this patch should address this but the
>> >> later patches should.
>> >
>> > We just have too many layers in networking code, and FIB/routing
>> > illustrates it well.
>>
>> I don't follow this comment. How does a lot of layers affect
>> the choice of implicit or explicit parameters? If you are maintaining
>> a patch outside the kernel I could see how there could be a win for
>> touching the least amount of code possible but for merged code that
>> you only have to go through once I don't see how the number of layers
>> affects things.
>
> I agree that implicit vs explicit parameters is a topic for discussion.
> From what you see from my patch, I vote for implicit ones in this case :)

Yes. I tend to be against implicit namespaces references mostly because
the explicit ones tend to make the code clearer.

> I was talking about layers because they imply changing more code,
> and usually imply adding more parameters to functions and passing these
> additional parameters to next layers.
> In "routing" code it goes from routing entry points, to routing cache, to
> general FIB functions, to table-specific code (FIB hash).

Yes. Although as I recall you don't have to pass anything down very far.
Because most functions once you have done the table lookup operate
on just a subset of the table, when they are getting the real work done.

> These additional parameters bloat the code to some extent.
> Sometimes it's possible to save here and there by fetching the parameter
> (namespace pointer) indirectly from structures you already have at hand,

> but it can't be done universally.
>
> One of the properties of implicit argument which I especially like
> is that both input and output paths are absolutely symmetric in how
> the namespace pointer is extracted.

There is an element of that. In the output path for the most part everything
works implicitly because you are in the proper context.

I need to dig out my code and start comparing to what you have done.

Eric

Subject: Re: [RFC][patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by dev on Tue, 27 Jun 2006 06:59:33 GMT
View Forum Message <> Reply to Message

>>Cleanup of dev_base list use, with the aim to make device list per-namespace.
>>In almost every occasion, use of dev_base variable and dev->next pointer
>>could be easily replaced by for_each_netdev loop.
>>A few most complicated places were converted to using
>>first_netdev()/next_netdev().
>
>
> As a proof of concept patch this is ok.
>
> As a real world patch this is much too big, which prevents review.
> Plus it takes a few actions that are more than replace just
> iterators through the device list.
Mmm, actually it is a whole changeset and should go as a one patch. I
didn't find it to be big and my review took only 5-10mins..
I also don't think that mailing each driver maintainer is a good idea.
Only if we want to make some buzz :)

Kirill

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Andrey Savochkin on Tue, 27 Jun 2006 09:25:32 GMT
View Forum Message <> Reply to Message

On Mon, Jun 26, 2006 at 10:05:14PM +0200, Herbert Poetzl wrote:
> On Mon, Jun 26, 2006 at 04:56:46PM +0200, Daniel Lezcano wrote:
> > Andrey Savochkin wrote:
> > >Structures related to IPv4 rounting (FIB and routing cache)
> > >are made per-namespace.

> >
> > How do you handle ICMP_REDIRECT ?
>
> and btw. how do you handle the beloved 'ping'
> (i.e. ICMP_ECHO_REQUEST/REPLY for and from
> guests?

I don't need to do anything special. They are just IP packets.
If packets are local in the current net namespace, they are delivered to
socket or handled by icmp_rcv.

Certainly, packet/raw sockets shouldn't see packets they aren't supposed to
see. For raw sockets, it implies making socket lookup aware of namespaces,
exactly like for TCP or UDP.

Andrey

Subject: Re: [RFC][patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by ebiederm on Tue, 27 Jun 2006 11:13:28 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>>Cleanup of dev_base list use, with the aim to make device list per-namespace.
>>>In almost every occasion, use of dev_base variable and dev->next pointer
>>>could be easily replaced by for_each_netdev loop.
>>>A few most complicated places were converted to using
>>>first_netdev()/next_netdev().
>> As a proof of concept patch this is ok.
>> As a real world patch this is much too big, which prevents review.
>> Plus it takes a few actions that are more than replace just
>> iterators through the device list.
> Mmm, actually it is a whole changeset and should go as a one patch. I didn't
> find it to be big and my review took only 5-10mins..
> I also don't think that mailing each driver maintainer is a good idea.
> Only if we want to make some buzz :)

Thanks for supporting my case. You reviewed it and missed the obvious typo.
I do agree that a patchset doing it all should happen at once.

As for not mailing the maintainers of the code we are changing. That
would just be irresponsible.

Eric

Subject: Re: [RFC][patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by dev on Tue, 27 Jun 2006 15:08:37 GMT
View Forum Message <> Reply to Message

>>>>Cleanup of dev_base list use, with the aim to make device list per-namespace.
>>>>In almost every occasion, use of dev_base variable and dev->next pointer
>>>>could be easily replaced by for_each_netdev loop.
>>>>A few most complicated places were converted to using
>>>>first_netdev()/next_netdev().
>>>
>>>As a proof of concept patch this is ok.
>>>As a real world patch this is much too big, which prevents review.
>>>Plus it takes a few actions that are more than replace just
>>>iterators through the device list.
>>
>>Mmm, actually it is a whole changeset and should go as a one patch. I didn't
>>find it to be big and my review took only 5-10mins..
>>I also don't think that mailing each driver maintainer is a good idea.
>>Only if we want to make some buzz :)
>
>
> Thanks for supporting my case. You reviewed it and missed the obvious typo.
> I do agree that a patchset doing it all should happen at once.
This doesn't support anything. e.g. I caught quite a lot of bugs after
Ingo Molnar, but this doesn't make his code "poor". People are people.
Anyway, I would be happy to see the typo.

> As for not mailing the maintainers of the code we are changing. That
> would just be irresponsible.

Kirill

Subject: Re: [RFC][patch 1/4] Network namespaces: cleanup of dev_base list use
Posted by ebiederm on Tue, 27 Jun 2006 16:54:45 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

> This doesn't support anything. e.g. I caught quite a lot of bugs after Ingo
> Molnar, but this doesn't make his code "poor". People are people.
> Anyway, I would be happy to see the typo.

Look up thread. You replied to the message where I commented on it.

There are two ways to argue this.
- It is the linux kernel development style to do small simple
  obviously patches that copy the maintainer of the code you are

changing.
- Explain why that is the style.

The basic idea is that on a simple patch that is well described, it is
trivial to check and trivial to verify.

Eric

Subject: [RFC] Network namespaces a path to mergable code.
Posted by ebiederm on Tue, 27 Jun 2006 17:20:40 GMT
View Forum Message <> Reply to Message

Thinking about this I am going to suggest a slightly different direction
for get a patchset we can merge.

First we concentrate on the fundamentals.
- How we mark a device as belonging to a specific network namespace.
- How we mark a socket as belonging to a specific network namespace.

As part of the fundamentals we add a patch to the generic socket code
that by default will disable it for protocol families that do not indicate
support for handling network namespaces, on a non-default network namespace.

I think that gives us a path that will allow us to convert the network stack
one protocol family at a time instead of in one big lump.

Stubbing off the sysfs and sysctl interfaces in the first round for the
non-default namespaces as you have done should be good enough.

The reason for the suggestion is that most of the work for the protocol
stacks ipv4 ipv6 af_packet af_unix is largely noise, and simple
replacement without real design work happening. Mostly it is just
tweaking the code to remove global variables, and doing a couple
lookups.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by Andrey Savochkin on Tue, 27 Jun 2006 17:58:59 GMT
View Forum Message <> Reply to Message

Eric,

On Tue, Jun 27, 2006 at 11:20:40AM -0600, Eric W. Biederman wrote:
>

> Thinking about this I am going to suggest a slightly different direction
> for get a patchset we can merge.
>
> First we concentrate on the fundamentals.
> - How we mark a device as belonging to a specific network namespace.
> - How we mark a socket as belonging to a specific network namespace.

I agree with the direction of your thoughts.
I was trying to do a similar thing, define clear steps in network
namespace merging.

My first patchset covers devices but not sockets.
The only difference from what you're suggesting is ipv4 routing.
For me, it is not less important than devices and sockets. May be even
more important, since routing exposes design deficiencies less obvious at
socket level.

>
> As part of the fundamentals we add a patch to the generic socket code
> that by default will disable it for protocol families that do not indicate
> support for handling network namespaces, on a non-default network namespace.

Fine

Can you summarize you objections against my way of handling devices, please?
And what was the typo you referred to in your letter to Kirill Korotaev?

Regards
Andrey

Subject: Re: Network namespaces a path to mergable code.
Posted by Sam Vilain on Tue, 27 Jun 2006 22:20:39 GMT
View Forum Message <> Reply to Message

Andrey Savochkin wrote:
> On Tue, Jun 27, 2006 at 11:20:40AM -0600, Eric W. Biederman wrote:
>
>> Thinking about this I am going to suggest a slightly different direction
>> for get a patchset we can merge.
>>
>> First we concentrate on the fundamentals.
>> - How we mark a device as belonging to a specific network namespace.
>> - How we mark a socket as belonging to a specific network namespace.
>>
>
> I agree with the direction of your thoughts.
> I was trying to do a similar thing, define clear steps in network

> namespace merging.
>
> My first patchset covers devices but not sockets.
> The only difference from what you're suggesting is ipv4 routing.
> For me, it is not less important than devices and sockets. May be even
> more important, since routing exposes design deficiencies less obvious at
> socket level.
>

It sounds then like it would be a good start to have general socket
namespaces, if it would merge more easily - perhaps then network device
namespaces would fall into place more easily.

AIUI socket namespaces are also necessary for situations where you want
containers to share IP addresses. AIUI, PlanetLab do something like this
with a module atop of VServer already (but read
http://openvz.org/pipermail/devel/2006-June/000666.html for a proper
explanation from Mark Huang)

>> As part of the fundamentals we add a patch to the generic socket code
>> that by default will disable it for protocol families that do not indicate
>> support for handling network namespaces, on a non-default network namespace.
>>
>
> Fine
>
> Can you summarize you objections against my way of handling devices, please?
>
There were many objections, the major one being the patch was too large for certainty of
adequate review.

Quoting what I perceived as a summary from Eric:
> When I went through this, my patchset just added an explicit
> continue if the devices was not in the appropriate namespace.
> I actually prefer the multiple list implementation but at
> the same time I think it is harder to get a clean implementation
> out of it.

You offered to re-do the patch without separate lists - I suggest that
this go ahead. No-one should really care; splitting it out into separate
lists can then be considered a performance optimization for later.

> And what was the typo you referred to in your letter to Kirill Korotaev?
>
I think this is the comment he refers to:
> These hunks should use for_each_netdev(ifp);

Both quotes are from http://lkml.org/lkml/2006/6/26/147

Though, in Kirill's defense, it seems a bit strange to expect him to
raise a fault that was just raised by Eric, in a reply to the message
where he raised it.

Sam.

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 04:20:32 GMT
View Forum Message <> Reply to Message

Andrey Savochkin <saw@swsoft.com> writes:

> Eric,
>
> On Tue, Jun 27, 2006 at 11:20:40AM -0600, Eric W. Biederman wrote:
>>
>> Thinking about this I am going to suggest a slightly different direction
>> for get a patchset we can merge.
>>
>> First we concentrate on the fundamentals.
>> - How we mark a device as belonging to a specific network namespace.
>> - How we mark a socket as belonging to a specific network namespace.
>
> I agree with the direction of your thoughts.
> I was trying to do a similar thing, define clear steps in network
> namespace merging.
>
> My first patchset covers devices but not sockets.
> The only difference from what you're suggesting is ipv4 routing.
> For me, it is not less important than devices and sockets. May be even
> more important, since routing exposes design deficiencies less obvious at
> socket level.

I agree we need to do it. I mostly want a base that allows us to
not need to convert the whole network stack at once and still be able
to merge code all the way to the stable kernel.

The routing code is important for understanding design choices. It
isn't important for merging if that makes sense.

For everyone looking at routing choices the IPv6 routing table is
interesting because it does not use a hash table, and seems quite
possibly to be an equally fast structure that scales better.

There is something to think about there.

>> As part of the fundamentals we add a patch to the generic socket code
>> that by default will disable it for protocol families that do not indicate
>> support for handling network namespaces, on a non-default network namespace.
>
> Fine
>
> Can you summarize you objections against my way of handling devices, please?
> And what was the typo you referred to in your letter to Kirill Korotaev?

I have no fundamental objects to the content I have seen so far.

Please read the first email Kirill responded too. I quoted a couple
of sections of code and described the bugs I saw with the patch.

All minor things. The typo I was referring to was a section where the
original iteration was on an ifp variable and you called it dev
without changing the rest of the code in that section.

The only big issue was that the patch too big, and should be split
into a patchset for better review. One patch for the new functions,
and the an additional patch for each driver/subsystem hunk describing
why that chunk needed to be changed.

I'm still curious why many of those chunks can't use existing helper
functions, to be cleaned up.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 04:33:48 GMT
View Forum Message <> Reply to Message

Sam Vilain <sam@vilain.net> writes:

> It sounds then like it would be a good start to have general socket
> namespaces, if it would merge more easily - perhaps then network device
> namespaces would fall into place more easily.

I guess I really see both sockets and devices as the fundamental
entities of a network namespace. Sockets need to be tagged because
in the general case there is no guarantee that a socket that you are
using was created in the network namespace of your current process.

In general it is possible to get file descriptors opened by someone

else because unix domain sockets allow file descriptor passing. Similarly
I think there are cases in both unshare and fork that allows you to sockets
open before you entered a namespace.

Since you can't create a new socket in a different network namespace
I can't see any real problems with allowing them to be used, but they
are something to be careful about in container creation code.

Something to examine here is that if both network devices and sockets
are tagged does that still allow implicit network namespace passing.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by abdallah.chatila on Wed, 28 Jun 2006 05:59:18 GMT
View Forum Message <> Reply to Message

On Tue, Jun 27, 2006 at 10:33:48PM -0600, Eric W. Biederman wrote:
>
> Something to examine here is that if both network devices and sockets
> are tagged does that still allow implicit network namespace passing.

I think avoiding implicit network namespace passing expresses more
power/flexibility plus it would make things clearer to what
container/namespace a given network resource belongs too.

>From our experience with an implementation of network containers [Virtual
Routing for ipv4/ipv6, with a complete isolation between containers where ip
addresses can overlap...], there is some problem domain in which you cannot
afford to duplicate a process/daemon in each container [a big process for
instance, scalability w.r.t. number of containers etc]

By having a proper namespace tag per socket, this can be solved by allowing
a process running in the host context to create sockets in that namespace
than moving them to the target guest namespaces [via a special setsockopt
for instance or unix domain socket as you said].

Regards

>
> Eric
> -
> To unsubscribe from this list: send the line "unsubscribe netdev" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html

Subject: Re: Network namespaces a path to mergable code.
Posted by Sam Vilain on Wed, 28 Jun 2006 06:19:19 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> In general it is possible to get file descriptors opened by someone
> else because unix domain sockets allow file descriptor passing. Similarly
> I think there are cases in both unshare and fork that allows you to sockets
> open before you entered a namespace.
>

This is an interesting point; it is known to be possible to do this on a traditional system, because with a Unix Domain socket, the other end is always in the same Unix Domain.

However what we're doing is saying that, well, the other end of the socket might not be in the same Unix Domain. In fact, we've already smashed to pieces this monolithic concept of a Unix Domain, to the point where the other end might be in a different network domain, but is in the same filesystem domain, for instance. Does it get to pass file descriptors through?

We would appear to be stretching the definition of "Unix Domain" somewhat if we allow these sockets to exist between network namespaces. Maybe it doesn't matter; this is just a VFS namespace feature/caveat.

Sam.

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 06:55:25 GMT
View Forum Message <> Reply to Message

Sam Vilain <sam@vilain.net> writes:

> Eric W. Biederman wrote:
>> In general it is possible to get file descriptors opened by someone
>> else because unix domain sockets allow file descriptor passing. Similarly
>> I think there are cases in both unshare and fork that allows you to sockets
>> open before you entered a namespace.
>>
>
> This is an interesting point; it is known to be possible to do this on a
> traditional system, because with a Unix Domain socket, the other end is
> always in the same Unix Domain.
>
> However what we're doing is saying that, well, the other end of the
> socket might not be in the same Unix Domain. In fact, we've already

> smashed to pieces this monolithic concept of a Unix Domain, to the point
> where the other end might be in a different network domain, but is in
> the same filesystem domain, for instance. Does it get to pass file
> descriptors through?

Despite what it might look like unix domain sockets do not live in the
filesystem. They store a cookie in the filesystem that roughly
corresponds to the port number of an AF_INET socket. When you open a
socket the lookup is done by the cookie retrieved from the filesystem.
So except for their cookies unix domain sockets are always in the
network stack.

Which means it is a royal pain to create a unix domain socket between
namespaces. Which is the generally desired behavior.

> We would appear to be stretching the definition of "Unix Domain"
> somewhat if we allow these sockets to exist between network namespaces.
> Maybe it doesn't matter; this is just a VFS namespace feature/caveat.

Unless I am mistaken this is something that can only be created (given
my describe semantics) when you create the container. So if you want
it you got it but you can't create it if you never had it.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by Cedric Le Goater on Wed, 28 Jun 2006 09:54:58 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:

> Despite what it might look like unix domain sockets do not live in the
> filesystem. They store a cookie in the filesystem that roughly
> corresponds to the port number of an AF_INET socket. When you open a
> socket the lookup is done by the cookie retrieved from the filesystem.

unix domain socket lookup uses a path_lookup for sockets in the filesystem
namespace and a find_by_name for socket in the abstract namespace.

> So except for their cookies unix domain sockets are always in the
> network stack.

what is that cookie ? the file dentry and mnt ref ?

so, ok, the resulting struct sock is part of the network namespace but
there is a bridge with the filesystem namespace which does not prevent
other namespaces to do a lookup. the lookup routine needs to be changed,

this is any way necessary for the abstract namespace.

I think we're reaching the limits of namespaces. It would be much easier
with a container id in each kernel object we want to isolate.

C.

Subject: Re: [RFC] Network namespaces a path to mergable code.
Posted by Cedric Le Goater on Wed, 28 Jun 2006 10:20:50 GMT
View Forum Message <> Reply to Message

Hello,

Eric W. Biederman wrote:
> Thinking about this I am going to suggest a slightly different direction
> for get a patchset we can merge.
>
> First we concentrate on the fundamentals.
> - How we mark a device as belonging to a specific network namespace.
> - How we mark a socket as belonging to a specific network namespace.
>
> As part of the fundamentals we add a patch to the generic socket code
> that by default will disable it for protocol families that do not indicate
> support for handling network namespaces, on a non-default network namespace.
>
> I think that gives us a path that will allow us to convert the network stack
> one protocol family at a time instead of in one big lump.
>
> Stubbing off the sysfs and sysctl interfaces in the first round for the
> non-default namespaces as you have done should be good enough.
>
> The reason for the suggestion is that most of the work for the protocol
> stacks ipv4 ipv6 af_packet af_unix is largely noise, and simple
> replacement without real design work happening. Mostly it is just
> tweaking the code to remove global variables, and doing a couple
> lookups.

How that proposal differs from the initial Daniel's patchset ? how far was
that patchset to reach a similar agreement ?

OK, i wear blue socks :), but I'm not advocating a patchset more than
another i'm just looking for a shorter path.

thanks,

C.

Subject: Re: Network namespaces a path to mergable code.
Posted by Andrey Savochkin on Wed, 28 Jun 2006 11:06:05 GMT
View Forum Message <> Reply to Message

Hi Eric,

On Tue, Jun 27, 2006 at 10:20:32PM -0600, Eric W. Biederman wrote:
> Andrey Savochkin <saw@swsoft.com> writes:
[snip]
> > My first patchset covers devices but not sockets.
> > The only difference from what you're suggesting is ipv4 routing.
> > For me, it is not less important than devices and sockets. May be even
> > more important, since routing exposes design deficiencies less obvious at
> > socket level.
>
> I agree we need to do it. I mostly want a base that allows us to
> not need to convert the whole network stack at once and still be able
> to merge code all the way to the stable kernel.
>
> The routing code is important for understanding design choices. It
> isn't important for merging if that makes sense.

Ok, fine.
Now I'm working on socket code.

We still have a question about implicit vs explicit function parameters.
This question becomes more important for sockets: if we want to allow to use
sockets belonging to namespaces other than the current one, we need to do
something about it.

One possible option to resolve this question is to show 2 relatively short
patches just introducing namespaces for sockets in 2 ways: with explicit
function parameters and using implicit current context.
Then people can compare them and vote.
Do you think it's worth the effort?

>
> For everyone looking at routing choices the IPv6 routing table is
> interesting because it does not use a hash table, and seems quite
> possibly to be an equally fast structure that scales better.
>
> There is something to think about there.

Sure

[snip]
> >
> > Can you summarize you objections against my way of handling devices, please?
> > And what was the typo you referred to in your letter to Kirill Korotaev?

>
> I have no fundamental objects to the content I have seen so far.
>
> Please read the first email Kirill responded too. I quoted a couple
> of sections of code and described the bugs I saw with the patch.

I found your comments, thank you!

>
> All minor things. The typo I was referring to was a section where the
> original iteration was on an ifp variable and you called it dev
> without changing the rest of the code in that section.
>
> The only big issue was that the patch too big, and should be split
> into a patchset for better review. One patch for the new functions,
> and the an additional patch for each driver/subsystem hunk describing
> why that chunk needed to be changed.

I'll split the patch.

> I'm still curious why many of those chunks can't use existing helper
> functions, to be cleaned up.

What helper functions are you referring to?

Best regards

Andrey

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Daniel Lezcano on Wed, 28 Jun 2006 13:51:32 GMT
View Forum Message <> Reply to Message

Daniel Lezcano wrote:
> Andrey Savochkin wrote:
>
>> Structures related to IPv4 rounting (FIB and routing cache)
>> are made per-namespace.

Hi Andrey,

if the ressources are private to the namespace, how do you will handle
NFS mounted before creating the network namespace ? Do you take care of
that or simply assume you can't access NFS anymore ?

Regards

-Daniel

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 14:03:41 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> Eric W. Biederman wrote:
>
>> Despite what it might look like unix domain sockets do not live in the
>> filesystem. They store a cookie in the filesystem that roughly
>> corresponds to the port number of an AF_INET socket. When you open a
>> socket the lookup is done by the cookie retrieved from the filesystem.
>
> unix domain socket lookup uses a path_lookup for sockets in the filesystem
> namespace and a find_by_name for socket in the abstract namespace.

Right. And the abstract namespace does nothing with the current
filesystem.

>> So except for their cookies unix domain sockets are always in the
>> network stack.
>
> what is that cookie ? the file dentry and mnt ref ?

The socket entry in the filesystem but really the socket
inode number in that entry. This entry has nothing to with dentry's
or mount refs so if I read the correctly every path to that socket
should yield the same entry.

> so, ok, the resulting struct sock is part of the network namespace but
> there is a bridge with the filesystem namespace which does not prevent
> other namespaces to do a lookup. the lookup routine needs to be changed,
> this is any way necessary for the abstract namespace.

Yep.

> I think we're reaching the limits of namespaces. It would be much easier
> with a container id in each kernel object we want to isolate.

Nope. Except for the fact that names are peculiar (sockets, network
device names, IP address, routes...) the network stack splits quite cleanly.

I did all of this in a proof of concept mode several months ago and
the code is still sitting in my git tree on kernel.org. I even got
the generic stack reference counting fixed.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by serue on Wed, 28 Jun 2006 14:15:39 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> > I think we're reaching the limits of namespaces. It would be much easier
> > with a container id in each kernel object we want to isolate.
>
> Nope. Except for the fact that names are peculiar (sockets, network
> device names, IP address, routes...) the network stack splits quite cleanly.
>
> I did all of this in a proof of concept mode several months ago and
> the code is still sitting in my git tree on kernel.org. I even got
> the generic stack reference counting fixed.
>
> Eric

Which branch?

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Herbert Poetzl on Wed, 28 Jun 2006 14:19:12 GMT
View Forum Message <> Reply to Message

On Wed, Jun 28, 2006 at 03:51:32PM +0200, Daniel Lezcano wrote:
> Daniel Lezcano wrote:
> >Andrey Savochkin wrote:
> >
> >>Structures related to IPv4 rounting (FIB and routing cache)
> >>are made per-namespace.
>
> Hi Andrey,
>
> if the ressources are private to the namespace, how do you will
> handle NFS mounted before creating the network namespace ?
> Do you take care of that or simply assume you can't access NFS anymore ?

considering that many providers put their guests
on NFS (or similar) filers, and run them on nodes
(for distributing the CPU load), that is indeed an
interesting question ...

what will happen to AOE or iSCSI btw?

best,
Herbert

> Regards
>
> -Daniel

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Andrey Savochkin on Wed, 28 Jun 2006 14:30:15 GMT
View Forum Message <> Reply to Message

Daniel,

On Wed, Jun 28, 2006 at 03:51:32PM +0200, Daniel Lezcano wrote:
> Daniel Lezcano wrote:
> > Andrey Savochkin wrote:
> >
> >> Structures related to IPv4 rounting (FIB and routing cache)
> >> are made per-namespace.
>
> Hi Andrey,
>
> if the ressources are private to the namespace, how do you will handle
> NFS mounted before creating the network namespace ? Do you take care of
> that or simply assume you can't access NFS anymore ?

This is a question that brings up another level of interaction between
networking and the rest of kernel code.
Solution that I use now makes the NFS communication part always run in
the root namespace. This is discussable, of course, but it's a far more
complicated matter than just device lists or routing :)

Best regards

Andrey

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by dev on Wed, 28 Jun 2006 14:34:33 GMT
View Forum Message <> Reply to Message

>>>>Structures related to IPv4 rounting (FIB and routing cache)
>>>>are made per-namespace.
>>
>>Hi Andrey,

>>
>>if the ressources are private to the namespace, how do you will handle
>>NFS mounted before creating the network namespace ? Do you take care of
>>that or simply assume you can't access NFS anymore ?
>
>
> This is a question that brings up another level of interaction between
> networking and the rest of kernel code.
> Solution that I use now makes the NFS communication part always run in
> the root namespace. This is discussable, of course, but it's a far more
> complicated matter than just device lists or routing :)
if we had containers (not namespaces) then it would be also possible to
run NFS in context of the appropriate container and thus each user could
  mount NFS itself with correct networking context.

it's another thing which ties subsytems and makes namespaces ugly :/

Kirill

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 14:56:26 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Eric W. Biederman (ebiederm@xmission.com):
>> > I think we're reaching the limits of namespaces. It would be much easier
>> > with a container id in each kernel object we want to isolate.
>>
>> Nope. Except for the fact that names are peculiar (sockets, network
>> device names, IP address, routes...) the network stack splits quite cleanly.
>>
>> I did all of this in a proof of concept mode several months ago and
>> the code is still sitting in my git tree on kernel.org. I even got
>> the generic stack reference counting fixed.
>>
>> Eric
>
> Which branch?

It should be the proof-of-concept branch. It is a development branch so the
history is ugly but the result was fairly decent.

Eric

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by ebiederm on Wed, 28 Jun 2006 15:05:50 GMT
View Forum Message <> Reply to Message

Daniel Lezcano <dlezcano@fr.ibm.com> writes:

> Daniel Lezcano wrote:
>> Andrey Savochkin wrote:
>>
>>> Structures related to IPv4 rounting (FIB and routing cache)
>>> are made per-namespace.
>
> Hi Andrey,
>
> if the ressources are private to the namespace, how do you will handle NFS
> mounted before creating the network namespace ?

Through the filesystem namespace. It is a weird case but it works :)

> Do you take care of that or simply assume you can't access NFS anymore ?

It is actually a noop. Unless I goofed this is basically handled by
looking at which socket NFS is using to communicate, and plucking the
namespace from there.

As I recall NFS gets the socket at mount time when it still has user
space context available.

So regardless if I implemented it correctly you can implement it that way
and always get the namespace context from whoever implemented it.

Eric

Subject: Re: [RFC] Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 15:20:26 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> How that proposal differs from the initial Daniel's patchset ? how far was
> that patchset to reach a similar agreement ?

My impression is as follows. The OpenVz implementation and mine work
on the same basic principles of handling the network stack at layer 2.

We have our implementation differences but the core ideas are about the
same.

Daniels patch still had elements of layer 3 handling as I recall
and that has problems.

> OK, i wear blue socks :), but I'm not advocating a patchset more than
> another i'm just looking for a shorter path.

Besides laying the foundations. The current conversation seems to be
about understanding the implications of the network stack when
we implement a network namespace.

There is a lot to the networking stack so it takes a while.
In addition this is one part of the problem that everyone has implemented,
so we have several more opinions on how it should be done and what
needs to happen.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 16:51:26 GMT
View Forum Message <> Reply to Message

Andrey Savochkin <saw@swsoft.com> writes:

> Ok, fine.
> Now I'm working on socket code.
>
> We still have a question about implicit vs explicit function parameters.
> This question becomes more important for sockets: if we want to allow to use
> sockets belonging to namespaces other than the current one, we need to do
> something about it.

There seems to be some real benefit to that. Especially for things like NFS,
that captures the context at mount time. It might as well keep the namespace
in it's socket.

> One possible option to resolve this question is to show 2 relatively short
> patches just introducing namespaces for sockets in 2 ways: with explicit
> function parameters and using implicit current context.
> Then people can compare them and vote.
> Do you think it's worth the effort?

Given that we have two strong opinions in different directions I think it
is worth the effort to resolve this.

In a slightly different vein your second patch introduced a lot
of #ifdef CONFIG_NET_NS in C files. That is something we need to look closely

at.

So I think the abstraction that we use to access per network namespace
variables needs some work if we are going to allow the ability to compile
out all of the namespace code. The explicit versus implicit lookup is just
one dimension of that problem.

>> All minor things. The typo I was referring to was a section where the
>> original iteration was on an ifp variable and you called it dev
>> without changing the rest of the code in that section.
>>
>> The only big issue was that the patch too big, and should be split
>> into a patchset for better review. One patch for the new functions,
>> and the an additional patch for each driver/subsystem hunk describing
>> why that chunk needed to be changed.
>
> I'll split the patch.

Thanks.

>> I'm still curious why many of those chunks can't use existing helper
>> functions, to be cleaned up.
>
> What helper functions are you referring to?

Basically most of the device list walker functions live in.
net/core/dev.c

I don't know if the cases you fixed could have used any of those
helper functions but it certainly has me asking that question.

A general pattern that happens in cleanups is the discovery
that code using an old interface in a problematic way really
could be done much better another way. I didn't dig enough
to see if that was the case in any of the code that you changed.

Eric

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Daniel Lezcano on Wed, 28 Jun 2006 16:56:55 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>>>>> Structures related to IPv4 rounting (FIB and routing cache)
>>>>> are made per-namespace.
>>>
>>>

>>> Hi Andrey,
>>>
>>> if the ressources are private to the namespace, how do you will
>>> handle NFS mounted before creating the network namespace ? Do you
>>> take care of that or simply assume you can't access NFS anymore ?
>>
>>
>>
>> This is a question that brings up another level of interaction between
>> networking and the rest of kernel code.
>> Solution that I use now makes the NFS communication part always run in
>> the root namespace. This is discussable, of course, but it's a far more
>> complicated matter than just device lists or routing :)
>
> if we had containers (not namespaces) then it would be also possible to
> run NFS in context of the appropriate container and thus each user could
> mount NFS itself with correct networking context.

I was asking the question because in some case, we want a lightweight
container for running applications (aka application container) who need
to share the filesystem and it will be too bad to have a network
namespace which brings isolation and prevents to implement application
containers. By the way, I agree from a point of view of a system
container, a complete network isolation is perfect.

Regards.

Daniel.

Subject: Re: [patch 3/4] Network namespaces: IPv4 FIB/routing in namespaces
Posted by Ben Greear on Wed, 28 Jun 2006 17:10:14 GMT
View Forum Message <> Reply to Message

Daniel Lezcano wrote:
> Kirill Korotaev wrote:
>
>>>>>> Structures related to IPv4 rounting (FIB and routing cache)
>>>>>> are made per-namespace.
>>>>
>>>>
>>>> Hi Andrey,
>>>>
>>>> if the ressources are private to the namespace, how do you will
>>>> handle NFS mounted before creating the network namespace ? Do you
>>>> take care of that or simply assume you can't access NFS anymore ?
>>>

>>>
>>>
>>>
>>> This is a question that brings up another level of interaction between
>>> networking and the rest of kernel code.
>>> Solution that I use now makes the NFS communication part always run in
>>> the root namespace. This is discussable, of course, but it's a far more
>>> complicated matter than just device lists or routing :)
>>
>>
>> if we had containers (not namespaces) then it would be also possible
>> to run NFS in context of the appropriate container and thus each user
>> could mount NFS itself with correct networking context.

With a relatively small patch, I was able to make NFS bind to a particular
local IP (poor man's namespace with existing code). I also changed it so
that multiple mounts to the same destination (and with unique local mount
points) are treated as unique mounts. This patch was done so that I could
stress test NFS servers, but similar logic might work for namespace isolation
as well...

Ben

--
Ben Greear <greearb@candelatech.com>
Candela Technologies Inc http://www.candelatech.com

Subject: Re: Network namespaces a path to mergable code.
Posted by Andrey Savochkin on Wed, 28 Jun 2006 17:22:40 GMT
View Forum Message <> Reply to Message

Hi Eric,

On Wed, Jun 28, 2006 at 10:51:26AM -0600, Eric W. Biederman wrote:
> Andrey Savochkin <saw@swsoft.com> writes:
>
> > One possible option to resolve this question is to show 2 relatively short
> > patches just introducing namespaces for sockets in 2 ways: with explicit
> > function parameters and using implicit current context.
> > Then people can compare them and vote.
> > Do you think it's worth the effort?
>
> Given that we have two strong opinions in different directions I think it
> is worth the effort to resolve this.

Do you have time to extract necessary parts of your old patch?
Or you aren't afraid of letting me draft an alternative version of socket

namespaces basing on your code? :)

>
> In a slightly different vein your second patch introduced a lot
> of #ifdef CONFIG_NET_NS in C files. That is something we need to look closely
> at.
>
> So I think the abstraction that we use to access per network namespace
> variables needs some work if we are going to allow the ability to compile
> out all of the namespace code. The explicit versus implicit lookup is just
> one dimension of that problem.

This is a good comment.

Those ifdef's mostly correspond to places where we walk over lists
and need to filter-out entities not belonging to a specific namespace.
Those places about the same in your and my implementation.
We can think what we can do with them.
One trick that I used on several occasions is net_ns_same macro
which doesn't evalute its arguments if CONFIG_NET_NS not defined,
and thus can be used without ifdef's.

Returning to implicit vs explicit function arguments, I belive that implicit
arguments are more promising in having zero impact on the code when
CONFIG_NET_NS is disabled.
Functions like inet_addr_type will translate into exactly the same code as
they did without net namespace patches.

>
> >> I'm still curious why many of those chunks can't use existing helper
> >> functions, to be cleaned up.
> >
> > What helper functions are you referring to?
>
> Basically most of the device list walker functions live in.
> net/core/dev.c
>
> I don't know if the cases you fixed could have used any of those
> helper functions but it certainly has me asking that question.
>
> A general pattern that happens in cleanups is the discovery
> that code using an old interface in a problematic way really
> could be done much better another way. I didn't dig enough
> to see if that was the case in any of the code that you changed.

Well, there is obvious improvement of this kind: many protocols walk over
device list to find devices with non-NULL protocol specific pointers.
For example, IPv6, decnet and others do it on module unloading to clean up.

Those places just ask for some simpler standard way of doing it, but I wasn't
bold enough for such radical change.
Do you think I should try?

Best regards

Andrey

Subject: Re: Network namespaces a path to mergable code.
Posted by Herbert Poetzl on Wed, 28 Jun 2006 17:40:06 GMT
View Forum Message <> Reply to Message

On Wed, Jun 28, 2006 at 09:22:40PM +0400, Andrey Savochkin wrote:
> Hi Eric,
>
> On Wed, Jun 28, 2006 at 10:51:26AM -0600, Eric W. Biederman wrote:
> > Andrey Savochkin <saw@swsoft.com> writes:
> >
> > > One possible option to resolve this question is to show 2
> > > relatively short patches just introducing namespaces for sockets
> > > in 2 ways: with explicit function parameters and using implicit
> > > current context. Then people can compare them and vote. Do you
> > > think it's worth the effort?
> >
> > Given that we have two strong opinions in different directions I
> > think it is worth the effort to resolve this.
>
> Do you have time to extract necessary parts of your old patch? Or you
> aren't afraid of letting me draft an alternative version of socket
> namespaces basing on your code? :)
>
> > In a slightly different vein your second patch introduced a lot of
> > #ifdef CONFIG_NET_NS in C files. That is something we need to look
> > closely at.
> >
> > So I think the abstraction that we use to access per network
> > namespace variables needs some work if we are going to allow the
> > ability to compile out all of the namespace code. The explicit
> > versus implicit lookup is just one dimension of that problem.
> This is a good comment.
>
> Those ifdef's mostly correspond to places where we walk over lists and
> need to filter-out entities not belonging to a specific namespace.
> Those places about the same in your and my implementation. We can
> think what we can do with them. One trick that I used on several
> occasions is net_ns_same macro which doesn't evalute its arguments if
> CONFIG_NET_NS not defined, and thus can be used without ifdef's.

yes, I think almost all of those cases can be avoided
while making the code even more readable by using
proper preprocessor (or even inline) mechanisms

> Returning to implicit vs explicit function arguments, I belive that
> implicit arguments are more promising in having zero impact on the
> code when CONFIG_NET_NS is disabled. Functions like inet_addr_type
> will translate into exactly the same code as they did without net
> namespace patches.

maybe a preprocessor wrapper can help here too ...

> > >> I'm still curious why many of those chunks can't use existing helper
> > >> functions, to be cleaned up.
> > >
> > > What helper functions are you referring to?
> >
> > Basically most of the device list walker functions live in.
> > net/core/dev.c
> >
> > I don't know if the cases you fixed could have used any of those
> > helper functions but it certainly has me asking that question.
> >
> > A general pattern that happens in cleanups is the discovery
> > that code using an old interface in a problematic way really
> > could be done much better another way. I didn't dig enough
> > to see if that was the case in any of the code that you changed.
>
> Well, there is obvious improvement of this kind: many protocols walk
> over device list to find devices with non-NULL protocol specific
> pointers. For example, IPv6, decnet and others do it on module
> unloading to clean up. Those places just ask for some simpler standard
> way of doing it, but I wasn't bold enough for such radical change.

> Do you think I should try?

IMHO it could not hurt to have some kind of protocol
helper library functions or macros ...

best,
Herbert

> Best regards
>
> Andrey

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 17:50:00 GMT
View Forum Message <> Reply to Message

Andrey Savochkin <saw@swsoft.com> writes:

>> A general pattern that happens in cleanups is the discovery
>> that code using an old interface in a problematic way really
>> could be done much better another way. I didn't dig enough
>> to see if that was the case in any of the code that you changed.
>
> Well, there is obvious improvement of this kind: many protocols walk over
> device list to find devices with non-NULL protocol specific pointers.
> For example, IPv6, decnet and others do it on module unloading to clean up.
> Those places just ask for some simpler standard way of doing it, but I wasn't
> bold enough for such radical change.
> Do you think I should try?

It probably makes sense to asses that after the patches are split up.
Unless you run into something obvious.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 18:11:24 GMT
View Forum Message <> Reply to Message

Andrey Savochkin <saw@swsoft.com> writes:

>> In a slightly different vein your second patch introduced a lot
>> of #ifdef CONFIG_NET_NS in C files. That is something we need to look closely
>> at.
>>
>> So I think the abstraction that we use to access per network namespace
>> variables needs some work if we are going to allow the ability to compile
>> out all of the namespace code. The explicit versus implicit lookup is just
>> one dimension of that problem.
>
> This is a good comment.
>
> Those ifdef's mostly correspond to places where we walk over lists
> and need to filter-out entities not belonging to a specific namespace.
> Those places about the same in your and my implementation.
> We can think what we can do with them.
> One trick that I used on several occasions is net_ns_same macro
> which doesn't evalute its arguments if CONFIG_NET_NS not defined,
> and thus can be used without ifdef's.

>
> Returning to implicit vs explicit function arguments, I belive that implicit
> arguments are more promising in having zero impact on the code when
> CONFIG_NET_NS is disabled.
> Functions like inet_addr_type will translate into exactly the same code as
> they did without net namespace patches.

Which brings us to a basic question. Does it make sense to have
a define that completely disables namespace support.

I know all of the simple namespaces have been implemented like that,
and it was relatively easy there. I'm not at all certain in the long
term we want a configuration option. Especially if simply enabling
the code doesn't have an impact on performance. Which I think is
a merge requirement anyway.

As for inet_addr_type and friends I do agree that implicit arguments
make for an easier implementation of CONFIG_NET_NS. My gut feel
is though that the code with explicit arguments is probably more
comprehensible in the long term. Especially as we find more weird
exceptions where the process we are running in does not have the correct
network namespace.

In general unnecessary CONFIG options are a problem because they make
the entire testing process much harder and make the code harder to
write (so that both cases work and work cleanly).

So my feeling is that we actually want to kill all of those CONFIG_XXX_NS
options.

Which simply leaves us with the problem of implementing the code cleanly.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Wed, 28 Jun 2006 18:14:41 GMT
View Forum Message <> Reply to Message

Andrey Savochkin <saw@swsoft.com> writes:

> Hi Eric,
>
> On Wed, Jun 28, 2006 at 10:51:26AM -0600, Eric W. Biederman wrote:
>> Andrey Savochkin <saw@swsoft.com> writes:
>>
>> > One possible option to resolve this question is to show 2 relatively short
>> > patches just introducing namespaces for sockets in 2 ways: with explicit

>> > function parameters and using implicit current context.
>> > Then people can compare them and vote.
>> > Do you think it's worth the effort?
>>
>> Given that we have two strong opinions in different directions I think it
>> is worth the effort to resolve this.
>
> Do you have time to extract necessary parts of your old patch?
> Or you aren't afraid of letting me draft an alternative version of socket
> namespaces basing on your code? :)

I'm not terribly afraid. I can always say you did it wrong. :)

I don't think I am going to have time today. But since this conversation
is slowing down and we are to getting into the technical details.
I will try and find some time.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by Andrey Savochkin on Wed, 28 Jun 2006 18:51:36 GMT
View Forum Message <> Reply to Message

On Wed, Jun 28, 2006 at 12:14:41PM -0600, Eric W. Biederman wrote:
> Andrey Savochkin <saw@swsoft.com> writes:
>
> > On Wed, Jun 28, 2006 at 10:51:26AM -0600, Eric W. Biederman wrote:
> >> Andrey Savochkin <saw@swsoft.com> writes:
> >>
> >> > One possible option to resolve this question is to show 2 relatively short
> >> > patches just introducing namespaces for sockets in 2 ways: with explicit
> >> > function parameters and using implicit current context.
> >> > Then people can compare them and vote.
> >> > Do you think it's worth the effort?
> >>
> >> Given that we have two strong opinions in different directions I think it
> >> is worth the effort to resolve this.
> >
> > Do you have time to extract necessary parts of your old patch?
> > Or you aren't afraid of letting me draft an alternative version of socket
> > namespaces basing on your code? :)
>
> I'm not terribly afraid. I can always say you did it wrong. :)

:)

> I don't think I am going to have time today. But since this conversation

> is slowing down and we are to getting into the technical details.
> I will try and find some time.

Good.
I'll focus on my part then.

Andrey

Subject: Re: Network namespaces a path to mergable code.
Posted by Daniel Lezcano on Wed, 28 Jun 2006 21:53:42 GMT
View Forum Message <> Reply to Message

Andrey Savochkin wrote:

> Ok, fine.
> Now I'm working on socket code.
>
> We still have a question about implicit vs explicit function parameters.
> This question becomes more important for sockets: if we want to allow to use
> sockets belonging to namespaces other than the current one, we need to do
> something about it.
>
> One possible option to resolve this question is to show 2 relatively short
> patches just introducing namespaces for sockets in 2 ways: with explicit
> function parameters and using implicit current context.
> Then people can compare them and vote.
> Do you think it's worth the effort?
>

The attached patch can have some part interesting for you for the socket
tagging. It is in the IPV4 isolation (part 5/6). With this and the
private routing table you will probably have a good IPV4 isolation.

Subject: Re: Network namespaces a path to mergable code.
Posted by James Morris on Wed, 28 Jun 2006 22:54:22 GMT
View Forum Message <> Reply to Message

On Wed, 28 Jun 2006, Daniel Lezcano wrote:

> The attached patch can have some part interesting for you for the socket
> tagging. It is in the IPV4 isolation (part 5/6). With this and the private
> routing table you will probably have a good IPV4 isolation.

Please send patches inline, do not attach them.

(Perhaps we should have a filter on vger which drops emails with
attachements).

All of this needs to be done in a way where it can be entirely disabled at
compile time, so there is zero overhead for people who don't want
network namespaces.

- James
--
James Morris
<jmorris@namei.org>

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Thu, 29 Jun 2006 00:19:05 GMT
View Forum Message <> Reply to Message

James Morris <jmorris@namei.org> writes:

> On Wed, 28 Jun 2006, Daniel Lezcano wrote:
>
>> The attached patch can have some part interesting for you for the socket
>> tagging. It is in the IPV4 isolation (part 5/6). With this and the private
>> routing table you will probably have a good IPV4 isolation.
>
> Please send patches inline, do not attach them.
>
> (Perhaps we should have a filter on vger which drops emails with
> attachements).
>
> All of this needs to be done in a way where it can be entirely disabled at
> compile time, so there is zero overhead for people who don't want
> network namespaces.

I agree with the principle of no overhead.

The goal is an implementation that has no measurable overhead when
there is only one network namespace.

If that goal is achieved and you can compile in the network namespace
code and not measure overhead there should be no need for a compile
time option.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by ebiederm on Thu, 29 Jun 2006 00:25:40 GMT
View Forum Message <> Reply to Message

Daniel Lezcano <dlezcano@fr.ibm.com> writes:

> Andrey Savochkin wrote:
>
>> Ok, fine.
>> Now I'm working on socket code.
>> We still have a question about implicit vs explicit function parameters.
>> This question becomes more important for sockets: if we want to allow to use
>> sockets belonging to namespaces other than the current one, we need to do
>> something about it.
>> One possible option to resolve this question is to show 2 relatively short
>> patches just introducing namespaces for sockets in 2 ways: with explicit
>> function parameters and using implicit current context.
>> Then people can compare them and vote.
>> Do you think it's worth the effort?
>>
>
> The attached patch can have some part interesting for you for the socket
> tagging. It is in the IPV4 isolation (part 5/6). With this and the private
> routing table you will probably have a good IPV4 isolation.
> This patch partially isolates ipv4 by adding the network namespace
> structure in the structure sock, bind bucket and skbuf.

Ugh. skbuf sounds very wrong. Per packet overhead?

> When a socket
> is created, the pointer to the network namespace is stored in the
> struct sock and the socket belongs to the namespace by this way. That
> allows to identify sockets related to a namespace for lookup and
> procfs.
>
> The lookup is extended with a network namespace pointer, in
> order to identify listen points binded to the same port. That allows
> to have several applications binded to INADDR_ANY:port in different
> network namespace without conflicting. The bind is checked against
> port and network namespace.

Yes. If we don't duplicate the hash table we need to extend the lookup.

> When an outgoing packet has the loopback destination addres, the
> skbuff is filled with the network namespace. So the loopback packets
> never go outside the namespace. This approach facilitate the migration
> of loopback because identification is done by network namespace and
> not by address. The loopback has been benchmarked by tbench and the
> overhead is roughly 1.5 %

Ugh. 1.5% is noticeable.

I think it is cheaper to have one loopback device per namespace.
Which removes the need for a skbuff tag.

Eric

Subject: Re: Network namespaces a path to mergable code.
Posted by Daniel Lezcano on Thu, 29 Jun 2006 09:42:37 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
>>When an outgoing packet has the loopback destination addres, the
>>skbuff is filled with the network namespace. So the loopback packets
>>never go outside the namespace. This approach facilitate the migration
>>of loopback because identification is done by network namespace and
>>not by address. The loopback has been benchmarked by tbench and the
>>overhead is roughly 1.5 %
>
>
> Ugh. 1.5% is noticeable.

We will see with all private network namespace ...
>
> I think it is cheaper to have one loopback device per namespace.
> Which removes the need for a skbuff tag.

Yes, probably.
