
Subject: [PATCH 1/3] The sysctl shadows
Posted by [Pavel Emelianov](#) on Tue, 20 Nov 2007 11:43:42 GMT
[View Forum Message](#) <> [Reply to Message](#)

The sysctl shadow is nothing but a `ctl_table_head`, that hides behind some other one and which is used in `proc_sys_readdir`, `proc_sys_lookup`, etc when scanning through the list of these heads.

Each "shadow" carries its own set of `ctl_tables` that are relevant to this shadow. The appropriate shadow is get via the `->shadow()` callback on the head.

When putting the shadow back (unuse it) one should work with the `shadow->origin` head.

The API:

- * `register_sysctl_table_shadow`: registers the `ctl` table and tells that it will be shadowed with the specified callback;
- * `create_sysctl_shadow`: clones the original head's table and creates a shadow for it. The caller then may move the `ctl_table->data` pointer on the new (shadow) tables to point to another variables. It may also change names for tables, etc;
- * `free_sysctl_shadow`: destroys the shadow.

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

```
diff --git a/include/linux/sysctl.h b/include/linux/sysctl.h
index 4f5047d..064f132 100644
--- a/include/linux/sysctl.h
+++ b/include/linux/sysctl.h
@@ -1057,9 +1057,16 @@ struct ctl_table_header
     struct list_head ctl_entry;
     int used;
     struct completion *unregistering;
+ struct ctl_table_header *(*shadow)(struct ctl_table_header *);
+ struct ctl_table_header *origin;
 };

 struct ctl_table_header *register_sysctl_table(struct ctl_table * table);
+struct ctl_table_header *register_sysctl_table_shadow(struct ctl_table * table,
+ struct ctl_table_header *(*shadow)(struct ctl_table_header *));
+
+struct ctl_table_header *create_sysctl_shadow(struct ctl_table_header *h);
```

```

+void free_sysctl_shadow(struct ctl_table_header *h);

void unregister_sysctl_table(struct ctl_table_header * table);
int sysctl_check_table(struct ctl_table *table);
diff --git a/kernel/sysctl.c b/kernel/sysctl.c
index 489b0d1..bfea4fa 100644
--- a/kernel/sysctl.c
+++ b/kernel/sysctl.c
@@ -1320,6 +1320,9 @@ void sysctl_head_finish(struct ctl_table_header *head)
{
    if (!head)
        return;
+ if (head->origin)
+ head = head->origin;
+
    spin_lock(&sysctl_lock);
    unuse_table(head);
    spin_unlock(&sysctl_lock);
@@ -1331,6 +1334,9 @@ struct ctl_table_header *sysctl_head_next(struct ctl_table_header
*prev)
    struct list_head *tmp;
    spin_lock(&sysctl_lock);
    if (prev) {
+ if (prev->origin != NULL)
+ prev = prev->origin;
+
        tmp = &prev->ctl_entry;
        unuse_table(prev);
        goto next;
@@ -1342,6 +1348,10 @@ struct ctl_table_header *sysctl_head_next(struct ctl_table_header
*prev)
    if (!use_table(head))
        goto next;
    spin_unlock(&sysctl_lock);
+
+ if (head->shadow)
+ head = head->shadow(head);
+
    return head;
next:
    tmp = tmp->next;
@@ -1582,7 +1592,8 @@ 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_table(struct ctl_table * table)
+struct ctl_table_header *register_sysctl_table_shadow(struct ctl_table * table,
+ struct ctl_table_header *(*shadow)(struct ctl_table_header *))

```

```

{
    struct ctl_table_header *tmp;
    tmp = kmalloc(sizeof(struct ctl_table_header), GFP_KERNEL);
@@ -1592,6 +1603,8 @@ struct ctl_table_header *register_sysctl_table(struct ctl_table * table)
    INIT_LIST_HEAD(&tmp->ctl_entry);
    tmp->used = 0;
    tmp->unregistering = NULL;
+ tmp->shadow = shadow;
+ tmp->origin = NULL;
    sysctl_set_parent(NULL, table);
    if (sysctl_check_table(tmp->ctl_table)) {
        kfree(tmp);
@@ -1603,6 +1616,11 @@ struct ctl_table_header *register_sysctl_table(struct ctl_table * table)
    return tmp;
}

+struct ctl_table_header *register_sysctl_table(struct ctl_table * table)
+{
+ return register_sysctl_table_shadow(table, NULL);
+}
+
/**
 * unregister_sysctl_table - unregister a sysctl table hierarchy
 * @header: the header returned from register_sysctl_table
@@ -1619,6 +1637,84 @@ void unregister_sysctl_table(struct ctl_table_header * header)
    kfree(header);
}

+/*
+ * ctl tables shadow management
+ */
+static void ctl_free_table(struct ctl_table *t)
+{
+ struct ctl_table *tmp;
+
+ for (tmp = t; tmp->ctl_name || tmp->procname; tmp++)
+ if (tmp->child)
+     ctl_free_table(tmp->child);
+
+ kfree(t);
+}
+
+static struct ctl_table *ctl_dup_table(struct ctl_table *parent,
+ struct ctl_table *t)
+{
+ int i;
+ struct ctl_table *copy;
+

```

```

+ for (copy = t, i = 1; copy->ctl_name || copy->procname; copy++, i++);
+
+ copy = kmemdup(t, i * sizeof(struct ctl_table), GFP_KERNEL);
+ if (copy == NULL)
+   goto out;
+
+ for (i = 0; copy[i].ctl_name || copy[i].procname; i++) {
+   copy[i].parent = parent;
+   if (copy[i].child == NULL)
+     continue;
+
+   copy[i].child = ctl_dup_table(&copy[i], t[i].child);
+   if (copy[i].child == NULL)
+     goto unroll;
+ }
+
+ return copy;
+
+unroll:
+ for (i--; i >= 0; i--)
+   if (copy[i].child)
+     ctl_free_table(copy[i].child);
+ kfree(copy);
+out:
+ return NULL;
+}
+
+struct ctl_table_header *create_sysctl_shadow(struct ctl_table_header *h)
+{
+ struct ctl_table_header *ret;
+ struct ctl_table *tbl;
+
+ ret = kmemdup(h, sizeof(struct ctl_table_header *), GFP_KERNEL);
+ if (ret == NULL)
+   goto err_header;
+
+ ret->shadow = NULL;
+ ret->origin = h;
+
+ tbl = ctl_dup_table(NULL, h->ctl_table);
+ if (tbl == NULL)
+   goto err_table;
+
+ ret->ctl_table = tbl;
+ return ret;
+
+err_table:
+ kfree(ret);

```

```
+err_header:
+ return NULL;
+}
+
+void free_sysctl_shadow(struct ctl_table_header *shadow)
+{
+ ctl_free_table(shadow->ctl_table);
+ kfree(shadow);
+}
+
+else /* !CONFIG_SYSCTL */
+ struct ctl_table_header *register_sysctl_table(struct ctl_table * table)
+ {
```

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