

---

Subject: [RFC] network namespaces

Posted by [Andrey Savochkin](#) on Tue, 15 Aug 2006 14:20:29 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Hi All,

I'd like to resurrect our discussion about network namespaces.

In our previous discussions it appeared that we have rather polar concepts which seemed hard to reconcile.

Now I have an idea how to look at all discussed concepts to enable everyone's usage scenario.

1. The most straightforward concept is complete separation of namespaces, covering device list, routing tables, netfilter tables, socket hashes, and everything else.

On input path, each packet is tagged with namespace right from the place where it appears from a device, and is processed by each layer in the context of this namespace.

Non-root namespaces communicate with the outside world in two ways: by owning hardware devices, or receiving packets forwarded them by their parent namespace via pass-through device.

This complete separation of namespaces is very useful for at least two purposes:

- allowing users to create and manage by their own various tunnels and VPNs, and
- enabling easier and more straightforward live migration of groups of processes with their environment.

2. People expressed concerns that complete separation of namespaces may introduce an undesired overhead in certain usage scenarios.

The overhead comes from packets traversing input path, then output path, then input path again in the destination namespace if root namespace acts as a router.

So, we may introduce short-cuts, when input packet starts to be processes in one namespace, but changes it at some upper layer.

The places where packet can change namespace are, for example: routing, post-routing netfilter hook, or even lookup in socket hash.

The cleanest example among them is post-routing netfilter hook.

Tagging of input packets there means that the packets is checked against root namespace's routing table, found to be local, and go directly to the socket hash lookup in the destination namespace.

In this scheme the ability to change routing tables or netfilter rules on a per-namespace basis is traded for lower overhead.

All other optimized schemes where input packets do not travel input-output-input paths in general case may be viewed as short-cuts in scheme (1). The remaining question is which exactly short-cuts make most sense, and how to make them consistent from the interface point of view.

My current idea is to reach some agreement on the basic concept, review patches, and then move on to implementing feasible short-cuts.

Opinions?

Next in this thread are patches introducing namespaces to device list, IPv4 routing, and socket hashes, and a pass-through device. Patches are against 2.6.18-rc4-mm1.

Best regards,

Andrey

---

Subject: [PATCH 1/9] network namespaces: core and device list  
Posted by [Andrey Savochkin](#) on Tue, 15 Aug 2006 14:48:36 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.

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

---

```
drivers/net/loopback.c | 69 ++++++++-----
include/linux/init_task.h | 9 ++
include/linux/net_ns.h | 82 ++++++++
include/linux/netdevice.h | 13 +++
include/linux/nsproxy.h | 3
include/linux/sched.h | 3
kernel/nsproxy.c | 14 ++++
net/Kconfig | 7 ++
net/core/dev.c | 150 ++++++++
net/core/net-sysfs.c | 24 ++++++
net/ipv4/devinet.c | 2
net/ipv6/addrconf.c | 2
net/ipv6/route.c | 9 +-
13 files changed, 349 insertions(+), 38 deletions(-)
```

--- ./drivers/net/loopback.c.vendsdev Mon Aug 14 17:02:18 2006

+++ ./drivers/net/loopback.c Mon Aug 14 17:18:20 2006

@@ -196,42 +196,55 @@ static struct ethtool\_ops loopback\_ethtool\_ops = {  
.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;
+ 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.vensdev Mon Aug 14 17:04:04 2006
+++ ./include/linux/init_task.h Mon Aug 14 17:18:21 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=0x1ffff (=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.vensdev Mon Aug 14 17:18:21 2006
+++ ./include/linux/net_ns.h Mon Aug 14 17:18:21 2006
@@ -0,0 +1,82 @@
+/*
+ * 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 net_device *dev_base_p, **dev_tail_p;
+ struct net_device *loopback;
+ unsigned int hash;
+ struct work_struct 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);
+}
+
+extern struct net_namespace init_net_ns;

```

```

+#define current_net_ns (current->net_context)
+
+#define push_net_ns(to, orig) do { \
+   struct task_struct *__cur; \
+   __cur = current; \
+   orig = __cur->net_context; \
+   __cur->net_context = to; \
+ } while (0)
+#define pop_net_ns(orig) do { \
+   current->net_context = orig; \
+ } while (0)
+#define switch_net_ns(to) do { \
+   current->net_context = to; \
+ } while (0)
+
+#define net_ns_match(target, context) ((target) == (context))
+#define net_ns_same(ns1, ns2) ((ns1) == (ns2))
+
+#define net_ns_hash(ns) ((ns)->hash)
+
+#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 push_net_ns(to, orig) do { \
+   orig = NULL; \
+ } while (0)
+#define pop_net_ns(orig) do { \
+   (void) orig; \
+ } while (0)
+#define switch_net_ns(to) do { \
+ } while (0)
+
+#define net_ns_match(target, context) ((void)(context), 1)
+#define net_ns_same(ns1, ns2) 1
+
+#define net_ns_hash(ns) 0
+
+#endif /* CONFIG_NET_NS */
+
+#define current_net_hash net_ns_hash(current_net_ns)
+
+#endif /* __LINUX_NET_NS__ */

```

```

--- ./include/linux/netdevice.h.vensdev Mon Aug 14 17:04:04 2006
+++ ./include/linux/netdevice.h Mon Aug 14 17:18:21 2006
@@ -374,6 +374,10 @@ struct net_device
    int  promiscuity;
    int  allmulti;

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

    /* Protocol specific pointers */

@@ -556,9 +560,16 @@ 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 net_device *dev_base; /* All devices */
#else
#define loopback_dev (*current_net_ns->loopback)
#define dev_base (current_net_ns->dev_base_p)
#endif
extern rwlock_t dev_base_lock; /* Device list lock */

extern int netdev_boot_setup_check(struct net_device *dev);
--- ./include/linux/nsproxy.h.vensdev Mon Aug 14 17:04:04 2006
+++ ./include/linux/nsproxy.h Mon Aug 14 17:18:21 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.vensdev Mon Aug 14 17:04:04 2006
+++ ./include/linux/sched.h Mon Aug 14 17:18:21 2006
@@ -917,6 +917,9 @@ 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.vensdev Mon Aug 14 17:04:05 2006
+++ ./kernel/nsproxy.c Mon Aug 14 17:18:21 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.vensdev Mon Aug 14 17:04:05 2006

```

```

+++ ./net/Kconfig Mon Aug 14 17:18:21 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.vensdev Mon Aug 14 17:04:05 2006
+++ ./net/core/dev.c Mon Aug 14 17:18:21 2006
@@ -90,6 +90,7 @@
#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>
@@ -174,11 +175,15 @@ static spinlock_t net_dma_event_lock;
 * unregister_netdevice(), which must be called with the rtnl
 * semaphore held.
 */
+#ifndef CONFIG_NET_NS
+struct net_device *dev_base;
+static struct net_device **dev_tail = &dev_base;
+DEFINE_RWLOCK(dev_base_lock);
+
+EXPORT_SYMBOL(dev_base);
+#else
+#define dev_tail (current_net_ns->dev_tail_p)
+#endif
+
+DEFINE_RWLOCK(dev_base_lock);
+EXPORT_SYMBOL(dev_base_lock);

#define NETDEV_HASHBITS 8
@@ -188,6 +193,7 @@ 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);
+ hash ^= current_net_hash;

```

```

    return &dev_name_head[hash & ((1<<NETDEV_HASHBITS)-1)];
}

```

```

@@ -212,10 +218,12 @@ 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

```

```

@@ -475,10 +483,13 @@ __setup("netdev=", netdev_boot_setup);
struct net_device *__dev_get_by_name(const char *name)
{
    struct hlist_node *p;
+ struct net_namespace *ns = 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_match(dev->net_ns, ns))
+ continue;
        if (!strncmp(dev->name, name, IFNAMSIZ))
            return dev;
    }
}

```

```

@@ -742,7 +753,7 @@ 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));

```

```

@@ -1683,7 +1694,10 @@ static void net_tx_action(struct softirq
    clear_bit(__LINK_STATE_SCHED, &dev->state);

```

```

    if (spin_trylock(&dev->queue_lock)) {
+ struct net_namespace *orig_net_ns;
+ push_net_ns(dev->net_ns, orig_net_ns);
        qdisc_run(dev);
+ pop_net_ns(orig_net_ns);
        spin_unlock(&dev->queue_lock);

```

```

    } else {
        netif_schedule(dev);
@@ -1770,6 +1784,7 @@ 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;
    int ret = NET_RX_DROP;
    unsigned short type;

@@ -1788,6 +1803,8 @@ int netif_receive_skb(struct sk_buff *sk
    if (!orig_dev)
        return NET_RX_DROP;

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

    skb->h.raw = skb->nh.raw = skb->data;
@@ -1858,6 +1875,7 @@ ncls:

out:
    rcu_read_unlock();
+ pop_net_ns(orig_net_ns);
    return ret;
}

@@ -2885,6 +2903,7 @@ int register_netdevice(struct net_device
{
    struct hlist_head *head;
    struct hlist_node *p;
+ struct net_namespace *ns = current_net_ns;
    int ret;

    BUG_ON(dev_boot_phase);
@@ -2902,9 +2921,19 @@ int register_netdevice(struct net_device
    spin_lock_init(&dev->ingress_lock);
#endif

#ifdef CONFIG_NET_NS
+ dev->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;

@@ -2932,6 +2961,8 @@ 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_match(d->net_ns, ns))
+ continue;
if (!strcmp(d->name, dev->name, IFNAMSIZ)) {
ret = -EEXIST;
goto out_err;
@@ -3007,6 +3038,12 @@ out:
return ret;
out_err:
free_divert_blk(dev);
+out_divert:
#ifdef CONFIG_NET_NS
+ if (dev != ns->loopback)
+ put_net_ns(ns);
+ dev->net_ns = NULL;
#endif
goto out;
}

@@ -3132,9 +3169,11 @@ static DEFINE_MUTEX(net_todo_run_mutex);
void netdev_run_todo(void)
{
struct list_head list;
+ struct net_namespace *orig_net_ns;

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

/* Not safe to do outside the semaphore. We must not return
* until all unregister events invoked by the local processor
@@ -3161,6 +3200,7 @@ void netdev_run_todo(void)
continue;
}

+ switch_net_ns(dev->net_ns);
netdev_unregister_sysfs(dev);

```

```

dev->reg_state = NETREG_UNREGISTERED;

@@ -3180,6 +3220,7 @@ void netdev_run_todo(void)
}

out:
+ pop_net_ns(orig_net_ns);
+ mutex_unlock(&net_todo_run_mutex);
}

@@ -3234,6 +3275,16 @@ 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) {
+ 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) {
@@ -3244,6 +3295,13 @@ 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
@@ -3489,6 +3547,90 @@ 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_p = &loopback_dev_static,
+ .dev_tail_p = &init_net_ns.dev_base_p,
+ .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;
+ struct task_struct *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);
+ ns->dev_base_p = NULL;
+ ns->dev_tail_p = &ns->dev_base_p;
+ 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);
+
+/* 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);
+ if (ns->dev_base_p != NULL) {
+ printk("NET_NS: BUG: context %p has devices! ref %d\n",
+ ns, atomic_read(&ns->active_ref));
+ pop_net_ns(orig_ns);
+ return;
+ }
+ pop_net_ns(orig_ns);
+ kfree(ns);
+}
+
+void net_ns_stop(struct net_namespace *ns)
+{
+ INIT_WORK(&ns->destroy_work, net_ns_destroy, ns);
+ schedule_work(&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.vensdev Mon Aug 14 17:02:49 2006
+++ ./net/core/net-sysfs.c Mon Aug 14 17:18:21 2006
@@ -12,6 +12,7 @@
#include <linux/capability.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>
@@ -444,6 +445,12 @@ 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));
}

@@ -453,6 +460,12 @@ 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;
@@ -473,6 +486,17 @@ 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.venusdev Mon Aug 14 17:04:07 2006
+++ ./net/ipv4/devinet.c Mon Aug 14 17:18:21 2006
@@ -197,7 +197,7 @@ 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.vensdev Mon Aug 14 17:04:10 2006
+++ ./net/ipv6/addrconf.c Mon Aug 14 17:18:21 2006
@@ -2331,7 +2331,7 @@ 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.vensdev Mon Aug 14 17:04:10 2006
+++ ./net/ipv6/route.c Mon Aug 14 17:18:21 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, },
@@ -147,7 +147,7 @@ struct rt6_info ip6_prohibit_entry = {
    .dst = {
        .__refcnt = ATOMIC_INIT(1),
        .__use = 1,
-    .dev = &loopback_dev,
+    /* .dev = &loopback_dev, */
        .obsolete = -1,
        .error = -EACCES,
        .metrics = { [RTAX_HOPLIMIT - 1] = 255, },
@@ -167,7 +167,7 @@ struct rt6_info ip6_blk_hole_entry = {
    .dst = {
        .__refcnt = ATOMIC_INIT(1),
        .__use = 1,
-    .dev = &loopback_dev,
+    /* .dev = &loopback_dev, */
        .obsolete = -1,
        .error = -EINVAL,
        .metrics = { [RTAX_HOPLIMIT - 1] = 255, },
@@ -2332,6 +2332,9 @@ void __init ip6_route_init(void)
#ifdef CONFIG_IPV6_MULTIPLE_TABLES
    fib6_rules_init();

```

```
#endif
+ ip6_null_entry.u.dst.dev = &loopback_dev;
+ ip6_prohibit_entry.u.dst.dev = &loopback_dev;
+ ip6_blk_hole_entry.u.dst.dev = &loopback_dev;
}
```

```
void ip6_route_cleanup(void)
```

---



---

Subject: [PATCH 2/9] network namespaces: IPv4 routing  
 Posted by [Andrey Savochkin](#) on Tue, 15 Aug 2006 14:48:42 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

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

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

---

```
include/linux/net_ns.h | 10 +++
include/net/flow.h      | 3 +
include/net/ip_fib.h    | 46 ++++++++-----
net/core/dev.c          | 8 ++
net/core/fib_rules.c    | 43 ++++++++-----
net/ipv4/Kconfig        | 4 -
net/ipv4/fib_frontend.c | 132 ++++++++-----
net/ipv4/fib_hash.c     | 13 +++-
net/ipv4/fib_rules.c    | 86 ++++++++-----
net/ipv4/fib_semantics.c | 99 ++++++++-----
net/ipv4/route.c        | 26 ++++++-
11 files changed, 375 insertions(+), 95 deletions(-)
```

--- ./include/linux/net\_ns.h.vensroute Mon Aug 14 17:18:59 2006

+++ ./include/linux/net\_ns.h Mon Aug 14 19:19:14 2006

```
@@ -14,7 +14,17 @@ struct net_namespace {
    atomic_t active_ref, use_ref;
    struct net_device *dev_base_p, **dev_tail_p;
    struct net_device *loopback;
+ #ifndef CONFIG_IP_MULTIPLE_TABLES
+ struct fib_table *fib4_local_table, *fib4_main_table;
+ #else
+ struct list_head fib_rules_ops_list;
+ struct fib_rules_ops *fib4_rules_ops;
+ struct hlist_head *fib4_tables;
+ #endif
+ struct hlist_head *fib4_hash, *fib4_laddrhash;
+ unsigned fib4_hash_size, fib4_info_cnt;
    unsigned int hash;
+ char destroying;
```

```

    struct work_struct destroy_work;
};

--- ./include/net/flow.h.vensroute Mon Aug 14 17:04:04 2006
+++ ./include/net/flow.h Mon Aug 14 17:18:59 2006
@@ -79,6 +79,9 @@ struct flowi {
    #define fl_icmp_code uli_u.icmpt.code
    #define fl_ipsec_spi uli_u.spi
    __u32          secid; /* used by xfrm; see secid.txt */
+#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.vensroute Mon Aug 14 17:04:04 2006
+++ ./include/net/ip_fib.h Tue Aug 15 11:53:22 2006
@@ -18,6 +18,7 @@

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

/* WARNING: The ordering of these elements must match ordering
@@ -171,14 +172,21 @@ struct fib_table {

#ifdef 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(u32 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(u32 id)
@@ -188,21 +196,29 @@ 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_get_table(RT_TABLE_LOCAL)
-#define ip_fib_main_table fib_get_table(RT_TABLE_MAIN)
+#define ip_fib_local_table_ns() fib_get_table(RT_TABLE_LOCAL)
+#define ip_fib_main_table_ns() fib_get_table(RT_TABLE_MAIN)

extern int fib_lookup(struct flowi *flp, struct fib_result *res);

@@ -214,6 +230,10 @@ 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_cleanup(void);
#endif
extern int inet_rtm_delroute(struct sk_buff *skb, struct nlmsg_hdr *nlh, void *arg);
extern int inet_rtm_newroute(struct sk_buff *skb, struct nlmsg_hdr *nlh, void *arg);

```

```

extern int inet_rtm_getroute(struct sk_buff *skb, struct nlmsg_hdr* nlh, void *arg);
@@ -231,6 +251,9 @@ extern int fib_sync_up(struct net_device
extern int fib_convert_rtnetentry(int cmd, struct nlmsg_hdr *nl, struct rtmsg *rtm,
    struct kern_rta *rta, struct rtnetentry *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(u32 id);
@@ -238,7 +261,10 @@ extern struct fib_table *fib_hash_init(u
#ifdef CONFIG_IP_MULTIPLE_TABLES
extern int fib4_rules_dump(struct sk_buff *skb, struct netlink_callback *cb);

-extern void __init fib4_rules_init(void);
+extern int fib4_rules_init(void);
#ifdef CONFIG_NET_NS
+extern void fib4_rules_cleanup(void);
#endif

#ifdef CONFIG_NET_CLS_ROUTE
extern u32 fib_rules_tclass(struct fib_result *res);
--- ./net/core/dev.c.vensroute Mon Aug 14 17:18:59 2006
+++ ./net/core/dev.c Mon Aug 14 17:18:59 2006
@@ -3548,6 +3548,8 @@ 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,
@@ -3588,6 +3590,9 @@ int net_ns_start(void)
    task = current;
    orig_ns = task->net_context;
    task->net_context = ns;
+ INIT_LIST_HEAD(&ns->fib_rules_ops_list);
+ if (ip_fib_struct_init())
+ goto out_fib4;
    err = register_netdev(dev);
    if (err)
        goto out_register;
@@ -3595,6 +3600,8 @@ int net_ns_start(void)
    return 0;

out_register:

```

```

+ ip_fib_struct_cleanup();
+out_fib4:
    dev->destructor(dev);
    task->net_context = orig_ns;
    BUG_ON(atomic_read(&ns->active_ref) != 1);
@@ -3619,6 +3626,7 @@ static void net_ns_destroy(void *data)
    pop_net_ns(orig_ns);
    return;
}
+ ip_fib_struct_cleanup();
    pop_net_ns(orig_ns);
    kfree(ns);
}
--- ./net/core/fib_rules.c.vensroute Mon Aug 14 17:04:05 2006
+++ ./net/core/fib_rules.c Mon Aug 14 17:18:59 2006
@@ -11,9 +11,15 @@
#include <linux/types.h>
#include <linux/kernel.h>
#include <linux/list.h>
+#include <linux/net_ns.h>
#include <net/fib_rules.h>

-static LIST_HEAD(rules_ops);
+#ifndef CONFIG_NET_NS
+static struct list_head rules_ops_static;
+#define rules_ops_ns() rules_ops_static
+#else
+#define rules_ops_ns() (current_net_ns->fib_rules_ops_list)
+#endif
static DEFINE_SPINLOCK(rules_mod_lock);

static void notify_rule_change(int event, struct fib_rule *rule,
@@ -21,10 +27,12 @@ static void notify_rule_change(int event

static struct fib_rules_ops *lookup_rules_ops(int family)
{
+ struct list_head *ops_list;
    struct fib_rules_ops *ops;

+ ops_list = &rules_ops_ns();
    rcu_read_lock();
- list_for_each_entry_rcu(ops, &rules_ops, list) {
+ list_for_each_entry_rcu(ops, ops_list, list) {
    if (ops->family == family) {
        if (!try_module_get(ops->owner))
            ops = NULL;
@@ -46,6 +54,7 @@ static void rules_ops_put(struct fib_rul
int fib_rules_register(struct fib_rules_ops *ops)

```

```

{
    int err = -EEXIST;
+ struct list_head *ops_list;
    struct fib_rules_ops *o;

    if (ops->rule_size < sizeof(struct fib_rule))
@@ -56,12 +65,13 @@ int fib_rules_register(struct fib_rules_
    ops->action == NULL)
    return -EINVAL;

+ ops_list = &rules_ops_ns();
    spin_lock(&rules_mod_lock);
- list_for_each_entry(o, &rules_ops, list)
+ list_for_each_entry(o, ops_list, list)
    if (ops->family == o->family)
        goto errout;

- list_add_tail_rcu(&ops->list, &rules_ops);
+ list_add_tail_rcu(&ops->list, ops_list);
    err = 0;
errout:
    spin_unlock(&rules_mod_lock);
@@ -84,10 +94,12 @@ static void cleanup_ops(struct fib_rules
int fib_rules_unregister(struct fib_rules_ops *ops)
{
    int err = 0;
+ struct list_head *ops_list;
    struct fib_rules_ops *o;

+ ops_list = &rules_ops_ns();
    spin_lock(&rules_mod_lock);
- list_for_each_entry(o, &rules_ops, list) {
+ list_for_each_entry(o, ops_list, list) {
    if (o == ops) {
        list_del_rcu(&o->list);
        cleanup_ops(ops);
@@ -114,6 +126,14 @@ int fib_rules_lookup(struct fib_rules_op

    rcu_read_lock();

+ err = -EINVAL;
+ if (ops->rules_list->next == NULL) {
+ if (net_ratelimit())
+ printk(" *** NULL head, ops %p, list %p\n",
+ ops, ops->rules_list);
+ goto out;
+ }
+

```

```

list_for_each_entry_rcu(rule, ops->rules_list, list) {
    if (rule->ifindex && (rule->ifindex != fl->iif))
        continue;
@@ -127,6 +147,12 @@ int fib_rules_lookup(struct fib_rules_op
    arg->rule = rule;
    goto out;
}
+ if (rule->list.next == NULL) {
+     if (net_ratelimit())
+         printk(" *** NULL, ops %p, list %p, item %p\n",
+             ops, ops->rules_list, rule);
+     goto out;
+ }
}

err = -ENETUNREACH;
@@ -382,19 +408,21 @@ static int fib_rules_event(struct notifi
    void *ptr)
{
    struct net_device *dev = ptr;
+ struct list_head *ops_list;
    struct fib_rules_ops *ops;

    ASSERT_RTNL();
    rcu_read_lock();

+ ops_list = &rules_ops_ns();
    switch (event) {
        case NETDEV_REGISTER:
- list_for_each_entry(ops, &rules_ops, list)
+ list_for_each_entry(ops, ops_list, list)
            attach_rules(ops->rules_list, dev);
        break;

        case NETDEV_UNREGISTER:
- list_for_each_entry(ops, &rules_ops, list)
+ list_for_each_entry(ops, ops_list, list)
            detach_rules(ops->rules_list, dev);
        break;
    }
@@ -410,6 +438,7 @@ static struct notifier_block fib_rules_n

static int __init fib_rules_init(void)
{
+ INIT_LIST_HEAD(&rules_ops_ns());
    return register_netdevice_notifier(&fib_rules_notifier);
}

```

```
--- ./net/ipv4/Kconfig.vensroute Mon Aug 14 17:04:07 2006
+++ ./net/ipv4/Kconfig Mon Aug 14 17:18:59 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.vensroute Mon Aug 14 17:04:07 2006
+++ ./net/ipv4/fib_frontend.c Tue Aug 15 11:53:22 2006
@@ -51,18 +51,23 @@
#define FFprint(a...) printk(KERN_DEBUG a)

#ifdef CONFIG_IP_MULTIPLE_TABLES
-
-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
#define FIB_TABLE_HASHSZ 1
-static struct hlist_head fib_table_hash[FIB_TABLE_HASHSZ];
-
-#else
-
#define FIB_TABLE_HASHSZ 256
-static struct hlist_head fib_table_hash[FIB_TABLE_HASHSZ];
+#endif

+#ifndef CONFIG_NET_NS
+static struct hlist_head fib_table_hash_static[FIB_TABLE_HASHSZ];
+#define fib_table_hash_ns() fib_table_hash_static
+#else
+#define fib_table_hash_ns() (current_net->fib4_tables)
```

```

+ #endif
+
+ #ifdef CONFIG_IP_MULTIPLE_TABLES
struct fib_table *fib_new_table(u32 id)
{
    struct fib_table *tb;
@@ -77,21 +82,23 @@ struct fib_table *fib_new_table(u32 id)
    if (!tb)
        return NULL;
    h = id & (FIB_TABLE_HASHSZ - 1);
- hlist_add_head_rcu(&tb->tb_hlist, &fib_table_hash[h]);
+ hlist_add_head_rcu(&tb->tb_hlist, &fib_table_hash_ns()[h]);
    return tb;
}

struct fib_table *fib_get_table(u32 id)
{
    struct fib_table *tb;
+ struct hlist_head *list;
    struct hlist_node *node;
    unsigned int h;

    if (id == 0)
        id = RT_TABLE_MAIN;
    h = id & (FIB_TABLE_HASHSZ - 1);
+ list = &fib_table_hash_ns()[h];
    rcu_read_lock();
- hlist_for_each_entry_rcu(tb, node, &fib_table_hash[h], tb_hlist) {
+ hlist_for_each_entry_rcu(tb, node, list, tb_hlist) {
    if (tb->tb_id == id) {
        rcu_read_unlock();
        return tb;
@@ -106,11 +113,13 @@ static void fib_flush(void)
{
    int flushed = 0;
    struct fib_table *tb;
+ struct hlist_head *list;
    struct hlist_node *node;
    unsigned int h;

    for (h = 0; h < FIB_TABLE_HASHSZ; h++) {
- hlist_for_each_entry(tb, node, &fib_table_hash[h], tb_hlist)
+ list = &fib_table_hash_ns()[h];
+ hlist_for_each_entry(tb, node, list, tb_hlist)
        flushed += tb->tb_flush(tb);
    }

@@ -126,14 +135,15 @@ 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;
@@ -150,6 +160,7 @@ 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))
@@ -161,10 +172,10 @@ unsigned inet_addr_type(u32 addr)
    res.r = NULL;
#endif

- if (ip_fib_local_table) {
+ 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);
    }
@@ -357,6 +368,7 @@ int inet_dump_fib(struct sk_buff *skb, s
    unsigned int h, s_h;
    unsigned int e = 0, s_e;
    struct fib_table *tb;
+ struct hlist_head *list;
    struct hlist_node *node;
    int dumped = 0;

@@ -369,7 +381,8 @@ int inet_dump_fib(struct sk_buff *skb, s

```



```

#else
- fib4_rules_init();
-#endif
+
+int ip_fib_struct_init(void)
+{
+ struct net_namespace *ns = current_net_ns;
+ struct hlist_head *tables;
+ unsigned int i;
+
+ tables = kmalloc(FIB_TABLE_HASHSZ * sizeof(*tables), GFP_KERNEL);
+ if (tables == NULL)
+ return -ENOMEM;
+ for (i = 0; i < FIB_TABLE_HASHSZ; i++)
+ INIT_HLIST_HEAD(&tables[i]);
+ ns->fib4_tables = tables;
+
+ if (fib4_rules_init())
+ goto out_fib4_rules;
+ return 0;
+
+out_fib4_rules:
+ kfree(tables);
+ ns->fib4_tables = NULL;
+ return -ENOMEM;
+}
+
+#endif /* CONFIG_NET_NS */
+
+void __init ip_fib_init(void)
+{
+ ip_fib_struct_init();

    register_netdevice_notifier(&fib_netdev_notifier);
    register_inetaddr_notifier(&fib_inetaddr_notifier);
    nl_fib_lookup_init();
}

#ifdef CONFIG_NET_NS
+void ip_fib_struct_cleanup(void)
+{
+ current_net_ns->destroying = 1;
+ rtnl_lock();
+#ifdef CONFIG_IP_MULTIPLE_TABLES
+ fib4_rules_cleanup();
+#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_table_hash_ns());
+ fib_table_hash_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.venusroute Mon Aug 14 17:04:07 2006
+++ ./net/ipv4/fib_hash.c Mon Aug 14 17:18:59 2006
@@ -627,6 +627,11 @@ 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;
@@ -636,7 +641,9 @@ 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)) {
@@ -817,7 +824,7 @@ 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;
@@ -956,7 +963,7 @@ 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.vensroute Mon Aug 14 17:04:08 2006
+++ ./net/ipv4/fib_rules.c Mon Aug 14 17:51:02 2006
@@ -32,7 +32,7 @@
#include <net/ip_fib.h>
#include <net/fib_rules.h>

-static struct fib_rules_ops fib4_rules_ops;
+static struct fib_rules_ops fib4_rules_ops_static;

struct fib4_rule
{
@@ -79,7 +79,12 @@ static struct fib4_rule local_rule = {
},
};

-static LIST_HEAD(fib4_rules);
+#ifndef CONFIG_NET_NS
+static LIST_HEAD(fib4_rules_static);
+#define fib4_rules_ops_ns() fib4_rules_ops_static
+#else
+#define fib4_rules_ops_ns() (*current_net_ns->fib4_rules_ops)
+#endif

#ifdef CONFIG_NET_CLS_ROUTE
u32 fib_rules_tclass(struct fib_result *res)
@@ -95,7 +100,7 @@ int fib_lookup(struct flowi *flp, struct
};
int err;

- err = fib_rules_lookup(&fib4_rules_ops, flp, 0, &arg);
+ err = fib_rules_lookup(&fib4_rules_ops_ns(), flp, 0, &arg);
    res->r = arg.rule;

    return err;
@@ -311,11 +316,13 @@ int fib4_rules_dump(struct sk_buff *skb,
static u32 fib4_rule_default_pref(void)
{
    struct list_head *pos;

```

```

+ struct list_head *fib4_rules;
+ struct fib_rule *rule;

- if (!list_empty(&fib4_rules)) {
- pos = fib4_rules.next;
- if (pos->next != &fib4_rules) {
+ fib4_rules = fib4_rules_ops_ns().rules_list;
+ if (!list_empty(fib4_rules)) {
+ pos = fib4_rules->next;
+ if (pos->next != fib4_rules) {
+ rule = list_entry(pos->next, struct fib_rule, list);
+ if (rule->pref)
+ return rule->pref - 1;
@@ -325,7 +332,7 @@ static u32 fib4_rule_default_pref(void)
+ return 0;
+ }

-static struct fib_rules_ops fib4_rules_ops = {
+static struct fib_rules_ops fib4_rules_ops_static = {
+ .family = AF_INET,
+ .rule_size = sizeof(struct fib4_rule),
+ .action = fib4_rule_action,
@@ -336,15 +343,68 @@ static struct fib_rules_ops fib4_rules_o
+ .default_pref = fib4_rule_default_pref,
+ .nlgroup = RTNLGRP_IPV4_RULE,
+ .policy = fib4_rule_policy,
- .rules_list = &fib4_rules,
+ .owner = THIS_MODULE,
+ };

-void __init fib4_rules_init(void)
+ #ifndef CONFIG_NET_NS
+ +
+ int fib4_rules_init(void)
+ {
+ fib4_rules_ops_static.rules_list = &fib4_rules_static;
+ list_add_tail(&local_rule.common.list, &fib4_rules_static);
+ list_add_tail(&main_rule.common.list, &fib4_rules_static);
+ list_add_tail(&default_rule.common.list, &fib4_rules_static);
+ fib_rules_register(&fib4_rules_ops_static);
+ return 0;
+ }
+
+ #else
+ +
+ static int fib4_rule_create(struct fib4_rule *orig, struct list_head *head)
+ {
+ struct fib4_rule *p;

```

```

+
+ p = kmalloc(sizeof(*p), GFP_KERNEL);
+ if (p == NULL)
+ return -1;
+ memcpy(p, orig, sizeof(*p));
+ list_add_tail_rcu(&p->common.list, head);
+ return 0;
+}
+
+int fib4_rules_init(void)
+{
- list_add_tail(&local_rule.common.list, &fib4_rules);
- list_add_tail(&main_rule.common.list, &fib4_rules);
- list_add_tail(&default_rule.common.list, &fib4_rules);
+ struct fib_rules_ops *ops;
+ struct list_head *rules;
+
+ ops = kmalloc(sizeof(*ops) + sizeof(*rules), GFP_KERNEL);
+ if (ops == NULL)
+ goto out;
+ memcpy(ops, &fib4_rules_ops_static, sizeof(*ops));
+ rules = (struct list_head *) (ops + 1);
+ INIT_LIST_HEAD(rules);
+ ops->rules_list = rules;
+ current_net_ns->fib4_rules_ops = ops;
+
+ fib_rules_register(ops);
+
+ if (fib4_rule_create(&local_rule, rules) ||
+     fib4_rule_create(&main_rule, rules) ||
+     fib4_rule_create(&default_rule, rules))
+ goto out_rule;
+
+ return 0;

- fib_rules_register(&fib4_rules_ops);
+out_rule:
+ fib_rules_unregister(ops); /* list cleanup is inside */
+ kfree(ops);
+out:
+ return -ENOMEM;
+}
+
+void fib4_rules_cleanup(void)
+{
+ fib_rules_unregister(&fib4_rules_ops_ns());
+}
+

```

```

+endif
--- ./net/ipv4/fib_semantics.c.venusroute Mon Aug 14 17:04:08 2006
+++ ./net/ipv4/fib_semantics.c Mon Aug 14 18:04:32 2006
@@ -50,10 +50,21 @@
#define FSprntk(a...)

static DEFINE_SPINLOCK(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)
@@ -153,7 +164,7 @@ 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(current_net_ns)--;
    kfree(fi);
}

@@ -196,9 +207,10 @@ static __inline__ int nh_comp(const struct
    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;
@@ -210,13 +222,14 @@ 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)
@@ -236,11 +249,13 @@ 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));
+ r ^= current_net_hash;
+ return r & mask;
}

/* Check, that the gateway is already configured.
@@ -563,9 +578,9 @@ 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;

```

```

}
@@ -594,17 +609,18 @@ 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;

    spin_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;

@@ -614,15 +630,15 @@ 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;

@@ -632,12 +648,12 @@ 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;

spin_unlock(&fib_info_lock);

@@ -646,11 +662,27 @@ static void fib_hash_move(struct hlist_head *old_h, struct hlist_head *new_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
@@ -684,8 +716,8 @@ fib_create_info(const struct rtmsg *r, struct kern_rta *rta, struct nlmsghdr *nlh, int *errp) {
#endif

    err = -ENOBUFFS;
- 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;
@@ -705,14 +737,14 @@ 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 = kzalloc(sizeof(*fi)+nhs*sizeof(struct fib_nh), GFP_KERNEL);
if (fi == NULL)
    goto failure;
- fib_info_cnt++;
+ fib_info_cnt(ns)++;

fi->fib_protocol = r->rtm_protocol;

@@ -822,11 +854,11 @@ link_it:
    atomic_inc(&fi->fib_clntref);
    spin_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) {
@@ -1165,15 +1197,16 @@ fib_convert_rtrentry(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.venusroute Mon Aug 14 17:04:10 2006
+++ ./net/ipv4/route.c Mon Aug 14 17:18:59 2006
@@ -266,6 +266,7 @@ 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;
@@ -278,21 +279,28 @@ static struct rtable *rt_cache_get_first
  break;
  rcu_read_unlock_bh();
}
+ if (r && !net_ns_match(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 = 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_match(r->fl.net_ns, ns))
+   goto next;
+out:
  return r;
}

@@ -563,6 +571,7 @@ 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;
}
@@ -1126,6 +1135,7 @@ void ip_rt_redirect(u32 old_gw, u32 daddr
    struct rtable *rth, **rthp;
    u32 skeys[2] = { saddr, 0 };
    int ikeys[2] = { dev->ifindex, 0 };
+ struct net_namespace *ns = current_net_ns;
    struct netevent_redirect netevent;

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

    if (rth->fl.fl4_dst != daddr ||
        rth->fl.fl4_src != skeys[i] ||
+   !net_ns_match(rth->fl.net_ns, ns) ||
        rth->fl.oif != ikeys[k] ||
        rth->fl.iif != 0) {
        rthp = &rth->u.rt_next;
@@ -1649,6 +1660,9 @@ 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;
@@ -1792,6 +1806,9 @@ 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;
@@ -2093,6 +2110,7 @@ int ip_route_input(struct sk_buff *skb,
    struct rtable *rth;
    unsigned hash;
    int iif = dev->ifindex;
+ struct net_namespace *ns = current_net_ns;

    tos &= IPTOS_RT_MASK;

```

```

hash = rt_hash_code(daddr, saddr ^ (iif << 5));
@@ -2102,6 +2120,7 @@ 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_match(rth->fl.net_ns, ns) &&
        rth->fl.iif == iif &&
        rth->fl.oif == 0 &&
#ifdef CONFIG_IP_ROUTE_FWMARK
@@ -2241,6 +2260,9 @@ 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;

@@ -2566,6 +2588,7 @@ int __ip_route_output_key(struct rtable
{
    unsigned hash;
    struct rtable *rth;
+ struct net_namespace *ns = current_net_ns;

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

@@ -2574,6 +2597,7 @@ 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_match(rth->fl.net_ns, ns) &&
        rth->fl.iif == 0 &&
        rth->fl.oif == flp->oif &&
#ifdef CONFIG_IP_ROUTE_FWMARK

```

---

Subject: [PATCH 4/9] network namespaces: socket hashes  
 Posted by [Andrey Savochkin](#) on Tue, 15 Aug 2006 14:48:44 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Socket hash lookups are made within namespace.  
 Hash tables are common for all namespaces, with  
 additional permutation of indexes.

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

---

include/linux/ipv6.h | 3 ++-

```

include/net/inet6_hashtables.h | 6 ++++--
include/net/inet_hashtables.h | 38 ++++++-----
include/net/inet_sock.h | 6 ++++--
include/net/inet_timewait_sock.h | 2 ++
include/net/sock.h | 4 ++++
include/net/udp.h | 12 ++++++---
net/core/sock.c | 5 ++++
net/ipv4/inet_connection_sock.c | 19 ++++++-----
net/ipv4/inet_hashtables.c | 29 ++++++-----
net/ipv4/inet_timewait_sock.c | 8 ++++++--
net/ipv4/raw.c | 2 ++
net/ipv4/udp.c | 20 ++++++-----
net/ipv6/inet6_connection_sock.c | 2 ++
net/ipv6/inet6_hashtables.c | 25 ++++++-----
net/ipv6/raw.c | 4 ++++
net/ipv6/udp.c | 21 ++++++-----
17 files changed, 151 insertions(+), 55 deletions(-)

```

```

--- ./include/linux/ipv6.h.venssock Mon Aug 14 17:02:45 2006
+++ ./include/linux/ipv6.h Tue Aug 15 13:38:47 2006
@@ -428,10 +428,11 @@ static inline struct raw6_sock *raw6_sk(
#define inet_v6_ip6only(__sk) 0
#endif /* defined(CONFIG_IPV6) || defined(CONFIG_IPV6_MODULE) */

-#define INET6_MATCH(__sk, __hash, __saddr, __daddr, __ports, __dif)\
+#define INET6_MATCH(__sk, __hash, __saddr, __daddr, __ports, __dif, __ns)\
  (((__sk)->sk_hash == (__hash)) && \
   ((*((__u32 *)&(inet_sk(__sk)->dport))) == (__ports)) && \
   ((__sk)->sk_family == AF_INET6) && \
+ net_ns_match((__sk)->sk_net_ns, __ns) && \
   ipv6_addr_equal(&inet6_sk(__sk)->daddr, (__saddr)) && \
   ipv6_addr_equal(&inet6_sk(__sk)->rcv_saddr, (__daddr)) && \
   (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
--- ./include/net/inet6_hashtables.h.venssock Mon Aug 14 17:02:47 2006
+++ ./include/net/inet6_hashtables.h Tue Aug 15 13:38:47 2006
@@ -26,11 +26,13 @@ struct inet_hashinfo;

```

```

/* I have no idea if this is a good hash for v6 or not. -DaveM */
static inline unsigned int inet6_eshashfn(const struct in6_addr *laddr, const u16 lport,
- const struct in6_addr *faddr, const u16 fport)
+ const struct in6_addr *faddr, const u16 fport,
+ struct net_namespace *ns)
{
    unsigned int hashent = (lport ^ fport);

    hashent ^= (laddr->s6_addr32[3] ^ faddr->s6_addr32[3]);
+ hashent ^= net_ns_hash(ns);
    hashent ^= hashent >> 16;

```

```

    hashent ^= hashent >> 8;
    return hashent;
@@ -44,7 +46,7 @@ static inline int inet6_sk_ehashfn(const
    const struct in6_addr *faddr = &np->daddr;
    const __u16 lport = inet->num;
    const __u16 fport = inet->dport;
- return inet6_ehashfn(laddr, lport, faddr, fport);
+ return inet6_ehashfn(laddr, lport, faddr, fport, current_net_ns);
}

extern void __inet6_hash(struct inet_hashinfo *hashinfo, struct sock *sk);
--- ./include/net/inet_hashtables.h.venussock Mon Aug 14 17:04:04 2006
+++ ./include/net/inet_hashtables.h Tue Aug 15 13:38:47 2006
@@ -74,6 +74,9 @@ struct inet_ehash_bucket {
    * ports are created in O(1) time? I thought so. ;-) -DaveM
    */
    struct inet_bind_bucket {
+ #ifdef CONFIG_NET_NS
+ struct net_namespace *net_ns;
+ #endif
    unsigned short port;
    signed short fastreuse;
    struct hlist_node node;
@@ -142,30 +145,34 @@ extern struct inet_bind_bucket *
extern void inet_bind_bucket_destroy(kmem_cache_t *cachep,
    struct inet_bind_bucket *tb);

-static inline int inet_bhashfn(const __u16 lport, const int bhash_size)
+static inline int inet_bhashfn(const __u16 lport,
+    struct net_namespace *ns,
+    const int bhash_size)
{
- return lport & (bhash_size - 1);
+ return (lport ^ net_ns_hash(ns)) & (bhash_size - 1);
}

extern void inet_bind_hash(struct sock *sk, struct inet_bind_bucket *tb,
    const unsigned short snum);

/* These can have wildcards, don't try too hard. */
-static inline int inet_lhashfn(const unsigned short num)
+static inline int inet_lhashfn(const unsigned short num,
+    struct net_namespace *ns)
{
- return num & (INET_LHTABLE_SIZE - 1);
+ return (num ^ net_ns_hash(ns)) & (INET_LHTABLE_SIZE - 1);
}

```

```
static inline int inet_sk_listen_hashfn(const struct sock *sk)
{
- return inet_lhashfn(inet_sk(sk)->num);
+ return inet_lhashfn(inet_sk(sk)->num, current_net_ns);
}
```

```
/* Caller must disable local BH processing. */
static inline void __inet_inherit_port(struct inet_hashinfo *table,
    struct sock *sk, struct sock *child)
{
- const int bhash = inet_bhashfn(inet_sk(child)->num, table->bhash_size);
+ const int bhash = inet_bhashfn(inet_sk(child)->num, current_net_ns,
+ table->bhash_size);
    struct inet_bind_hashbucket *head = &table->bhash[bhash];
    struct inet_bind_bucket *tb;
```

```
@@ -299,29 +306,33 @@ static inline struct sock *inet_lookup_l
#define INET_ADDR_COOKIE(__name, __saddr, __daddr) \
    const __u64 __name = (((__u64)(__daddr)) << 32) | ((__u64)(__saddr));
#endif /* __BIG_ENDIAN */
-#define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif)\
+#define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif, __ns)\
    (((__sk)->sk_hash == (__hash)) && \
    ((*((__u64 *)&(inet_sk(__sk)->daddr))) == (__cookie)) && \
    ((*((__u32 *)&(inet_sk(__sk)->dport))) == (__ports)) && \
+ net_ns_match((__sk)->sk_net_ns, __ns) && \
    (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
-#define INET_TW_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif)\
+#define INET_TW_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif, __ns)\
    (((__sk)->sk_hash == (__hash)) && \
    ((*((__u64 *)&(inet_twsk(__sk)->tw_daddr))) == (__cookie)) && \
    ((*((__u32 *)&(inet_twsk(__sk)->tw_dport))) == (__ports)) && \
+ net_ns_match((__sk)->sk_net_ns, __ns) && \
    (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
#else /* 32-bit arch */
#define INET_ADDR_COOKIE(__name, __saddr, __daddr)
-#define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif) \
+#define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif, __ns) \
    (((__sk)->sk_hash == (__hash)) && \
    (inet_sk(__sk)->daddr == (__saddr)) && \
    (inet_sk(__sk)->rcv_saddr == (__daddr)) && \
    ((*((__u32 *)&(inet_sk(__sk)->dport))) == (__ports)) && \
+ net_ns_match((__sk)->sk_net_ns, __ns) && \
    (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
-#define INET_TW_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif) \
+#define INET_TW_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif, __ns) \
    (((__sk)->sk_hash == (__hash)) && \
    (inet_twsk(__sk)->tw_daddr == (__saddr)) && \
```

```

(inet_twsk(__sk)->tw_rcv_saddr == (__daddr)) && \
((*(__u32 *)&(inet_twsk(__sk)->tw_dport)) == (__ports)) && \
+ net_ns_match((__sk)->sk_net_ns, __ns) && \
(!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif)))
#endif /* 64-bit arch */

```

```

@@ -341,22 +352,23 @@ static inline struct sock *
const __u32 ports = INET_COMBINED_PORTS(sport, hnum);
struct sock *sk;
const struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;
/* Optimize here for direct hit, only listening connections can
 * have wildcards anyways.
 */
- unsigned int hash = inet_ehashfn(daddr, hnum, saddr, sport);
+ unsigned int hash = inet_ehashfn(daddr, hnum, saddr, sport, ns);
struct inet_ehash_bucket *head = inet_ehash_bucket(hashinfo, hash);

```

```

prefetch(head->chain.first);
read_lock(&head->lock);
sk_for_each(sk, node, &head->chain) {
- if (INET_MATCH(sk, hash, acookie, saddr, daddr, ports, dif))
+ if (INET_MATCH(sk, hash, acookie, saddr, daddr, ports, dif, ns))
goto hit; /* You sunk my battleship! */
}

```

```

/* Must check for a TIME_WAIT'er before going to listener hash. */
sk_for_each(sk, node, &(head + hashinfo->ehash_size)->chain) {
- if (INET_TW_MATCH(sk, hash, acookie, saddr, daddr, ports, dif))
+ if (INET_TW_MATCH(sk, hash, acookie, saddr, daddr, ports, dif, ns))
goto hit;
}
sk = NULL;

```

```

--- ./include/net/inet_sock.h.venussock Mon Aug 14 17:04:04 2006
+++ ./include/net/inet_sock.h Tue Aug 15 13:38:47 2006
@@ -168,9 +168,11 @@ static inline void inet_sk_copy_descenda
extern int inet_sk_rebuild_header(struct sock *sk);

```

```

static inline unsigned int inet_ehashfn(const __u32 laddr, const __u16 lport,
- const __u32 faddr, const __u16 fport)
+ const __u32 faddr, const __u16 fport,
+ struct net_namespace *ns)
{
unsigned int h = (laddr ^ lport) ^ (faddr ^ fport);
+ h ^= net_ns_hash(ns);
h ^= h >> 16;
h ^= h >> 8;
return h;

```

```

@@ -184,7 +186,7 @@ static inline int inet_sk_ehashfn(const
    const __u32 faddr = inet->daddr;
    const __u16 fport = inet->dport;

- return inet_ehashfn(laddr, lport, faddr, fport);
+ return inet_ehashfn(laddr, lport, faddr, fport, current_net_ns);
}

#endif /* _INET_SOCKET_H */
--- ./include/net/inet_timewait_sock.h.venssock Mon Aug 14 17:02:47 2006
+++ ./include/net/inet_timewait_sock.h Tue Aug 15 13:38:47 2006
@@ -115,6 +115,7 @@ struct inet_timewait_sock {
    #define tw_refcnt    __tw_common.skc_refcnt
    #define tw_hash      __tw_common.skc_hash
    #define tw_prot      __tw_common.skc_prot
+ #define tw_net_ns     __tw_common.skc_net_ns
    volatile unsigned char tw_substate;
    /* 3 bits hole, try to pack */
    unsigned char tw_rcv_wscale;
@@ -200,6 +201,7 @@ static inline void inet_twsk_put(struct
    printk(KERN_DEBUG "%s timewait_sock %p released\n",
           tw->tw_prot->name, tw);
#endif
+ put_net_ns(tw->tw_net_ns);
    kmem_cache_free(tw->tw_prot->twsk_prot->twsk_slab, tw);
    module_put(owner);
}
--- ./include/net/sock.h.venssock Mon Aug 14 17:04:05 2006
+++ ./include/net/sock.h Tue Aug 15 13:38:47 2006
@@ -118,6 +118,9 @@ struct sock_common {
    atomic_t skc_refcnt;
    unsigned int skc_hash;
    struct proto *skc_prot;
+ #ifdef CONFIG_NET_NS
+ struct net_namespace *skc_net_ns;
+ #endif
};

/**
@@ -194,6 +197,7 @@ struct sock {
    #define sk_refcnt    __sk_common.skc_refcnt
    #define sk_hash      __sk_common.skc_hash
    #define sk_prot      __sk_common.skc_prot
+ #define sk_net_ns     __sk_common.skc_net_ns
    unsigned char sk_shutdown : 2,
        sk_no_check : 2,
        sk_userlocks : 4;
--- ./include/net/udp.h.venssock Mon Mar 20 08:53:29 2006

```

```

+++ ./include/net/udp.h Tue Aug 15 13:38:47 2006
@@ -39,13 +39,19 @@ extern rwlock_t udp_hash_lock;

extern int udp_port_rover;

-static inline int udp_lport_inuse(u16 num)
+static inline int udp_hashfn(u16 num, struct net_namespace *ns)
+{
+ return (num ^ net_ns_hash(ns)) & (UDP_HTABLE_SIZE - 1);
+}
+
+static inline int udp_lport_inuse(u16 num, struct net_namespace *ns)
+{
+ struct sock *sk;
+ struct hlist_node *node;

- sk_for_each(sk, node, &udp_hash[num & (UDP_HTABLE_SIZE - 1)])
- if (inet_sk(sk)->num == num)
+ sk_for_each(sk, node, &udp_hash[udp_hashfn(num, ns)])
+ if (inet_sk(sk)->num == num &&
+     net_ns_match(sk->sk_net_ns, ns))
+     return 1;
+     return 0;
+ }
--- ./net/core/sock.c.vensock Mon Aug 14 17:04:05 2006
+++ ./net/core/sock.c Tue Aug 15 13:38:47 2006
@@ -858,6 +858,9 @@ struct sock *sk_alloc(int family, gfp_t
 */
+ sk->sk_prot = sk->sk_prot_creator = prot;
+ sock_lock_init(sk);
+#ifdef CONFIG_NET_NS
+ sk->sk_net_ns = get_net_ns(current_net_ns);
+#endif
+ }

+ if (security_sk_alloc(sk, family, priority))
@@ -897,6 +900,7 @@ void sk_free(struct sock *sk)
+ __FUNCTION__, atomic_read(&sk->sk_omem_alloc));

+ security_sk_free(sk);
+ put_net_ns(sk->sk_net_ns);
+ if (sk->sk_prot_creator->slab != NULL)
+     kmem_cache_free(sk->sk_prot_creator->slab, sk);
+ else
@@ -932,6 +936,7 @@ struct sock *sk_clone(const struct sock
+ lockdep_set_class(&newsk->sk_callback_lock,
+     af_callback_keys + newsk->sk_family);

```

```

+ (void) get_net_ns(newsk->sk_net_ns);
  newsk->sk_dst_cache = NULL;
  newsk->sk_wmem_queued = 0;
  newsk->sk_forward_alloc = 0;
--- ./net/ipv4/inet_connection_sock.c.venussock Mon Aug 14 17:04:08 2006
+++ ./net/ipv4/inet_connection_sock.c Tue Aug 15 13:38:47 2006
@@ -43,10 +43,12 @@ int inet_csk_bind_conflict(const struct
  struct sock *sk2;
  struct hlist_node *node;
  int reuse = sk->sk_reuse;
+ struct net_namespace *ns = current_net_ns;

  sk_for_each_bound(sk2, node, &tb->owners) {
    if (sk != sk2 &&
        !inet_v6_ipv6only(sk2) &&
+   net_ns_match(sk2->sk_net_ns, ns) &&
        (!sk->sk_bound_dev_if ||
         !sk2->sk_bound_dev_if ||
         sk->sk_bound_dev_if == sk2->sk_bound_dev_if)) {
@@ -75,6 +77,7 @@ int inet_csk_get_port(struct inet_hashin
  struct inet_bind_hashbucket *head;
  struct hlist_node *node;
  struct inet_bind_bucket *tb;
+ struct net_namespace *ns = current_net_ns;
  int ret;

  local_bh_disable();
@@ -85,11 +88,15 @@ int inet_csk_get_port(struct inet_hashin
  int rover = net_random() % (high - low) + low;

  do {
-   head = &hashinfo->bhash[inet_bhashfn(rover, hashinfo->bhash_size)];
+   head = &hashinfo->bhash[inet_bhashfn(rover, ns,
+   hashinfo->bhash_size)];
    spin_lock(&head->lock);
-   inet_bind_bucket_for_each(tb, node, &head->chain)
+   inet_bind_bucket_for_each(tb, node, &head->chain) {
+     if (!net_ns_match(tb->net_ns, ns))
+       continue;
+     if (tb->port == rover)
+       goto next;
+   }
    break;
  next:
    spin_unlock(&head->lock);
@@ -112,11 +119,15 @@ int inet_csk_get_port(struct inet_hashin
  */
  snum = rover;

```

```

    } else {
-   head = &hashinfo->bhash[inet_bhashfn(snum, hashinfo->bhash_size)];
+   head = &hashinfo->bhash[inet_bhashfn(snum, ns,
+   hashinfo->bhash_size)];
    spin_lock(&head->lock);
-   inet_bind_bucket_for_each(tb, node, &head->chain)
+   inet_bind_bucket_for_each(tb, node, &head->chain) {
+   if (!net_ns_match(tb->net_ns, ns))
+   continue;
    if (tb->port == snum)
        goto tb_found;
+   }
    }
    tb = NULL;
    goto tb_not_found;
--- ./net/ipv4/inet_hashtables.c.venussock Mon Aug 14 17:04:08 2006
+++ ./net/ipv4/inet_hashtables.c Tue Aug 15 13:43:20 2006
@@ -36,6 +36,9 @@ struct inet_bind_bucket *inet_bind_bucke
    if (tb != NULL) {
        tb->port = snum;
        tb->fastreuse = 0;
+ifdef CONFIG_NET_NS
+   tb->net_ns = get_net_ns(current_net_ns);
+endif
        INIT_HLIST_HEAD(&tb->owners);
        hlist_add_head(&tb->node, &head->chain);
    }
@@ -49,6 +52,7 @@ void inet_bind_bucket_destroy(kmem_cache
{
    if (hlist_empty(&tb->owners)) {
        __hlist_del(&tb->node);
+   put_net_ns(tb->net_ns);
        kmem_cache_free(cache, tb);
    }
}
@@ -66,7 +70,8 @@ void inet_bind_hash(struct sock *sk, str
*/
static void __inet_put_port(struct inet_hashinfo *hashinfo, struct sock *sk)
{
-   const int bhash = inet_bhashfn(inet_sk(sk)->num, hashinfo->bhash_size);
+   const int bhash = inet_bhashfn(inet_sk(sk)->num, current_net_ns,
+   hashinfo->bhash_size);
    struct inet_bind_hashbucket *head = &hashinfo->bhash[bhash];
    struct inet_bind_bucket *tb;

@@ -130,12 +135,15 @@ static struct sock *inet_lookup_listener
    const int dif)
{

```

```

    struct sock *result = NULL, *sk;
+ struct net_namespace *ns = current_net_ns;
    const struct hlist_node *node;
    int hiscore = -1;

    sk_for_each(sk, node, head) {
        const struct inet_sock *inet = inet_sk(sk);

+ if (!net_ns_match(sk->sk_net_ns, ns))
+ continue;
        if (inet->num == hnum && !ipv6_only_sock(sk)) {
            const __u32 rcv_saddr = inet->rcv_saddr;
            int score = sk->sk_family == PF_INET ? 1 : 0;
@@ -168,14 +176,16 @@ struct sock *__inet_lookup_listener(stru
        {
            struct sock *sk = NULL;
            const struct hlist_head *head;
+ struct net_namespace *ns = current_net_ns;

            read_lock(&hashinfo->lhash_lock);
- head = &hashinfo->listening_hash[inet_lhashfn(hnum)];
+ head = &hashinfo->listening_hash[inet_lhashfn(hnum, ns)];
            if (!hlist_empty(head)) {
                const struct inet_sock *inet = inet_sk((sk = __sk_head(head)));

                if (inet->num == hnum && !sk->sk_node.next &&
                    (!inet->rcv_saddr || inet->rcv_saddr == daddr) &&
+ net_ns_match(sk->sk_net_ns, ns) &&
                    (sk->sk_family == PF_INET || !ipv6_only_sock(sk)) &&
                    !sk->sk_bound_dev_if)
                    goto sherry_cache;
@@ -202,7 +212,8 @@ static int __inet_check_established(stru
            int dif = sk->sk_bound_dev_if;
            INET_ADDR_COOKIE(acookie, saddr, daddr)
            const __u32 ports = INET_COMBINED_PORTS(inet->dport, lport);
- unsigned int hash = inet_ehashfn(daddr, lport, saddr, inet->dport);
+ struct net_namespace *ns = current_net_ns;
+ unsigned int hash = inet_ehashfn(daddr, lport, saddr, inet->dport, ns);
            struct inet_ehash_bucket *head = inet_ehash_bucket(hinfo, hash);
            struct sock *sk2;
            const struct hlist_node *node;
@@ -215,7 +226,7 @@ static int __inet_check_established(stru
            sk_for_each(sk2, node, &(head + hinfo->ehash_size)->chain) {
                tw = inet_twsk(sk2);

- if (INET_TW_MATCH(sk2, hash, acookie, saddr, daddr, ports, dif)) {
+ if (INET_TW_MATCH(sk2, hash, acookie, saddr, daddr, ports, dif, ns)) {
                    if (twsk_unique(sk, sk2, twp))

```

```

    goto unique;
else
@@ -226,7 +237,7 @@ static int __inet_check_established(stru

/* And established part... */
sk_for_each(sk2, node, &head->chain) {
- if (INET_MATCH(sk2, hash, acookie, saddr, daddr, ports, dif))
+ if (INET_MATCH(sk2, hash, acookie, saddr, daddr, ports, dif, ns))
    goto not_unique;
}

@@ -274,6 +285,7 @@ int inet_hash_connect(struct inet_timewa
{
    struct inet_hashinfo *hinfo = death_row->hashinfo;
    const unsigned short snum = inet_sk(sk)->num;
+ struct net_namespace *ns = current_net_ns;
    struct inet_bind_hashbucket *head;
    struct inet_bind_bucket *tb;
    int ret;
@@ -292,7 +304,8 @@ int inet_hash_connect(struct inet_timewa
    local_bh_disable();
    for (i = 1; i <= range; i++) {
        port = low + (i + offset) % range;
-    head = &hinfo->bhash[inet_bhashfn(port, hinfo->bhash_size)];
+    head = &hinfo->bhash[inet_bhashfn(port, ns,
+    hinfo->bhash_size)];
        spin_lock(&head->lock);

/* Does not bother with rcv_saddr checks,
@@ -300,6 +313,8 @@ int inet_hash_connect(struct inet_timewa
    * unique enough.
    */
    inet_bind_bucket_for_each(tb, node, &head->chain) {
+ if (!net_ns_match(tb->net_ns, ns))
+ continue;
        if (tb->port == port) {
            BUG_TRAP(!hlist_empty(&tb->owners));
            if (tb->fastreuse >= 0)
@@ -347,7 +362,7 @@ ok:
        goto out;
    }

- head = &hinfo->bhash[inet_bhashfn(snum, hinfo->bhash_size)];
+ head = &hinfo->bhash[inet_bhashfn(snum, ns, hinfo->bhash_size)];
    tb = inet_csk(sk)->icsk_bind_hash;
    spin_lock_bh(&head->lock);
    if (sk_head(&tb->owners) == sk && !sk->sk_bind_node.next) {
--- ./net/ipv4/inet_timewait_sock.c.venussock Mon Aug 14 17:02:49 2006

```

```

+++ ./net/ipv4/inet_timewait_sock.c Tue Aug 15 13:38:47 2006
@@ -31,7 +31,7 @@ void __inet_twsk_kill(struct inet_timewa
    write_unlock(&thead->lock);

    /* Disassociate with bind bucket. */
- bhead = &hashinfo->bhash[inet_bhashfn(tw->tw_num, hashinfo->bhash_size)];
+ bhead = &hashinfo->bhash[inet_bhashfn(tw->tw_num, current_net_ns, hashinfo->bhash_size)];
    spin_lock(&bhead->lock);
    tb = tw->tw_tb;
    __hlist_del(&tw->tw_bind_node);
@@ -65,7 +65,8 @@ void __inet_twsk_hashdance(struct inet_t
    Note, that any socket with inet->num != 0 MUST be bound in
    binding cache, even if it is closed.
    */
- bhead = &hashinfo->bhash[inet_bhashfn(inet->num, hashinfo->bhash_size)];
+ bhead = &hashinfo->bhash[inet_bhashfn(inet->num, current_net_ns,
+    hashinfo->bhash_size)];
    spin_lock(&bhead->lock);
    tw->tw_tb = icsk->icsk_bind_hash;
    BUG_TRAP(icsk->icsk_bind_hash);
@@ -109,6 +110,9 @@ struct inet_timewait_sock *inet_twsk_all
    tw->tw_hash = sk->sk_hash;
    tw->tw_ipv6only = 0;
    tw->tw_prot = sk->sk_prot_creator;
#ifdef CONFIG_NET_NS
+ tw->tw_net_ns = get_net_ns(current_net_ns);
#endif
    atomic_set(&tw->tw_refcnt, 1);
    inet_twsk_dead_node_init(tw);
    __module_get(tw->tw_prot->owner);
--- ./net/ipv4/raw.c.venssock Mon Aug 14 17:04:10 2006
+++ ./net/ipv4/raw.c Tue Aug 15 13:38:47 2006
@@ -106,6 +106,7 @@ struct sock *__raw_v4_lookup(struct sock
    int dif)
    {
        struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;

        sk_for_each_from(sk, node) {
            struct inet_sock *inet = inet_sk(sk);
@@ -113,6 +114,7 @@ struct sock *__raw_v4_lookup(struct sock
            if (inet->num == num &&
                !(inet->daddr && inet->daddr != raddr) &&
                !(inet->rcv_saddr && inet->rcv_saddr != laddr) &&
+ net_ns_match(sk->sk_net_ns, ns) &&
                !(sk->sk_bound_dev_if && sk->sk_bound_dev_if != dif))
                goto found; /* gotcha */
        }
    }

```

```

--- ./net/ipv4/udp.c.venussock Mon Aug 14 17:04:10 2006
+++ ./net/ipv4/udp.c Tue Aug 15 13:38:47 2006
@@ -125,6 +125,7 @@ static int udp_v4_get_port(struct sock *
{
    struct hlist_node *node;
    struct sock *sk2;
+ struct net_namespace *ns = current_net_ns;
    struct inet_sock *inet = inet_sk(sk);

    write_lock_bh(&udp_hash_lock);
@@ -140,7 +141,7 @@ static int udp_v4_get_port(struct sock *
    struct hlist_head *list;
    int size;

- list = &udp_hash[result & (UDP_HTABLE_SIZE - 1)];
+ list = &udp_hash[udp_hashfn(result, ns)];
    if (hlist_empty(list)) {
        if (result > sysctl_local_port_range[1])
            result = sysctl_local_port_range[0] +
@@ -162,7 +163,7 @@ static int udp_v4_get_port(struct sock *
        result = sysctl_local_port_range[0]
            + ((result - sysctl_local_port_range[0]) &
              (UDP_HTABLE_SIZE - 1));
- if (!udp_lport_inuse(result))
+ if (!udp_lport_inuse(result, ns))
        break;
    }
    if (i >= (1 << 16) / UDP_HTABLE_SIZE)
@@ -170,13 +171,13 @@ static int udp_v4_get_port(struct sock *
gotit:
    udp_port_rover = snum = result;
} else {
- sk_for_each(sk2, node,
-             &udp_hash[snum & (UDP_HTABLE_SIZE - 1)]) {
+ sk_for_each(sk2, node, &udp_hash[udp_hashfn(snum, ns)]) {
    struct inet_sock *inet2 = inet_sk(sk2);

    if (inet2->num == snum &&
        sk2 != sk &&
        !ipv6_only_sock(sk2) &&
+ net_ns_match(sk2->sk_net_ns, ns) &&
        (!sk2->sk_bound_dev_if ||
         !sk->sk_bound_dev_if ||
         sk2->sk_bound_dev_if == sk->sk_bound_dev_if) &&
@@ -189,7 +190,7 @@ static int udp_v4_get_port(struct sock *
}
inet->num = snum;
if (sk_unhashed(sk)) {

```

```

- struct hlist_head *h = &udp_hash[snum & (UDP_HTABLE_SIZE - 1)];
+ struct hlist_head *h = &udp_hash[udp_hashfn(snum, ns)];

    sk_add_node(sk, h);
    sock_prot_inc_use(sk->sk_prot);
@@ -225,12 +226,15 @@ static struct sock *udp_v4_lookup_longwa
{
    struct sock *sk, *result = NULL;
    struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;
    unsigned short hnum = ntohs(dport);
    int badness = -1;

- sk_for_each(sk, node, &udp_hash[hnum & (UDP_HTABLE_SIZE - 1)]) {
+ sk_for_each(sk, node, &udp_hash[udp_hashfn(hnum, ns)]) {
    struct inet_sock *inet = inet_sk(sk);

+ if (!net_ns_match(sk->sk_net_ns, ns))
+ continue;
    if (inet->num == hnum && !ipv6_only_sock(sk)) {
        int score = (sk->sk_family == PF_INET ? 1 : 0);
        if (inet->rcv_saddr) {
@@ -285,6 +289,7 @@ static inline struct sock *udp_v4_mcast_
{
    struct hlist_node *node;
    struct sock *s = sk;
+ struct net_namespace *ns = current_net_ns;
    unsigned short hnum = ntohs(loc_port);

    sk_for_each_from(s, node) {
@@ -295,6 +300,7 @@ static inline struct sock *udp_v4_mcast_
    (inet->dport != rmt_port && inet->dport) ||
    (inet->rcv_saddr && inet->rcv_saddr != loc_addr) ||
    ipv6_only_sock(s) ||
+ !net_ns_match(sk->sk_net_ns, ns) ||
    (s->sk_bound_dev_if && s->sk_bound_dev_if != dif))
        continue;
    if (!ip_mc_sf_allow(s, loc_addr, rmt_addr, dif))
@@ -1063,7 +1069,7 @@ static int udp_v4_mcast_deliver(struct s
int dif;

    read_lock(&udp_hash_lock);
- sk = sk_head(&udp_hash[ntohs(uh->dest) & (UDP_HTABLE_SIZE - 1)]);
+ sk = sk_head(&udp_hash[udp_hashfn(ntohs(uh->dest), current_net_ns)]);
    dif = skb->dev->ifindex;
    sk = udp_v4_mcast_next(sk, uh->dest, daddr, uh->source, saddr, dif);
    if (sk) {
--- ./net/ipv6/inet6_connection_sock.c.venssock Mon Aug 14 17:04:10 2006

```

```

+++ ./net/ipv6/inet6_connection_sock.c Tue Aug 15 13:38:47 2006
@@ -31,10 +31,12 @@ int inet6_csk_bind_conflict(const struct
{
    const struct sock *sk2;
    const struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;

    /* We must walk the whole port owner list in this case. -DaveM */
    sk_for_each_bound(sk2, node, &tb->owners) {
        if (sk != sk2 &&
+         net_ns_match(sk2->sk_net_ns, ns) &&
            (!sk->sk_bound_dev_if ||
             !sk2->sk_bound_dev_if ||
             sk->sk_bound_dev_if == sk2->sk_bound_dev_if) &&
--- ./net/ipv6/inet6_hashtables.c.venssock Mon Aug 14 17:02:50 2006
+++ ./net/ipv6/inet6_hashtables.c Tue Aug 15 13:38:47 2006
@@ -65,17 +65,18 @@ struct sock *__inet6_lookup_established(
    struct sock *sk;
    const struct hlist_node *node;
    const __u32 ports = INET_COMBINED_PORTS(sport, hnum);
+ struct net_namespace *ns = current_net_ns;
    /* Optimize here for direct hit, only listening connections can
     * have wildcards anyways.
     */
- unsigned int hash = inet6_ehashfn(daddr, hnum, saddr, sport);
+ unsigned int hash = inet6_ehashfn(daddr, hnum, saddr, sport, ns);
    struct inet_ehash_bucket *head = inet_ehash_bucket(hashinfo, hash);

    prefetch(head->chain.first);
    read_lock(&head->lock);
    sk_for_each(sk, node, &head->chain) {
        /* For IPV6 do the cheaper port and family tests first. */
- if (INET6_MATCH(sk, hash, saddr, daddr, ports, dif))
+ if (INET6_MATCH(sk, hash, saddr, daddr, ports, dif, ns))
            goto hit; /* You sunk my battleship! */
    }
    /* Must check for a TIME_WAIT'er before going to listener hash. */
@@ -83,6 +84,7 @@ struct sock *__inet6_lookup_established(
    const struct inet_timewait_sock *tw = inet_twsk(sk);

    if((__u32 *)&(tw->tw_dport)) == ports &&
+ net_ns_match(sk->sk_net_ns, ns) &&
        sk->sk_family == PF_INET6) {
        const struct inet6_timewait_sock *tw6 = inet6_twsk(sk);

@@ -107,12 +109,15 @@ struct sock *inet6_lookup_listener(struct
    const unsigned short hnum, const int dif)
{

```

```

    struct sock *sk;
+ struct net_namespace *ns = current_net_ns;
    const struct hlist_node *node;
    struct sock *result = NULL;
    int score, hiscore = 0;

    read_lock(&hashinfo->lhash_lock);
- sk_for_each(sk, node, &hashinfo->listening_hash[inet_lhashfn(hnum)]) {
+ sk_for_each(sk, node, &hashinfo->listening_hash[inet_lhashfn(hnum, ns)]) {
+ if (!net_ns_match(sk->sk_net_ns, ns))
+ continue;
    if (inet_sk(sk)->num == hnum && sk->sk_family == PF_INET6) {
        const struct ipv6_pinfo *np = inet6_sk(sk);

@@ -172,8 +177,9 @@ static int __inet6_check_established(str
    const struct in6_addr *saddr = &np->daddr;
    const int dif = sk->sk_bound_dev_if;
    const u32 ports = INET_COMBINED_PORTS(inet->dport, lport);
+ struct net_namespace *ns = current_net_ns;
    const unsigned int hash = inet6_eshashfn(daddr, inet->num, saddr,
- inet->dport);
+ inet->dport, ns);
    struct inet_eshash_bucket *head = inet_eshash_bucket(hinfo, hash);
    struct sock *sk2;
    const struct hlist_node *node;
@@ -190,6 +196,7 @@ static int __inet6_check_established(str

    if((__u32 *)&(tw->tw_dport)) == ports &&
        sk2->sk_family == PF_INET6 &&
+ net_ns_match(sk2->sk_net_ns, ns) &&
    ipv6_addr_equal(&tw6->tw_v6_daddr, saddr) &&
    ipv6_addr_equal(&tw6->tw_v6_rcv_saddr, daddr) &&
    sk2->sk_bound_dev_if == sk->sk_bound_dev_if) {
@@ -203,7 +210,7 @@ static int __inet6_check_established(str

    /* And established part... */
    sk_for_each(sk2, node, &head->chain) {
- if (INET6_MATCH(sk2, hash, saddr, daddr, ports, dif))
+ if (INET6_MATCH(sk2, hash, saddr, daddr, ports, dif, ns))
        goto not_unique;
    }

@@ -249,6 +256,7 @@ int inet6_hash_connect(struct inet_timew
{
    struct inet_hashinfo *hinfo = death_row->hashinfo;
    const unsigned short snum = inet_sk(sk)->num;
+ struct net_namespace *ns = current_net_ns;
    struct inet_bind_hashbucket *head;

```

```

    struct inet_bind_bucket *tb;
    int ret;
@@ -266,7 +274,8 @@ int inet6_hash_connect(struct inet_timew
    local_bh_disable();
    for (i = 1; i <= range; i++) {
        port = low + (i + offset) % range;
-   head = &hinfo->bhash[inet_bhashfn(port, hinfo->bhash_size)];
+   head = &hinfo->bhash[inet_bhashfn(port, ns,
+   hinfo->bhash_size)];
        spin_lock(&head->lock);

        /* Does not bother with rcv_saddr checks,
@@ -274,6 +283,8 @@ int inet6_hash_connect(struct inet_timew
        * unique enough.
        */
        inet_bind_bucket_for_each(tb, node, &head->chain) {
+   if (!net_ns_match(tb->net_ns, ns))
+   continue;
        if (tb->port == port) {
            BUG_TRAP(!hlist_empty(&tb->owners));
            if (tb->fastreuse >= 0)
@@ -322,7 +333,7 @@ ok:
        goto out;
    }

-   head = &hinfo->bhash[inet_bhashfn(snum, hinfo->bhash_size)];
+   head = &hinfo->bhash[inet_bhashfn(snum, ns, hinfo->bhash_size)];
    tb = inet_csk(sk)->icsk_bind_hash;
    spin_lock_bh(&head->lock);

--- ./net/ipv6/raw.c.venussock Mon Aug 14 17:04:10 2006
+++ ./net/ipv6/raw.c Tue Aug 15 13:38:47 2006
@@ -87,11 +87,15 @@ struct sock *__raw_v6_lookup(struct sock
{
    struct hlist_node *node;
    int is_multicast = ipv6_addr_is_multicast(loc_addr);
+   struct net_namespace *ns = current_net_ns;

    sk_for_each_from(sk, node)
        if (inet_sk(sk)->num == num) {
            struct ipv6_pinfo *np = inet6_sk(sk);

+   if (!net_ns_match(sk->sk_net_ns, ns))
+   continue;
+
            if (!ipv6_addr_any(&np->daddr) &&
                !ipv6_addr_equal(&np->daddr, rmt_addr))
                continue;

```

```

--- ./net/ipv6/udp.c.venussock Mon Aug 14 17:04:10 2006
+++ ./net/ipv6/udp.c Tue Aug 15 13:38:47 2006
@@ -68,6 +68,7 @@ static int udp_v6_get_port(struct sock *
{
    struct sock *sk2;
    struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;

    write_lock_bh(&udp_hash_lock);
    if (snum == 0) {
@@ -82,7 +83,7 @@ static int udp_v6_get_port(struct sock *
    int size;
    struct hlist_head *list;

- list = &udp_hash[result & (UDP_HTABLE_SIZE - 1)];
+ list = &udp_hash[udp_hashfn(result, ns)];
    if (hlist_empty(list)) {
        if (result > sysctl_local_port_range[1])
            result = sysctl_local_port_range[0] +
@@ -104,7 +105,7 @@ static int udp_v6_get_port(struct sock *
        result = sysctl_local_port_range[0]
            + ((result - sysctl_local_port_range[0]) &
              (UDP_HTABLE_SIZE - 1));
- if (!udp_lport_inuse(result))
+ if (!udp_lport_inuse(result, ns))
        break;
    }
    if (i >= (1 << 16) / UDP_HTABLE_SIZE)
@@ -112,10 +113,10 @@ static int udp_v6_get_port(struct sock *
gotit:
    udp_port_rover = snum = result;
} else {
- sk_for_each(sk2, node,
-             &udp_hash[snum & (UDP_HTABLE_SIZE - 1)]) {
+ sk_for_each(sk2, node, &udp_hash[udp_hashfn(snum, ns)]) {
    if (inet_sk(sk2)->num == snum &&
        sk2 != sk &&
+ net_ns_match(sk2->sk_net_ns, ns) &&
        (!sk2->sk_bound_dev_if ||
         !sk->sk_bound_dev_if ||
         sk2->sk_bound_dev_if == sk->sk_bound_dev_if) &&
@@ -127,7 +128,7 @@ gotit:

    inet_sk(sk)->num = snum;
    if (sk_unhashed(sk)) {
- sk_add_node(sk, &udp_hash[snum & (UDP_HTABLE_SIZE - 1)]);
+ sk_add_node(sk, &udp_hash[udp_hashfn(snum, ns)]);
    sock_prot_inc_use(sk->sk_prot);

```

```

}
write_unlock_bh(&udp_hash_lock);
@@ -158,13 +159,16 @@ static struct sock *udp_v6_lookup(struct
{
    struct sock *sk, *result = NULL;
    struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;
    unsigned short hnum = ntohs(dport);
    int badness = -1;

    read_lock(&udp_hash_lock);
- sk_for_each(sk, node, &udp_hash[hnum & (UDP_HTABLE_SIZE - 1)]) {
+ sk_for_each(sk, node, &udp_hash[udp_hashfn(hnum, ns)]) {
    struct inet_sock *inet = inet_sk(sk);

+ if (!net_ns_match(sk->sk_net_ns, ns))
+ continue;
    if (inet->num == hnum && sk->sk_family == PF_INET6) {
        struct ipv6_pinfo *np = inet6_sk(sk);
        int score = 0;
@@ -372,6 +376,7 @@ static struct sock *udp_v6_mcast_next(st
{
    struct hlist_node *node;
    struct sock *s = sk;
+ struct net_namespace *ns = current_net_ns;
    unsigned short num = ntohs(loc_port);

    sk_for_each_from(s, node) {
@@ -383,6 +388,8 @@ static struct sock *udp_v6_mcast_next(st
    if (inet->dport != rmt_port)
        continue;
    }
+ if (!net_ns_match(sk->sk_net_ns, ns))
+ continue;
    if (!ipv6_addr_any(&np->daddr) &&
        !ipv6_addr_equal(&np->daddr, rmt_addr))
        continue;
@@ -414,7 +421,7 @@ static void udpv6_mcast_deliver(struct u
int dif;

    read_lock(&udp_hash_lock);
- sk = sk_head(&udp_hash[ntohs(uh->dest) & (UDP_HTABLE_SIZE - 1)]);
+ sk = sk_head(&udp_hash[udp_hashfn(uh->dest, current_net_ns)]);
    dif = skb->dev->ifindex;
    sk = udp_v6_mcast_next(sk, uh->dest, daddr, uh->source, saddr, dif);
    if (!sk) {

```

---



---

Subject: [PATCH 5/9] network namespaces: async socket operations  
Posted by [Andrey Savochkin](#) on Tue, 15 Aug 2006 14:48:45 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Non-trivial part of socket namespaces: asynchronous events  
should be run in proper context.

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

---

```
af_inet.c      | 10 ++++++++
inet_timewait_sock.c | 8 ++++++
tcp_timer.c    | 9 ++++++++
3 files changed, 27 insertions(+)
```

--- ./net/ipv4/af\_inet.c.vensock-asyn Mon Aug 14 17:04:07 2006

+++ ./net/ipv4/af\_inet.c Tue Aug 15 13:45:44 2006

@@ -366,10 +366,17 @@ out\_rcu\_unlock:

```
int inet_release(struct socket *sock)
{
    struct sock *sk = sock->sk;
+   struct net_namespace *ns, *orig_net_ns;
```

```
    if (sk) {
        long timeout;
```

```
+   /* Need to change context here since protocol ->close
+    * operation may send packets.
+    */
```

```
+   ns = get_net_ns(sk->sk_net_ns);
+   push_net_ns(ns, orig_net_ns);
+   /* Applications forget to leave groups before exiting */
+   ip_mc_drop_socket(sk);
```

@@@ -386,6 +393,9 @@@ int inet\_release(struct socket \*sock)

```
    timeout = sk->sk_lingertime;
    sock->sk = NULL;
    sk->sk_prot->close(sk, timeout);
```

```
+
+   pop_net_ns(orig_net_ns);
+   put_net_ns(ns);
+   }
+   return 0;
+   }
```

--- ./net/ipv4/inet\_timewait\_sock.c.vensock-asyn Tue Aug 15 13:45:44 2006

+++ ./net/ipv4/inet\_timewait\_sock.c Tue Aug 15 13:45:44 2006

@@@ -129,6 +129,7 @@@ static int inet\_twdr\_do\_twkill\_work(stru

```
{
    struct inet_timewait_sock *tw;
```

```

    struct hlist_node *node;
+ struct net_namespace *orig_net_ns;
    unsigned int killed;
    int ret;

@@ -140,8 +141,10 @@ static int inet_twdr_do_twkill_work(stru
    */
    killed = 0;
    ret = 0;
+ push_net_ns(current_net_ns, orig_net_ns);
    rescan:
    inet_twsd_for_each_inmate(tw, node, &twdr->cells[slot]) {
+ switch_net_ns(tw->tw_net_ns);
    __inet_twsd_del_dead_node(tw);
    spin_unlock(&twdr->death_lock);
    __inet_twsd_kill(tw, twdr->hashinfo);
@@ -164,6 +167,7 @@ rescan:

    twdr->tw_count -= killed;
    NET_ADD_STATS_BH(LINUX_MIB_TIMEWAITED, killed);
+ pop_net_ns(orig_net_ns);

    return ret;
}
@@ -338,10 +342,12 @@ void inet_twdr_twcal_tick(unsigned long
    int n, slot;
    unsigned long j;
    unsigned long now = jiffies;
+ struct net_namespace *orig_net_ns;
    int killed = 0;
    int adv = 0;

    twdr = (struct inet_timewait_death_row *)data;
+ push_net_ns(current_net_ns, orig_net_ns);

    spin_lock(&twdr->death_lock);
    if (twdr->twcal_hand < 0)
@@ -357,6 +363,7 @@ void inet_twdr_twcal_tick(unsigned long

    inet_twsd_for_each_inmate_safe(tw, node, safe,
        &twdr->twcal_row[slot]) {
+ switch_net_ns(tw->tw_net_ns);
    __inet_twsd_del_dead_node(tw);
    __inet_twsd_kill(tw, twdr->hashinfo);
    inet_twsd_put(tw);
@@ -384,6 +391,7 @@ out:
    del_timer(&twdr->tw_timer);
    NET_ADD_STATS_BH(LINUX_MIB_TIMEWAITKILLED, killed);

```

```

    spin_unlock(&twdr->death_lock);
+ pop_net_ns(orig_net_ns);
}

EXPORT_SYMBOL_GPL(inet_twdr_twcal_tick);
--- ./net/ipv4/tcp_timer.c.venussock-asyn Mon Aug 14 16:43:51 2006
+++ ./net/ipv4/tcp_timer.c Tue Aug 15 13:45:44 2006
@@ -171,7 +171,9 @@ static void tcp_delack_timer(unsigned lo
    struct sock *sk = (struct sock*)data;
    struct tcp_sock *tp = tcp_sk(sk);
    struct inet_connection_sock *icsk = inet_csk(sk);
+ struct net_namespace *orig_net_ns;

+ push_net_ns(sk->sk_net_ns, orig_net_ns);
    bh_lock_sock(sk);
    if (sock_owned_by_user(sk)) {
        /* Try again later. */
@@ -225,6 +227,7 @@ out:
    out_unlock:
        bh_unlock_sock(sk);
        sock_put(sk);
+ pop_net_ns(orig_net_ns);
    }

static void tcp_probe_timer(struct sock *sk)
@@ -384,8 +387,10 @@ static void tcp_write_timer(unsigned lon
{
    struct sock *sk = (struct sock*)data;
    struct inet_connection_sock *icsk = inet_csk(sk);
+ struct net_namespace *orig_net_ns;
    int event;

+ push_net_ns(sk->sk_net_ns, orig_net_ns);
    bh_lock_sock(sk);
    if (sock_owned_by_user(sk)) {
        /* Try again later */
@@ -419,6 +424,7 @@ out:
    out_unlock:
        bh_unlock_sock(sk);
        sock_put(sk);
+ pop_net_ns(orig_net_ns);
    }

/*
@@ -447,9 +453,11 @@ static void tcp_keepalive_timer (unsigne
{
    struct sock *sk = (struct sock *) data;
    struct inet_connection_sock *icsk = inet_csk(sk);

```

```

+ struct net_namespace *orig_net_ns;
  struct tcp_sock *tp = tcp_sk(sk);
  __u32 elapsed;

+ push_net_ns(sk->sk_net_ns, orig_net_ns);
  /* Only process if socket is not in use. */
  bh_lock_sock(sk);
  if (sock_owned_by_user(sk)) {
@@ -521,4 +529,5 @@ death:
out:
  bh_unlock_sock(sk);
  sock_put(sk);
+ put_net_ns(orig_net_ns);
  }

```

---

Subject: [PATCH 6/9] allow proc\_dir\_entries to have destructor  
 Posted by [Andrey Savochkin](#) on Tue, 15 Aug 2006 14:48:46 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Destructor field added proc\_dir\_entries,  
 standard destructor kfree'ing data introduced.

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

---

```

fs/proc/generic.c      | 10 ++++++---
fs/proc/root.c         |  1 +
include/linux/proc_fs.h |  4 ++++
3 files changed, 13 insertions(+), 2 deletions(-)

```

```

--- ./fs/proc/generic.c.veprocdtor Mon Aug 14 16:43:41 2006
+++ ./fs/proc/generic.c Tue Aug 15 13:45:51 2006
@@ -608,6 +608,11 @@ static struct proc_dir_entry *proc_creat
  return ent;
  }

```

```

+void proc_data_destructor(struct proc_dir_entry *ent)
+{
+ kfree(ent->data);
+}
+
struct proc_dir_entry *proc_symlink(const char *name,
  struct proc_dir_entry *parent, const char *dest)
{
@@ -620,6 +625,7 @@ struct proc_dir_entry *proc_symlink(const
  ent->data = kmalloc((ent->size=strlen(dest))+1, GFP_KERNEL);
  if (ent->data) {
    strcpy((char*)ent->data, dest);
  }

```

```

+ ent->destructor = proc_data_destructor;
  if (proc_register(parent, ent) < 0) {
    kfree(ent->data);
    kfree(ent);
@@ -698,8 +704,8 @@ void free_proc_entry(struct proc_dir_entry

  release_inode_number(ino);

- if (S_ISLNK(de->mode) && de->data)
- kfree(de->data);
+ if (de->destructor)
+ de->destructor(de);
  kfree(de);
}

--- ./fs/proc/root.c.veprocdtor Mon Aug 14 17:02:38 2006
+++ ./fs/proc/root.c Tue Aug 15 13:45:51 2006
@@ -154,6 +154,7 @@ EXPORT_SYMBOL(proc_symlink);
EXPORT_SYMBOL(proc_mkdir);
EXPORT_SYMBOL(create_proc_entry);
EXPORT_SYMBOL(remove_proc_entry);
+EXPORT_SYMBOL(proc_data_destructor);
EXPORT_SYMBOL(proc_root);
EXPORT_SYMBOL(proc_root_fs);
EXPORT_SYMBOL(proc_net);
--- ./include/linux/proc_fs.h.veprocdtor Mon Aug 14 17:02:47 2006
+++ ./include/linux/proc_fs.h Tue Aug 15 13:45:51 2006
@@ -46,6 +46,8 @@ typedef int (read_proc_t)(char *page, ch
typedef int (write_proc_t)(struct file *file, const char __user *buffer,
    unsigned long count, void *data);
typedef int (get_info_t)(char *, char **, off_t, int);
+struct proc_dir_entry;
+typedef void (destroy_proc_t)(struct proc_dir_entry *);

struct proc_dir_entry {
  unsigned int low_ino;
@@ -65,6 +67,7 @@ struct proc_dir_entry {
  read_proc_t *read_proc;
  write_proc_t *write_proc;
  atomic_t count; /* use count */
+ destroy_proc_t *destructor;
  int deleted; /* delete flag */
  void *set;
};
@@ -109,6 +112,7 @@ char *task_mem(struct mm_struct *, char
extern struct proc_dir_entry *create_proc_entry(const char *name, mode_t mode,
    struct proc_dir_entry *parent);
extern void remove_proc_entry(const char *name, struct proc_dir_entry *parent);

```

```
+extern void proc_data_destructor(struct proc_dir_entry *);
```

```
extern struct vfsmount *proc_mnt;
```

```
extern int proc_fill_super(struct super_block *,void *,int);
```

---

---

Subject: [PATCH 7/9] net\_device seq\_file

Posted by [Andrey Savochkin](#) on Tue, 15 Aug 2006 14:48:47 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Library function to create a seq\_file in proc filesystem,  
showing some information for each netdevice.

This code is present in the kernel in about 10 instances, and  
all of them can be converted to using introduced library function.

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

---

```
include/linux/netdevice.h | 7 +++
```

```
net/core/dev.c | 96 ++++++++++++++++++++++++++++++++++++++
```

```
2 files changed, 103 insertions(+)
```

--- ./include/linux/netdevice.h.venetproc Tue Aug 15 13:46:08 2006

+++ ./include/linux/netdevice.h Tue Aug 15 13:46:08 2006

@@ -592,6 +592,13 @@ extern int register\_netdevice(struct ne

```
extern int unregister_netdevice(struct net_device *dev);
```

```
extern void free_netdev(struct net_device *dev);
```

```
extern void synchronize_net(void);
```

```
+#ifdef CONFIG_PROC_FS
```

```
+extern int netdev_proc_create(char *name,
```

```
+ int (*show)(struct seq_file *,
```

```
+ struct net_device *, void *),
```

```
+ void *data, struct module *mod);
```

```
+void netdev_proc_remove(char *name);
```

```
+#endif
```

```
extern int register_netdevice_notifier(struct notifier_block *nb);
```

```
extern int unregister_netdevice_notifier(struct notifier_block *nb);
```

```
extern int call_netdevice_notifiers(unsigned long val, void *v);
```

--- ./net/core/dev.c.venetproc Tue Aug 15 13:46:08 2006

+++ ./net/core/dev.c Tue Aug 15 13:46:08 2006

@@ -2100,6 +2100,102 @@ static int dev\_ifconf(char \_\_user \*arg)

```
}
```

```
#ifdef CONFIG_PROC_FS
```

```
+
```

```
+struct netdev_proc_data {
```

```
+ struct file_operations fops;
```

```
+ int (*show)(struct seq_file *, struct net_device *, void *);
```

```
+ void *data;
```

```

+};
+
+static void *netdev_proc_seq_start(struct seq_file *seq, loff_t *pos)
+{
+ struct net_device *dev;
+ loff_t off;
+
+ read_lock(&dev_base_lock);
+ if (*pos == 0)
+ return SEQ_START_TOKEN;
+ for (dev = dev_base, off = 1; dev = dev->next, off++) {
+ if (*pos == off)
+ return dev;
+ }
+ return NULL;
+}
+
+static void *netdev_proc_seq_next(struct seq_file *seq, void *v, loff_t *pos)
+{
+ ++*pos;
+ return (v == SEQ_START_TOKEN) ? dev_base
+ : ((struct net_device *)v)->next;
+}
+
+static void netdev_proc_seq_stop(struct seq_file *seq, void *v)
+{
+ read_unlock(&dev_base_lock);
+}
+
+static int netdev_proc_seq_show(struct seq_file *seq, void *v)
+{
+ struct netdev_proc_data *p;
+
+ p = seq->private;
+ return (*p->show)(seq, v, p->data);
+}
+
+static struct seq_operations netdev_proc_seq_ops = {
+ .start = netdev_proc_seq_start,
+ .next = netdev_proc_seq_next,
+ .stop = netdev_proc_seq_stop,
+ .show = netdev_proc_seq_show,
+};
+
+static int netdev_proc_open(struct inode *inode, struct file *file)
+{
+ int err;
+ struct seq_file *p;

```

```

+
+ err = seq_open(file, &netdev_proc_seq_ops);
+ if (!err) {
+   p = file->private_data;
+   p->private = (struct netdev_proc_data *)PDE(inode)->data;
+ }
+ return err;
+}
+
+int netdev_proc_create(char *name,
+ int (*show)(struct seq_file *, struct net_device *, void *),
+ void *data, struct module *mod)
+{
+ struct netdev_proc_data *p;
+ struct proc_dir_entry *ent;
+
+ p = kzalloc(sizeof(*p), GFP_KERNEL);
+ p->fops.owner = mod;
+ p->fops.open = netdev_proc_open;
+ p->fops.read = seq_read;
+ p->fops.llseek = seq_lseek;
+ p->fops.release = seq_release;
+ p->show = show;
+ p->data = data;
+ ent = create_proc_entry(name, S_IRUGO, proc_net);
+ if (ent == NULL) {
+   kfree(p);
+   return -EINVAL;
+ }
+ ent->data = p;
+ ent->destructor = proc_data_destructor;
+ smp_wmb();
+ ent->proc_fops = &p->fops;
+ return 0;
+}
+EXPORT_SYMBOL(netdev_proc_create);
+
+void netdev_proc_remove(char *name)
+{
+ proc_net_remove(name);
+}
+EXPORT_SYMBOL(netdev_proc_remove);
+
+/*
+ * This is invoked by the /proc filesystem handler to display a device
+ * in detail.

```

---



---

Subject: [PATCH 8/9] network namespaces: device to pass packets between namespaces

Posted by [Andrey Savochkin](#) on Tue, 15 Aug 2006 14:48:48 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

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

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

---

```
Makefile | 3
veth.c | 327 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++
2 files changed, 330 insertions(+)
```

--- ./drivers/net/Makefile.veveth Mon Aug 14 17:03:45 2006

+++ ./drivers/net/Makefile Tue Aug 15 13:46:15 2006

@@ -124,6 +124,9 @@ obj-\$(CONFIG\_SLIP) += slip.o

obj-\$(CONFIG\_SLHC) += slhc.o

obj-\$(CONFIG\_DUMMY) += dummy.o

+ifeq (\$(CONFIG\_NET\_NS),y)

+obj-m += veth.o

+endif

obj-\$(CONFIG\_IFB) += ifb.o

obj-\$(CONFIG\_DE600) += de600.o

obj-\$(CONFIG\_DE620) += de620.o

--- ./drivers/net/veth.c.veveth Tue Aug 15 13:44:46 2006

+++ ./drivers/net/veth.c Tue Aug 15 13:46:15 2006

@@ -0,0 +1,327 @@

+/\*

+ \* Copyright (C) 2006 SWsoft

+ \*

+ \* Written by Andrey Savochkin <saw@sw.ru>,

+ \* reusing code by Andrey Mirkin <amirkin@sw.ru>.

+ \*/

+#include <linux/list.h>

+#include <linux/spinlock.h>

+#include <linux/ctype.h>

+#include <asm/semaphore.h>

+#include <linux/netdevice.h>

+#include <linux/etherdevice.h>

+#include <linux/proc\_fs.h>

+#include <linux/seq\_file.h>

+#include <net/dst.h>

+#include <net/xfrm.h>

+

+struct veth\_struct

+{

+ struct net\_device \*pair;

```

+ struct net_device_stats stats;
+};
+
+#define veth_from_netdev(dev) ((struct veth_struct *)(netdev_priv(dev)))
+
+/* ----- *
+ *
+ * Device functions
+ *
+ * ----- */
+
+static struct net_device_stats *get_stats(struct net_device *dev);
+static int veth_xmit(struct sk_buff *skb, struct net_device *dev)
+{
+ struct net_device_stats *stats;
+ struct veth_struct *entry;
+ struct net_device *rcv;
+ struct net_namespace *orig_net_ns;
+ int length;
+
+ stats = get_stats(dev);
+ entry = veth_from_netdev(dev);
+ rcv = entry->pair;
+
+ if (!(rcv->flags & IFF_UP))
+ /* Target namespace does not want to receive packets */
+ goto outf;
+
+ dst_release(skb->dst);
+ skb->dst = NULL;
+ secpath_reset(skb);
+ skb_orphan(skb);
+#ifdef CONFIG_NETFILTER
+ nf_conntrack_put(skb->nfct);
+#if defined(CONFIG_NF_CONNTRACK) || defined(CONFIG_NF_CONNTRACK_MODULE)
+ nf_conntrack_put_reasm(skb->nfct_reasm);
+#endif
+#ifdef CONFIG_BRIDGE_NETFILTER
+ nf_bridge_put(skb->nf_bridge);
+#endif
+#endif
+
+ push_net_ns(rcv->net_ns, orig_net_ns);
+ skb->dev = rcv;
+ skb->pkt_type = PACKET_HOST;
+ skb->protocol = eth_type_trans(skb, rcv);
+
+ length = skb->len;

```

```

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

```

```

+}
+
+static void veth_setup(struct net_device *dev)
+{
+ dev->init = veth_init_dev;
+}
+
+static inline int is_veth_dev(struct net_device *dev)
+{
+ return dev->init == veth_init_dev;
+}
+
+/* ----- *
+ *
+ * Management interface
+ *
+ * ----- */
+
+struct net_device *veth_dev_alloc(char *name, char *addr)
+{
+ struct net_device *dev;
+
+ dev = alloc_netdev(sizeof(struct veth_struct), name, veth_setup);
+ if (dev != NULL) {
+ memcpy(dev->dev_addr, addr, ETH_ALEN);
+ dev->addr_len = ETH_ALEN;
+ }
+ return dev;
+}
+
+int veth_entry_add(char *parent_name, char *parent_addr,
+ char *child_name, char *child_addr,
+ struct net_namespace *child_ns)
+{
+ struct net_device *parent_dev, *child_dev;
+ struct net_namespace *parent_ns;
+ int err;
+
+ err = -ENOMEM;
+ if ((parent_dev = veth_dev_alloc(parent_name, parent_addr)) == NULL)
+ goto out_alocp;
+ if ((child_dev = veth_dev_alloc(child_name, child_addr)) == NULL)
+ goto out_alocc;
+ veth_from_netdev(parent_dev)->pair = child_dev;
+ veth_from_netdev(child_dev)->pair = parent_dev;
+
+ /*
+ * About serialization, see comments to veth_pair_del().

```

```

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

```

```

+ unregister_netdevice(child_dev);
+ pop_net_ns(parent_ns);
+
+ put_net_ns(child_ns);
+}
+
+int veth_entry_del(char *parent_name)
+{
+ struct net_device *dev;
+
+ if ((dev = dev_get_by_name(parent_name)) == NULL)
+ return -ENODEV;
+
+ rtnl_lock();
+ veth_pair_del(dev);
+ dev_put(dev);
+ rtnl_unlock();
+
+ return 0;
+}
+
+void veth_entry_del_all(void)
+{
+ struct net_device **p, *dev;
+
+ rtnl_lock();
+ for (p = &dev_base; (dev = *p) != NULL; ) {
+ if (!is_veth_dev(dev)) {
+ p = &dev->next;
+ continue;
+ }
+
+ dev_hold(dev);
+ veth_pair_del(dev);
+ dev_put(dev);
+ }
+ rtnl_unlock();
+}
+
+/* ----- */
+ *
+ * Information in proc
+ *
+ * ----- */
+
+#ifdef CONFIG_PROC_FS
+
+#define ADDR_FMT "%02x:%02x:%02x:%02x:%02x:%02x"

```



```
+ * ----- */
+
+int __init veth_init(void)
+{
+ veth_proc_create();
+ return 0;
+}
+
+void __exit veth_exit(void)
+{
+ veth_proc_remove();
+ veth_entry_del_all();
+}
+
+module_init(veth_init)
+module_exit(veth_exit)
+
+MODULE_DESCRIPTION("Virtual Ethernet Device");
+MODULE_LICENSE("GPL v2");
```

---

Subject: [PATCH 9/9] network namespaces: playing with pass-through device

Posted by [Andrey Savochkin](#) on Tue, 15 Aug 2006 14:48:49 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Temporary code to debug and play with pass-through device.

Create device pair by

```
modprobe veth
```

```
echo 'add veth1 0:1:2:3:4:1 eth0 0:1:2:3:4:2' >/proc/net/veth_ctl
```

and your shell will appear into a new namespace with `eth0' device.

Configure device in this namespace

```
ip l s eth0 up
```

```
ip a a 1.2.3.4/24 dev eth0
```

and in the root namespace

```
ip l s veth1 up
```

```
ip a a 1.2.3.1/24 dev veth1
```

to establish a communication channel between root namespace and the newly created one.

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

---

```
veth.c | 113 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
+++++
```

```
1 files changed, 113 insertions(+)
```

```
--- ./drivers/net/veth.c.veveth-dbg Tue Aug 15 13:47:48 2006
```

```
+++ ./drivers/net/veth.c Tue Aug 15 14:08:04 2006
```

```
@@ -251,6 +251,116 @@ void veth_entry_del_all(void)
```

```

/* ----- *
 *
+ * Temporary interface to create veth devices
+ *
+ * ----- */
+
+ #ifdef CONFIG_PROC_FS
+
+ static int veth_debug_open(struct inode *inode, struct file *file)
+ {
+     return 0;
+ }
+
+ static char *parse_addr(char *s, char *addr)
+ {
+     int i, v;
+
+     for (i = 0; i < ETH_ALEN; i++) {
+         if (!isxdigit(*s))
+             return NULL;
+         *addr = 0;
+         v = isdigit(*s) ? *s - '0' : toupper(*s) - 'A' + 10;
+         s++;
+         if (isxdigit(*s)) {
+             *addr += v << 16;
+             v = isdigit(*s) ? *s - '0' : toupper(*s) - 'A' + 10;
+             s++;
+         }
+         *addr++ += v;
+         if (i < ETH_ALEN - 1 && ispunct(*s))
+             s++;
+     }
+     return s;
+ }
+
+ extern int net_ns_start(void);
+ static ssize_t veth_debug_write(struct file *file, const char __user *user_buf,
+     size_t size, loff_t *ppos)
+ {
+     char buf[128], *s, *parent_name, *child_name;
+     char parent_addr[ETH_ALEN], child_addr[ETH_ALEN];
+     struct net_namespace *parent_ns, *child_ns;
+     int err;
+
+     s = buf;
+     err = -EINVAL;
+     if (size >= sizeof(buf))

```

```

+ goto out;
+ err = -EFAULT;
+ if (copy_from_user(buf, user_buf, size))
+ goto out;
+ buf[size] = 0;
+
+ err = -EBADRQC;
+ if (!strncmp(buf, "add ", 4)) {
+ parent_name = buf + 4;
+ if ((s = strchr(parent_name, ' ')) == NULL)
+ goto out;
+ *s = 0;
+ if ((s = parse_addr(s + 1, parent_addr)) == NULL)
+ goto out;
+ if (!*s)
+ goto out;
+ child_name = s + 1;
+ if ((s = strchr(child_name, ' ')) == NULL)
+ goto out;
+ *s = 0;
+ if ((s = parse_addr(s + 1, child_addr)) == NULL)
+ goto out;
+
+ parent_ns = get_net_ns(current_net_ns);
+ err = net_ns_start();
+ if (err)
+ goto out;
+ /* return to parent context */
+ push_net_ns(parent_ns, child_ns);
+ err = veth_entry_add(parent_name, parent_addr,
+ child_name, child_addr, child_ns);
+ pop_net_ns(child_ns);
+ put_net_ns(parent_ns);
+ if (!err)
+ err = size;
+ }
+out:
+ return err;
+}
+
+static struct file_operations veth_debug_ops = {
+ .open = &veth_debug_open,
+ .write = &veth_debug_write,
+};
+
+static int veth_debug_create(void)
+{
+ proc_net_fops_create("veth_ctl", 0200, &veth_debug_ops);

```

```

+ return 0;
+}
+
+static void veth_debug_remove(void)
+{
+ proc_net_remove("veth_ctl");
+}
+
+
+##else
+
+static int veth_debug_create(void) { return -1; }
+static void veth_debug_remove(void) { }
+
+##endif
+
+/* ----- */
+ *
+ * Information in proc
+ *
+ * ----- */
@@ -310,12 +420,15 @@ static inline void veth_proc_remove(void

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

void __exit veth_exit(void)
{
+ veth_debug_remove();
  veth_proc_remove();
  veth_entry_del_all();
}

```

---

Subject: Re: [RFC] network namespaces  
 Posted by [serue](#) on Wed, 16 Aug 2006 11:53:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Quoting Andrey Savochkin (saw@sw.ru):

> Hi All,

>

> I'd like to resurrect our discussion about network namespaces.

> In our previous discussions it appeared that we have rather polar concepts

> which seemed hard to reconcile.

- > Now I have an idea how to look at all discussed concepts to enable everyone's
- > usage scenario.
- >
- > 1. The most straightforward concept is complete separation of namespaces,
- > covering device list, routing tables, netfilter tables, socket hashes, and
- > everything else.
- >
- > On input path, each packet is tagged with namespace right from the
- > place where it appears from a device, and is processed by each layer
- > in the context of this namespace.
- > Non-root namespaces communicate with the outside world in two ways: by
- > owning hardware devices, or receiving packets forwarded them by their parent
- > namespace via pass-through device.
- >
- > This complete separation of namespaces is very useful for at least two
- > purposes:
- > - allowing users to create and manage by their own various tunnels and
- > VPNs, and
- > - enabling easier and more straightforward live migration of groups of
- > processes with their environment.

I conceptually prefer this approach, but I seem to recall there were actual problems in using this for checkpoint/restart of lightweight (application) containers. Performance aside, are there any reasons why this approach would be problematic for c/r?

I'm afraid Daniel may be on vacation, and don't know who else other than Eric might have thoughts on this.

- > 2. People expressed concerns that complete separation of namespaces
- > may introduce an undesired overhead in certain usage scenarios.
- > The overhead comes from packets traversing input path, then output path,
- > then input path again in the destination namespace if root namespace
- > acts as a router.
- >
- > So, we may introduce short-cuts, when input packet starts to be processes
- > in one namespace, but changes it at some upper layer.
- > The places where packet can change namespace are, for example:
- > routing, post-routing netfilter hook, or even lookup in socket hash.
- >
- > The cleanest example among them is post-routing netfilter hook.
- > Tagging of input packets there means that the packets is checked against
- > root namespace's routing table, found to be local, and go directly to
- > the socket hash lookup in the destination namespace.
- > In this scheme the ability to change routing tables or netfilter rules on
- > a per-namespace basis is traded for lower overhead.
- >
- > All other optimized schemes where input packets do not travel

- > input-output-input paths in general case may be viewed as short-cuts in
- > scheme (1). The remaining question is which exactly short-cuts make most
- > sense, and how to make them consistent from the interface point of view.
- >
- > My current idea is to reach some agreement on the basic concept, review
- > patches, and then move on to implementing feasible short-cuts.
- >
- > Opinions?
- >
- > Next in this thread are patches introducing namespaces to device list,
- > IPv4 routing, and socket hashes, and a pass-through device.
- > Patches are against 2.6.18-rc4-mm1.

Just to provide the extreme other end of implementation options, here is the bsdjail based version I've been using for some testing while waiting for network namespaces to show up in -mm :)

(Not intended for *any* sort of inclusion consideration :)

Example usage:

```
ifconfig eth0:0 192.168.1.16
echo -n "ip 192.168.1.16" > /proc/$$/attr/exec
exec /bin/sh
```

-serge

From: Serge E. Hallyn <hallyn@sergelap.(none)>

Date: Wed, 26 Jul 2006 21:47:13 -0500

Subject: [PATCH 1/1] bsdjail: define bsdjail lsm

Define the actual bsdjail LSM.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>

---

```
security/Kconfig | 11
security/Makefile | 1
security/bsdjail.c | 1351 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
3 files changed, 1363 insertions(+), 0 deletions(-)
```

```
diff --git a/security/Kconfig b/security/Kconfig
index 67785df..fa30e40 100644
--- a/security/Kconfig
+++ b/security/Kconfig
@@ -105,6 +105,17 @@ config SECURITY_SECLVL
```

If you are unsure how to answer this question, answer N.

```
+config SECURITY_BSDJAIL
```

```

+ tristate "BSD Jail LSM"
+ depends on SECURITY
+ select SECURITY_NETWORK
+ help
+ Provides BSD Jail compartmentalization functionality.
+ See Documentation/bsdjail.txt for more information and
+ usage instructions.
+
+ If you are unsure how to answer this question, answer N.
+
source security/selinux/Kconfig

```

```

endmenu
diff --git a/security/Makefile b/security/Makefile
index 8cbbf2f..050b588 100644
--- a/security/Makefile
+++ b/security/Makefile
@@ -17,3 +17,4 @@ obj-$(CONFIG_SECURITY_SELINUX) += selin
obj-$(CONFIG_SECURITY_CAPABILITIES) += commoncap.o capability.o
obj-$(CONFIG_SECURITY_ROOTPLUG) += commoncap.o root_plug.o
obj-$(CONFIG_SECURITY_SECLVL) += seclvl.o
+obj-$(CONFIG_SECURITY_BSDJAIL) += bsdjail.o
diff --git a/security/bsdjail.c b/security/bsdjail.c
new file mode 100644
index 0000000..d800610
--- /dev/null
+++ b/security/bsdjail.c
@@ -0,0 +1,1351 @@
+/*
+ * File: linux/security/bsdjail.c
+ * Author: Serge Hallyn (serue@us.ibm.com)
+ * Date: Sep 12, 2004
+ *
+ * (See Documentation/bsdjail.txt for more information)
+ *
+ * Copyright (C) 2004 International Business Machines <serue@us.ibm.com>
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ */
+
+#include <linux/config.h>
+#include <linux/module.h>
+#include <linux/kernel.h>
+#include <linux/init.h>
+#include <linux/security.h>

```

```

#include <linux/namei.h>
#include <linux/namespace.h>
#include <linux/proc_fs.h>
#include <linux/in.h>
#include <linux/in6.h>
#include <linux/pagemap.h>
#include <linux/ip.h>
#include <net/ipv6.h>
#include <linux/mount.h>
#include <linux/netdevice.h>
#include <linux/inetdevice.h>
#include <linux/seq_file.h>
#include <linux/un.h>
#include <linux/smp_lock.h>
#include <linux/kref.h>
#include <asm/uaccess.h>
+
+static int jail_debug;
+module_param(jail_debug, int, 0);
+MODULE_PARM_DESC(jail_debug, "Print bsd jail debugging messages.\n");
+
+#define DBG 0
+#define WARN 1
+#define bsdj_debug(how, fmt, arg... ) \
+ do { \
+  if ( how || jail_debug ) \
+   printk(KERN_NOTICE "%s: %s: " fmt, \
+    MY_NAME, __FUNCTION__ , \
+    ## arg ); \
+ } while ( 0 )
+
+#define MY_NAME "bsdjail"
+
+/* flag to keep track of how we were registered */
+static int secondary;
+
+/*
+ * The task structure holding jail information.
+ * Taskp->security points to one of these (or is null).
+ * There is exactly one jail_struct for each jail.  If >1 process
+ * are in the same jail, they share the same jail_struct.
+ */
+struct jail_struct {
+ struct kref kref;
+
+ /* these are set on writes to /proc/<pid>/attr/exec */
+ char *ip4_addr_name; /* char * containing ip4 addr to use for jail */
+ char *ip6_addr_name; /* char * containing ip6 addr to use for jail */

```

```

+
+ /* these are set when a jail becomes active */
+ __u32 addr4; /* internal form of ip4_addr_name */
+ struct in6_addr addr6; /* internal form of ip6_addr_name */
+
+ /* Resource limits. 0 = no limit */
+ int max_nrtask; /* maximum number of tasks within this jail. */
+ int cur_nrtask; /* current number of tasks within this jail. */
+ long maxtimeslice; /* max timeslice in ms for procs in this jail */
+ long nice; /* nice level for processes in this jail */
+ long max_data, max_memlock; /* equivalent to RLIMIT_{DATA, MEMLOCK} */
+ /* values for the jail_flags field */
+ #define IN_USE 1 /* if 0, task is setting up jail, not yet in it */
+ #define GOT_IPV4 2
+ #define GOT_IPV6 4 /* if 0, ipv4, else ipv6 */
+ char jail_flags;
+};
+
+ /*
+ * disable_jail: A jail which was in use, but has no references
+ * left, is disabled - we free up the mountpoint and dentry, and
+ * give up our reference on the module.
+ *
+ * don't need to put namespace, it will be done automatically
+ * when the last process in jail is put.
+ * DO need to put the dentry and vfsmount
+ */
+static void
+disable_jail(struct jail_struct *tsec)
+{
+ module_put(THIS_MODULE);
+}
+
+static void free_jail(struct jail_struct *tsec)
+{
+ if (!tsec)
+ return;
+
+ kfree(tsec->ip4_addr_name);
+ kfree(tsec->ip6_addr_name);
+ kfree(tsec);
+}
+
+ /* release_jail:
+ * Callback for kref_put to use for releasing a jail when its
+ * last user exits.
+ */

```

```

+static void release_jail(struct kref *kref)
+{
+ struct jail_struct *tsec;
+
+ tsec = container_of(kref, struct jail_struct, kref);
+ disable_jail(tsec);
+ free_jail(tsec);
+}
+
+/*
+ * jail_task_free_security: this is the callback hooked into LSM.
+ * If there was no task->security field for bsdjail, do nothing.
+ * If there was, but it was never put into use, free the jail.
+ * If there was, and the jail is in use, then decrement the usage
+ * count, and disable and free the jail if the usage count hits 0.
+ */
+static void jail_task_free_security(struct task_struct *task)
+{
+ struct jail_struct *tsec = task->security;
+
+ if (!tsec)
+ return;
+
+ if (!(tsec->jail_flags & IN_USE)) {
+ /*
+  * someone did 'echo -n x > /proc/<pid>/attr/exec' but
+  * then forked before execing. Nuke the old info.
+  */
+ free_jail(tsec);
+ task->security = NULL;
+ return;
+ }
+ tsec->cur_nrtask--;
+ /* If this was the last process in the jail, delete the jail */
+ kref_put(&tsec->kref, release_jail);
+}
+
+static struct jail_struct *
+alloc_task_security(struct task_struct *tsk)
+{
+ struct jail_struct *tsec;
+
+ tsec = kmalloc(sizeof(struct jail_struct), GFP_KERNEL);
+ if (tsec) {
+ memset(tsec, 0, sizeof(struct jail_struct));
+ tsk->security = tsec;
+ }
+ return tsec;

```

```

+}
+
+static inline int
+in_jail(struct task_struct *t)
+{
+ struct jail_struct *tsec = t->security;
+
+ if (tsec && (tsec->jail_flags & IN_USE))
+ return 1;
+
+ return 0;
+}
+
+/*
+ * If a network address was passed into /proc/<pid>/attr/exec,
+ * then process in its jail will only be allowed to bind/listen
+ * to that address.
+ */
+static void
+setup_netaddress(struct jail_struct *tsec)
+{
+ unsigned int a, b, c, d, i;
+ unsigned int x[8];
+
+ tsec->jail_flags &= ~(GOT_IPV4 | GOT_IPV6);
+ tsec->addr4 = 0;
+ ipv6_addr_set(&tsec->addr6, 0, 0, 0, 0);
+
+ if (tsec->ip4_addr_name) {
+ if (sscanf(tsec->ip4_addr_name, "%u.%u.%u.%u",
+ &a, &b, &c, &d) != 4)
+ return;
+ if (a>255 || b>255 || c>255 || d>255)
+ return;
+ tsec->addr4 = htonl((a<<24) | (b<<16) | (c<<8) | d);
+ tsec->jail_flags |= GOT_IPV4;
+ bsdj_debug(DBG, "Network (ipv4) set up (%s)\n",
+ tsec->ip4_addr_name);
+ }
+
+ if (tsec->ip6_addr_name) {
+ if (sscanf(tsec->ip6_addr_name, "%x:%x:%x:%x:%x:%x:%x:%x",
+ &x[0], &x[1], &x[2], &x[3], &x[4], &x[5], &x[6],
+ &x[7]) != 8) {
+ printk(KERN_INFO "%s: bad ipv6 addr %s\n", __FUNCTION__,
+ tsec->ip6_addr_name);
+ return;
+ }
+ }

```

```

+ for (i=0; i<8; i++) {
+   if (x[i] > 65535) {
+     printk("%s: %x > 65535 at %d\n", __FUNCTION__, x[i], i);
+     return;
+   }
+   tsec->addr6.in6_u.u6_addr16[i] = htons(x[i]);
+ }
+ tsec->jail_flags |= GOT_IPV6;
+ bsdj_debug(DBG, "Network (ipv6) set up (%s)\n",
+   tsec->ip6_addr_name);
+ }
+}
+
+/*
+ * enable_jail:
+ * Called when a process is placed into a new jail to handle the
+ * actual creation of the jail.
+ * Creates namespace
+ * Stores the requested ip address
+ * Registers a unique pseudo-proc filesystem for this jail
+ */
+static int enable_jail(struct task_struct *tsk)
+{
+   struct jail_struct *tsec = tsk->security;
+   int retval = -EFAULT;
+
+   if (!tsec)
+     goto out;
+
+   /* set up networking */
+   if (tsec->ip4_addr_name || tsec->ip6_addr_name)
+     setup_netaddress(tsec);
+
+   tsec->cur_nrtask = 1;
+   if (tsec->nice)
+     set_user_nice(current, tsec->nice);
+   if (tsec->max_data) {
+     current->signal->rlim[RLIMIT_DATA].rlim_cur = tsec->max_data;
+     current->signal->rlim[RLIMIT_DATA].rlim_max = tsec->max_data;
+   }
+   if (tsec->max_memlock) {
+     current->signal->rlim[RLIMIT_MEMLOCK].rlim_cur =
+       tsec->max_memlock;
+     current->signal->rlim[RLIMIT_MEMLOCK].rlim_max =
+       tsec->max_memlock;
+   }
+   if (tsec->maxtimeslice) {
+     current->signal->rlim[RLIMIT_CPU].rlim_cur = tsec->maxtimeslice;

```

```

+ current->signal->rlim[RLIMIT_CPU].rlim_max = tsec->maxtimeslice;
+ }
+ /* success and end */
+ kref_init(&tsec->kref);
+ tsec->jail_flags |= IN_USE;
+
+ /* won't let ourselves be removed until this jail goes away */
+ try_module_get(THIS_MODULE);
+
+ return 0;
+
+out:
+ return retval;
+}
+
+/*
+ * LSM /proc/<pid>/attr hooks.
+ * You may write into /proc/<pid>/attr/exec:
+ *   lock (no value, just to specify a jail)
+ *   ip 2.2.2.2
+ * etc...
+ * These values will be used on the next exec() to set up your jail
+ * (assuming you're not already in a jail)
+ */
+static int
+jail_setprocattr(struct task_struct *p, char *name, void *value, size_t rsize)
+{
+ struct jail_struct *tsec = current->security;
+ long val;
+ char *v = value;
+ int start, len;
+ size_t size = rsize;
+
+ if (tsec && (tsec->jail_flags & IN_USE))
+ return -EINVAL; /* let them guess why */
+
+ if (p != current || strcmp(name, "exec"))
+ return -EPERM;
+
+ if (!tsec) {
+ tsec = alloc_task_security(current);
+ if (!tsec)
+ return -ENOMEM;
+ }
+
+ if (v[size-1] == '\n')
+ size--;
+
+

```

```

+ if (strncmp(value, "ip ", 3) == 0) {
+   kfree(tsec->ip4_addr_name);
+   start = 3;
+   len = size - start + 1;
+   tsec->ip4_addr_name = kmalloc(len, GFP_KERNEL);
+   if (!tsec->ip4_addr_name)
+     return -ENOMEM;
+   strcpy(tsec->ip4_addr_name, value+start, len);
+ } else if (strncmp(value, "ip6 ", 4) == 0) {
+   kfree(tsec->ip6_addr_name);
+   start = 4;
+   len = size - start + 1;
+   tsec->ip6_addr_name = kmalloc(len, GFP_KERNEL);
+   if (!tsec->ip6_addr_name)
+     return -ENOMEM;
+   strcpy(tsec->ip6_addr_name, value+start, len);
+
+   /* the next two are equivalent */
+ } else if (strncmp(value, "slice ", 6) == 0) {
+   val = simple_strtoul(value+6, NULL, 0);
+   tsec->maxtimeslice = val;
+ } else if (strncmp(value, "timeslice ", 10) == 0) {
+   val = simple_strtoul(value+10, NULL, 0);
+   tsec->maxtimeslice = val;
+ } else if (strncmp(value, "nrtask ", 7) == 0) {
+   val = (int) simple_strtol(value+7, NULL, 0);
+   if (val < 1)
+     return -EINVAL;
+   tsec->max_nrtask = val;
+ } else if (strncmp(value, "memlock ", 8) == 0) {
+   val = simple_strtoul(value+8, NULL, 0);
+   tsec->max_memlock = val;
+ } else if (strncmp(value, "data ", 5) == 0) {
+   val = simple_strtoul(value+5, NULL, 0);
+   tsec->max_data = val;
+ } else if (strncmp(value, "nice ", 5) == 0) {
+   val = simple_strtoul(value+5, NULL, 0);
+   tsec->nice = val;
+ } else if (strncmp(value, "lock", 4) != 0)
+   return -EINVAL;
+
+ return rsize;
+}
+
+static int print_jail_net_info(struct jail_struct *j, char *buf, int maxcnt)
+{
+   int len = 0;
+
+

```

```

+ if (j->ip4_addr_name)
+ len += snprintf(buf, maxcnt, "%s\n", j->ip4_addr_name);
+ if (j->ip6_addr_name)
+ len += snprintf(buf, maxcnt-len, "%s\n", j->ip6_addr_name);
+
+ if (len)
+ return len;
+
+ return snprintf(buf, maxcnt, "No network information\n");
+}
+
+/*
+ * LSM /proc/<pid>/attr read hook.
+ *
+ * /proc/$$/attr/current output:
+ * If the reading process, say process 1001, is in a jail, then
+ * cat /proc/999/attr/current
+ * will print networking information.
+ * If the reading process, say process 1001, is not in a jail, then
+ * cat /proc/999/attr/current
+ * will return
+ * ip: (ip address of jail)
+ * if 999 is in a jail, or
+ * -EINVAL
+ * if 999 is not in a jail.
+ *
+ * /proc/$$/attr/exec output:
+ * A process in a jail gets -EINVAL for /proc/$$/attr/exec.
+ * A process not in a jail gets hints on starting a jail.
+ */
+static int
+jail_getprocattr(struct task_struct *p, char *name, void *value, size_t size)
+{
+ struct jail_struct *tsec;
+ int err = 0;
+
+ if (in_jail(current)) {
+ if (strcmp(name, "current") == 0) {
+ /* provide network info */
+ err = print_jail_net_info(current->security, value,
+ size);
+ return err;
+ }
+ return -EINVAL; /* let them guess why */
+ }
+
+ if (strcmp(name, "exec") == 0) {
+ /* Print usage some help */

```

```

+ err = snprintf(value, size,
+ "Valid keywords:\n"
+ "lock\n"
+ "ip    <ip4-addr>\n"
+ "ip6   <ip6-addr>\n"
+ "nrtask <max number of tasks in this jail>\n"
+ "nice   <nice level for processes in this jail>\n"
+ "slice  <max timeslice per process in msecs>\n"
+ "data   <max data size per process in bytes>\n"
+ "memlock <max lockable memory per process in bytes>\n");
+ return err;
+ }
+
+ if (strcmp(name, "current"))
+ return -EPERM;
+
+ tsec = p->security;
+ if (!tsec || !(tsec->jail_flags & IN_USE)) {
+ err = snprintf(value, size, "Not Jailed\n");
+ } else {
+ err = snprintf(value, size,
+ "IPv4: %s\nIPv6: %s\n"
+ "max_nrtask %d current nrtask %d max_timeslice %lu "
+ "nice %lu\n"
+ "max_memlock %lu max_data %lu\n",
+ tsec->ip4_addr_name ? tsec->ip4_addr_name : "(none)",
+ tsec->ip6_addr_name ? tsec->ip6_addr_name : "(none)",
+ tsec->max_nrtask, tsec->cur_nrtask, tsec->maxtimeslice,
+ tsec->nice, tsec->max_data, tsec->max_memlock);
+ }
+
+ return err;
+}
+
+/*
+ * Forbid a process in a jail from sending a signal to a process in another
+ * (or no) jail through file sigio.
+ *
+ * We consider the process which set the fowner to be the one sending the
+ * signal, rather than the one writing to the file. Therefore we store the
+ * jail of a process during jail_file_set_fowner, then check that against
+ * the jail of the process receiving the signal.
+ */
+static int
+jail_file_send_sigiotask(struct task_struct *tsk, struct fown_struct *fown,
+ int sig)
+{
+ struct file *file;

```

```

+
+ if (!in_jail(current))
+ return 0;
+
+ file = container_of(fown, struct file, f_owner);
+ if (file->f_security != tsk->security)
+ return -EPERM;
+
+ return 0;
+}
+
+static int
+jail_file_set_fowner(struct file *file)
+{
+ struct jail_struct *tsec;
+
+ tsec = current->security;
+ file->f_security = tsec;
+ if (tsec)
+ kref_get(&tsec->kref);
+
+ return 0;
+}
+
+static void free_ipc_security(struct kern_ipc_perm *ipc)
+{
+ struct jail_struct *tsec;
+
+ tsec = ipc->security;
+ if (!tsec)
+ return;
+ kref_put(&tsec->kref, release_jail);
+ ipc->security = NULL;
+}
+
+static void free_file_security(struct file *file)
+{
+ struct jail_struct *tsec;
+
+ tsec = file->f_security;
+ if (!tsec)
+ return;
+ kref_put(&tsec->kref, release_jail);
+ file->f_security = NULL;
+}
+
+static void free_inode_security(struct inode *inode)
+{

```

```

+ struct jail_struct *tsec;
+
+ tsec = inode->i_security;
+ if (!tsec)
+ return;
+ kref_put(&tsec->kref, release_jail);
+ inode->i_security = NULL;
+}
+
+/*
+ * LSM ptrace hook:
+ * process in jail may not ptrace process not in the same jail
+ */
+static int
+jail_ptrace (struct task_struct *tracer, struct task_struct *tracee)
+{
+ struct jail_struct *tsec = tracer->security;
+
+ if (tsec && (tsec->jail_flags & IN_USE)) {
+ if (tsec == tracee->security)
+ return 0;
+ return -EPERM;
+ }
+ return 0;
+}
+
+/*
+ * process in jail may only use one (aliased) ip address. If they try to
+ * attach to 127.0.0.1, that is remapped to their own address. If some
+ * other address (and not their own), deny permission
+ */
+static int jail_socket_unix_bind(struct socket *sock, struct sockaddr *address,
+ int addrlen);
+
+#define loopbackaddr htonl((127 << 24) | 1)
+
+static inline int jail_inet4_bind(struct socket *sock, struct sockaddr *address,
+ int addrlen, struct jail_struct *tsec)
+{
+ struct sockaddr_in *inaddr;
+ __u32 sin_addr, jailaddr;
+
+ if (!(tsec->jail_flags & GOT_IPV4))
+ return -EPERM;
+
+ inaddr = (struct sockaddr_in *) address;
+ sin_addr = inaddr->sin_addr.s_addr;
+ jailaddr = tsec->addr4;

```

```

+
+ if (sin_addr == jailaddr)
+ return 0;
+
+ if (sin_addr == loopbackaddr || !sin_addr) {
+ bsdj_debug(DBG, "Got a loopback or 0 address\n");
+ sin_addr = jailaddr;
+ bsdj_debug(DBG, "Converted to: %u.%u.%u.%u\n",
+ NIPQUAD(sin_addr));
+ return 0;
+ }
+
+ return -EPERM;
+}
+
+static inline int
+jail_inet6_bind(struct socket *sock, struct sockaddr *address, int addrlen,
+ struct jail_struct *tsec)
+{
+ struct sockaddr_in6 *inaddr6;
+ struct in6_addr *sin6_addr, *jailaddr;
+
+ if (!(tsec->jail_flags & GOT_IPV6))
+ return -EPERM;
+
+ inaddr6 = (struct sockaddr_in6 *) address;
+ sin6_addr = &inaddr6->sin6_addr;
+ jailaddr = &tsec->addr6;
+
+ if (ipv6_addr_cmp(jailaddr, sin6_addr) == 0)
+ return 0;
+
+ if (ipv6_addr_cmp(sin6_addr, &in6addr_loopback) == 0) {
+ ipv6_addr_copy(sin6_addr, jailaddr);
+ return 0;
+ }
+
+ printk(KERN_NOTICE "%s: DENYING\n", __FUNCTION__);
+ printk(KERN_NOTICE "%s: a %04x:%04x:%04x:%04x:%04x:%04x:%04x: "
+ "j %04x:%04x:%04x:%04x:%04x:%04x:%04x:%04x\n",
+ __FUNCTION__,
+ NIP6(*sin6_addr),
+ NIP6(*jailaddr));
+
+ return -EPERM;
+}
+
+static int

```

```

+jail_socket_bind(struct socket *sock, struct sockaddr *address, int addrlen)
+{
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return 0;
+
+ if (sock->sk->sk_family == AF_UNIX)
+ return jail_socket_unix_bind(sock, address, addrlen);
+
+ if (!(tsec->jail_flags & (GOT_IPV4 | GOT_IPV6)))
+ /* If we want to be strict, we could just
+  * deny net access when lacking a pseudo ip.
+  * For now we just allow it. */
+ return 0;
+
+ switch(address->sa_family) {
+ case AF_INET:
+ return jail_inet4_bind(sock, address, addrlen, tsec);
+
+ case AF_INET6:
+ return jail_inet6_bind(sock, address, addrlen, tsec);
+
+ default:
+ return 0;
+ }
+}
+
+/*
+ * If locked in an ipv6 jail, don't let them use ipv4, and vice versa
+ */
+static int
+jail_socket_create(int family, int type, int protocol, int kern)
+{
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || kern || !(tsec->jail_flags & IN_USE) ||
+     !(tsec->jail_flags & (GOT_IPV4 | GOT_IPV6)))
+ return 0;
+
+ switch(family) {
+ case AF_INET:
+ if (tsec->jail_flags & GOT_IPV4)
+ return 0;
+ return -EPERM;
+ case AF_INET6:
+ if (tsec->jail_flags & GOT_IPV6)
+ return 0;

```

```

+ return -EPERM;
+ default:
+ return 0;
+ };
+
+ return 0;
+}
+
+static void
+jail_socket_post_create(struct socket *sock, int family, int type,
+ int protocol, int kern)
+{
+ struct inet_sock *inet;
+ struct ipv6_pinfo *inet6;
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || kern || !(tsec->jail_flags & IN_USE) ||
+ !(tsec->jail_flags & (GOT_IPV4 | GOT_IPV6)))
+ return;
+
+ switch(family) {
+ case AF_INET:
+ inet = inet_sk(sock->sk);
+ inet->saddr = tsec->addr4;
+ break;
+ case AF_INET6:
+ inet6 = inet6_sk(sock->sk);
+ ipv6_addr_copy(&inet6->saddr, &tsec->addr6);
+ break;
+ default:
+ break;
+ };
+
+ return;
+}
+
+static int
+jail_socket_listen(struct socket *sock, int backlog)
+{
+ struct inet_sock *inet;
+ struct ipv6_pinfo *inet6;
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE) ||
+ !(tsec->jail_flags & (GOT_IPV4 | GOT_IPV6)))
+ return 0;
+
+ switch (sock->sk->sk_family) {

```

```

+ case AF_INET:
+   inet = inet_sk(sock->sk);
+   if (inet->saddr == tsec->addr4)
+     return 0;
+   return -EPERM;
+
+ case AF_INET6:
+   inet6 = inet6_sk(sock->sk);
+   if (ipv6_addr_cmp(&inet6->saddr, &tsec->addr6) == 0)
+     return 0;
+   return -EPERM;
+
+ default:
+   return 0;
+
+ }
+}
+
+static void free_sock_security(struct sock *sk)
+{
+   struct jail_struct *tsec;
+
+   tsec = sk->sk_security;
+   if (!tsec)
+     return;
+   kref_put(&tsec->kref, release_jail);
+   sk->sk_security = NULL;
+}
+
+/*
+ * The next three (socket) hooks prevent a process in a jail from sending
+ * data to a abstract unix domain socket which was bound outside the jail.
+ */
+static int
+jail_socket_unix_bind(struct socket *sock, struct sockaddr *address,
+   int addrlen)
+{
+   struct sockaddr_un *sunaddr;
+   struct jail_struct *tsec;
+
+   if (sock->sk->sk_family != AF_UNIX)
+     return 0;
+
+   sunaddr = (struct sockaddr_un *) address;
+   if (sunaddr->sun_path[0] != 0)
+     return 0;
+
+   tsec = current->security;

```

```

+ sock->sk->sk_security = tsec;
+ if (tsec)
+   kref_get(&tsec->kref);
+ return 0;
+}
+
+/*
+ * Note - we deny sends both from unjailed to jailed, and from jailed
+ * to unjailed. As well as, of course between different jails.
+ */
+static int
+jail_socket_unix_may_send(struct socket *sock, struct socket *other)
+{
+ struct jail_struct *tsec, *ssec;
+
+ tsec = current->security; /* jail of sending process */
+ ssec = other->sk->sk_security; /* jail of receiver */
+
+ if (tsec != ssec)
+   return -EPERM;
+
+ return 0;
+}
+
+static int
+jail_socket_unix_stream_connect(struct socket *sock,
+ struct socket *other, struct sock *newsk)
+{
+ struct jail_struct *tsec, *ssec;
+
+ tsec = current->security; /* jail of sending process */
+ ssec = other->sk->sk_security; /* jail of receiver */
+
+ if (tsec != ssec)
+   return -EPERM;
+
+ return 0;
+}
+
+static int
+jail_mount(char * dev_name, struct nameidata *nd, char * type,
+ unsigned long flags, void * data)
+{
+ if (in_jail(current))
+   return -EPERM;
+
+ return 0;
+}

```

```

+
+static int
+jail_umount(struct vfsmount *mnt, int flags)
+{
+ if (in_jail(current))
+ return -EPERM;
+
+ return 0;
+}
+
+/*
+ * process in jail may not:
+ * use nice
+ * change network config
+ * load/unload modules
+ */
+static int
+jail_capable (struct task_struct *tsk, int cap)
+{
+ if (in_jail(tsk)) {
+ if (cap == CAP_SYS_NICE)
+ return -EPERM;
+ if (cap == CAP_NET_ADMIN)
+ return -EPERM;
+ if (cap == CAP_SYS_MODULE)
+ return -EPERM;
+ if (cap == CAP_SYS_RAWIO)
+ return -EPERM;
+ }
+
+ if (cap_is_fs_cap (cap) ? tsk->fsuid == 0 : tsk->euid == 0)
+ return 0;
+ return -EPERM;
+}
+
+/*
+ * jail_security_task_create:
+ *
+ * If the current process is in a jail, and that jail is about to exceed a
+ * maximum number of processes, then refuse to fork. If the maximum number
+ * of jails is listed as 0, then there is no limit for this jail, and we allow
+ * all forks.
+ */
+static inline int
+jail_security_task_create (unsigned long clone_flags)
+{
+ struct jail_struct *tsec = current->security;
+
+

```

```

+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return 0;
+
+ if (tsec->max_nrtask && tsec->cur_nrtask >= tsec->max_nrtask)
+ return -EPERM;
+ return 0;
+}
+
+/*
+ * The child of a process in a jail belongs in the same jail
+ */
+static int
+jail_task_alloc_security(struct task_struct *tsk)
+{
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return 0;
+
+ tsk->security = tsec;
+ kref_get(&tsec->kref);
+ tsec->cur_nrtask++;
+ if (tsec->maxtimeslice) {
+ tsk->signal->rlim[RLIMIT_CPU].rlim_max = tsec->maxtimeslice;
+ tsk->signal->rlim[RLIMIT_CPU].rlim_cur = tsec->maxtimeslice;
+ }
+ if (tsec->max_data) {
+ tsk->signal->rlim[RLIMIT_CPU].rlim_max = tsec->max_data;
+ tsk->signal->rlim[RLIMIT_CPU].rlim_cur = tsec->max_data;
+ }
+ if (tsec->max_memlock) {
+ tsk->signal->rlim[RLIMIT_CPU].rlim_max = tsec->max_memlock;
+ tsk->signal->rlim[RLIMIT_CPU].rlim_cur = tsec->max_memlock;
+ }
+ if (tsec->nice)
+ set_user_nice(current, tsec->nice);
+
+ return 0;
+}
+
+static int
+jail_bprm_alloc_security(struct linux_binprm *bprm)
+{
+ struct jail_struct *tsec = current->security;
+ int ret;
+
+ if (!tsec)
+ return 0;

```

```

+
+ if (tsec->jail_flags & IN_USE)
+ return 0;
+
+ ret = enable_jail(current);
+ if (ret) {
+ /* if we failed, nix out the ip requests */
+ jail_task_free_security(current);
+ return ret;
+ }
+ return 0;
+}
+
+/*
+ * Process in jail may not create devices
+ * Thanks to Brad Spender for pointing out fifos should be allowed.
+ */
+/* TODO: We may want to allow /dev/log, at least... */
+static int
+jail_inode_mknod(struct inode *dir, struct dentry *dentry, int mode, dev_t dev)
+{
+ if (!in_jail(current))
+ return 0;
+
+ if (S_ISFIFO(mode))
+ return 0;
+
+ return -EPERM;
+}
+
+/* yanked from fs/proc/base.c */
+static unsigned name_to_int(struct dentry *dentry)
+{
+ const char *name = dentry->d_name.name;
+ int len = dentry->d_name.len;
+ unsigned n = 0;
+
+ if (len > 1 && *name == '0')
+ goto out;
+ while (len-- > 0) {
+ unsigned c = *name++ - '0';
+ if (c > 9)
+ goto out;
+ if (n >= (~0U-9)/10)
+ goto out;
+ n *= 10;
+ n += c;
+ }

```

```

+ return n;
+out:
+ return ~0U;
+}
+
+/*
+ * jail_proc_inode_permission:
+ *   called only when current is in a jail, and is trying to reach
+ *   /proc/<pid>. We check whether <pid> is in the same jail as
+ *   current. If not, permission is denied.
+ *
+ * NOTE: On the one hand, the task_to_inode(inode)->i_security
+ * approach seems cleaner, but on the other, this prevents us
+ * from unloading bsdjail for awhile...
+ */
+static int
+jail_proc_inode_permission(struct inode *inode, int mask,
+    struct nameidata *nd)
+{
+ struct jail_struct *tsec = current->security;
+ struct dentry *dentry = nd->dentry;
+ unsigned pid;
+
+ pid = name_to_int(dentry);
+ if (pid == ~0U) {
+ return 0;
+ }
+
+ if (dentry->d_parent != dentry->d_sb->s_root)
+ return 0;
+ if (inode->i_security != tsec)
+ return -ENOENT;
+
+ return 0;
+}
+
+/*
+ * security_task_to_inode:
+ * Set inode->security = task's jail.
+ */
+static void jail_task_to_inode(struct task_struct *p, struct inode *inode)
+{
+ struct jail_struct *tsec = p->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return;
+ if (inode->i_security)
+ return;

```

```

+ kref_get(&tsec->kref);
+ inode->i_security = tsec;
+}
+
+/*
+ * inode_permission:
+ * If we are trying to look into certain /proc files from in a jail, we
+ * may deny permission.
+ */
+static int
+jail_inode_permission(struct inode *inode, int mask,
+    struct nameidata *nd)
+{
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return 0;
+
+ if (!nd)
+ return 0;
+
+ if (nd->dentry &&
+     strcmp(nd->dentry->d_sb->s_type->name, "proc") == 0) {
+ return jail_proc_inode_permission(inode, mask, nd);
+ }
+
+ return 0;
+}
+
+/*
+ * A function which returns -ENOENT if dentry is the dentry for
+ * a /proc/<pid> directory. It returns 0 otherwise.
+ */
+static inline int
+generic_procpid_check(struct dentry *dentry)
+{
+ struct jail_struct *jail = current->security;
+ unsigned pid = name_to_int(dentry);
+
+ if (!jail || !(jail->jail_flags & IN_USE))
+ return 0;
+ if (pid == ~0U)
+ return 0;
+ if (strcmp(dentry->d_sb->s_type->name, "proc") != 0)
+ return 0;
+ if (dentry->d_parent != dentry->d_sb->s_root)
+ return 0;

```

```

+ if (dentry->d_inode->i_security != jail)
+ return -ENOENT;
+ return 0;
+}
+
+/*
+ * We want getattr to fail on /proc/<pid> to prevent leakage through, for
+ * instance, ls -d.
+ */
+static int
+jail_inode_getattr(struct vfsmount *mnt, struct dentry *dentry)
+{
+ return generic_procpid_check(dentry);
+}
+
+/* This probably is not necessary - /proc does not support xattrs? */
+static int
+jail_inode_getxattr(struct dentry *dentry, char *name)
+{
+ return generic_procpid_check(dentry);
+}
+
+/* process in jail may not send signal to process not in the same jail */
+static int
+jail_task_kill(struct task_struct *p, struct siginfo *info, int sig, u32 secid)
+{
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return 0;
+
+ if (tsec == p->security)
+ return 0;
+
+ if (sig==SIGCHLD)
+ return 0;
+
+ return -EPERM;
+}
+
+/*
+ * LSM hooks to limit jailed process' abilities to muck with resource
+ * limits
+ */
+static int jail_task_setrlimit (unsigned int resource, struct rlimit *new_rlim)
+{
+ if (!in_jail(current))
+ return 0;

```

```

+
+ return -EPERM;
+}
+
+static int jail_task_setscheduler (struct task_struct *p, int policy,
+    struct sched_param *lp)
+{
+ if (!in_jail(current))
+ return 0;
+
+ return -EPERM;
+}
+
+/*
+ * LSM hooks to limit IPC access.
+ */
+
+static inline int
+basic_ipc_security_check(struct kern_ipc_perm *p, struct task_struct *target)
+{
+ struct jail_struct *tsec = target->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return 0;
+
+ if (p->security != tsec)
+ return -EPERM;
+
+ return 0;
+}
+
+static int
+jail_ipc_permission(struct kern_ipc_perm *ipcp, short flag)
+{
+ return basic_ipc_security_check(ipcp, current);
+}
+
+static int
+jail_shm_alloc_security (struct shmid_kernel *shp)
+{
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return 0;
+
+ shp->shm_perm.security = tsec;
+ kref_get(&tsec->kref);
+ return 0;
+}

```

```

+
+static void
+jail_shm_free_security (struct shmid_kernel *shp)
+{
+ free_ipc_security(&shp->shm_perm);
+}
+
+static int
+jail_shm_associate (struct shmid_kernel *shp, int shmflg)
+{
+ return basic_ipc_security_check(&shp->shm_perm, current);
+}
+
+static int
+jail_shm_shmctl(struct shmid_kernel *shp, int cmd)
+{
+ if (cmd == IPC_INFO || cmd == SHM_INFO)
+ return 0;
+
+ return basic_ipc_security_check(&shp->shm_perm, current);
+}
+
+static int
+jail_shm_shmat(struct shmid_kernel *shp, char *shmaddr, int shmflg)
+{
+ return basic_ipc_security_check(&shp->shm_perm, current);
+}
+
+static int
+jail_msg_queue_alloc(struct msg_queue *msq)
+{
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return 0;
+ msq->q_perm.security = tsec;
+ kref_get(&tsec->kref);
+ return 0;
+}
+
+static void
+jail_msg_queue_free(struct msg_queue *msq)
+{
+ free_ipc_security(&msq->q_perm);
+}
+
+static int jail_msg_queue_associate(struct msg_queue *msq, int flag)
+{

```

```

+ return basic_ipc_security_check(&msq->q_perm, current);
+}
+
+static int
+jail_msg_queue_msgctl(struct msg_queue *msq, int cmd)
+{
+ if (cmd == IPC_INFO || cmd == MSG_INFO)
+ return 0;
+
+ return basic_ipc_security_check(&msq->q_perm, current);
+}
+
+static int
+jail_msg_queue_msgsnd(struct msg_queue *msq, struct msg_msg *msg, int msqflg)
+{
+ return basic_ipc_security_check(&msq->q_perm, current);
+}
+
+static int
+jail_msg_queue_msgrcv(struct msg_queue *msq, struct msg_msg *msg,
+ struct task_struct *target, long type, int mode)
+
+{
+ return basic_ipc_security_check(&msq->q_perm, target);
+}
+
+static int
+jail_sem_alloc_security(struct sem_array *sma)
+{
+ struct jail_struct *tsec = current->security;
+
+ if (!tsec || !(tsec->jail_flags & IN_USE))
+ return 0;
+ sma->sem_perm.security = tsec;
+ kref_get(&tsec->kref);
+ return 0;
+}
+
+static void
+jail_sem_free_security(struct sem_array *sma)
+{
+ free_ipc_security(&sma->sem_perm);
+}
+
+static int
+jail_sem_associate(struct sem_array *sma, int semflg)
+{
+ return basic_ipc_security_check(&sma->sem_perm, current);

```

```

+}
+
+static int
+jail_sem_semctl(struct sem_array *sma, int cmd)
+{
+ if (cmd == IPC_INFO || cmd == SEM_INFO)
+ return 0;
+ return basic_ipc_security_check(&sma->sem_perm, current);
+}
+
+static int
+jail_sem_semop(struct sem_array *sma, struct sembuf *sops, unsigned nsops,
+ int alter)
+{
+ return basic_ipc_security_check(&sma->sem_perm, current);
+}
+
+static int
+jail_sysctl(struct ctl_table *table, int op)
+{
+ if (!in_jail(current))
+ return 0;
+
+ if (op & 002)
+ return -EPERM;
+
+ return 0;
+}
+
+static struct security_operations bsdjail_security_ops = {
+ .ptrace = jail_ptrace,
+ .capable = jail_capable,
+
+ .task_kill = jail_task_kill,
+ .task_alloc_security = jail_task_alloc_security,
+ .task_free_security = jail_task_free_security,
+ .bprm_alloc_security = jail_bprm_alloc_security,
+ .task_create = jail_security_task_create,
+ .task_to_inode = jail_task_to_inode,
+
+ .task_setrlimit = jail_task_setrlimit,
+ .task_setscheduler = jail_task_setscheduler,
+
+ .setprocattr = jail_setprocattr,
+ .getprocattr = jail_getprocattr,
+
+ .file_set_fowner = jail_file_set_fowner,
+ .file_send_sigiotask = jail_file_send_sigiotask,

```

```

+ .file_free_security = free_file_security,
+
+ .socket_bind = jail_socket_bind,
+ .socket_listen = jail_socket_listen,
+ .socket_create = jail_socket_create,
+ .socket_post_create = jail_socket_post_create,
+ .unix_stream_connect = jail_socket_unix_stream_connect,
+ .unix_may_send = jail_socket_unix_may_send,
+ .sk_free_security = free_sock_security,
+
+ .inode_mknod = jail_inode_mknod,
+ .inode_permission = jail_inode_permission,
+ .inode_free_security = free_inode_security,
+ .inode_getattr = jail_inode_getattr,
+ .inode_getxattr = jail_inode_getxattr,
+ .sb_mount = jail_mount,
+ .sb_umount = jail_umount,
+
+ .ipc_permission = jail_ipc_permission,
+ .shm_alloc_security = jail_shm_alloc_security,
+ .shm_free_security = jail_shm_free_security,
+ .shm_associate = jail_shm_associate,
+ .shm_shmctl = jail_shm_shmctl,
+ .shm_shmat = jail_shm_shmat,
+
+ .msg_queue_alloc_security = jail_msg_queue_alloc,
+ .msg_queue_free_security = jail_msg_queue_free,
+ .msg_queue_associate = jail_msg_queue_associate,
+ .msg_queue_msgctl = jail_msg_queue_msgctl,
+ .msg_queue_msgsnd = jail_msg_queue_msgsnd,
+ .msg_queue_msgrcv = jail_msg_queue_msgrcv,
+
+ .sem_alloc_security = jail_sem_alloc_security,
+ .sem_free_security = jail_sem_free_security,
+ .sem_associate = jail_sem_associate,
+ .sem_semctl = jail_sem_semctl,
+ .sem_semop = jail_sem_semop,
+
+ .sysctl = jail_sysctl,
+};
+
+static int __init bsdjail_init (void)
+{
+ int rc = 0;
+
+ if (register_security (&bsdjail_security_ops)) {
+ printk (KERN_INFO
+ "Failure registering BSD Jail module with the kernel\n");

```

```

+
+ rc = mod_reg_security(MY_NAME, &bsdjail_security_ops);
+ if (rc < 0) {
+   printk (KERN_INFO "Failure registering BSD Jail "
+    " module with primary security module.\n");
+   return -EINVAL;
+ }
+ secondary = 1;
+ }
+ printk (KERN_INFO "BSD Jail module initialized.\n");
+
+ return 0;
+}
+
+static void __exit bsdjail_exit (void)
+{
+ if (secondary) {
+   if (mod_unreg_security (MY_NAME, &bsdjail_security_ops))
+     printk (KERN_INFO "Failure unregistering BSD Jail "
+      " module with primary module.\n");
+ } else {
+   if (unregister_security (&bsdjail_security_ops)) {
+     printk (KERN_INFO "Failure unregistering BSD Jail "
+      "module with the kernel\n");
+   }
+ }
+ }
+
+ printk (KERN_INFO "BSD Jail module removed\n");
+}
+
+security_initcall (bsdjail_init);
+module_exit (bsdjail_exit);
+
+MODULE_DESCRIPTION("BSD Jail LSM.");
+MODULE_LICENSE("GPL");
+
+--
1.4.1.1

```

---

Subject: Re: [PATCH 1/9] network namespaces: core and device list  
 Posted by [Dave Hansen](#) on Wed, 16 Aug 2006 14:46:43 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 2006-08-15 at 18:48 +0400, Andrey Savochkin wrote:

```

>
>   /* 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;
>      }
```

With this much surgery it might be best to start using things that have come along since this code was touched last, like `kzalloc()`.

-- Dave

---

---

Subject: Re: [RFC] network namespaces  
Posted by [Alexey Kuznetsov](#) on Wed, 16 Aug 2006 15:12:26 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello!

> (application) containers. Performance aside, are there any reasons why  
> this approach would be problematic for c/r?

This approach is just perfect for c/r.

Probably, this is the only approach when migration can be done in a clean and self-consistent way.

Alexey

---

---

Subject: Re: [PATCH 1/9] network namespaces: core and device list  
Posted by [Stephen Hemminger](#) on Wed, 16 Aug 2006 16:45:28 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, 16 Aug 2006 07:46:43 -0700  
Dave Hansen <haveblue@us.ibm.com> wrote:

> On Tue, 2006-08-15 at 18:48 +0400, Andrey Savochkin wrote:

```
> >
> >      /* 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;
```

> > }

>

> With this much surgery it might be best to start using things that have  
> come along since this code was touched last, like kcalloc().

>

>

If you are going to make the loopback device dynamic, it MUST use  
alloc\_netdev().

---

---

Subject: Re: [RFC] network namespaces

Posted by [ebiederm](#) on Wed, 16 Aug 2006 17:35:41 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Alexey Kuznetsov <kuznet@ms2.inr.ac.ru> writes:

> Hello!

>

>> (application) containers. Performance aside, are there any reasons why  
>> this approach would be problematic for c/r?

>

> This approach is just perfect for c/r.

Yes. For c/r you need to take your state with you.

> Probably, this is the only approach when migration can be done  
> in a clean and self-consistent way.

Basically there are currently 3 approaches that have been proposed.

The trivial bsdjail style as implemented by Serge and in a slightly  
more sophisticated version in vserver. This approach as it does not  
touch the packets has little to no packet level overhead. Basically  
this is what I have called the Level 3 approach.

The more in depth approach where we modify the packet processing based  
upon which network interface the packet comes in on, and it looks like  
each namespace has it's own instance of the network stack. Roughly  
what was proposed earlier in this thread the Level 2 approach. This  
potentially has per packet overhead so we need to watch the implementation  
very carefully.

Some weird hybrid as proposed by Daniel, that I was never clear on the  
semantics.

>From the previous conversations my impression was that as long as  
we could get a Layer 2 approach that did not slow down the networking

stack and was clean, everyone would be happy.

I'm buried in the process id namespace at the moment, and except to be so for the rest of the month, so I'm not going to be very helpful except for a few stray comments.

Eric

---

---

Subject: Re: [RFC] network namespaces

Posted by [dev](#) on Thu, 17 Aug 2006 08:28:03 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

> Basically there are currently 3 approaches that have been proposed.

>

> The trivial bsdjail style as implemented by Serge and in a slightly  
> more sophisticated version in vserver. This approach as it does not  
> touch the packets has little to no packet level overhead. Basically  
> this is what I have called the Level 3 approach.

>

> The more in depth approach where we modify the packet processing based  
> upon which network interface the packet comes in on, and it looks like  
> each namespace has it's own instance of the network stack. Roughly  
> what was proposed earlier in this thread the Level 2 approach. This  
> potentially has per packet overhead so we need to watch the implementation  
> very carefully.

>

> Some weird hybrid as proposed by Daniel, that I was never clear on the  
> semantics.

The good thing is that these approaches do not contradict each other.  
We discussed it with Daniel during the summit and as Andrey proposed  
some shortcuts can be created to avoid double stack traversing.

>>From the previous conversations my impression was that as long as  
> we could get a Layer 2 approach that did not slow down the networking  
> stack and was clean, everyone would be happy.  
agree.

> I'm buried in the process id namespace at the moment, and except  
> to be so for the rest of the month, so I'm not going to be  
> very helpful except for a few stray comments.

I will be very much obliged if you find some time to review these new  
patches so that we could make some progress here.

Thanks,  
Kirill

---

---

Subject: Re: [RFC] network namespaces

Posted by [Daniel Lezcano](#) on Tue, 05 Sep 2006 13:34:40 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Hi all,

>> This complete separation of namespaces is very useful for at least two  
>> purposes:  
>> - allowing users to create and manage by their own various tunnels and  
>> VPNs, and  
>> - enabling easier and more straightforward live migration of groups of  
>> processes with their environment.

>

>

> I conceptually prefer this approach, but I seem to recall there were  
> actual problems in using this for checkpoint/restart of lightweight  
> (application) containers. Performance aside, are there any reasons why  
> this approach would be problematic for c/r?

I agree with this approach too, separated namespaces is the best way to  
identify the network resources for a specific container.

> I'm afraid Daniel may be on vacation, and don't know who else other than  
> Eric might have thoughts on this.

Yes, I was in "vacation", but I am back :)

>>2. People expressed concerns that complete separation of namespaces  
>> may introduce an undesired overhead in certain usage scenarios.  
>> The overhead comes from packets traversing input path, then output path,  
>> then input path again in the destination namespace if root namespace  
>> acts as a router.

Yes, performance is probably one issue.

My concerns was for layer 2 / layer 3 virtualization. I agree a layer 2  
isolation/virtualization is the best for the "system container".  
But there is another family of container called "application container",  
it is not a system which is run inside a container but only the  
application. If you want to run a oracle database inside a container,  
you can run it inside an application container without launching <init>  
and all the services.

This family of containers are used too for HPC (high performance  
computing) and for distributed checkpoint/restart. The cluster runs  
hundred of jobs, spawning them on different hosts inside an application  
container. Usually the jobs communicates with broadcast and multicast.  
Application containers does not care of having different MAC address and  
rely on a layer 3 approach.

Are application containers comfortable with a layer 2 virtualization ? I don't think so, because several jobs running inside the same host communicate via broadcast/multicast between them and between other jobs running on different hosts. The IP consumption is a problem too: 1 container == 2 IP (one for the root namespace/ one for the container), multiplied with the number of jobs. Furthermore, lot of jobs == lot of virtual devices.

However, after a discussion with Kirill at the OLS, it appears we can merge the layer 2 and 3 approaches if the level of network virtualization is tunable and we can choose layer 2 or layer 3 when doing the "unshare". The determination of the namespace for the incoming traffic can be done with an specific iptable module as a first step. While looking at the network namespace patches, it appears that the TCP/UDP part is **very** similar at what is needed for a layer 3 approach.

Any thoughts ?

Daniel

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Tue, 05 Sep 2006 14:45:39 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

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

>>>2. People expressed concerns that complete separation of namespaces  
>>> may introduce an undesired overhead in certain usage scenarios.  
>>> The overhead comes from packets traversing input path, then output path,  
>>> then input path again in the destination namespace if root namespace  
>>> acts as a router.  
>  
> Yes, performance is probably one issue.  
>  
> My concerns was for layer 2 / layer 3 virtualization. I agree a layer 2  
> isolation/virtualization is the best for the "system container".  
> But there is another family of container called "application container", it is  
> not a system which is run inside a container but only the application. If you  
> want to run a oracle database inside a container, you can run it inside an  
> application container without launching <init> and all the services.  
>  
> This family of containers are used too for HPC (high performance computing) and  
> for distributed checkpoint/restart. The cluster runs hundred of jobs, spawning  
> them on different hosts inside an application container. Usually the jobs  
> communicates with broadcast and multicast.  
> Application containers does not care of having different MAC address and rely on

> a layer 3 approach.

>

> Are application containers comfortable with a layer 2 virtualization ? I don't  
> think so, because several jobs running inside the same host communicate via  
> broadcast/multicast between them and between other jobs running on different  
> hosts. The IP consumption is a problem too: 1 container == 2 IP (one for the  
> root namespace/ one for the container), multiplied with the number of  
> jobs. Furthermore, lot of jobs == lot of virtual devices.

>

> However, after a discussion with Kirill at the OLS, it appears we can merge the  
> layer 2 and 3 approaches if the level of network virtualization is tunable and  
> we can choose layer 2 or layer 3 when doing the "unshare". The determination of  
> the namespace for the incoming traffic can be done with an specific iptable  
> module as a first step. While looking at the network namespace patches, it  
> appears that the TCP/UDP part is **very** similar at what is needed for a layer  
> 3 approach.

>

> Any thoughts ?

For HPC if you are interested in migration you need a separate IP per container. If you can take you IP address with you migration of networking state is simple. If you can't take your IP address with you a network container is nearly pointless from a migration perspective.

Beyond that from everything I have seen layer 2 is just much cleaner than any layer 3 approach short of Serge's bind filtering.

Beyond that I have yet to see a clean semantics for anything resembling your layer 2 layer 3 hybrid approach. If we can't have clear semantics it is by definition impossible to implement correctly because no one understands what it is supposed to do.

Note. A true layer 3 approach has no impact on TCP/UDP filtering because it filters at bind time not at packet reception time. Once you start inspecting packets I don't see what the gain is from not going all of the way to layer 2.

Eric

---

Subject: Re: [RFC] network namespaces

Posted by [Daniel Lezcano](#) on Tue, 05 Sep 2006 15:32:01 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

> For HPC if you are interested in migration you need a separate IP per  
> container. If you can take you IP address with you migration of  
> networking state is simple. If you can't take your IP address with

> you a network container is nearly pointless from a migration  
> perspective.

Eric, please, I know... I showed you a migration demo at OLS ;)

> Beyond that from everything I have seen layer 2 is just much cleaner  
> than any layer 3 approach short of Serge's bind filtering.

> Beyond that I have yet to see a clean semantics for anything  
> resembling your layer 2 layer 3 hybrid approach. If we can't have  
> clear semantics it is by definition impossible to implement correctly  
> because no one understands what it is supposed to do.

> Note. A true layer 3 approach has no impact on TCP/UDP filtering  
> because it filters at bind time not at packet reception time. Once  
> you start inspecting packets I don't see what the gain is from not  
> going all of the way to layer 2.

The bsdjail was just for information ...

- Daniel

---

Subject: Re: [RFC] network namespaces  
Posted by [dev](#) on Tue, 05 Sep 2006 15:44:19 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

> Yes, performance is probably one issue.

>

> My concerns was for layer 2 / layer 3 virtualization. I agree a layer 2  
> isolation/virtualization is the best for the "system container".

> But there is another family of container called "application container",  
> it is not a system which is run inside a container but only the  
> application. If you want to run a oracle database inside a container,  
> you can run it inside an application container without launching <init>  
> and all the services.

>

> This family of containers are used too for HPC (high performance  
> computing) and for distributed checkpoint/restart. The cluster runs  
> hundred of jobs, spawning them on different hosts inside an application  
> container. Usually the jobs communicates with broadcast and multicast.  
> Application containers does not care of having different MAC address and  
> rely on a layer 3 approach.

>

> Are application containers comfortable with a layer 2 virtualization ? I  
> don't think so, because several jobs running inside the same host  
> communicate via broadcast/multicast between them and between other jobs

> running on different hosts. The IP consumption is a problem too: 1  
> container == 2 IP (one for the root namespace/ one for the container),  
> multiplied with the number of jobs. Furthermore, lot of jobs == lot  
> of virtual devices.  
>  
> However, after a discussion with Kirill at the OLS, it appears we can  
> merge the layer 2 and 3 approaches if the level of network  
> virtualization is tunable and we can choose layer 2 or layer 3 when  
> doing the "unshare". The determination of the namespace for the incoming  
> traffic can be done with an specific iptable module as a first step.  
> While looking at the network namespace patches, it appears that the  
> TCP/UDP part is **\*\*very\*\*** similar at what is needed for a layer 3 approach.  
>

> Any thoughts ?

My humble opinion is that your approach doesn't intersect with this one.

So we can freely go with both \*if needed\*.

And hear the comments from network guru guys and what and how to improve.

So I suggest you at least to send the patches, so we could discuss it.

Thanks,  
Kirill

---

Subject: Re: [RFC] network namespaces

Posted by [Herbert Poetzl](#) on Tue, 05 Sep 2006 16:53:28 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, Sep 05, 2006 at 08:45:39AM -0600, Eric W. Biederman wrote:

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

>

> >>>2. People expressed concerns that complete separation of namespaces

> >>> may introduce an undesired overhead in certain usage scenarios.

> >>> The overhead comes from packets traversing input path, then output path,

> >>> then input path again in the destination namespace if root namespace

> >>> acts as a router.

> >

> > Yes, performance is probably one issue.

> >

> > My concerns was for layer 2 / layer 3 virtualization. I agree

> > a layer 2 isolation/virtualization is the best for the "system

> > container". But there is another family of container called

> > "application container", it is not a system which is run inside a

> > container but only the application. If you want to run a oracle

> > database inside a container, you can run it inside an application

> > container without launching <init> and all the services.

> >

> > This family of containers are used too for HPC (high performance

> > computing) and for distributed checkpoint/restart. The cluster  
> > runs hundred of jobs, spawning them on different hosts inside an  
> > application container. Usually the jobs communicates with broadcast  
> > and multicast. Application containers does not care of having  
> > different MAC address and rely on a layer 3 approach.  
> >  
> > Are application containers comfortable with a layer 2 virtualization  
> > ? I don't think so, because several jobs running inside the same  
> > host communicate via broadcast/multicast between them and between  
> > other jobs running on different hosts. The IP consumption is a  
> > problem too: 1 container == 2 IP (one for the root namespace/  
> > one for the container), multiplied with the number of jobs.  
> > Furthermore, lot of jobs == lot of virtual devices.  
> >  
> > However, after a discussion with Kirill at the OLS, it appears we  
> > can merge the layer 2 and 3 approaches if the level of network  
> > virtualization is tunable and we can choose layer 2 or layer 3 when  
> > doing the "unshare". The determination of the namespace for the  
> > incoming traffic can be done with an specific iptable module as  
> > a first step. While looking at the network namespace patches, it  
> > appears that the TCP/UDP part is **\*\*very\*\*** similar at what is needed  
> > for a layer  
> > 3 approach.  
> >  
> > Any thoughts ?  
>  
> For HPC if you are interested in migration you need a separate IP  
> per container. If you can take you IP address with you migration of  
> networking state is simple. If you can't take your IP address with you  
> a network container is nearly pointless from a migration perspective.  
>  
> Beyond that from everything I have seen layer 2 is just much cleaner  
> than any layer 3 approach short of Serge's bind filtering.

well, the 'ip subset' approach Linux-VServer and other Jail solutions use is very clean, it just does not match your expectations of a virtual interface (as there is none) and it does not cope well with all kinds of per context 'requirements', which IMHO do not really exist on the application layer (only on the whole system layer)

> Beyond that I have yet to see a clean semantics for anything  
> resembling your layer 2 layer 3 hybrid approach. If we can't have  
> clear semantics it is by definition impossible to implement correctly  
> because no one understands what it is supposed to do.

IMHO that would be quite simple, have a 'namespace'

for limiting port binds to a subset of the available  
ips and another one which does complete network  
virtualization with all the whistles and bells, IMHO  
most of them are orthogonal and can easily be combined

- full network virtualization
- lightweight ip subset
- both

> Note. A true layer 3 approach has no impact on TCP/UDP filtering  
> because it filters at bind time not at packet reception time. Once you  
> start inspecting packets I don't see what the gain is from not going  
> all of the way to layer 2.

IMHO this requirement only arises from the full system  
virtualization approach, just look at the other jail  
solutions (solaris, bsd, ...) some of them do not even  
allow for more than a single ip but they work quite  
well when used properly ...

best,  
Herbert

> Eric

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Tue, 05 Sep 2006 17:09:21 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

> This family of containers are used too for HPC (high performance computing) and  
> for distributed checkpoint/restart. The cluster runs hundred of jobs, spawning  
> them on different hosts inside an application container. Usually the jobs  
> communicates with broadcast and multicast.  
> Application containers does not care of having different MAC address and rely on  
> a layer 3 approach.

Ok I think to understand this we need some precise definitions.  
In the normal case it is an error for a job to communication with a different  
job.

The basic advantage with a different MAC is that you can found out who the  
intended recipient is sooner in the networking stack and you have truly  
separate network devices. Allowing for a cleaner implementation.

Changing the MAC after migration is likely to be fine.

> Are application containers comfortable with a layer 2 virtualization ? I don't

> think so, because several jobs running inside the same host communicate via  
> broadcast/multicast between them and between other jobs running on different  
> hosts. The IP consumption is a problem too: 1 container == 2 IP (one for the  
> root namespace/ one for the container), multiplied with the number of  
> jobs. Furthermore, lot of jobs == lot of virtual devices.

First if you hook you network namespaces with ethernet bridging  
you don't need any extra IPs.

Second don't see the conflict you perceive between application containers  
and layer 2 containment.

The bottom line is that you need at least one loopback interface per non-trivial  
network namespace. One you get that having a virtual is no big deal. In  
addition network devices don't consume less memory than a process. So lots  
of network devices should not be a problem.

Eric

---

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Tue, 05 Sep 2006 18:27:20 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Herbert Poetzl <herbert@13thfloor.at> writes:

> On Tue, Sep 05, 2006 at 08:45:39AM -0600, Eric W. Biederman wrote:  
>> Daniel Lezcano <dlezcano@fr.ibm.com> writes:  
>>  
>> For HPC if you are interested in migration you need a separate IP  
>> per container. If you can take you IP address with you migration of  
>> networking state is simple. If you can't take your IP address with you  
>> a network container is nearly pointless from a migration perspective.  
>>  
>> Beyond that from everything I have seen layer 2 is just much cleaner  
>> than any layer 3 approach short of Serge's bind filtering.  
>  
> well, the 'ip subset' approach Linux-VServer and  
> other Jail solutions use is very clean, it just does  
> not match your expectations of a virtual interface  
> (as there is none) and it does not cope well with  
> all kinds of per context 'requirements', which IMHO  
> do not really exist on the application layer (only  
> on the whole system layer)

I probably expressed that wrong. There are currently three  
basic approaches under discussion.  
Layer 3 (Basically bind filtering) nothing at the packet level.

The approach taken by Serge's version of bsdjails and Vserver.

Layer 2.5 What Daniel proposed.

Layer 2. (Trivially mapping each packet to a different interface)  
And then treating everything as multiple instances of the network stack.

Roughly what OpenVZ and I have implemented.

You can get into some weird complications at layer 3 but because it doesn't touch each packet the proof it is fast is trivial.

>> Beyond that I have yet to see a clean semantics for anything  
>> resembling your layer 2 layer 3 hybrid approach. If we can't have  
>> clear semantics it is by definition impossible to implement correctly  
>> because no one understands what it is supposed to do.

>  
> IMHO that would be quite simple, have a 'namespace'  
> for limiting port binds to a subset of the available  
> ips and another one which does complete network  
> virtualization with all the whistles and bells, IMHO  
> most of them are orthogonal and can easily be combined  
>  
> - full network virtualization  
> - lightweight ip subset  
> - both

Quite possibly. The LSM will stay for a while so we do have a clean way to restrict port binds.

>> Note. A true layer 3 approach has no impact on TCP/UDP filtering  
>> because it filters at bind time not at packet reception time. Once you  
>> start inspecting packets I don't see what the gain is from not going  
>> all of the way to layer 2.

>  
> IMHO this requirement only arises from the full system  
> virtualization approach, just look at the other jail  
> solutions (solaris, bsd, ...) some of them do not even  
> allow for more than a single ip but they work quite  
> well when used properly ...

Yes they do. Currently I am strongly opposed to Daniel Layer 2.5 approach as I see no redeeming value in it. A good clean layer 3 approach I avoid only because I think we can do better.

Eric

---

Subject: Re: [RFC] network namespaces  
Posted by [Daniel Lezcano](#) on Wed, 06 Sep 2006 09:10:23 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi Herbert,

> well, the 'ip subset' approach Linux-VServer and  
> other Jail solutions use is very clean, it just does  
> not match your expectations of a virtual interface  
> (as there is none) and it does not cope well with  
> all kinds of per context 'requirements', which IMHO  
> do not really exist on the application layer (only  
> on the whole system layer)  
>  
> IMHO that would be quite simple, have a 'namespace'  
> for limiting port binds to a subset of the available  
> ips and another one which does complete network  
> virtualization with all the whistles and bells, IMHO  
> most of them are orthogonal and can easily be combined  
>  
> - full network virtualization  
> - lightweight ip subset  
> - both  
>  
> IMHO this requirement only arises from the full system  
> virtualization approach, just look at the other jail  
> solutions (solaris, bsd, ...) some of them do not even  
> allow for more than a single ip but they work quite  
> well when used properly ...

As far as I see, vserver use a layer 3 solution but, when needed, the veth "component", made by Nestor Pena, is used to provide a layer 2 virtualization. Right ?

Having the two solutions, you have certainly a lot of information about use cases. From the point of view of vserver, can you give some examples of when a layer 3 solution is better/worst than a layer 2 solution ? Who wants a layer 2/3 virtualization and why ?

These informations will be very useful.

Regards

-- Daniel

---

---

Subject: Re: [RFC] network namespaces

Posted by [dev](#) on Wed, 06 Sep 2006 14:52:50 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

>>On Tue, Sep 05, 2006 at 08:45:39AM -0600, Eric W. Biederman wrote:

>>

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

>>>

>>>For HPC if you are interested in migration you need a separate IP  
>>>per container. If you can take you IP address with you migration of  
>>>networking state is simple. If you can't take your IP address with you  
>>>a network container is nearly pointless from a migration perspective.  
>>>

>>>Beyond that from everything I have seen layer 2 is just much cleaner  
>>>than any layer 3 approach short of Serge's bind filtering.

>>

>>well, the 'ip subset' approach Linux-VServer and  
>>other Jail solutions use is very clean, it just does  
>>not match your expectations of a virtual interface  
>>(as there is none) and it does not cope well with  
>>all kinds of per context 'requirements', which IMHO  
>>do not really exist on the application layer (only  
>>on the whole system layer)

>

>

> I probably expressed that wrong. There are currently three  
> basic approaches under discussion.  
> Layer 3 (Basically bind filtering) nothing at the packet level.  
> The approach taken by Serge's version of bsdjails and Vserver.

>

> Layer 2.5 What Daniel proposed.

>

> Layer 2. (Trivially mapping each packet to a different interface)  
> And then treating everything as multiple instances of the  
> network stack.  
> Roughly what OpenVZ and I have implemented.

I think classifying network virtualization by Layer X is not good enough.  
OpenVZ has Layer 3 (venet) and Layer 2 (veth) implementations, but  
in both cases networking stack inside VE remains fully virtualized.

Thanks,  
Kirill

---

Containers mailing list

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [kir](#) on Wed, 06 Sep 2006 15:09:28 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Kirill Korotaev wrote:

> I think classifying network virtualization by Layer X is not good enough.  
> OpenVZ has Layer 3 (venet) and Layer 2 (veth) implementations, but  
> in both cases networking stack inside VE remains fully virtualized.  
>

Let's describe all those (three?) approaches at  
<http://wiki.openvz.org/Containers/Networking>

Everyone is able to read and contribute to that, and (I hope) we will  
come to the common understanding. I have started the article, please  
enlarge.

---

---

Subject: Re: [RFC] network namespaces  
Posted by [Herbert Poetzl](#) on Wed, 06 Sep 2006 16:56:43 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, Sep 06, 2006 at 11:10:23AM +0200, Daniel Lezcano wrote:

> Hi Herbert,

>  
> >well, the 'ip subset' approach Linux-VServer and  
> >other Jail solutions use is very clean, it just does  
> >not match your expectations of a virtual interface  
> >(as there is none) and it does not cope well with  
> >all kinds of per context 'requirements', which IMHO  
> >do not really exist on the application layer (only  
> >on the whole system layer)

> >  
> >IMHO that would be quite simple, have a 'namespace'  
> >for limiting port binds to a subset of the available  
> >ips and another one which does complete network  
> >virtualization with all the whistles and bells, IMHO  
> >most of them are orthogonal and can easily be combined

> >  
> > - full network virtualization  
> > - lightweight ip subset  
> > - both

> >  
> >IMHO this requirement only arises from the full system  
> >virtualization approach, just look at the other jail

> >solutions (solaris, bsd, ...) some of them do not even  
> >allow for more than a single ip but they work quite  
> >well when used properly ...  
>  
> As far as I see, vserver use a layer 3 solution but, when needed, the  
> veth "component", made by Nestor Pena, is used to provide a layer 2  
> virtualization. Right ?

well, no, we do not explicitly use the VETH daemon  
for networking, although some folks probably make use  
of it, mainly because if you realize that this kind  
of isolation is something different and partially  
complementary to network virtualization, you can do  
live without the layer 2 virtualization in almost  
all cases, nevertheless, for certain purposes layer  
2/3 virtualization is required and/or makes perfect  
sense

> Having the two solutions, you have certainly a lot of information  
> about use cases.

> From the point of view of vserver, can you give some  
> examples of when a layer 3 solution is better/worst than  
> a layer 2 solution ?

my point (until we have an implementation which clearly  
shows that performance is equal/better to isolation)  
is simply this:

of course, you can 'simulate' or 'construct' all the  
isolation scenarios with kernel bridging and routing  
and tricky injection/marketing of packets, but, this  
usually comes with an overhead ...

> Who wants a layer 2/3 virtualization and why ?

there are some reasons for virtualization instead of  
pure isolation (as Linux-VServer does it for now)

- context migration/snapshot (probably reason #1)
- creating network devices inside a guest  
(can help with vpn and similar)
- allowing non IP protocols (like DHCP, ICMP, etc)

the problem which arises with this kind of network  
virtualization is that you need some additional policy  
for example to avoid sending 'evil' packets and/or  
(D)DoSing one guest from another, which again adds

further overhead, so basically if you 'just' want to have network isolation, you have to do this:

- create a 'copy' of your hosts networking inside the guest (with virtual interfaces)
- assign all the same (subset) ips and this to the virtual guest interfaces
- activate some smart bridging code which 'knows' what ports can be used and/or mapped
- add policy to block unwanted connections and/or packets to/from the guest

all this sounds very intrusive and for sure (please proove me wrong here :) adds a lot of overhead to the networking itself, while a 'simple' isolation approach for IP (tcp/udp) is (almost) without any cost, certainly without overhead once a connection is established.

> These informations will be very useful.

HTH,  
Herbert

> Regards  
>  
> -- Daniel

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [kir](#) on Wed, 06 Sep 2006 17:36:17 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Herbert Poetzl wrote:

> my point (until we have an implementation which clearly  
> shows that performance is equal/better to isolation)  
> is simply this:  
>  
> of course, you can 'simulate' or 'construct' all the  
> isolation scenarios with kernel bridging and routing  
> and tricky injection/marking of packets, but, this  
> usually comes with an overhead ...  
>

Well, TANSTAAFL\*, and pretty much everything comes with an overhead. Multitasking comes with the (scheduler, context switch, CPU cache, etc.) overhead -- is that the reason to abandon it? OpenVZ and Linux-VServer resource management also adds some overhead -- do we want to throw it away?

The question is not just "equal or better performance", the question is

"what do we get and how much we pay for it".

Finally, as I understand both network isolation and network virtualization (both level2 and level3) can happily co-exist. We do have several filesystems in kernel. Let's have several network virtualization approaches, and let a user choose. Is that makes sense?

\* -- <http://en.wikipedia.org/wiki/TANSTAAFL>

---

---

Subject: Re: [RFC] network namespaces

Posted by [ebiederm](#) on Wed, 06 Sep 2006 17:58:39 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Herbert Poetzl <[herbert@13thfloor.at](mailto:herbert@13thfloor.at)> writes:

> On Wed, Sep 06, 2006 at 11:10:23AM +0200, Daniel Lezcano wrote:

>>

>> As far as I see, vserver use a layer 3 solution but, when needed, the  
>> veth "component", made by Nestor Pena, is used to provide a layer 2  
>> virtualization. Right ?

>

> well, no, we do not explicitly use the VETH daemon  
> for networking, although some folks probably make use  
> of it, mainly because if you realize that this kind  
> of isolation is something different and partially  
> complementary to network virtualization, you can do  
> live without the layer 2 virtualization in almost  
> all cases, nevertheless, for certain purposes layer  
> 2/3 virtualization is required and/or makes perfect  
> sense

>

>> Having the two solutions, you have certainly a lot if information  
>> about use cases.

>

>> From the point of view of vserver, can you give some  
>> examples of when a layer 3 solution is better/worst than  
>> a layer 2 solution ?

>

> my point (until we have an implementation which clearly  
> shows that performance is equal/better to isolation)  
> is simply this:

>

> of course, you can 'simulate' or 'construct' all the  
> isolation scenarios with kernel bridging and routing  
> and tricky injection/marking of packets, but, this  
> usually comes with an overhead ...

>  
 >> Who wants a layer 2/3 virtualization and why ?  
 >  
 > there are some reasons for virtualization instead of  
 > pure isolation (as Linux-VServer does it for now)  
 >  
 > - context migration/snapshot (probably reason #1)  
 > - creating network devices inside a guest  
 > (can help with vpn and similar)  
 > - allowing non IP protocols (like DHCP, ICMP, etc)  
 >  
 > the problem which arises with this kind of network  
 > virtualization is that you need some additional policy  
 > for example to avoid sending 'evil' packets and/or  
 > (D)DoSing one guest from another, which again adds  
 > further overhead, so basically if you 'just' want  
 > to have network isolation, you have to do this:  
 >  
 > - create a 'copy' of your hosts networking inside  
 > the guest (with virtual interfaces)  
 > - assign all the same (subset) ips and this to  
 > the virtual guest interfaces  
 > - activate some smart bridging code which 'knows'  
 > what ports can be used and/or mapped  
 > - add policy to block unwanted connections and/or  
 > packets to/from the guest  
 >  
 > all this sounds very intrusive and for sure (please  
 > prove me wrong here :) adds a lot of overhead to the  
 > networking itself, while a 'simple' isolation approach  
 > for IP (tcp/udp) is (almost) without any cost, certainly  
 > without overhead once a connection is established.

Thanks, for the good summary of the situation.

I think we can prove you wrong but it is going to take  
 some doing to build a good implementation and take  
 the necessary measurements.

Hmm. I wonder if the filtering layer 3 style of isolation can be built with  
 netfilter rules. Just skimming it looks we may be able to do it with something  
 like the netfilter owner module, possibly in conjunction with the connmark or  
 conntrack modules. If not if the infrastructure is close enough we can write  
 our own module.

Has anyone looked at network isolation from the netfilter perspective?

Eric

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Wed, 06 Sep 2006 18:34:59 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Kir Kolyshkin <[kir@openvz.org](mailto:kir@openvz.org)> writes:

> Herbert Poetzl wrote:  
>> my point (until we have an implementation which clearly  
>> shows that performance is equal/better to isolation)  
>> is simply this:  
>>  
>> of course, you can 'simulate' or 'construct' all the  
>> isolation scenarios with kernel bridging and routing  
>> and tricky injection/marking of packets, but, this  
>> usually comes with an overhead ...  
>>  
> Well, TANSTAAFL\*, and pretty much everything comes with an overhead.  
> Multitasking comes with the (scheduler, context switch, CPU cache, etc.)  
> overhead -- is that the reason to abandon it? OpenVZ and Linux-VServer  
> resource management also adds some overhead -- do we want to throw it away?  
>  
> The question is not just "equal or better performance", the question is  
> "what do we get and how much we pay for it".

Equal or better performance is certainly required when we have the code compiled in but aren't using it. We must not penalize the current code.

> Finally, as I understand both network isolation and network  
> virtualization (both level2 and level3) can happily co-exist. We do have  
> several filesystems in kernel. Let's have several network virtualization  
> approaches, and let a user choose. Is that makes sense?

If there are not compelling arguments for using both ways of doing it is silly to merge both, as it is more maintenance overhead.

That said I think there is a real chance if we can look at the bind filtering and find a way to express that in the networking stack through iptables. Using the security hooks conflicts with things like selinux. Although it would be interesting to see if selinux can already implement general purpose layer 3 filtering.

The more I look the gut feel I have is that the way to proceed would be to add a new table that filters binds, and connects. Plus a new module that would look at a process creating a socket and tell us if it is the appropriate group of processes. With a little care that would be a general solution to the layer 3 filtering problem.

Eric

---

---

Subject: Re: [RFC] network namespaces  
Posted by [kir](#) on Wed, 06 Sep 2006 18:56:43 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Eric W. Biederman wrote:

> Kir Kolyshkin <[kir@openvz.org](mailto:kir@openvz.org)> writes:

>

>

>> Herbert Poetzl wrote:

>>

>>> my point (until we have an implementation which clearly

>>> shows that performance is equal/better to isolation)

>>> is simply this:

>>>

>>> of course, you can 'simulate' or 'construct' all the

>>> isolation scenarios with kernel bridging and routing

>>> and tricky injection/marketing of packets, but, this

>>> usually comes with an overhead ...

>>>

>>>

>> Well, TANSTAAFL\*, and pretty much everything comes with an overhead.

>> Multitasking comes with the (scheduler, context switch, CPU cache, etc.)

>> overhead -- is that the reason to abandon it? OpenVZ and Linux-VServer

>> resource management also adds some overhead -- do we want to throw it away?

>>

>> The question is not just "equal or better performance", the question is

>> "what do we get and how much we pay for it".

>>

>

> Equal or better performance is certainly required when we have the code

> compiled in but aren't using it. We must not penalize the current code.

>

That's a valid argument. Although it's not applicable here (at least for both network virtualization types which OpenVZ offers). Kirill/Andrey, please correct me if I'm wrong here.

>> Finally, as I understand both network isolation and network

>> virtualization (both level2 and level3) can happily co-exist. We do have

>> several filesystems in kernel. Let's have several network virtualization

>> approaches, and let a user choose. Is that makes sense?

>>  
> 0  
> If there are not compelling arguments for using both ways of doing  
> it is silly to merge both, as it is more maintenance overhead.  
>  
Definitely a valid argument as well.

I am not sure about "network isolation" (used by Linux-VServer), but as it comes for level2 vs. level3 virtualization, I see a need for both.  
Here is the easy-to-understand comparison which can shed some light:  
[http://wiki.openvz.org/Differences\\_between\\_venet\\_and\\_veth](http://wiki.openvz.org/Differences_between_venet_and_veth)

Here are a couple of examples

- \* Do we want to let container's owner (i.e. root) to add/remove IP addresses? Most probably not, but in some cases we want that.
- \* Do we want to be able to run DHCP server and/or DHCP client inside a container? Sometimes...but not always.
- \* Do we want to let container's owner to create/manage his own set of iptables? In half of the cases we do.

The problem here is single solution will not cover all those scenarios.

> That said I think there is a real chance if we can look at the bind  
> filtering and find a way to express that in the networking stack  
> through iptables. Using the security hooks conflicts with things  
> like selinux. Although it would be interesting to see if selinux  
> can already implement general purpose layer 3 filtering.  
>  
> The more I look the gut feel I have is that the way to proceed would  
> be to add a new table that filters binds, and connects. Plus a new  
> module that would look at a process creating a socket and tell us if  
> it is the appropriate group of processes. With a little care that  
> would be a general solution to the layer 3 filtering problem.  
>  
> Eric  
>

---

Subject: Re: [RFC] network namespaces  
Posted by [Cedric Le Goater](#) on Wed, 06 Sep 2006 20:25:10 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Eric W. Biederman wrote:

>> This family of containers are used too for HPC (high performance computing) and  
>> for distributed checkpoint/restart. The cluster runs hundred of jobs, spawning  
>> them on different hosts inside an application container. Usually the jobs  
>> communicates with broadcast and multicast.  
>> Application containers does not care of having different MAC address and rely on  
>> a layer 3 approach.

>  
> Ok I think to understand this we need some precise definitions.  
> In the normal case it is an error for a job to communication with a different  
> job.

hmm ? What about an MPI application ?

I would expect each MPI task to be run in its container on different nodes  
or on the same node. These individual tasks `_communicate_` between each  
other through the MPI layer (not only TCP btw) to complete a large calculation.

> The basic advantage with a different MAC is that you can found out who the  
> intended recipient is sooner in the networking stack and you have truly  
> separate network devices. Allowing for a cleaner implementation.  
>  
> Changing the MAC after migration is likely to be fine.

indeed.

C.

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Wed, 06 Sep 2006 20:40:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

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

> Eric W. Biederman wrote:  
>  
> hmm ? What about an MPI application ?  
>  
> I would expect each MPI task to be run in its container on different nodes  
> or on the same node. These individual tasks `_communicate_` between each  
> other through the MPI layer (not only TCP btw) to complete a large calculation.

All parts of the MPI application are part of the same job. Communication between  
processes on multiple machines that are part of the job is fine.

At least that is how I used job in HPC computer context.

Eric

---

Subject: Re: [RFC] network namespaces  
Posted by [Cedric Le Goater](#) on Wed, 06 Sep 2006 20:53:08 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Kir Kolyshkin wrote:

<snip>

> I am not sure about "network isolation" (used by Linux-VServer), but as  
> it comes for level2 vs. level3 virtualization, I see a need for both.  
> Here is the easy-to-understand comparison which can shed some light:  
> [http://wiki.openvz.org/Differences\\_between\\_venet\\_and\\_veth](http://wiki.openvz.org/Differences_between_venet_and_veth)

thanks kir,

> Here are a couple of examples  
> \* Do we want to let container's owner (i.e. root) to add/remove IP  
> addresses? Most probably not, but in some cases we want that.  
> \* Do we want to be able to run DHCP server and/or DHCP client inside a  
> container? Sometimes...but not always.  
> \* Do we want to let container's owner to create/manage his own set of  
> iptables? In half of the cases we do.  
>  
> The problem here is single solution will not cover all those scenarios.

some would argue that there is one single solution : Xen or similar.

IMO, I think containers should try to leverage their difference,  
performance, and not try to simulate a real hardware environment.

Restricting the network environment of a container should be considered  
acceptable if this is for the sake of performance. The network interface(s)  
could be pre-configured and provided to the container. Protocol(s) could be  
forbidden.

Now, if you need more network power in a container, you will need a real or  
a virtualized interface.

But let's consider both alternatives.

thanks,

C.

---

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

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Daniel Lezcano](#) on Wed, 06 Sep 2006 21:44:35 GMT

---

Kir Kolyshkin wrote:

> Herbert Poetzl wrote:

>

>> my point (until we have an implementation which clearly

>> shows that performance is equal/better to isolation)

>> is simply this:

>>

>> of course, you can 'simulate' or 'construct' all the

>> isolation scenarios with kernel bridging and routing

>> and tricky injection/marking of packets, but, this

>> usually comes with an overhead ...

>>

>

> Well, TANSTAAFL\*, and pretty much everything comes with an overhead.

> Multitasking comes with the (scheduler, context switch, CPU cache, etc.)

> overhead -- is that the reason to abandon it? OpenVZ and Linux-VServer

> resource management also adds some overhead -- do we want to throw it away?

>

> The question is not just "equal or better performance", the question is

> "what do we get and how much we pay for it".

>

> Finally, as I understand both network isolation and network

> virtualization (both level2 and level3) can happily co-exist. We do have

> several filesystems in kernel. Let's have several network virtualization

> approaches, and let a user choose. Is that makes sense?

Definitely yes, I agree.

---

Subject: RE: [RFC] network namespaces

Posted by [Caitlin Bestler](#) on Wed, 06 Sep 2006 23:06:16 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

ebiederm@xmission.com wrote:

>

>> Finally, as I understand both network isolation and network

>> virtualization (both level2 and level3) can happily co-exist. We do

>> have several filesystems in kernel. Let's have several network

>> virtualization approaches, and let a user choose. Is that makes

>> sense?

>

> If there are not compelling arguments for using both ways of

> doing it is silly to merge both, as it is more maintenance overhead.

>

My reading is that full virtualization (Xen, etc.) calls for implementing L2 switching between the partitions and the physical NIC(s).

The tradeoffs between L2 and L3 switching are indeed complex, but there are two implications of doing L2 switching between partitions:

- 1) Do we really want to ask device drivers to support L2 switching for partitions and something \*different\* for containers?
- 2) Do we really want any single packet to traverse an L2 switch (for the partition-style virtualization layer) and then an L3 switch (for the container-style layer)?

The full virtualization solution calls for virtual NICs with distinct MAC addresses. Is there any reason why this same solution cannot work for containers (just creating more than one VNIC for the partition, and then assigning each VNIC to a container?)

---

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Wed, 06 Sep 2006 23:25:50 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

"Caitlin Bestler" <[caitlinb@broadcom.com](mailto:caitlinb@broadcom.com)> writes:

> ebiederm@xmission.com wrote:

>

>>

>>> Finally, as I understand both network isolation and network virtualization (both level2 and level3) can happily co-exist. We do >>> have several filesystems in kernel. Let's have several network >>> virtualization approaches, and let a user choose. Is that makes >>> sense?

>>

>> If there are not compelling arguments for using both ways of >> doing it is silly to merge both, as it is more maintenance overhead. >>

>

> My reading is that full virtualization (Xen, etc.) calls for > implementing > L2 switching between the partitions and the physical NIC(s). >

> The tradeoffs between L2 and L3 switching are indeed complex, but > there are two implications of doing L2 switching between partitions: >

- > 1) Do we really want to ask device drivers to support L2 switching for > partitions and something \*different\* for containers?

No.

- > 2) Do we really want any single packet to traverse an L2 switch (for
- > the partition-style virtualization layer) and then an L3 switch
- > (for the container-style layer)?

In general what has been done with layer 3 is to simply filter which processes can use which IP addresses and it all happens at socket creation time. So it is very cheap, and it can be done purely in the network layer without any driver intervention.

Basically think of what is happening at layer 3 as an extremely light-weight version of traffic filtering.

- > The full virtualization solution calls for virtual NICs with distinct
- > MAC addresses. Is there any reason why this same solution cannot work
- > for containers (just creating more than one VNIC for the partition,
- > and then assigning each VNIC to a container?)

The VNIC approach is the fundamental idea with the layer two networking and if we can push the work down into the device driver it so different destination macs show up a in different packet queues it should be as fast as a normal networking stack.

Implementing VNICs so far is the only piece of containers that has come close to device drivers, and we can likely do it without device driver support (but with more cost). Basically this optimization is a subset of the Grand Unified Lookup idea.

I think we can do a mergeable implementation without noticeable cost without when not using containers without having to resort to a grand unified lookup but I may be wrong.

Eric

---

Subject: Re: [RFC] network namespaces  
Posted by [Stephen Hemminger](#) on Thu, 07 Sep 2006 00:53:12 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, 06 Sep 2006 17:25:50 -0600  
ebiederm@xmission.com (Eric W. Biederman) wrote:

- > "Caitlin Bestler" <caitlinb@broadcom.com> writes:
- >
- > > ebiederm@xmission.com wrote:
- > >

> >>

> >>> Finally, as I understand both network isolation and network  
> >>> virtualization (both level2 and level3) can happily co-exist. We do  
> >>> have several filesystems in kernel. Let's have several network  
> >>> virtualization approaches, and let a user choose. Is that makes  
> >>> sense?

> >>

> >> If there are not compelling arguments for using both ways of  
> >> doing it is silly to merge both, as it is more maintenance overhead.

> >>

> >

> > My reading is that full virtualization (Xen, etc.) calls for  
> > implementing  
> > L2 switching between the partitions and the physical NIC(s).

> >

> > The tradeoffs between L2 and L3 switching are indeed complex, but  
> > there are two implications of doing L2 switching between partitions:

> >

> > 1) Do we really want to ask device drivers to support L2 switching for  
> > partitions and something \*different\* for containers?

>

> No.

>

> > 2) Do we really want any single packet to traverse an L2 switch (for  
> > the partition-style virtualization layer) and then an L3 switch  
> > (for the container-style layer)?

>

> In general what has been done with layer 3 is to simply filter which  
> processes can use which IP addresses and it all happens at socket  
> creation time. So it is very cheap, and it can be done purely  
> in the network layer without any driver intervention.

>

> Basically think of what is happening at layer 3 as an extremely light-weight  
> version of traffic filtering.

>

> > The full virtualization solution calls for virtual NICs with distinct  
> > MAC addresses. Is there any reason why this same solution cannot work  
> > for containers (just creating more than one VNIC for the partition,  
> > and then assigning each VNIC to a container?)

>

> The VNIC approach is the fundamental idea with the layer two networking  
> and if we can push the work down into the device driver it so different  
> destination macs show up a in different packet queues it should be  
> as fast as a normal networking stack.

>

> Implementing VNICs so far is the only piece of containers that has  
> come close to device drivers, and we can likely do it without device  
> driver support (but with more cost). Basically this optimization

> is a subset of the Grand Unified Lookup idea.  
>  
> I think we can do a mergeable implementation without noticeable cost without  
> when not using containers without having to resort to a grand unified lookup  
> but I may be wrong.  
>  
> Eric

The problem with VNIC's is it won't work for all devices (without lots of work), and for many device's it requires putting the device in promiscuous mode. It also plays havoc with network access control devices.

---

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Thu, 07 Sep 2006 05:11:27 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Stephen Hemminger <[shemminger@osdl.org](mailto:shemminger@osdl.org)> writes:

> The problem with VNIC's is it won't work for all devices (without lots of  
> work), and for many device's it requires putting the device in promiscuous  
> mode. It also plays havoc with network access control devices.

Which is fine. If it works it is a cool performance optimization.  
But it doesn't stop anything from my side if it doesn't.

Eric

---

---

Subject: Re: [RFC] network namespaces  
Posted by [Daniel Lezcano](#) on Thu, 07 Sep 2006 08:25:56 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Caitlin Bestler wrote:

> ebiederm@xmission.com wrote:  
>  
>  
>>>Finally, as I understand both network isolation and network  
>>>virtualization (both level2 and level3) can happily co-exist. We do  
>>>have several filesystems in kernel. Let's have several network  
>>>virtualization approaches, and let a user choose. Is that makes  
>>>sense?  
>>  
>>If there are not compelling arguments for using both ways of  
>>doing it is silly to merge both, as it is more maintenance overhead.  
>>  
>

- >
- > My reading is that full virtualization (Xen, etc.) calls for
- > implementing
- > L2 switching between the partitions and the physical NIC(s).
- >
- > The tradeoffs between L2 and L3 switching are indeed complex, but
- > there are two implications of doing L2 switching between partitions:
- >
- > 1) Do we really want to ask device drivers to support L2 switching for
- > partitions and something \*different\* for containers?
- >
- > 2) Do we really want any single packet to traverse an L2 switch (for
- > the partition-style virtualization layer) and then an L3 switch
- > (for the container-style layer)?
- >
- > The full virtualization solution calls for virtual NICs with distinct
- > MAC addresses. Is there any reason why this same solution cannot work
- > for containers (just creating more than one VNIC for the partition,
- > and then assigning each VNIC to a container?)

IHMO, I think there is one reason. The unsharing mechanism is not only for containers, its aim other kind of isolation like a "bsdjail" for example. The unshare syscall is flexible, shall the network unsharing be one-block solution ? For example, we want to launch an application using TCP/IP and we want to have an IP address only used by the application, nothing more.

With a layer 2, we must after unsharing:

- 1) create a virtual device into the application namespace
- 2) assign an IP address
- 3) create a virtual device pass-through in the root namespace
- 4) set the virtual device IP

All this stuff, need a lot of administration (check mac addresses conflicts, check interface names collision in root namespace, ...) for a simple network isolation.

With a layer 3:

- 1) assign an IP address

In the other hand, a layer 3 isolation is not sufficient to reach the level of isolation/virtualization needed for the system containers.

Very soon, I will commit more info at:

<http://wiki.openvz.org/Containers/Networking>

So the consensus is based on the fact that there is a lot of common code for the layer 2 and layer 3 isolation/virtualization and we can find a

way to merge the 2 implementation in order to have a flexible network virtualization/isolation.

-- Regards

Daniel.

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [dev](#) on Thu, 07 Sep 2006 16:20:47 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

>>Herbert Poetzl wrote:

>>

>>>my point (until we have an implementation which clearly  
>>>shows that performance is equal/better to isolation)

>>>is simply this:

>>>

>>> of course, you can 'simulate' or 'construct' all the  
>>> isolation scenarios with kernel bridging and routing  
>>> and tricky injection/marking of packets, but, this  
>>> usually comes with an overhead ...

>>>

>>

>>Well, TANSTAAFL\*, and pretty much everything comes with an overhead.

>>Multitasking comes with the (scheduler, context switch, CPU cache, etc.)

>>overhead -- is that the reason to abandon it? OpenVZ and Linux-VServer

>>resource management also adds some overhead -- do we want to throw it away?

>>

>>The question is not just "equal or better performance", the question is

>>"what do we get and how much we pay for it".

>

>

> Equal or better performance is certainly required when we have the code  
> compiled in but aren't using it. We must not penalize the current code.  
you talk about host system performance.

Both approaches do not introduce overhead to host networking.

>>Finally, as I understand both network isolation and network

>>virtualization (both level2 and level3) can happily co-exist. We do have

>>several filesystems in kernel. Let's have several network virtualization

>>approaches, and let a user choose. Is that makes sense?

>

>

> If there are not compelling arguments for using both ways of doing  
> it is silly to merge both, as it is more maintenance overhead.

>

> That said I think there is a real chance if we can look at the bind

> filtering and find a way to express that in the networking stack  
> through iptables. Using the security hooks conflicts with things  
> like selinux. Although it would be interesting to see if selinux  
> can already implement general purpose layer 3 filtering.  
>  
> The more I look the gut feel I have is that the way to proceed would  
> be to add a new table that filters binds, and connects. Plus a new  
> module that would look at a process creating a socket and tell us if  
> it is the appropriate group of processes. With a little care that  
> would be a general solution to the layer 3 filtering problem.  
Huh, you will still have to insert lots of access checks into different  
parts of code like RAW sockets, netlinks, protocols which are not inserted,  
netfilters (to not allow create iptables rules :) ) and many many other places.

I see Dave Miller looking at such a patch and my ears hear his rude words :)

Thanks,  
Kirill

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Herbert Poetzl](#) on Thu, 07 Sep 2006 17:27:59 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Thu, Sep 07, 2006 at 08:23:53PM +0400, Kirill Korotaev wrote:  
> >>Herbert Poetzl wrote:  
> >>  
> >>>my point (until we have an implementation which clearly  
> >>>shows that performance is equal/better to isolation)  
> >>>is simply this:  
> >>>  
> >>> of course, you can 'simulate' or 'construct' all the  
> >>> isolation scenarios with kernel bridging and routing  
> >>> and tricky injection/marking of packets, but, this  
> >>> usually comes with an overhead ...  
> >>>  
> >>Well, TANSTAAFL\*, and pretty much everything comes with an overhead.  
> >>Multitasking comes with the (scheduler, context switch, CPU cache, etc.)  
> >>overhead -- is that the reason to abandon it? OpenVZ and Linux-VServer  
> >>resource management also adds some overhead -- do we want to throw it away?  
> >>  
> >>The question is not just "equal or better performance", the question is  
> >>"what do we get and how much we pay for it".  
> >  
> >  
> > Equal or better performance is certainly required when we have the code  
> > compiled in but aren't using it. We must not penalize the current code.

> you talk about host system performance.  
> Both approaches do not introduce overhead to host networking.  
>  
> >> Finally, as I understand both network isolation and network  
> >> virtualization (both level2 and level3) can happily co-exist. We do have  
> >> several filesystems in kernel. Let's have several network virtualization  
> >> approaches, and let a user choose. Is that makes sense?  
> >  
> >  
> > If there are not compelling arguments for using both ways of doing  
> > it is silly to merge both, as it is more maintenance overhead.  
> >  
> > That said I think there is a real chance if we can look at the bind  
> > filtering and find a way to express that in the networking stack  
> > through iptables. Using the security hooks conflicts with things  
> > like selinux. Although it would be interesting to see if selinux  
> > can already implement general purpose layer 3 filtering.  
> >  
> > The more I look the gut feel I have is that the way to proceed would  
> > be to add a new table that filters binds, and connects. Plus a new  
> > module that would look at a process creating a socket and tell us if  
> > it is the appropriate group of processes. With a little care that  
> > would be a general solution to the layer 3 filtering problem.  
  
> Huh, you will still have to insert lots of access checks into  
> different parts of code like RAW sockets, netlinks, protocols which  
> are not inserted, netfilters (to not allow create iptables rules :) )  
> and many many other places.

well, who said that you need to have things like RAW sockets  
or other protocols except IP, not to speak of iptable and  
routing entries ...

folks who want full network virtualization can use the  
more complete virtual setup and be happy ...

best,  
Herbert

> I see Dave Miller looking at such a patch and my ears hear his rude  
> words :)  
>  
> Thanks,  
> Kirill  
> \_\_\_\_\_  
> Containers mailing list  
> Containers@lists.osdl.org  
> <https://lists.osdl.org/mailman/listinfo/containers>

---

---

Subject: Re: [RFC] network namespaces

Posted by [ebiederm](#) on Thu, 07 Sep 2006 18:29:21 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

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

>  
> IHMO, I think there is one reason. The unsharing mechanism is not only for  
> containers, its aim other kind of isolation like a "bsdjail" for example. The  
> unshare syscall is flexible, shall the network unsharing be one-block solution ?  
> For example, we want to launch an application using TCP/IP and we want to have  
> an IP address only used by the application, nothing more.  
> With a layer 2, we must after unsharing:  
> 1) create a virtual device into the application namespace  
> 2) assign an IP address  
> 3) create a virtual device pass-through in the root namespace  
> 4) set the virtual device IP  
>  
> All this stuff, need a lot of administration (check mac addresses conflicts,  
> check interface names collision in root namespace, ...) for a simple network  
> isolation.

Yes, and even more it is hard to show that it will perform as well.

Although by dropping CAP\_NET\_ADMIN the actual runtime administration  
is about the same.

> With a layer 3:  
> 1) assign an IP address  
>  
> In the other hand, a layer 3 isolation is not sufficient to reach the level of  
> isolation/virtualization needed for the system containers.

Agreed.

> Very soon, I will commit more info at:  
>  
> <http://wiki.openvz.org/Containers/Networking>  
>  
> So the consensus is based on the fact that there is a lot of common code for the  
> layer 2 and layer 3 isolation/virtualization and we can find a way to merge the  
> 2 implementation in order to have a flexible network virtualization/isolation.

NACK In a real level 3 implementation there is very little common code with  
a layer 2 implementation. You don't need to muck with the socket handling  
code as you are not allowed to dup addresses between containers. Look  
at what Serge did that is layer 3.

A layer 3 isolation implementation should either be a new security module  
or a new form of iptables. The problem with using the lsm is that it  
seems to be an all or nothing mechanism so is a very coarse grained

tool for this job.

A layer 2 implementation (where you have network devices isolated and not sockets) should be a namespace.

Eric

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Thu, 07 Sep 2006 19:50:55 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Herbert Poetzi <herbert@13thfloor.at> writes:

> On Thu, Sep 07, 2006 at 08:23:53PM +0400, Kirill Korotaev wrote:  
>  
> well, who said that you need to have things like RAW sockets  
> or other protocols except IP, not to speak of iptable and  
> routing entries ...  
>  
> folks who \_want\_ full network virtualization can use the  
> more complete virtual setup and be happy ...

Exactly this was a proposal for isolation for containers that don't get CAP\_NET\_ADMIN, with a facility that could easily be general purpose.

Eric

---

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

---

---

Subject: Re: [RFC] network namespaces  
Posted by [Herbert Poetzi](#) on Fri, 08 Sep 2006 06:02:00 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Thu, Sep 07, 2006 at 12:29:21PM -0600, Eric W. Biederman wrote:  
> Daniel Lezcano <dlezcano@fr.ibm.com> writes:  
> >  
> > IHMO, I think there is one reason. The unsharing mechanism is  
> > not only for containers, its aim other kind of isolation like a  
> > "bsdjail" for example. The unshare syscall is flexible, shall the  
> > network unsharing be one-block solution ? For example, we want to  
> > launch an application using TCP/IP and we want to have  
> > an IP address only used by the application, nothing more.

> > With a layer 2, we must after unsharing:

- > > 1) create a virtual device into the application namespace
- > > 2) assign an IP address
- > > 3) create a virtual device pass-through in the root namespace
- > > 4) set the virtual device IP

> >

> > All this stuff, need a lot of administration (check mac addresses  
> > conflicts, check interface names collision in root namespace, ...)  
> > for a simple network isolation.

>

> Yes, and even more it is hard to show that it will perform as well.  
> Although by dropping CAP\_NET\_ADMIN the actual runtime administration  
> is about the same.

>

> > With a layer 3:

- > > 1) assign an IP address

> >

> > In the other hand, a layer 3 isolation is not sufficient to reach  
> > the level of isolation/virtualization needed for the system  
> > containers.

>

> Agreed.

>

> > Very soon, I will commit more info at:

> >

> > <http://wiki.openvz.org/Containers/Networking>

> >

> > So the consensus is based on the fact that there is a lot of common  
> > code for the layer 2 and layer 3 isolation/virtualization and we can  
> > find a way to merge the 2 implementation in order to have a flexible  
> > network virtualization/isolation.

>

> NACK In a real level 3 implementation there is very little common  
> code with a layer 2 implementation. You don't need to muck with the  
> socket handling code as you are not allowed to dup addresses between  
> containers. Look at what Serge did that is layer 3.

>

> A layer 3 isolation implementation should either be a new security  
> module or a new form of iptables. The problem with using the lsm is  
> that it seems to be an all or nothing mechanism so is a very coarse  
> grained tool for this job.

IMHO LSM was never an option for that, because it is

- a) very complicated to use it for that purpose
- b) missing many hooks you definitely need to make this work
- c) is not really efficient and/or performant

with something 'like' iptables, this could be done, but

I'm not sure that is the best approach either ...

best,  
Herbert

> A layer 2 implementation (where you have network devices isolated and  
> not sockets) should be a namespace.

>

> Eric

>

---

> Containers mailing list

> Containers@lists.osdl.org

> <https://lists.osdl.org/mailman/listinfo/containers>

---

Containers mailing list

Containers@lists.osdl.org

<https://lists.osdl.org/mailman/listinfo/containers>

---

---

Subject: Re: Re: [RFC] network namespaces

Posted by [Mishin Dmitry](#) on Fri, 08 Sep 2006 13:10:08 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On Thursday 07 September 2006 21:27, Herbert Poetzl wrote:

> well, who said that you need to have things like RAW sockets

> or other protocols except IP, not to speak of iptable and

> routing entries ...

>

> folks who \_want\_ full network virtualization can use the

> more complete virtual setup and be happy ...

Let's think about how to implement this.

As I understood VServer's design, your proposal is to split CAP\_NET\_ADMIN to multiple capabilities and use them if required. So, for your light-weight container it is enough to implement context isolation for protected by CAP\_NET\_IP capability (for example) code and put 'if (!capable(CAP\_NET\_\*))' checks to all other places. But this could be easily implemented over OpenVZ code by CAP\_VE\_NET\_ADMIN split.

So, the question is:

Could you point out the places in Andrey's implementation of network namespaces, which prevents you to add CAP\_NET\_ADMIN separation later?

--

Thanks,  
Dmitry.

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Herbert Poetzl](#) on Fri, 08 Sep 2006 18:11:54 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Fri, Sep 08, 2006 at 05:10:08PM +0400, Dmitry Mishin wrote:

> On Thursday 07 September 2006 21:27, Herbert Poetzl wrote:  
> > well, who said that you need to have things like RAW sockets  
> > or other protocols except IP, not to speak of iptable and  
> > routing entries ...  
> >  
> > folks who \_want\_ full network virtualization can use the  
> > more complete virtual setup and be happy ...

> Let's think about how to implement this.

> As I understood VServer's design, your proposal is to split  
> CAP\_NET\_ADMIN to multiple capabilities and use them if required. So,  
> for your light-weight container it is enough to implement context  
> isolation for protected by CAP\_NET\_IP capability (for example) code  
> and put 'if (!capable(CAP\_NET\_\*))' checks to all other places.

actually the light-weight ip isolation runs perfectly  
fine \_without\_ CAP\_NET\_ADMIN, as you do not want the  
guest to be able to mess with the 'configured' ips at  
all (not to speak of interfaces here)

best,  
Herbert

> But this could be easily implemented over OpenVZ code by  
> CAP\_VE\_NET\_ADMIN split.  
>  
> So, the question is:  
> Could you point out the places in Andrey's implementation of network  
> namespaces, which prevents you to add CAP\_NET\_ADMIN separation later?  
>  
> --  
> Thanks,  
> Dmitry.

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Mishin Dmitry](#) on Sat, 09 Sep 2006 07:57:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Friday 08 September 2006 22:11, Herbert Poetzl wrote:

> actually the light-weight ip isolation runs perfectly  
> fine \_without\_ CAP\_NET\_ADMIN, as you do not want the

> guest to be able to mess with the 'configured' ips at

> all (not to speak of interfaces here)

It was only an example. I'm thinking about how to implement flexible solution, which permits light-weight ip isolation as well as full-fledged network virtualization. Another solution is to split CONFIG\_NET\_NAMESPACE. Is it good for you?

--

Thanks,

Dmitry.

---

Subject: Re: Re: [RFC] network namespaces

Posted by [Herbert Poetzl](#) on Sun, 10 Sep 2006 02:47:09 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On Sat, Sep 09, 2006 at 11:57:24AM +0400, Dmitry Mishin wrote:

> On Friday 08 September 2006 22:11, Herbert Poetzl wrote:

> > actually the light-weight ip isolation runs perfectly

> > fine \_without\_ CAP\_NET\_ADMIN, as you do not want the

> > guest to be able to mess with the 'configured' ips at

> > all (not to speak of interfaces here)

> It was only an example. I'm thinking about how to implement flexible

> solution, which permits light-weight ip isolation as well as

> full-fledged network virtualization. Another solution is to split

> CONFIG\_NET\_NAMESPACE. Is it good for you?

well, I think it would be best to have both, as they are complementary to some degree, and IMHO both, the full virtualization \_and\_ the isolation will require a separate namespace to work, I also think that limiting the isolation to something very simple (like one IP + network or so) would be acceptable for a start, because especially multi IP or network range checks require a little more effort to get them right ...

I do not think that folks would want to recompile their kernel just to get a light-weight guest or a fully virtualized one

best,

Herbert

> --

> Thanks,

> Dmitry.

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Sun, 10 Sep 2006 03:41:35 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Herbert Poetzl <herbert@13thfloor.at> writes:

> On Sat, Sep 09, 2006 at 11:57:24AM +0400, Dmitry Mishin wrote:  
>> On Friday 08 September 2006 22:11, Herbert Poetzl wrote:  
>> > actually the light-weight ip isolation runs perfectly  
>> > fine \_without\_ CAP\_NET\_ADMIN, as you do not want the  
>> > guest to be able to mess with the 'configured' ips at  
>> > all (not to speak of interfaces here)  
>  
>> It was only an example. I'm thinking about how to implement flexible  
>> solution, which permits light-weight ip isolation as well as  
>> full-fledged network virtualization. Another solution is to split  
>> CONFIG\_NET\_NAMESPACE. Is it good for you?  
>  
> well, I think it would be best to have both, as  
> they are complementary to some degree, and IMHO  
> both, the full virtualization \_and\_ the isolation  
> will require a separate namespace to work, I also  
> think that limiting the isolation to something  
> very simple (like one IP + network or so) would  
> be acceptable for a start, because especially  
> multi IP or network range checks require a little  
> more effort to get them right ...  
>  
> I do not think that folks would want to recompile  
> their kernel just to get a light-weight guest or  
> a fully virtualized one

I certainly agree that we are not at a point where a final decision can be made. A major piece of that is that a layer 2 approach has not shown to be without a performance penalty.

A practical question. Do the IPs assigned to guests ever get used by anything besides the guest?

Eric

---

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

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Mishin Dmitry](#) on Sun, 10 Sep 2006 07:45:35 GMT

On Sunday 10 September 2006 06:47, Herbert Poetzl wrote:

> well, I think it would be best to have both, as  
> they are complementary to some degree, and IMHO  
> both, the full virtualization \_and\_ the isolation  
> will require a separate namespace to work,  
[snip]

> I do not think that folks would want to recompile  
> their kernel just to get a light-weight guest or  
> a fully virtualized one

In this case light-weight guest will have unnecessary overhead.

For example, instead of using static pointer, we have to find the required common namespace before. And there will be no advantages for such guest over full-featured.

>  
> best,  
> Herbert  
>  
> > --  
> > Thanks,  
> > Dmitry.

--  
Thanks,  
Dmitry.

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Mishin Dmitry](#) on Sun, 10 Sep 2006 08:11:35 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Sunday 10 September 2006 07:41, Eric W. Biederman wrote:

> I certainly agree that we are not at a point where a final decision  
> can be made. A major piece of that is that a layer 2 approach has  
> not shown to be without a performance penalty.  
But it is required. Why to limit possible usages?

> A practical question. Do the IPs assigned to guests ever get used  
> by anything besides the guest?  
In case of level2 virtualization - no.

--  
Thanks,  
Dmitry.

---

Containers mailing list

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Sun, 10 Sep 2006 11:48:02 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Dmitry Mishin <dim@openvz.org> writes:

> On Sunday 10 September 2006 07:41, Eric W. Biederman wrote:  
>> I certainly agree that we are not at a point where a final decision  
>> can be made. A major piece of that is that a layer 2 approach has  
>> not shown to be without a performance penalty.  
> But it is required. Why to limit possible usages?

Wrong perspective.

The point is that we need to dig in and show that there is no measurable penalty for the current cases. Showing that there is little penalty for the advanced configurations is a plus.

The practical question is, do we need to implement the grand unified lookup before we can do this cheaply, or can we implement this without needing that optimization?

To get a perspective, to get a good implementation of the pid namespace I am having to refactor significant parts of the kernel so it uses abstractions that can cleanly express what we are doing. The networking stack is in better shape but there is a lot of it.

>> A practical question. Do the IPs assigned to guests ever get used  
>> by anything besides the guest?  
> In case of level2 virtualization - no.

Actually that is one of the benefits of a layer 2 implementation you can set up weird things like shared IPs, that various types of fail over scenarios want.

My question was really about the layer 3 bind filtering techniques, and how people are using them.

The basic attraction with layer 3 is that you can do a simple implementation, and it will run very fast, and it doesn't need to conflict with the layer 2 work at all. If you can make that layer 3 implementation clean and generally mergeable as well it is worth pursuing.

Eric

---

Containers mailing list

Containers@lists.osdl.org

<https://lists.osdl.org/mailman/listinfo/containers>

---

---

Subject: Re: Re: [RFC] network namespaces

Posted by [Herbert Poetzl](#) on Sun, 10 Sep 2006 19:19:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On Sat, Sep 09, 2006 at 09:41:35PM -0600, Eric W. Biederman wrote:

> Herbert Poetzl <herbert@13thfloor.at> writes:

>

> > On Sat, Sep 09, 2006 at 11:57:24AM +0400, Dmitry Mishin wrote:

> >> On Friday 08 September 2006 22:11, Herbert Poetzl wrote:

> >> > actually the light-weight ip isolation runs perfectly

> >> > fine \_without\_ CAP\_NET\_ADMIN, as you do not want the

> >> > guest to be able to mess with the 'configured' ips at

> >> > all (not to speak of interfaces here)

> >

> >> It was only an example. I'm thinking about how to implement flexible

> >> solution, which permits light-weight ip isolation as well as

> >> full-fledged network virtualization. Another solution is to split

> >> CONFIG\_NET\_NAMESPACE. Is it good for you?

> >

> > well, I think it would be best to have both, as

> > they are complementary to some degree, and IMHO

> > both, the full virtualization \_and\_ the isolation

> > will require a separate namespace to work, I also

> > think that limiting the isolation to something

> > very simple (like one IP + network or so) would

> > be acceptable for a start, because especially

> > multi IP or network range checks require a little

> > more effort to get them right ...

> >

> > I do not think that folks would want to recompile

> > their kernel just to get a light-weight guest or

> > a fully virtualized one

>

> I certainly agree that we are not at a point where a final decision

> can be made. A major piece of that is that a layer 2 approach has

> not shown to be without a performance penalty.

>

> A practical question. Do the IPs assigned to guests ever get used

> by anything besides the guest?

only in special setups and for testing routing and

general operation of course, i.e. one typical failure scenario is this:

- 'provider' has a bunch of ips assigned
- 'host' ip works perfectly
- 'guest' ip is not routed (by the external router)

in this case, for example, I always suggest to test on the host with a guest ip, simplest example:

```
ping -I <guest-ip> google.com
```

but for 'normal' operation, the guest ip is reserved for the guests, unless some service like named is shared between guests ...

HTH,  
Herbert

> Eric

---

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

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Herbert Poetzl](#) on Sun, 10 Sep 2006 19:22:44 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Sun, Sep 10, 2006 at 11:45:35AM +0400, Dmitry Mishin wrote:

> On Sunday 10 September 2006 06:47, Herbert Poetzl wrote:

> > well, I think it would be best to have both, as  
> > they are complementary to some degree, and IMHO  
> > both, the full virtualization \_and\_ the isolation  
> > will require a separate namespace to work,  
> [snip]  
> > I do not think that folks would want to recompile  
> > their kernel just to get a light-weight guest or  
> > a fully virtualized one

> In this case light-weight guest will have unnecessary overhead. For  
> example, instead of using static pointer, we have to find the required  
> common namespace before.

this is only required at 'bind' time, which makes  
a non measurable fraction of the actual connection  
usage (unless you keep binding ports over and over

without ever using them)

> And there will be no advantages for such guest over full-featured.

the advantage is in the flexibility, simplicity of  
setup and the basically non-existent overhead on  
the hot (connection/transfer) part ...

> > best,  
> > Herbert  
> >  
> > > --  
> > > Thanks,  
> > > Dmitry.  
>  
> --  
> Thanks,  
> Dmitry.

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Daniel Lezcano](#) on Mon, 11 Sep 2006 14:40:59 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Dmitry Mishin wrote:

> On Friday 08 September 2006 22:11, Herbert Poetzl wrote:  
>  
>> actually the light-weight ip isolation runs perfectly  
>> fine \_without\_ CAP\_NET\_ADMIN, as you do not want the  
>> guest to be able to mess with the 'configured' ips at  
>> all (not to speak of interfaces here)  
>  
> It was only an example. I'm thinking about how to implement flexible solution,  
> which permits light-weight ip isolation as well as full-fledged network  
> virtualization. Another solution is to split CONFIG\_NET\_NAMESPACE. Is it good  
> for you?

Hi Dmitry,

I am currently working on this and I am finishing a prototype bringing  
isolation at the ip layer. The prototype code is very close to Andrey's  
patches at TCP/UDP level. So the next step is to merge the prototype  
code with the existing network namespace layer 2 isolation.

IHMO, the solution of splitting CONFIG\_NET\_NS into CONFIG\_L2\_NET\_NS and  
CONFIG\_L3\_NET\_NS is for me not acceptable because you will need to  
recompile the kernel. The proper way is certainly to have a specific  
flag for the unshare, something like CLONE\_NEW\_L2\_NET and

CLONE\_NEW\_L3\_NET for example.

-- Daniel

---

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

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Herbert Poetzl](#) on Mon, 11 Sep 2006 14:57:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Mon, Sep 11, 2006 at 04:40:59PM +0200, Daniel Lezcano wrote:

> Dmitry Mishin wrote:

> > On Friday 08 September 2006 22:11, Herbert Poetzl wrote:

> >

> >> actually the light-weight ip isolation runs perfectly

> >> fine \_without\_ CAP\_NET\_ADMIN, as you do not want the

> >> guest to be able to mess with the 'configured' ips at

> >> all (not to speak of interfaces here)

> >

> > It was only an example. I'm thinking about how to implement flexible

> > solution, which permits light-weight ip isolation as well as full-fledged

> > network virtualization. Another solution is to split CONFIG\_NET\_NAMESPACE.

> > Is it good for you?

>

> Hi Dmitry,

>

> I am currently working on this and I am finishing a prototype bringing

> isolation at the ip layer. The prototype code is very closed to

> Andrey's patches at TCP/UDP level. So the next step is to merge the

> prototype code with the existing network namespace layer 2 isolation.

you might want to take a look at the current Linux-VServer  
implementation for the network isolation too, should be  
quite similar to Andrey's approach, but maybe you can  
gather some additional information from there

> IHMO, the solution of splitting CONFIG\_NET\_NS into CONFIG\_L2\_NET\_NS

> and CONFIG\_L3\_NET\_NS is for me not acceptable because you will need

> to recompile the kernel. The proper way is certainly to have a

> specific flag for the unshare, something like CLONE\_NEW\_L2\_NET and

> CLONE\_NEW\_L3\_NET for example.

I completely agree here, we need a separate namespace  
for that, so that we can combine isolation and virtualization

as needed, unless the bind restrictions can be completely expressed with an additional mangle or filter table (as was suggested)

best,  
Herbert

> -- Daniel

---

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

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Daniel Lezcano](#) on Mon, 11 Sep 2006 15:04:38 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Herbert Poetzl wrote:

> On Mon, Sep 11, 2006 at 04:40:59PM +0200, Daniel Lezcano wrote:  
>

>>I am currently working on this and I am finishing a prototype bringing  
>>isolation at the ip layer. The prototype code is very closed to  
>>Andrey's patches at TCP/UDP level. So the next step is to merge the  
>>prototype code with the existing network namespace layer 2 isolation.

>

>

> you might want to take a look at the current Linux-VServer  
> implementation for the network isolation too, should be  
> quite similar to Andrey's approach, but maybe you can  
> gather some additional information from there

ok, thanks. I will.

>>IHMO, the solution of splitting CONFIG\_NET\_NS into CONFIG\_L2\_NET\_NS  
>>and CONFIG\_L3\_NET\_NS is for me not acceptable because you will need  
>>to recompile the kernel. The proper way is certainly to have a  
>>specific flag for the unshare, something like CLONE\_NEW\_L2\_NET and  
>>CLONE\_NEW\_L3\_NET for example.

>

>

> I completely agree here, we need a separate namespace  
> for that, so that we can combine isolation and virtualization  
> as needed, unless the bind restrictions can be completely  
> expressed with an additional mangle or filter table (as  
> was suggested)

What is the bind restriction ? Do you want to force binding to a specific source address ?

-- Daniel

---

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

---

---

Subject: Re: Re: [RFC] network namespaces  
Posted by [Mishin Dmitry](#) on Mon, 11 Sep 2006 15:10:31 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Monday 11 September 2006 18:57, Herbert Poetzl wrote:  
> I completely agree here, we need a separate namespace  
> for that, so that we can combine isolation and virtualization  
> as needed, unless the bind restrictions can be completely  
> expressed with an additional mangle or filter table (as  
> was suggested)  
iptables are designed for packet flow decisions and filtering, it has nothing common with bind restrictions. So, it may be only packet flow scheduling/filtering, but it will not help to resolve bind-time IP conflicts.

--  
Thanks,  
Dmitry.

---

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

---

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Tue, 12 Sep 2006 03:26:52 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Dmitry Mishin <[dim@openvz.org](mailto:dim@openvz.org)> writes:

> On Sunday 10 September 2006 06:47, Herbert Poetzl wrote:  
>> well, I think it would be best to have both, as  
>> they are complementary to some degree, and IMHO  
>> both, the full virtualization \_and\_ the isolation  
>> will require a separate namespace to work,  
> [snip]  
>> I do not think that folks would want to recompile  
>> their kernel just to get a light-weight guest or

>> a fully virtualized one  
> In this case light-weight guest will have unnecessary overhead.  
> For example, instead of using static pointer, we have to find the required  
> common namespace before. And there will be no advantages for such guest over  
> full-featured.

Dmitry that just isn't true if implemented properly.

Eric

---

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

---

---

Subject: Re: [RFC] network namespaces  
Posted by [ebiederm](#) on Tue, 12 Sep 2006 03:28:47 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Dmitry Mishin <dim@openvz.org> writes:

> On Monday 11 September 2006 18:57, Herbert Poetzl wrote:  
>> I completely agree here, we need a separate namespace  
>> for that, so that we can combine isolation and virtualization  
>> as needed, unless the bind restrictions can be completely  
>> expressed with an additional mangle or filter table (as  
>> was suggested)  
>  
> iptables are designed for packet flow decisions and filtering, it has nothing  
> common with bind restrictions. So, it may be only packet flow  
> scheduling/filtering, but it will not help to resolve bind-time IP conflicts.

Please read the archive, where the suggestion was made.

What was suggested was a new table, with it's own set of chains.  
So we could make filtering decisions on where sockets could be bound.

That is not a far stretch from where iptables is today.

Do you have some concrete arguments against the proposal?

Eric

---

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

---

Subject: Re: [RFC] network namespaces

Posted by [Mishin Dmitry](#) on Tue, 12 Sep 2006 07:38:16 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Sorry, don't understand your proposal correctly from the previous talk. :)  
But...

On Tuesday 12 September 2006 07:28, Eric W. Biederman wrote:

> Do you have some concrete arguments against the proposal?

Yes, I have. I think it is unnecessary complication. This complication will followed in additional bugs. Especially if we'll accept rules creation in userspace. Why we need complex solution, if there are only two approaches to socket bound - isolation and virtualization? These approaches could co-exist without hooks. Or you probably have thoughts about other ways?

--

Thanks,  
Dmitry.

---

Containers mailing list

[Containers@lists.osdl.org](mailto:Containers@lists.osdl.org)

<https://lists.osdl.org/mailman/listinfo/containers>

---

---

Subject: Re: [PATCH 4/9] network namespaces: socket hashes

Posted by [Daniel Lezcano](#) on Mon, 18 Sep 2006 15:12:49 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Andrey Savochkin wrote:

> Socket hash lookups are made within namespace.

> Hash tables are common for all namespaces, with

> additional permutation of indexes.

Hi Andrey,

why is the hash table common and not instanciated multiple times for each namespace like the routes ?

---

---

Subject: Re: [PATCH 4/9] network namespaces: socket hashes

Posted by [Andrey Savochkin](#) on Wed, 20 Sep 2006 16:32:22 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Hi,

On Mon, Sep 18, 2006 at 05:12:49PM +0200, Daniel Lezcano wrote:

> Andrey Savochkin wrote:

> > Socket hash lookups are made within namespace.

> > Hash tables are common for all namespaces, with  
> > additional permutation of indexes.  
>  
> Hi Andrey,  
>  
> why is the hash table common and not instantiated multiple times for  
> each namespace like the routes ?

The main reason is that socket hash tables should be large enough to work efficiently, but it isn't good to waste a lot of memory for each namespace. Namespaces should be cheap enough, to allow to have hundreds of them. This reason of memory efficiency, of course, has a priority unless/until socket hash tables start to resize automatically.

Another point is that routing lookup is much more complicated than the socket's one to add another search key.  
Routing also have additional routines for deleting entries matching some patterns, and so on.  
In short, routing is much more complicated, and it already quite efficient for various sizes of routing tables.

Andrey

---

---

Subject: Re: [PATCH 4/9] network namespaces: socket hashes  
Posted by [Daniel Lezcano](#) on Thu, 21 Sep 2006 12:34:53 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Andrey Savochkin wrote:

> The main reason is that socket hash tables should be large enough to work  
> efficiently, but it isn't good to waste a lot of memory for each namespace.  
> Namespaces should be cheap enough, to allow to have hundreds of them.  
> This reason of memory efficiency, of course, has a priority unless/until  
> socket hash tables start to resize automatically.  
>  
> Another point is that routing lookup is much more complicated than the  
> socket's one to add another search key.  
> Routing also have additional routines for deleting entries matching some  
> patterns, and so on.  
> In short, routing is much more complicated, and it already quite efficient  
> for various sizes of routing tables.

That makes sense, thx for the explanation.

Cheers.  
-- Daniel.

---

---

Subject: Re: [PATCH 5/9] network namespaces: async socket operations

Posted by [Daniel Lezcano](#) on Fri, 22 Sep 2006 15:33:56 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Andrey Savochkin wrote:

> Non-trivial part of socket namespaces: asynchronous events

> should be run in proper context.

>

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

> ---

> af\_inet.c | 10 ++++++++

> inet\_timewait\_sock.c | 8 ++++++

> tcp\_timer.c | 9 ++++++++

> 3 files changed, 27 insertions(+)

>

> --- ./net/ipv4/af\_inet.c.venssock-asyn Mon Aug 14 17:04:07 2006

> +++ ./net/ipv4/af\_inet.c Tue Aug 15 13:45:44 2006

> @@ -366,10 +366,17 @@ out\_rcu\_unlock:

> int inet\_release(struct socket \*sock)

> {

> struct sock \*sk = sock->sk;

> + struct net\_namespace \*ns, \*orig\_net\_ns;

>

> if (sk) {

> long timeout;

>

> + /\* Need to change context here since protocol ->close

> + \* operation may send packets.

> + \*/

> + ns = get\_net\_ns(sk->sk\_net\_ns);

> + push\_net\_ns(ns, orig\_net\_ns);

> +

Is it not a race condition here ? What happens if you have a packet incoming during the namespace context switching ?

IHMO doing namespace switching is something dangerous, you can probably handle that with locks but it will be difficult and will decrease all network performance.

In an other hand, I don't see how you can handle the "sk->sk\_prot->close" after ...

-- Cheers

---

Subject: Re: [PATCH 5/9] network namespaces: async socket operations

Posted by [Andrey Savochkin](#) on Sat, 23 Sep 2006 13:16:43 GMT

On Fri, Sep 22, 2006 at 05:33:56PM +0200, Daniel Lezcano wrote:

```
> Andrey Savochkin wrote:
> > Non-trivial part of socket namespaces: asynchronous events
> > should be run in proper context.
> >
> > Signed-off-by: Andrey Savochkin <saw@swsoft.com>
> > ---
> > af_inet.c          | 10 ++++++++
> > inet_timewait_sock.c |  8 ++++++++
> > tcp_timer.c         |  9 ++++++++
> > 3 files changed, 27 insertions(+)
> >
> > --- ./net/ipv4/af_inet.c.venussock-asyn Mon Aug 14 17:04:07 2006
> > +++ ./net/ipv4/af_inet.c Tue Aug 15 13:45:44 2006
> > @@ -366,10 +366,17 @@ out_rcu_unlock:
> > int inet_release(struct socket *sock)
> > {
> >     struct sock *sk = sock->sk;
> > + struct net_namespace *ns, *orig_net_ns;
> >
> >     if (sk) {
> >         long timeout;
> >
> > + /* Need to change context here since protocol ->close
> > + * operation may send packets.
> > + */
> > + ns = get_net_ns(sk->sk_net_ns);
> > + push_net_ns(ns, orig_net_ns);
> > +
>
> Is it not a race condition here ? What happens if you have a packet
> incoming during the namespace context switching ?
```

All asynchronous operations (RX softirq, timers) should set their context explicitly, and can't rely on the current context being the right one (or a valid pointer at all).

Andrey

---

---

Subject: Re: [RFC] network namespaces  
Posted by [Daniel Lezcano](#) on Wed, 04 Oct 2006 09:40:32 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Andrey Savochkin wrote:  
> Hi All,

- >
- > I'd like to resurrect our discussion about network namespaces.
- > In our previous discussions it appeared that we have rather polar concepts
- > which seemed hard to reconcile.
- > Now I have an idea how to look at all discussed concepts to enable everyone's
- > usage scenario.

Hi Andrey,

I have a few questions ... sorry for asking so late ;)

- >
- > 1. The most straightforward concept is complete separation of namespaces,
- > covering device list, routing tables, netfilter tables, socket hashes, and
- > everything else.
- >
- > On input path, each packet is tagged with namespace right from the
- > place where it appears from a device, and is processed by each layer
- > in the context of this namespace.

If you have the namespace where is coming the packet, why do you tag the packet instead of switching to the right namespace ?

- > Non-root namespaces communicate with the outside world in two ways: by
- > owning hardware devices, or receiving packets forwarded them by their parent
- > namespace via pass-through device.

Do you will do proxy arp and ip forwarding into the root namespace in order to make non-root namespace visible from the outside world ?

Regards.

-- Daniel