
Subject: [PATCH] net: Basic network namespace infrastructure.

Posted by [ebiederm](#) on Fri, 22 Jun 2007 21:22:47 GMT

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

This is the basic infrastructure needed to support network namespaces. This infrastructure is:

- Registration functions to support initializing per network namespace data when a network namespaces is created or destroyed.
- struct net. The network namespace datastructure.
This structure will grow as variables are made per network namespace but this is the minimal starting point.
- Functions to grab a reference to the network namespace.
I provide both get/put functions that keep a network namespace from being freed. And hold/release functions serve as weak references and will warn if their count is not zero when the data structure is freed. Useful for dealing with more complicated data structures like the ipv4 route cache.
- A list of all of the network namespaces so we can iterate over them.
- A slab for the network namespace data structures allowing leaks to be spotted.

I have deliberately chosen to not make it possible to compile out the code as the support for per-network namespace initialization and uninitialization needs to always be compiled in once code has started using it (even if we don't have network namespaces, and because no one has ever measured any performance overhead specific to network namespace infrastructure. As code to compile out the network namespace pointers etc is complicated it is best to avoid that code unless that complexity is justified.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
include/net/net_namespace.h | 66 ++++++++
net/core/Makefile           | 2 +-
net/core/net_namespace.c    | 291 +++++++++++++++++++++++++++++++++++++
3 files changed, 358 insertions(+), 1 deletions(-)
create mode 100644 include/net/net_namespace.h
create mode 100644 net/core/net_namespace.c
```

```
diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
new file mode 100644
index 0000000..c909b3a
--- /dev/null
+++ b/include/net/net_namespace.h
```

```

@@ -0,0 +1,66 @@
+/*
+ * Operations on the network namespace
+ */
+#ifndef __NET_NET_NAMESPACE_H
+#define __NET_NET_NAMESPACE_H
+
+#include <asm/atomic.h>
+#include <linux/workqueue.h>
+#include <linux/list.h>
+
+struct net {
+ atomic_t count; /* To decided when the network namespace
+  * should go
+  */
+ atomic_t use_count; /* For references we destroy on demand */
+ struct list_head list; /* list of network namespace structures */
+ struct work_struct work; /* work struct for freeing */
+};
+
+extern struct net init_net;
+extern struct list_head net_namespace_list;
+
+extern void __put_net(struct net *net);
+
+static inline struct net *get_net(struct net *net)
+{
+ atomic_inc(&net->count);
+ return net;
+}
+
+static inline void put_net(struct net *net)
+{
+ if (atomic_dec_and_test(&net->count))
+ __put_net(net);
+}
+
+static inline struct net *hold_net(struct net *net)
+{
+ atomic_inc(&net->use_count);
+ return net;
+}
+
+static inline void release_net(struct net *net)
+{
+ atomic_dec(&net->use_count);
+}
+

```

```

+extern void net_lock(void);
+extern void net_unlock(void);
+
+#define for_each_net(VAR) \
+ list_for_each_entry(VAR, &net_namespace_list, list);
+
+
+struct pernet_operations {
+ struct list_head list;
+ int (*init)(struct net *net);
+ void (*exit)(struct net *net);
+};
+
+extern int register_pernet_subsys(struct pernet_operations *);
+extern void unregister_pernet_subsys(struct pernet_operations *);
+extern int register_pernet_device(struct pernet_operations *);
+extern void unregister_pernet_device(struct pernet_operations *);
+
+#endif /* __NET_NET_NAMESPACE_H */
diff --git a/net/core/Makefile b/net/core/Makefile
index 4751613..ea9b3f3 100644
--- a/net/core/Makefile
+++ b/net/core/Makefile
@@ -3,7 +3,7 @@
#

obj-y := sock.o request_sock.o skbuff.o iovector.o datagram.o stream.o scm.o \
- gen_stats.o gen_estimator.o
+ gen_stats.o gen_estimator.o net_namespace.o

obj-$(CONFIG_SYSCTL) += sysctl_net_core.o

diff --git a/net/core/net_namespace.c b/net/core/net_namespace.c
new file mode 100644
index 0000000..397c15f
--- /dev/null
+++ b/net/core/net_namespace.c
@@ -0,0 +1,291 @@
+#include <linux/workqueue.h>
+#include <linux/rtnetlink.h>
+#include <linux/cache.h>
+#include <linux/slab.h>
+#include <linux/list.h>
+#include <linux/delay.h>
+#include <net/net_namespace.h>
+
+/*
+ * Our network namespace constructor/destructor lists

```

```

+ */
+
+static LIST_HEAD(pernet_list);
+static struct list_head *first_device = &pernet_list;
+static DEFINE_MUTEX(net_mutex);
+
+static DEFINE_MUTEX(net_list_mutex);
+LIST_HEAD(net_namespace_list);
+
+static struct kmem_cache *net_cachep;
+
+struct net init_net;
+
+void net_lock(void)
+{
+ mutex_lock(&net_list_mutex);
+}
+
+void net_unlock(void)
+{
+ mutex_unlock(&net_list_mutex);
+}
+
+static struct net *net_alloc(void)
+{
+ return kmem_cache_alloc(net_cachep, GFP_KERNEL);
+}
+
+static void net_free(struct net *net)
+{
+ if (!net)
+ return;
+
+ if (unlikely(atomic_read(&net->use_count) != 0)) {
+ printk(KERN_EMERG "network namespace not free! Usage: %d\n",
+ atomic_read(&net->use_count));
+ return;
+ }
+
+ kmem_cache_free(net_cachep, net);
+}
+
+static void cleanup_net(struct work_struct *work)
+{
+ struct pernet_operations *ops;
+ struct list_head *ptr;
+ struct net *net;
+

```

```

+ net = container_of(work, struct net, work);
+
+ mutex_lock(&net_mutex);
+
+ /* Don't let anyone else find us. */
+ net_lock();
+ list_del(&net->list);
+ net_unlock();
+
+ /* Run all of the network namespace exit methods */
+ list_for_each_prev(ptr, &pernet_list) {
+ ops = list_entry(ptr, struct pernet_operations, list);
+ if (ops->exit)
+ ops->exit(net);
+ }
+
+ mutex_unlock(&net_mutex);
+
+ /* Ensure there are no outstanding rcu callbacks using this
+ * network namespace.
+ */
+ rcu_barrier();
+
+ /* Finally it is safe to free my network namespace structure */
+ net_free(net);
+}
+
+void __put_net(struct net *net)
+{
+ /* Cleanup the network namespace in process context */
+ INIT_WORK(&net->work, cleanup_net);
+ schedule_work(&net->work);
+}
+EXPORT_SYMBOL_GPL(__put_net);
+
+/*
+ * setup_net runs the initializers for the network namespace object.
+ */
+static int setup_net(struct net *net)
+{
+ /* Must be called with net_mutex held */
+ struct pernet_operations *ops;
+ struct list_head *ptr;
+ int error;
+
+ memset(net, 0, sizeof(struct net));
+ atomic_set(&net->count, 1);

```

```

+ atomic_set(&net->use_count, 0);
+
+ error = 0;
+ list_for_each(ptr, &pernet_list) {
+ ops = list_entry(ptr, struct pernet_operations, list);
+ if (ops->init) {
+ error = ops->init(net);
+ if (error < 0)
+ goto out_undo;
+ }
+ }
+out:
+ return error;
+out_undo:
+ /* Walk through the list backwards calling the exit functions
+  * for the pernet modules whose init functions did not fail.
+  */
+ for (ptr = ptr->prev; ptr != &pernet_list; ptr = ptr->prev) {
+ ops = list_entry(ptr, struct pernet_operations, list);
+ if (ops->exit)
+ ops->exit(net);
+ }
+ goto out;
+}
+
+static int __init net_ns_init(void)
+{
+ int err;
+
+ printk(KERN_INFO "net_namespace: %zd bytes\n", sizeof(struct net));
+ net_cache = kmem_cache_create("net_namespace", sizeof(struct net),
+ SMP_CACHE_BYTES,
+ SLAB_PANIC, NULL, NULL);
+ mutex_lock(&net_mutex);
+ err = setup_net(&init_net);
+
+ net_lock();
+ list_add_tail(&init_net.list, &net_namespace_list);
+ net_unlock();
+
+ mutex_unlock(&net_mutex);
+ if (err)
+ panic("Could not setup the initial network namespace");
+
+ return 0;
+}
+
+pure_initcall(net_ns_init);

```

```

+
+static int register_pernet_operations(struct list_head *list,
+    struct pernet_operations *ops)
+{
+ struct net *net, *undo_net;
+ int error;
+
+ error = 0;
+ list_add_tail(&ops->list, list);
+ for_each_net(net) {
+ if (ops->init) {
+ error = ops->init(net);
+ if (error)
+ goto out_undo;
+ }
+ }
+out:
+ return error;
+
+out_undo:
+ /* If I have an error cleanup all namespaces I initialized */
+ list_del(&ops->list);
+ for_each_net(undo_net) {
+ if (undo_net == net)
+ goto undone;
+ if (ops->exit)
+ ops->exit(undo_net);
+ }
+undone:
+ goto out;
+}
+
+static void unregister_pernet_operations(struct pernet_operations *ops)
+{
+ struct net *net;
+
+ list_del(&ops->list);
+ for_each_net(net)
+ if (ops->exit)
+ ops->exit(net);
+}
+
+/**
+ * register_pernet_subsys - register a network namespace subsystem
+ * @ops: pernet operations structure for the subsystem
+ *
+ * Register a subsystem which has init and exit functions
+ * that are called when network namespaces are created and

```

```

+ * destroyed respectively.
+ *
+ * When registered all network namespace init functions are
+ * called for every existing network namespace. Allowing kernel
+ * modules to have a race free view of the set of network namespaces.
+ *
+ * When a new network namespace is created all of the init
+ * methods are called in the order in which they were registered.
+ *
+ * When a network namespace is destroyed all of the exit methods
+ * are called in the reverse of the order with which they were
+ * registered.
+ */
+int register_pernet_subsys(struct pernet_operations *ops)
+{
+ int error;
+ mutex_lock(&net_mutex);
+ error = register_pernet_operations(first_device, ops);
+ mutex_unlock(&net_mutex);
+ return error;
+}
+EXPORT_SYMBOL_GPL(register_pernet_subsys);
+
+/**
+ *   unregister_pernet_subsys - unregister a network namespace subsystem
+ * @ops: pernet operations structure to manipulate
+ *
+ * Remove the pernet operations structure from the list to be
+ * used when network namespaces are created or destroyed. In
+ * addition run the exit method for all existing network
+ * namespaces.
+ */
+void unregister_pernet_subsys(struct pernet_operations *module)
+{
+ mutex_lock(&net_mutex);
+ unregister_pernet_operations(module);
+ mutex_unlock(&net_mutex);
+}
+EXPORT_SYMBOL_GPL(unregister_pernet_subsys);
+
+/**
+ *   register_pernet_device - register a network namespace device
+ * @ops: pernet operations structure for the subsystem
+ *
+ * Register a device which has init and exit functions
+ * that are called when network namespaces are created and
+ * destroyed respectively.
+ *

```



```

+ * When registered all network namespace init functions are
+ * called for every existing network namespace. Allowing kernel
+ * modules to have a race free view of the set of network namespaces.
+ *
+ * When a new network namespace is created all of the init
+ * methods are called in the order in which they were registered.
+ *
+ * When a network namespace is destroyed all of the exit methods
+ * are called in the reverse of the order with which they were
+ * registered.
+ */
+int register_pernet_device(struct pernet_operations *ops)
+{
+ int error;
+ mutex_lock(&net_mutex);
+ error = register_pernet_operations(&pernet_list, ops);
+ if (!error && (first_device == &pernet_list))
+ first_device = &ops->list;
+ mutex_unlock(&net_mutex);
+ return error;
+}
+EXPORT_SYMBOL_GPL(register_pernet_device);
+
+/**
+ *   unregister_pernet_device - unregister a network namespace netdevice
+ * @ops: pernet operations structure to manipulate
+ *
+ * Remove the pernet operations structure from the list to be
+ * used when network namespaces are created or destroyed. In
+ * addition run the exit method for all existing network
+ * namespaces.
+ */
+void unregister_pernet_device(struct pernet_operations *ops)
+{
+ mutex_lock(&net_mutex);
+ if (&ops->list == first_device)
+ first_device = first_device->next;
+ unregister_pernet_operations(ops);
+ mutex_unlock(&net_mutex);
+}
+EXPORT_SYMBOL_GPL(unregister_pernet_device);
--
1.5.1.1.181.g2de0

```