Subject: [patch 11/12] net namespace : debugfs - add net_ns debugfs
Posted by Daniel Lezcano on Fri, 19 Jan 2007 15:47:25 GMT
View Forum Message <> Reply to Message

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

For debug purpose only, this is not intended to be included.
Add /sys/kernel/debug/net_ns.

Creation of network namespace:

echo <level> > /sys/kernel/debug/net_ns/start

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>

```
---
 fs/debugfs/Makefile |    2
 fs/debugfs/net_ns.c |  335 +++++++++++++++++++++++++++++++++++++++++++++++++++
 net/Kconfig         |    4
 3 files changed, 340 insertions(+), 1 deletion(-)

Index: 2.6.20-rc4-mm1/fs/debugfs/Makefile
===================================================================
--- 2.6.20-rc4-mm1.orig/fs/debugfs/Makefile
+++ 2.6.20-rc4-mm1/fs/debugfs/Makefile
@@ -1,4 +1,4 @@
 debugfs-objs := inode.o file.o

 obj-$(CONFIG_DEBUG_FS) += debugfs.o
-
+obj-$(CONFIG_NET_NS_DEBUG) += net_ns.o
Index: 2.6.20-rc4-mm1/fs/debugfs/net_ns.c
===================================================================
--- /dev/null
+++ 2.6.20-rc4-mm1/fs/debugfs/net_ns.c
@@ -0,0 +1,335 @@
+/*
+ *  net_ns.c - adds a net_ns/ directory to debug NET namespaces
+ *
+ *  Author: Daniel Lezcano <dlezcano@fr.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, version 2 of the
+ *     License.
+ */
+
+#include <linux/module.h>
```

```
+#include <linux/kernel.h>
+#include <linux/pagemap.h>
+#include <linux/debugfs.h>
+#include <linux/sched.h>
+#include <linux/netdevice.h>
+#include <linux/inetdevice.h>
+#include <linux/syscalls.h>
+#include <linux/net_namespace.h>
+#include <linux/rtnetlink.h>
+
+static struct dentry *net_ns_dentry;
+static struct dentry *net_ns_dentry_dev;
+static struct dentry *net_ns_dentry_start;
+static struct dentry *net_ns_dentry_info;
+
+static ssize_t net_ns_dev_read_file(struct file *file, char __user *user_buf,
+        size_t count, loff_t *ppos)
+{
+ return 0;
+}
+
+static ssize_t net_ns_dev_write_file(struct file *file,
+         const char __user *user_buf,
+         size_t count, loff_t *ppos)
+{
+ return 0;
+}
+
+static int net_ns_dev_open_file(struct inode *inode, struct file *file)
+{
+ return 0;
+}
+
+static int net_ns_start_open_file(struct inode *inode, struct file *file)
+{
+ return 0;
+}
+
+static ssize_t net_ns_start_read_file(struct file *file, char __user *user_buf,
+          size_t count, loff_t *ppos)
+{
+ return 0;
+}
+
+static ssize_t net_ns_start_write_file(struct file *file,
+          const char __user *user_buf,
+          size_t count, loff_t *ppos)
+{
```

```
+ int err;
+ size_t len;
+ const char __user *p;
+ char c;
+ unsigned long flags;
+ struct net_namespace *net, *new_net;
+  struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
+
+ if (current_net_ns != &init_net_ns)
+  return -EBUSY;
+
+ len = 0;
+ p = user_buf;
+ while (len < count) {
+  if (get_user(c, p++))
+   return -EFAULT;
+  if (c == 0 || c == '\n')
+   break;
+  len++;
+ }
+
+ if (len > 1)
+  return -EINVAL;
+
+ if (copy_from_user(&c, user_buf, sizeof(c)))
+  return -EFAULT;
+
+ if (c != '2' && c != '3')
+  return -EINVAL;
+
+ flags = (c=='2'?CLONE_NEWNET2:CLONE_NEWNET3);
+  err = unshare_net_ns(flags, &new_net);
+ if (err)
+  return err;
+
+  old_nsproxy = current->nsproxy;
+  new_nsproxy = dup_namespaces(old_nsproxy);
+
+  if (!new_nsproxy) {
+   put_net_ns(new_net);
+   task_unlock(current);
+   return -ENOMEM;
+  }
+
+  task_lock(current);
+
+  if (new_nsproxy) {
+   current->nsproxy = new_nsproxy;
```

```
+   new_nsproxy = old_nsproxy;
+  }
+
+  net = current->nsproxy->net_ns;
+  current->nsproxy->net_ns = new_net;
+ pop_net_ns(new_net);
+  new_net = net;
+
+  task_unlock(current);
+
+  put_nsproxy(new_nsproxy);
+  put_net_ns(new_net);
+
+ return count;
+}
+
+static int net_ns_info_open_file(struct inode *inode, struct file *file)
+{
+ return 0;
+}
+
+static ssize_t net_ns_info_read_file(struct file *file, char __user *user_buf,
+         size_t count, loff_t *ppos)
+{
+ const unsigned int length = 256;
+ size_t len;
+ char buff[length];
+ char *level;
+ struct net_namespace *net_ns = current_net_ns;
+ struct nsproxy *ns = current->nsproxy;
+
+ if (*ppos < 0)
+  return -EINVAL;
+ if (*ppos >= count)
+  return 0;
+ if (!count)
+  return 0;
+
+ switch (net_ns->level) {
+ case NET_NS_LEVEL2:
+  level = "layer 2";
+  break;
+ case NET_NS_LEVEL3:
+  level = "layer 3";
+  break;
+ default:
+  level = "unknown";
+  break;
```

```
+ }
+
+  sprintf(buff,
+   "nsproxy: %p\nnsproxy refcnt: %d\nnet_ns: %p\nnet_ns refcnt: %d\nlevel: %s\n",
+   ns,
+   atomic_read(&ns->count),
+   net_ns,
+   atomic_read(&net_ns->kref.refcount),
+   level);
+
+ len = strlen(buff);
+ if (len > count)
+  len = count;
+
+ if (copy_to_user(user_buf, buff, len))
+  return -EINVAL;
+
+ *ppos += count;
+
+ return count;
+}
+
+static ssize_t net_ns_info_write_file(struct file *file,
+          const char __user *user_buf,
+          size_t count, loff_t *ppos)
+{
+ struct net_namespace *net_ns = current_net_ns;
+ struct net_device *dev;
+ struct in_device *in_dev;
+ struct in_ifaddr **ifap = NULL;
+ struct in_ifaddr *ifa = NULL;
+ char *colon;
+ int err;
+
+ char buff[1024];
+ char *eth, *addr, *s;
+ __be32 address = 0;
+ __be32 p;
+
+ if (!capable(CAP_NET_ADMIN))
+  return -EPERM;
+
+ if (net_ns->level != NET_NS_LEVEL3)
+  return -EPERM;
+
+ if (count > sizeof(buff))
+  return -EINVAL;
+
```

```
+ if (copy_from_user(buff, user_buf, count))
+  return -EFAULT;
+
+ buff[count] = '\0';
+
+       eth = buff;
+       s = strchr(eth, ' ');
+       if (!s)
+               return -EINVAL;
+       *s = 0;
+
+       addr = s + 1;
+       s = strchr(addr, '.');
+       if (!s)
+               return -EINVAL;
+       *s = 0;
+       p = simple_strtoul(addr, NULL, 0);
+       ((char *)&address)[3] = p;
+       addr = s + 1;
+
+       s = strchr(addr, '.');
+       if (!s)
+               return -EINVAL;
+       *s = 0;
+       p = simple_strtoul(addr, NULL, 0);
+       ((char *)&address)[2] = p;
+       addr = s + 1;
+
+       s = strchr(addr, '.');
+       if (!s)
+               return -EINVAL;
+       *s = 0;
+       p = simple_strtoul(addr, NULL, 0);
+       ((char *)&address)[1] = p;
+       addr = s + 1;
+
+       p = simple_strtoul(addr, NULL, 0);
+       ((char *)&address)[0]  = p;
+
+ colon = strchr(eth, ':');
+ if (colon)
+  *colon = 0;
+
+       address = htonl(address);
+
+ rtnl_lock();
+
+ err = -ENODEV;
```

```
+ dev = __dev_get_by_name(eth);
+ if (!dev)
+  goto out;
+
+ if (colon)
+  *colon = ':';
+
+ err = -EADDRNOTAVAIL;
+ in_dev = __in_dev_get_rtnl(dev);
+ if (!in_dev)
+  goto out;
+
+ for (ifap = &in_dev->ifa_list; (ifa = *ifap) != NULL;
+      ifap = &ifa->ifa_next)
+  if (!strcmp(eth, ifa->ifa_label) &&
+      address == ifa->ifa_local)
+   break;
+ if (!ifa)
+  goto out;
+
+ ifa->ifa_net_ns = net_ns;
+
+ err = count;
+out:
+ rtnl_unlock();
+ return err;
+}
+
+
+static struct file_operations net_ns_dev_fops = {
+       .read =         net_ns_dev_read_file,
+       .write =        net_ns_dev_write_file,
+       .open =         net_ns_dev_open_file,
+};
+
+static struct file_operations net_ns_start_fops = {
+       .read =         net_ns_start_read_file,
+       .write =        net_ns_start_write_file,
+       .open =         net_ns_start_open_file,
+};
+
+static struct file_operations net_ns_info_fops = {
+       .read =         net_ns_info_read_file,
+       .write =        net_ns_info_write_file,
+       .open =         net_ns_info_open_file,
+};
+
+static int __init net_ns_init(void)
```

```
+{
+ net_ns_dentry = debugfs_create_dir("net_ns", NULL);
+
+ net_ns_dentry_dev = debugfs_create_file("dev", 0444,
+       net_ns_dentry,
+       NULL,
+       &net_ns_dev_fops);
+
+ net_ns_dentry_start = debugfs_create_file("start", 0666,
+         net_ns_dentry,
+         NULL,
+         &net_ns_start_fops);
+
+ net_ns_dentry_info = debugfs_create_file("info", 0444,
+        net_ns_dentry,
+        NULL,
+        &net_ns_info_fops);
+
+ return 0;
+}
+
+static void __exit net_ns_exit(void)
+{
+ debugfs_remove(net_ns_dentry_info);
+ debugfs_remove(net_ns_dentry_start);
+ debugfs_remove(net_ns_dentry_dev);
+ debugfs_remove(net_ns_dentry);
+}
+
+module_init(net_ns_init);
+module_exit(net_ns_exit);
+
+MODULE_DESCRIPTION("NET namespace debugfs");
+MODULE_AUTHOR("Daniel Lezcano <dlezcano@fr.ibm.com>");
+MODULE_LICENSE("GPL");
Index: 2.6.20-rc4-mm1/net/Kconfig
===================================================================
--- 2.6.20-rc4-mm1.orig/net/Kconfig
+++ 2.6.20-rc4-mm1/net/Kconfig
@@ -60,6 +60,10 @@

    Short answer: say Y.

+config NET_NS_DEBUG
+ bool "Debug fs for network namespace"
+ depends on DEBUG_FS && NET_NS
+
 if INET
```

source "net/ipv4/Kconfig"
source "net/ipv6/Kconfig"

--

_____

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