Subject: [PATCH net-2.6.25 2/2][NEIGH] Use the ctl paths to create neighbours sysctls
Posted by Pavel Emelianov on Fri, 30 Nov 2007 17:29:16 GMT
View Forum Message <> Reply to Message

The appropriate path is prepared right inside this function. It
is prepared similar to how the ctl tables were.

Since the path is modified, it is put on the stack, to avoid
possible races with multiple calls to neigh_sysctl_register() : it
is called by protocols and I didn't find any protection in this
case. Did I overlooked the rtnl lock?.

The stack growth of the neigh_sysctl_register() is 40 bytes. I
believe this is OK, since this is not that much and this function
is not called with the deep stack (device/protocols register).

The device's name is stored on the template to free it later.

This will help with the net namespaces, as each namespace should
have its own set of these ctls.

Besides, this saves ~350 bytes from the neigh template :)

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

```
diff --git a/net/core/neighbour.c b/net/core/neighbour.c
index 5dbe26f..4b6dd1e 100644
--- a/net/core/neighbour.c
+++ b/net/core/neighbour.c
@@ -2484,11 +2484,8 @@ void neigh_app_ns(struct neighbour *n)

 static struct neigh_sysctl_table {
  struct ctl_table_header *sysctl_header;
- ctl_table  neigh_vars[__NET_NEIGH_MAX];
- ctl_table  neigh_dev[2];
- ctl_table  neigh_neigh_dir[2];
- ctl_table  neigh_proto_dir[2];
- ctl_table  neigh_root_dir[2];
+ struct ctl_table neigh_vars[__NET_NEIGH_MAX];
+ char *dev_name;
 } neigh_sysctl_template __read_mostly = {
  .neigh_vars = {
   {
@@ -2619,32 +2616,7 @@ static struct neigh_sysctl_table {
    .mode  = 0644,
```

```
    .proc_handler = &proc_dointvec,
   },
-  {}
- },
- .neigh_dev = {
-  {
-   .ctl_name = NET_PROTO_CONF_DEFAULT,
-   .procname = "default",
-   .mode  = 0555,
-  },
- },
- .neigh_neigh_dir = {
-  {
-   .procname = "neigh",
-   .mode  = 0555,
-  },
- },
- .neigh_proto_dir = {
-  {
-   .mode  = 0555,
-  },
- },
- .neigh_root_dir = {
-  {
-   .ctl_name = CTL_NET,
-   .procname = "net",
-   .mode  = 0555,
-  },
+  {},
  },
 };

@@ -2654,7 +2626,19 @@ int neigh_sysctl_register(struct net_device *dev, struct neigh_parms
*p,
 {
  struct neigh_sysctl_table *t;
  const char *dev_name_source = NULL;
- char *dev_name = NULL;
+
+#define NEIGH_CTL_PATH_ROOT 0
+#define NEIGH_CTL_PATH_PROTO 1
+#define NEIGH_CTL_PATH_NEIGH 2
+#define NEIGH_CTL_PATH_DEV 3
+
+ struct ctl_path neigh_path[] = {
+  { .procname = "net",  .ctl_name = CTL_NET, },
+  { .procname = "proto",  .ctl_name = 0, },
+  { .procname = "neigh",  .ctl_name = 0, },
```

```
+  { .procname = "default", .ctl_name = NET_PROTO_CONF_DEFAULT, },
+  { },
+ };

  t = kmemdup(&neigh_sysctl_template, sizeof(*t), GFP_KERNEL);
  if (!t)
@@ -2677,11 +2661,11 @@ int neigh_sysctl_register(struct net_device *dev, struct neigh_parms
*p,

  if (dev) {
   dev_name_source = dev->name;
-  t->neigh_dev[0].ctl_name = dev->ifindex;
+  neigh_path[NEIGH_CTL_PATH_DEV].ctl_name = dev->ifindex;
   /* Terminate the table early */
   memset(&t->neigh_vars[14], 0, sizeof(t->neigh_vars[14]));
  } else {
-  dev_name_source = t->neigh_dev[0].procname;
+  dev_name_source = neigh_path[NEIGH_CTL_PATH_DEV].procname;
   t->neigh_vars[14].data = (int *)(p + 1);
   t->neigh_vars[15].data = (int *)(p + 1) + 1;
   t->neigh_vars[16].data = (int *)(p + 1) + 2;
@@ -2716,23 +2700,16 @@ int neigh_sysctl_register(struct net_device *dev, struct neigh_parms
*p,
    t->neigh_vars[13].ctl_name = CTL_UNNUMBERED;
  }

- dev_name = kstrdup(dev_name_source, GFP_KERNEL);
- if (!dev_name)
+ t->dev_name = kstrdup(dev_name_source, GFP_KERNEL);
+ if (!t->dev_name)
   goto free;

- t->neigh_dev[0].procname = dev_name;
-
- t->neigh_neigh_dir[0].ctl_name = pdev_id;
-
- t->neigh_proto_dir[0].procname = p_name;
- t->neigh_proto_dir[0].ctl_name = p_id;
-
- t->neigh_dev[0].child        = t->neigh_vars;
- t->neigh_neigh_dir[0].child    = t->neigh_dev;
- t->neigh_proto_dir[0].child    = t->neigh_neigh_dir;
- t->neigh_root_dir[0].child     = t->neigh_proto_dir;
+ neigh_path[NEIGH_CTL_PATH_DEV].procname = t->dev_name;
+ neigh_path[NEIGH_CTL_PATH_NEIGH].ctl_name = pdev_id;
+ neigh_path[NEIGH_CTL_PATH_PROTO].procname = p_name;
+ neigh_path[NEIGH_CTL_PATH_PROTO].ctl_name = p_id;
```

```
- t->sysctl_header = register_sysctl_table(t->neigh_root_dir);
+ t->sysctl_header = register_sysctl_paths(neigh_path, t->neigh_vars);
  if (!t->sysctl_header)
   goto free_procname;

@@ -2740,7 +2717,7 @@ int neigh_sysctl_register(struct net_device *dev, struct neigh_parms
*p,
  return 0;

 free_procname:
- kfree(dev_name);
+ kfree(t->dev_name);
 free:
  kfree(t);
 err:
@@ -2753,7 +2730,7 @@ void neigh_sysctl_unregister(struct neigh_parms *p)
   struct neigh_sysctl_table *t = p->sysctl_table;
   p->sysctl_table = NULL;
   unregister_sysctl_table(t->sysctl_header);
-  kfree(t->neigh_dev[0].procname);
+  kfree(t->dev_name);
   kfree(t);
  }
 }
--
1.5.3.4
```

Subject: Re: [PATCH net-2.6.25 2/2][NEIGH] Use the ctl paths to create neighbours sysctls

Posted by Herbert Xu on Sat, 01 Dec 2007 13:09:23 GMT

View Forum Message <> Reply to Message

On Fri, Nov 30, 2007 at 08:29:16PM +0300, Pavel Emelyanov wrote:
> Since the path is modified, it is put on the stack, to avoid
> possible races with multiple calls to neigh_sysctl_register() : it
> is called by protocols and I didn't find any protection in this
> case. Did I overlooked the rtnl lock?.

I think the only caller that can be a module is IPv6 :)

> The stack growth of the neigh_sysctl_register() is 40 bytes. I
> believe this is OK, since this is not that much and this function
> is not called with the deep stack (device/protocols register).

Yes it's fine.

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

Both applied to net-2.6.25. Thanks Pavel!
--
Visit Openswan at http://www.openswan.org/
Email: Herbert Xu ~{PmV>HI~} <herbert@gondor.apana.org.au>
Home Page: http://gondor.apana.org.au/~herbert/
PGP Key: http://gondor.apana.org.au/~herbert/pubkey.txt
