
Subject: Re: [PATCH 0/3] Sysctl shadow management
Posted by [ebiederm](#) on Tue, 20 Nov 2007 15:21:06 GMT
[View Forum Message](#) <> [Reply to Message](#)

To be very very very clear.

This is the way I think we should do the core sysctl infrastructure.

On top of the register_sysctl_table patch, getting all of the infrastructure in at once.

With a list of lists so we don't kill ourselves when we try to implement sysctls that are per network devices.

I'm not yet finished testing and reviewing the code yet but I think this is pretty close.

>From 7a0ab8b4d471fb1f2721150a5b1737a6e407b7b8 Mon Sep 17 00:00:00 2001
From: Eric W. Biederman <ebiederm@xmission.com>
Date: Tue, 20 Nov 2007 07:51:50 -0700
Subject: [PATCH] sysctl: Infrastructure for per namespace sysctls

This patch implements the basic infrastructure for per namespace sysctls.

A list of lists of sysctl headers is added, allowing each namespace to have it's own list of sysctl headers.

Each list of sysctl headers has a lookup function to find the first sysctl header in the list, allowing the lists to have a per namespace instance.

register_sysctl_root is added to tell sysctl.c about additional lists of lists. As all of the users are expected to be in kernel no unregister function is provided.

sysctl_head_next is updated to walk through the list of lists.

__reregister_sysctl_paths is added to add a new sysctl table on a non-default sysctl list.

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

```
include/linux/sysctl.h | 16 +++++++-
kernel/sysctl.c        | 92 ++++++++++++++++++++++++++++++++++++++-----
kernel/sysctl_check.c  | 25 ++++++-----
3 files changed, 108 insertions(+), 25 deletions(-)
```

```
diff --git a/include/linux/sysctl.h b/include/linux/sysctl.h
```

index 8b2e9e0..eeea2bb 100644

--- a/include/linux/sysctl.h

+++ b/include/linux/sysctl.h

@@ -951,7 +951,9 @@ enum

```
/* For the /proc/sys support */
struct ctl_table;
+struct nsproxy;
extern struct ctl_table_header *sysctl_head_next(struct ctl_table_header *prev);
+extern struct ctl_table_header *__sysctl_head_next(struct nsproxy *namespaces, struct
ctl_table_header *prev);
extern void sysctl_head_finish(struct ctl_table_header *prev);
extern int sysctl_perm(struct ctl_table *table, int op);
```

@@ -1055,6 +1057,12 @@ struct ctl_table

```
void *extra2;
};
```

```
+struct ctl_table_root {
+ struct list_head list;
+ struct ctl_table_header *ctl_header;
+ struct ctl_table_header *(*lookup)(struct nsproxy *namespaces);
+};
+
```

```
/* struct ctl_table_header is used to maintain dynamic lists of
   struct ctl_table trees. */
struct ctl_table_header
```

@@ -1064,6 +1072,7 @@ struct ctl_table_header

```
int used;
struct completion *unregistering;
struct ctl_table *ctl_table_arg;
+ struct ctl_table_root *root;
};
```

```
/* struct ctl_path describes where in the hierarchy a table is added */
@@ -1073,12 +1082,17 @@ struct ctl_path
int ctl_name;
};
```

```
+void register_sysctl_root(struct ctl_table_root *root);
+struct ctl_table_header *__register_sysctl_paths(
+ struct ctl_table_root *root, struct nsproxy *namespaces,
+ const struct ctl_path *path, struct ctl_table *table);
struct ctl_table_header *register_sysctl_table(struct ctl_table *table);
struct ctl_table_header *register_sysctl_paths(const struct ctl_path *path,
struct ctl_table *table);
```

```
void unregister_sysctl_table(struct ctl_table_header *table);
```

```

-int sysctl_check_table(struct ctl_table *table);
+int sysctl_check_table(struct nsproxy *namespaces, struct ctl_table *table);
+

#else /* __KERNEL__ */

diff --git a/kernel/sysctl.c b/kernel/sysctl.c
index ef023b5..d5e77ec 100644
--- a/kernel/sysctl.c
+++ b/kernel/sysctl.c
@@ -156,8 +156,16 @@ static int proc_dointvec_taint(struct ctl_table *table, int write, struct file *
#endif

static struct ctl_table root_table[];
-static struct ctl_table_header root_table_header =
- { root_table, LIST_HEAD_INIT(root_table_header.ctl_entry) };
+static struct ctl_table_root sysctl_table_root;
+static struct ctl_table_header root_table_header = {
+ .ctl_table = root_table,
+ .ctl_entry = LIST_HEAD_INIT(root_table_header.ctl_entry),
+ .root = &sysctl_table_root,
+};
+static struct ctl_table_root sysctl_table_root = {
+ .list = LIST_HEAD_INIT(sysctl_table_root.list),
+ .ctl_header = &root_table_header,
+};

static struct ctl_table kern_table[];
static struct ctl_table vm_table[];
@@ -1300,10 +1308,13 @@ void sysctl_head_finish(struct ctl_table_header *head)
    spin_unlock(&sysctl_lock);
}

-struct ctl_table_header *sysctl_head_next(struct ctl_table_header *prev)
+struct ctl_table_header *__sysctl_head_next(struct nsproxy *namespaces,
+      struct ctl_table_header *prev)
{
- struct ctl_table_header *head;
+ struct ctl_table_header *head;
+ struct ctl_table_root *root;
+ struct ctl_table_header *first;
    struct list_head *tmp;
+
    spin_lock(&sysctl_lock);
    if (prev) {
        tmp = &prev->ctl_entry;
@@ -1320,13 +1331,42 @@ struct ctl_table_header *sysctl_head_next(struct ctl_table_header
*prev)
    return head;

```

```

    next:
    tmp = tmp->next;
-   if (tmp == &root_table_header.ctl_entry)
-   break;
+   head = list_entry(tmp, struct ctl_table_header, ctl_entry);
+   root = head->root;
+   first = root->ctl_header;
+   if (root->lookup)
+   first = root->lookup(namespaces);
+
+   if (head == first) {
+   next_root:
+   root = list_entry(root->list.next,
+       struct ctl_table_root, list);
+   if (root == &sysctl_table_root)
+   break;
+   first = root->ctl_header;
+   if (root->lookup)
+   first = root->lookup(namespaces);
+   if (!first)
+   goto next_root;
+   tmp = &first->ctl_entry;
+   }
    }
    spin_unlock(&sysctl_lock);
    return NULL;
}

+struct ctl_table_header *sysctl_head_next(struct ctl_table_header *prev)
+{
+   return __sysctl_head_next(current->nsproxy, prev);
+}
+
+void register_sysctl_root(struct ctl_table_root *root)
+{
+   spin_lock(&sysctl_lock);
+   list_add_tail(&sysctl_table_root.list, &root->list);
+   spin_unlock(&sysctl_lock);
+}
+
#ifdef CONFIG_SYSCTL_SYSCALL
int do_sysctl(int __user *name, int nlen, void __user *oldval, size_t __user *oldlenp,
    void __user *newval, size_t newlen)
@@ -1483,14 +1523,16 @@ static __init int sysctl_init(void)
{
    int err;
    sysctl_set_parent(NULL, root_table);
-   err = sysctl_check_table(root_table);

```

```

+ err = sysctl_check_table(current->nsproxy, root_table);
  return 0;
}

core_initcall(sysctl_init);

/**
 * register_sysctl_paths - register a sysctl hierarchy
 * __register_sysctl_paths - register a sysctl hierarchy
 * @root: List of sysctl headers to register on
 * @namespaces: Data to compute which lists of sysctl entries are visible
 * @path: The path to the directory the sysctl table is in.
 * @table: the top-level table structure
 *
@@ -1558,10 +1600,12 @@ core_initcall(sysctl_init);
 * This routine returns %NULL on a failure to register, and a pointer
 * to the table header on success.
 */
-struct ctl_table_header *register_sysctl_paths(const struct ctl_path *path,
- struct ctl_table *table)
+struct ctl_table_header *__register_sysctl_paths(
+ struct ctl_table_root *root,
+ struct nsproxy *namespaces,
+ const struct ctl_path *path, struct ctl_table *table)
{
- struct ctl_table_header *header;
+ struct ctl_table_header *header, *first;
  struct ctl_table *new, **prevp;
  unsigned int n, npath;

@@ -1603,19 +1647,40 @@ struct ctl_table_header *register_sysctl_paths(const struct ctl_path
*path,
  INIT_LIST_HEAD(&header->ctl_entry);
  header->used = 0;
  header->unregistering = NULL;
+ header->root = root;
  sysctl_set_parent(NULL, header->ctl_table);
- if (sysctl_check_table(header->ctl_table)) {
+ if (sysctl_check_table(namespaces, header->ctl_table)) {
  kfree(header);
  return NULL;
}
  spin_lock(&sysctl_lock);
- list_add_tail(&header->ctl_entry, &root_table_header.ctl_entry);
+ first = root->ctl_header;
+ if (root->lookup)
+ first = root->lookup(namespaces);
+ list_add_tail(&header->ctl_entry, &first->ctl_entry);

```

```
spin_unlock(&sysctl_lock);

return header;
}

/**
 * register_sysctl_table_path - register a sysctl table hierarchy
 * @path: The path to the directory the sysctl table is in.
 * @table: the top-level table structure
 *
 * Register a sysctl table hierarchy. @table should be a filled in ctl_table
 * array. A completely 0 filled entry terminates the table.
 *
 * See __register_sysctl_paths for more details.
 */
+struct ctl_table_header *register_sysctl_paths(const struct ctl_path *path,
+      struct ctl_table *table)
+{
+ return __register_sysctl_paths(&sysctl_table_root, current->nsproxy,
+   path, table);
+}
+
+/**
 * register_sysctl_table - register a sysctl table hierarchy
 * @table: the top-level table structure
 *
@@ -1648,6 +1713,7 @@ void unregister_sysctl_table(struct ctl_table_header * header)
    kfree(header);
}

+
+else /* !CONFIG_SYSCTL */
+struct ctl_table_header *register_sysctl_table(struct ctl_table * table)
+{
diff --git a/kernel/sysctl_check.c b/kernel/sysctl_check.c
index 5a2f2b2..9761561 100644
--- a/kernel/sysctl_check.c
+++ b/kernel/sysctl_check.c
@@ -1383,7 +1383,8 @@ static void sysctl_repair_table(struct ctl_table *table)
 }
 }

-static struct ctl_table *sysctl_check_lookup(struct ctl_table *table)
+static struct ctl_table *sysctl_check_lookup(struct nsproxy *namespaces,
+      struct ctl_table *table)
+{
    struct ctl_table_header *head;
    struct ctl_table *ref, *test;
```

```
@@ -1391,8 +1392,8 @@ static struct ctl_table *sysctl_check_lookup(struct ctl_table *table)
```

```
    depth = sysctl_depth(table);
```

```
- for (head = sysctl_head_next(NULL); head;
-     head = sysctl_head_next(head)) {
+ for (head = __sysctl_head_next(namespaces, NULL); head;
+     head = __sysctl_head_next(namespaces, head)) {
    cur_depth = depth;
    ref = head->ctl_table;
```

```
    repeat:
```

```
@@ -1437,13 +1438,14 @@ static void set_fail(const char **fail, struct ctl_table *table, const
char *str
```

```
    *fail = str;
}
```

```
-static int sysctl_check_dir(struct ctl_table *table)
+static int sysctl_check_dir(struct nsproxy *namespaces,
+ struct ctl_table *table)
{
    struct ctl_table *ref;
    int error;
```

```
    error = 0;
```

```
- ref = sysctl_check_lookup(table);
+ ref = sysctl_check_lookup(namespaces, table);
    if (ref) {
        int match = 0;
        if ((!table->procname && !ref->procname) ||
```

```
@@ -1468,11 +1470,12 @@ static int sysctl_check_dir(struct ctl_table *table)
    return error;
}
```

```
-static void sysctl_check_leaf(struct ctl_table *table, const char **fail)
+static void sysctl_check_leaf(struct nsproxy *namespaces,
+ struct ctl_table *table, const char **fail)
{
    struct ctl_table *ref;
```

```
- ref = sysctl_check_lookup(table);
+ ref = sysctl_check_lookup(namespaces, table);
    if (ref && (ref != table))
        set_fail(fail, table, "Sysctl already exists");
}
```

```
@@ -1496,7 +1499,7 @@ static void sysctl_check_bin_path(struct ctl_table *table, const char
**fail)
}
}
```

```

-int sysctl_check_table(struct ctl_table *table)
+int sysctl_check_table(struct nsproxy *namespaces, struct ctl_table *table)
{
    int error = 0;
    for (; table->ctl_name || table->procname; table++) {
@@ -1526,7 +1529,7 @@ int sysctl_check_table(struct ctl_table *table)
        set_fail(&fail, table, "Directory with extra1");
        if (table->extra2)
            set_fail(&fail, table, "Directory with extra2");
-    if (sysctl_check_dir(table))
+    if (sysctl_check_dir(namespaces, table))
        set_fail(&fail, table, "Inconsistent directory names");
    } else {
        if ((table->strategy == sysctl_data) ||
@@ -1575,7 +1578,7 @@ int sysctl_check_table(struct ctl_table *table)
        if (!table->procname && table->proc_handler)
            set_fail(&fail, table, "proc_handler without procname");
    #endif
-    sysctl_check_leaf(table, &fail);
+    sysctl_check_leaf(namespaces, table, &fail);
    }
    sysctl_check_bin_path(table, &fail);
    if (fail) {
@@ -1583,7 +1586,7 @@ int sysctl_check_table(struct ctl_table *table)
        error = -EINVAL;
    }
    if (table->child)
-    error |= sysctl_check_table(table->child);
+    error |= sysctl_check_table(namespaces, table->child);
    }
    return error;
}
--
1.5.3.rc6.17.g1911

```

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
