Subject: [PATCH 2/14][NETNS]: Generic per-net pointers.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:43:30 GMT
View Forum Message <> Reply to Message

Add the elastic array of void * pointer to the struct net.
The access rules are simple:

1. register the ops with register_pernet_gen_device to get
   the id of your private pointer
2. call net_assign_generic() to put the private data on the
   struct net (most preferably this should be done in the
   ->init callback of the ops registered)
3. do not change this pointer while the net is alive;
4. use the net_generic() to get the pointer.

When adding a new pointer, I copy the old array, replace it
with a new one and schedule the old for kfree after an RCU
grace period.

Since the net_generic explores the net->gen array inside rcu
read section and once set the net->gen->ptr[x] pointer never
changes, this grants us a safe access to generic pointers.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 include/net/net_namespace.h |    2 +
 include/net/netns/generic.h |   49 ++++++++++++++++++++++++++++++++++++++++++
 net/core/net_namespace.c    |   62 ++++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 113 insertions(+), 0 deletions(-)
 create mode 100644 include/net/netns/generic.h

diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
index 6971fdb..e3d4eb4 100644
--- a/include/net/net_namespace.h
+++ b/include/net/net_namespace.h
@@ -19,6 +19,7 @@ struct proc_dir_entry;
 struct net_device;
 struct sock;
 struct ctl_table_header;
+struct net_generic;

 struct net {
  atomic_t  count;  /* To decided when the network
@@ -57,6 +58,7 @@ struct net {
 #ifdef CONFIG_NETFILTER
  struct netns_xt  xt;
 #endif

```
+ struct net_generic *gen;
 };


diff --git a/include/net/netns/generic.h b/include/net/netns/generic.h
new file mode 100644
index 0000000..e8a6d27
--- /dev/null
+++ b/include/net/netns/generic.h
@@ -0,0 +1,49 @@
+/*
+ * generic net pointers
+ */
+
+#ifndef __NET_GENERIC_H__
+#define __NET_GENERIC_H__
+
+#include <linux/rcupdate.h>
+
+/*
+ * Generic net pointers are to be used by modules
+ * to put some private stuff on the struct net without
+ * explicit struct net modification
+ *
+ * The rules are simple:
+ * 1. register the ops with register_pernet_gen_device to get
+ *    the id of your private pointer
+ * 2. call net_assign_generic() to put the private data on the
+ *    struct net (most preferably this should be done in the
+ *    ->init callback of the ops registered)
+ * 3. do not change this pointer while the net is alive.
+ *
+ * After accomplishing all of the above, the private pointer
+ * can be accessed with the net_generic() call.
+ */
+
+struct net_generic {
+ unsigned int len;
+ struct rcu_head rcu;
+
+ void *ptr[0];
+};
+
+static inline void *net_generic(struct net *net, int id)
+{
+ struct net_generic *ng;
+ void *ptr;
+
```

```
+ rcu_read_lock();
+ ng = rcu_dereference(net->gen);
+ BUG_ON(id == 0 || id > ng->len);
+ ptr = ng->ptr[id - 1];
+ rcu_read_unlock();
+
+ return ptr;
+}
+
+extern int net_assign_generic(struct net *net, int id, void *data);
+#endif
diff --git a/net/core/net_namespace.c b/net/core/net_namespace.c
index 7ef3bac..b384840 100644
--- a/net/core/net_namespace.c
+++ b/net/core/net_namespace.c
@@ -7,6 +7,7 @@
 #include <linux/sched.h>
 #include <linux/idr.h>
 #include <net/net_namespace.h>
+#include <net/netns/generic.h>

 /*
  * Our network namespace constructor/destructor lists
@@ -21,6 +22,8 @@ LIST_HEAD(net_namespace_list);
 struct net init_net;
 EXPORT_SYMBOL(init_net);

+#define INITIAL_NET_GEN_PTRS 13 /* +1 for len +2 for rcu_head */
+
 /*
  * setup_net runs the initializers for the network namespace object.
  */
@@ -29,10 +32,21 @@ static __net_init int setup_net(struct net *net)
  /* Must be called with net_mutex held */
  struct pernet_operations *ops;
  int error;
+ struct net_generic *ng;

  atomic_set(&net->count, 1);
  atomic_set(&net->use_count, 0);

+ error = -ENOMEM;
+ ng = kzalloc(sizeof(struct net_generic) +
+   INITIAL_NET_GEN_PTRS * sizeof(void *), GFP_KERNEL);
+ if (ng == NULL)
+  goto out;
+
+ ng->len = INITIAL_NET_GEN_PTRS;
```

```
+ INIT_RCU_HEAD(&ng->rcu);
+ rcu_assign_pointer(net->gen, ng);
+
  error = 0;
  list_for_each_entry(ops, &pernet_list, list) {
   if (ops->init) {
@@ -54,6 +68,7 @@ out_undo:
  }

  rcu_barrier();
+ kfree(ng);
  goto out;
 }

@@ -384,3 +399,50 @@ void unregister_pernet_gen_device(int id, struct pernet_operations
*ops)
  mutex_unlock(&net_mutex);
 }
 EXPORT_SYMBOL_GPL(unregister_pernet_gen_device);
+
+static void net_generic_release(struct rcu_head *rcu)
+{
+ struct net_generic *ng;
+
+ ng = container_of(rcu, struct net_generic, rcu);
+ kfree(ng);
+}
+
+int net_assign_generic(struct net *net, int id, void *data)
+{
+ struct net_generic *ng, *old_ng;
+
+ BUG_ON(!mutex_is_locked(&net_mutex));
+ BUG_ON(id == 0);
+
+ ng = old_ng = net->gen;
+ if (old_ng->len >= id)
+  goto assign;
+
+ ng = kzalloc(sizeof(struct net_generic) +
+   id * sizeof(void *), GFP_KERNEL);
+ if (ng == NULL)
+  return -ENOMEM;
+
+ /*
+  * Some synchronisation notes:
+  *
+  * The net_generic explores the net->gen array inside rcu
```

```
+  * read section. Besides once set the net->gen->ptr[x]
+  * pointer never changes (see rules in netns/generic.h).
+  *
+  * That said, we simply duplicate this array and schedule
+  * the old copy for kfree after a grace period.
+  */
+
+ ng->len = id;
+ INIT_RCU_HEAD(&ng->rcu);
+ memcpy(&ng->ptr, &old_ng->ptr, old_ng->len);
+
+ rcu_assign_pointer(net->gen, ng);
+ call_rcu(&old_ng->rcu, net_generic_release);
+assign:
+ ng->ptr[id - 1] = data;
+ return 0;
+}
+EXPORT_SYMBOL_GPL(net_assign_generic);
--
1.5.3.4
```