
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;
#ifdef 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, strlen(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 (!strcmp(dev->name, name, IFNAMSIZ))
            return dev;
    }
@@ -740,7 +751,7 @@ int dev_change_name(struct net_device *d
else
    strcpy(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 (!strcmp(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);
+
+
+/*
+ * 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)
```
