Subject: [PATCH 0/3] Sysctl shadow management
Posted by Pavel Emelianov on Tue, 20 Nov 2007 11:38:36 GMT
View Forum Message <> Reply to Message

Hi guys!

You all know, that with multiple namespaces we have to take special care about sysctls. E.g. IPC sysctl handlers are equipped with kludges to alter the sysctl parameters of appropriate namespace. The same thing should be done for UTS namespace (but it is not - we have a BUG in mainstream) and (!) for network namespaces.

Unlike all the other namespaces, network will have to not just address different variables via same sysctl names, but to have different tables with different sysctl names. E.g. /proc/sys/net/conf have entries for devices, which differ across namespaces.

Eric currently have some work done in that directions, I like the approach in general very much, but it looks rather raw (Eric, take this in good part). You know, ifdefs in the middle of the code, explicit references to net namespace and so on and so forth.

So here's the RFC for a bit better sysctls shadow management.

I will provide 3 patches:
1. the sysctl shadows themselves;
2. using shadows in UTS namespace;
3. using shadows in IPC namespace;

If someone want I can send

4. example on how to create a /proc/sys/net/conf/-like structure with different names.

Using them in net namespace is already checked (I created sysctl entries with different names), but I don't have any patches against any David's tree yet. If we're OK with this set I will start talking to Andrew and David about who to send these patches to and making shadows for net-related sysctl variables.

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

_____

Containers mailing list
Containers@lists.linux-foundation.org

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 is 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

---

Subject: [PATCH 2/3] Switch UTS namespace to use shadows
Posted by Pavel Emelianov on Tue, 20 Nov 2007 11:45:51 GMT
View Forum Message <> Reply to Message

The uts sysctl table contains two writable fields
(domainname and nodename), so split the table into
common (read-only) part and writable (shadowed).

This fixes the BUG! You may create a namespace and
then writing to /proc/sys/hostname will cause an
init_uts_ns overwrite.

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

```
diff --git a/include/linux/utsname.h b/include/linux/utsname.h
index 923db99..7517b36 100644
--- a/include/linux/utsname.h
+++ b/include/linux/utsname.h
@@ -40,6 +40,7 @@ struct new_utsname {
 struct uts_namespace {
  struct kref kref;
  struct new_utsname name;
+ struct ctl_table_header *ctl_header;
```

```
 };
 extern struct uts_namespace init_uts_ns;

@@ -66,6 +67,9 @@ static inline struct new_utsname *init_utsname(void)
  return &init_uts_ns.name;
 }

+int clone_uts_sysctl(struct uts_namespace *ns);
+void free_uts_sysctl(struct uts_namespace *ns);
+
 extern struct rw_semaphore uts_sem;

 #endif /* __KERNEL__ */
diff --git a/kernel/utsname.c b/kernel/utsname.c
index 816d7b2..22e40bb 100644
--- a/kernel/utsname.c
+++ b/kernel/utsname.c
@@ -26,13 +26,21 @@ static struct uts_namespace *clone_uts_ns(struct uts_namespace
*old_ns)

  ns = kmalloc(sizeof(struct uts_namespace), GFP_KERNEL);
  if (!ns)
-  return ERR_PTR(-ENOMEM);
+  goto err_alloc;
+
+ if (clone_uts_sysctl(ns))
+  goto err_sysctl;

  down_read(&uts_sem);
  memcpy(&ns->name, &old_ns->name, sizeof(ns->name));
  up_read(&uts_sem);
  kref_init(&ns->kref);
  return ns;
+
+err_sysctl:
+ kfree(ns);
+err_alloc:
+ return ERR_PTR(-ENOMEM);
 }

 /*
@@ -62,5 +70,6 @@ void free_uts_ns(struct kref *kref)
  struct uts_namespace *ns;

  ns = container_of(kref, struct uts_namespace, kref);
+ free_uts_sysctl(ns);
  kfree(ns);
 }
```

```
diff --git a/kernel/utsname_sysctl.c b/kernel/utsname_sysctl.c
index c76c064..8a06f0b 100644
--- a/kernel/utsname_sysctl.c
+++ b/kernel/utsname_sysctl.c
@@ -75,6 +75,11 @@ static int sysctl_uts_string(ctl_table *table, int __user *name, int nlen,
 #define sysctl_uts_string NULL
 #endif

+static struct ctl_table_header *uts_sysctl_shadow(struct ctl_table_header *h)
+{
+ return current->nsproxy->uts_ns->ctl_header;
+}
+
 static struct ctl_table uts_kern_table[] = {
  {
   .ctl_name = KERN_OSTYPE,
@@ -103,6 +108,20 @@ static struct ctl_table uts_kern_table[] = {
   .proc_handler = proc_do_uts_string,
   .strategy = sysctl_uts_string,
  },
+ {}
+};
+
+static struct ctl_table uts_root_table[] = {
+ {
+  .ctl_name = CTL_KERN,
+  .procname = "kernel",
+  .mode  = 0555,
+  .child  = uts_kern_table,
+ },
+ {}
+};
+
+static struct ctl_table uts_kern_table_sh[] = {
  {
   .ctl_name = KERN_NODENAME,
   .procname = "hostname",
@@ -124,19 +143,44 @@ static struct ctl_table uts_kern_table[] = {
  {}
 };

-static struct ctl_table uts_root_table[] = {
+static struct ctl_table uts_root_table_sh[] = {
  {
   .ctl_name = CTL_KERN,
   .procname = "kernel",
   .mode  = 0555,
-  .child  = uts_kern_table,
```

```
+  .child  = uts_kern_table_sh,
  },
  {}
 };

+int clone_uts_sysctl(struct uts_namespace *ns)
+{
+ struct ctl_table_header *h;
+ struct ctl_table *tbl;
+
+ h = create_sysctl_shadow(init_uts_ns.ctl_header);
+ if (h == NULL)
+  return -ENOMEM;
+
+ tbl = h->ctl_table->child;
+
+ tbl[0].data = ns->name.nodename;
+ tbl[1].data = ns->name.domainname;
+
+ ns->ctl_header = h;
+ return 0;
+}
+
+void free_uts_sysctl(struct uts_namespace *ns)
+{
+ free_sysctl_shadow(ns->ctl_header);
+}
+
 static int __init utsname_sysctl_init(void)
 {
  register_sysctl_table(uts_root_table);
+ init_uts_ns.ctl_header = register_sysctl_table_shadow(uts_root_table_sh,
+    uts_sysctl_shadow);
  return 0;
 }
```

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

Subject: [PATCH 3/3] Switch IPC namespace to use sysctl shadows
Posted by Pavel Emelianov on Tue, 20 Nov 2007 11:47:41 GMT
View Forum Message <> Reply to Message

The same as with UTS - use the shadows. With this

there's no longer need in having kludged ->proc_handler
and ->strategy for ctl_tables.

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

```
diff --git a/include/linux/ipc.h b/include/linux/ipc.h
index 408696e..a9fef83 100644
--- a/include/linux/ipc.h
+++ b/include/linux/ipc.h
@@ -118,6 +118,8 @@ struct ipc_namespace {
  size_t  shm_ctlall;
  int  shm_ctlmni;
  int  shm_tot;
+
+ struct ctl_table_header *ctl_head;
 };

 extern struct ipc_namespace init_ipc_ns;
diff --git a/ipc/ipc_sysctl.c b/ipc/ipc_sysctl.c
index 79e24e8..5affdfe 100644
--- a/ipc/ipc_sysctl.c
+++ b/ipc/ipc_sysctl.c
@@ -15,84 +15,6 @@
 #include <linux/sysctl.h>
 #include <linux/uaccess.h>

-static void *get_ipc(ctl_table *table)
-{
- char *which = table->data;
- struct ipc_namespace *ipc_ns = current->nsproxy->ipc_ns;
- which = (which - (char *)&init_ipc_ns) + (char *)ipc_ns;
- return which;
-}
-
-#ifdef CONFIG_PROC_FS
-static int proc_ipc_dointvec(ctl_table *table, int write, struct file *filp,
- void __user *buffer, size_t *lenp, loff_t *ppos)
-{
- struct ctl_table ipc_table;
- memcpy(&ipc_table, table, sizeof(ipc_table));
- ipc_table.data = get_ipc(table);
-
- return proc_dointvec(&ipc_table, write, filp, buffer, lenp, ppos);
-}
-
-static int proc_ipc_doulongvec_minmax(ctl_table *table, int write,
```

```
- struct file *filp, void __user *buffer, size_t *lenp, loff_t *ppos)
-{
- struct ctl_table ipc_table;
- memcpy(&ipc_table, table, sizeof(ipc_table));
- ipc_table.data = get_ipc(table);
-
- return proc_doulongvec_minmax(&ipc_table, write, filp, buffer,
-     lenp, ppos);
-}
-
-#else
-#define proc_ipc_doulongvec_minmax NULL
-#define proc_ipc_dointvec    NULL
-#endif
-
-#ifdef CONFIG_SYSCTL_SYSCALL
-/* The generic sysctl ipc data routine. */
-static int sysctl_ipc_data(ctl_table *table, int __user *name, int nlen,
-  void __user *oldval, size_t __user *oldlenp,
-  void __user *newval, size_t newlen)
-{
- size_t len;
- void *data;
-
- /* Get out of I don't have a variable */
- if (!table->data || !table->maxlen)
-  return -ENOTDIR;
-
- data = get_ipc(table);
- if (!data)
-  return -ENOTDIR;
-
- if (oldval && oldlenp) {
-  if (get_user(len, oldlenp))
-   return -EFAULT;
-  if (len) {
-   if (len > table->maxlen)
-    len = table->maxlen;
-   if (copy_to_user(oldval, data, len))
-    return -EFAULT;
-   if (put_user(len, oldlenp))
-    return -EFAULT;
-  }
- }
-
- if (newval && newlen) {
-  if (newlen > table->maxlen)
-   newlen = table->maxlen;
```

```
-
-  if (copy_from_user(data, newval, newlen))
-   return -EFAULT;
- }
- return 1;
-}
-#else
-#define sysctl_ipc_data NULL
-#endif
-
 static struct ctl_table ipc_kern_table[] = {
  {
   .ctl_name = KERN_SHMMAX,
@@ -100,8 +22,8 @@ static struct ctl_table ipc_kern_table[] = {
   .data  = &init_ipc_ns.shm_ctlmax,
   .maxlen  = sizeof (init_ipc_ns.shm_ctlmax),
   .mode  = 0644,
-  .proc_handler = proc_ipc_doulongvec_minmax,
-  .strategy = sysctl_ipc_data,
+  .proc_handler = proc_doulongvec_minmax,
+  .strategy = sysctl_data,
  },
  {
   .ctl_name = KERN_SHMALL,
@@ -109,8 +31,8 @@ static struct ctl_table ipc_kern_table[] = {
   .data  = &init_ipc_ns.shm_ctlall,
   .maxlen  = sizeof (init_ipc_ns.shm_ctlall),
   .mode  = 0644,
-  .proc_handler = proc_ipc_doulongvec_minmax,
-  .strategy = sysctl_ipc_data,
+  .proc_handler = proc_doulongvec_minmax,
+  .strategy = sysctl_data,
  },
  {
   .ctl_name = KERN_SHMMNI,
@@ -118,8 +40,8 @@ static struct ctl_table ipc_kern_table[] = {
   .data  = &init_ipc_ns.shm_ctlmni,
   .maxlen  = sizeof (init_ipc_ns.shm_ctlmni),
   .mode  = 0644,
-  .proc_handler = proc_ipc_dointvec,
-  .strategy = sysctl_ipc_data,
+  .proc_handler = proc_dointvec,
+  .strategy = sysctl_data,
  },
  {
   .ctl_name = KERN_MSGMAX,
@@ -127,8 +49,8 @@ static struct ctl_table ipc_kern_table[] = {
   .data  = &init_ipc_ns.msg_ctlmax,
```

```
   .maxlen  = sizeof (init_ipc_ns.msg_ctlmax),
   .mode  = 0644,
-  .proc_handler = proc_ipc_dointvec,
-  .strategy = sysctl_ipc_data,
+  .proc_handler = proc_dointvec,
+  .strategy = sysctl_data,
  },
  {
   .ctl_name = KERN_MSGMNI,
@@ -136,8 +58,8 @@ static struct ctl_table ipc_kern_table[] = {
   .data  = &init_ipc_ns.msg_ctlmni,
   .maxlen  = sizeof (init_ipc_ns.msg_ctlmni),
   .mode  = 0644,
-  .proc_handler = proc_ipc_dointvec,
-  .strategy = sysctl_ipc_data,
+  .proc_handler = proc_dointvec,
+  .strategy = sysctl_data,
  },
  {
   .ctl_name = KERN_MSGMNB,
@@ -145,8 +67,8 @@ static struct ctl_table ipc_kern_table[] = {
   .data  = &init_ipc_ns.msg_ctlmnb,
   .maxlen  = sizeof (init_ipc_ns.msg_ctlmnb),
   .mode  = 0644,
-  .proc_handler = proc_ipc_dointvec,
-  .strategy = sysctl_ipc_data,
+  .proc_handler = proc_dointvec,
+  .strategy = sysctl_data,
  },
  {
   .ctl_name = KERN_SEM,
@@ -154,8 +76,8 @@ static struct ctl_table ipc_kern_table[] = {
   .data  = &init_ipc_ns.sem_ctls,
   .maxlen  = 4*sizeof (int),
   .mode  = 0644,
-  .proc_handler = proc_ipc_dointvec,
-  .strategy = sysctl_ipc_data,
+  .proc_handler = proc_dointvec,
+  .strategy = sysctl_data,
  },
  {}
 };
@@ -170,9 +92,43 @@ static struct ctl_table ipc_root_table[] = {
 };

+int ipc_clone_sysctl(struct ipc_namespace *ns)
+{
```

```
+ struct ctl_table_header *h;
+ struct ctl_table *t;
+
+ h = create_sysctl_shadow(init_ipc_ns.ctl_head);
+ if (h == NULL)
+  return -ENOMEM;
+
+ t = h->ctl_table->child;
+
+ t[0].data = &ns->shm_ctlmax;
+ t[1].data = &ns->shm_ctlall;
+ t[2].data = &ns->shm_ctlmni;
+ t[3].data = &ns->msg_ctlmax;
+ t[4].data = &ns->msg_ctlmni;
+ t[5].data = &ns->msg_ctlmnb;
+ t[6].data = &ns->sem_ctls;
+
+ ns->ctl_head = h;
+ return 0;
+}
+
+void ipc_free_sysctl(struct ipc_namespace *ns)
+{
+ free_sysctl_shadow(ns->ctl_head);
+}
+
+static struct ctl_table_header *ipc_sysctl_shadow(struct ctl_table_header *h)
+{
+ return current->nsproxy->ipc_ns->ctl_head;
+}
+
 static int __init ipc_sysctl_init(void)
 {
- register_sysctl_table(ipc_root_table);
+ init_ipc_ns.ctl_head = register_sysctl_table_shadow(ipc_root_table,
+   ipc_sysctl_shadow);
  return 0;
 }

diff --git a/ipc/util.c b/ipc/util.c
index 76c1f34..a8cb99d 100644
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -61,6 +61,9 @@ static struct ipc_namespace *clone_ipc_ns(struct ipc_namespace *old_ns)
  if (ns == NULL)
   goto err_mem;

+ if (ipc_clone_sysctl(ns))
```

```
+  goto err_ctl;
+
  err = sem_init_ns(ns);
  if (err)
   goto err_sem;
@@ -79,6 +82,8 @@ err_shm:
 err_msg:
  sem_exit_ns(ns);
 err_sem:
+ ipc_free_sysctl(ns);
+err_ctl:
  kfree(ns);
 err_mem:
  return ERR_PTR(err);
@@ -108,6 +113,7 @@ void free_ipc_ns(struct kref *kref)
  sem_exit_ns(ns);
  msg_exit_ns(ns);
  shm_exit_ns(ns);
+ ipc_free_sysctl(ns);
  kfree(ns);
 }

diff --git a/ipc/util.h b/ipc/util.h
index 9ffea40..dfd53c6 100644
--- a/ipc/util.h
+++ b/ipc/util.h
@@ -28,6 +28,9 @@ void sem_exit_ns(struct ipc_namespace *ns);
 void msg_exit_ns(struct ipc_namespace *ns);
 void shm_exit_ns(struct ipc_namespace *ns);

+int ipc_clone_sysctl(struct ipc_namespace *ns);
+void ipc_free_sysctl(struct ipc_namespace *ns);
+
 struct ipc_ids {
  int in_use;
  unsigned short seq;
```

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

---

Subject: Re: [PATCH 0/3] Sysctl shadow management
Posted by ebiederm on Tue, 20 Nov 2007 13:05:58 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov <xemul@openvz.org> writes:

> Hi guys!
>
> You all know, that with multiple namespaces we have to take
> special care about sysctls. E.g. IPC sysctl handlers are
> equipped with kludges to alter the sysctl parameters of
> appropriate namespace. The same thing should be done for UTS
> namespace (but it is not - we have a BUG in mainstream) and
> (!) for network namespaces.

The bug in mainstream was introduced by commit:
7d69a1f4a72b18876c99c697692b78339d491568

Thee code should read:

```
 static void *get_uts(ctl_table *table, int write)
 {
  char *which = table->data;
+ struct uts_namespace *uts_ns = current->nsproxy->uts_ns;
+ which = (which - (char *)&init_uts_ns) + (char *)uts_ns;

  if (!write)
   down_read(&uts_sem);
  else
   down_write(&uts_sem);
  return which;
 }
```

And for 2.6.24 we should just restore the two missing lines.

> Unlike all the other namespaces, network will have to not
> just address different variables via same sysctl names, but
> to have different tables with different sysctl names. E.g.
> /proc/sys/net/conf have entries for devices, which differ
> across namespaces.
>
> Eric currently have some work done in that directions, I
> like the approach in general very much, but it looks rather
> raw (Eric, take this in good part). You know, ifdefs in the
> middle of the code, explicit references to net namespace
> and so on and so forth.

Sure. I don't in principle have any problem with the set of
roots we go through be dynamic at run time instead of compile
time. That is probably a cleaner approach and likely to solve
my problem with sched_debug registering sysctls before the
sysctl subsystem is initialized.

One direction I have always intended to expand things when
there was a bit of time to modify /proc/sys so that it
is a symlink to /proc/self/sys. Then make /proc/<pid>/sys
have cachable dentries for the sysctls.

To achieve that we need to pass our namespaces into
your shadow function. Instead of always using current.

So I am thinking something like:

```
struct ctl_table_root {
 struct list_head ctl_entry;
 struct ctl_table_header *ctl_head;
 struct ctl_table_header *lookup_ctl_head(struct nsproxy *namespaces);
};
```

To actually handle the set of network devices in a namespace we need
to have a list so making sysctl_head_next just loop over a list of
lists should be no extra work and make implementing the users
easier.

Beyond that through from my quick skim I have a preference for
the way I am handling it.

We really need to add the tables with some variant of
register_sysctl_table or else we will have module unload races.

Introducing register_sysctl_paths is a very useful cleanup in
it's own right and it helps quite a bit. In most cases it removes the
need for your create_sysctl_shadow function, and it always reduced the
amount of code for tables.

Further simply using kmemdup instead of a custom crafted function
is a little more straight forward and is the idiom already
established throughout the networking code.

Then we will just need a base sysctl function:

```
struct ctl_table_header *
register_sysctl_rooted_paths(struct ctl_table_root *root,
    struct nsproxy *namespaces,
    struct ctl_path *path,
    struct ctl_table *table)
```

For the simple namespaces we can call it once per namespace
after registering our root (we can't use current because we initialize

things before we update nsproxy), and we still need to run
sysctl_check.

For the more complex namespaces (i.e. the network namespace). We can
write a simple wrapper around register_sysctl_rooted_paths.

And of course non-namespace specific sysctls can just use a wrapper
that assumes the default global list of sysctl_headers.

> So here's the RFC for a bit better sysctls shadow management.
>
> I will provide 3 patches:
> 1. the sysctl shadows themselves;
> 2. using shadows in UTS namespace;
> 3. using shadows in IPC namespace;

Your patches look fairly reasonable. Other then the pieces
I have mentioned already.

> Using them in net namespace is already checked (I created
> sysctl entries with different names), but I don't have any
> patches against any David's tree yet. If we're OK with this
> set I will start talking to Andrew and David about who to
> send these patches to and making shadows for net-related
> sysctl variables.

I think we need another round. My hunch is that it will be easiest
if David collects them up, and then Andrew updates his tree, but
we will see.

Eric

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

---

Subject: Re: [PATCH 0/3] Sysctl shadow management
Posted by Pavel Emelianov on Tue, 20 Nov 2007 13:21:21 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelyanov <xemul@openvz.org> writes:
>
>> Hi guys!
>>
>> You all know, that with multiple namespaces we have to take

>> special care about sysctls. E.g. IPC sysctl handlers are
>> equipped with kludges to alter the sysctl parameters of
>> appropriate namespace. The same thing should be done for UTS
>> namespace (but it is not - we have a BUG in mainstream) and
>> (!) for network namespaces.
>
>
> The bug in mainstream was introduced by commit:
> 7d69a1f4a72b18876c99c697692b78339d491568
>
> Thee code should read:
> static void *get_uts(ctl_table *table, int write)
> {
>   char *which = table->data;
> + struct uts_namespace *uts_ns = current->nsproxy->uts_ns;
> + which = (which - (char *)&init_uts_ns) + (char *)uts_ns;
>
>   if (!write)
>   down_read(&uts_sem);
>   else
>   down_write(&uts_sem);
>   return which;
> }
>
> And for 2.6.24 we should just restore the two missing lines.

Yup. I didn't want to submit this set as a fix.
On the other hand, I do not quite like the kludges
like above...

>
>> Unlike all the other namespaces, network will have to not
>> just address different variables via same sysctl names, but
>> to have different tables with different sysctl names. E.g.
>> /proc/sys/net/conf have entries for devices, which differ
>> across namespaces.
>>
>> Eric currently have some work done in that directions, I
>> like the approach in general very much, but it looks rather
>> raw (Eric, take this in good part). You know, ifdefs in the
>> middle of the code, explicit references to net namespace
>> and so on and so forth.
>
> Sure. I don't in principle have any problem with the set of
> roots we go through be dynamic at run time instead of compile
> time. That is probably a cleaner approach and likely to solve
> my problem with sched_debug registering sysctls before the
> sysctl subsystem is initialized.

>
> One direction I have always intended to expand things when
> there was a bit of time to modify /proc/sys so that it
> is a symlink to /proc/self/sys. Then make /proc/<pid>/sys
> have cachable dentries for the sysctls.

In this case you implicitly create some "sysctl namespace".
But this is not true. You may have tasks sharing net
namespace with their different sysctl entries and with
common ipc namespace with their common entries. And tasks
in same net namespace with different ipc ones and many
many over mixtures... This is unclean on how to treat
such per-process sysctl tree.

Shadows just make sysctls work somewhat independently
from namespaces.

> To achieve that we need to pass our namespaces into
> your shadow function. Instead of always using current.

Current is OK for now. At least this gives you 100%
guarantee that one namespace will never hack sysctls
from the other ones.

> So I am thinking something like:
>
> struct ctl_table_root {
> struct list_head ctl_entry;
> struct ctl_table_header *ctl_head;
> struct ctl_table_header *lookup_ctl_head(struct nsproxy *namespaces);
> };
>
> To actually handle the set of network devices in a namespace we need
> to have a list so making sysctl_head_next just loop over a list of
> lists should be no extra work and make implementing the users
> easier.
>
>
>
> Beyond that through from my quick skim I have a preference for
> the way I am handling it.
>
> We really need to add the tables with some variant of
> register_sysctl_table or else we will have module unload races.
>
> Introducing register_sysctl_paths is a very useful cleanup in

Agree. This piece of work is a "must-have". Are you going to go

on with it or may we take care?

But the shadows allow us move further with the net namespaces,
while paths are mostly cleanups. Important but cleanups.

> it's own right and it helps quite a bit. In most cases it removes the
> need for your create_sysctl_shadow function, and it always reduced the
> amount of code for tables.
>
> Further simply using kmemdup instead of a custom crafted function
> is a little more straight forward and is the idiom already
> established throughout the networking code.
>
> Then we will just need a base sysctl function:
> struct ctl_table_header *
> register_sysctl_rooted_paths(struct ctl_table_root *root,
>   struct nsproxy *namespaces,
>   struct ctl_path *path,
>   struct ctl_table *table)
>
>
> For the simple namespaces we can call it once per namespace
> after registering our root (we can't use current because we initialize
> things before we update nsproxy), and we still need to run
> sysctl_check.
>
> For the more complex namespaces (i.e. the network namespace). We can
> write a simple wrapper around register_sysctl_rooted_paths.
>
> And of course non-namespace specific sysctls can just use a wrapper
> that assumes the default global list of sysctl_headers.
>
>> So here's the RFC for a bit better sysctls shadow management.
>>
>> I will provide 3 patches:
>>   1. the sysctl shadows themselves;
>>   2. using shadows in UTS namespace;
>>   3. using shadows in IPC namespace;
>
>
> Your patches look fairly reasonable.   Other then the pieces
> I have mentioned already.

OK. This is just a launchpad. Sure, this will be expanded further.

>> Using them in net namespace is already checked (I created
>> sysctl entries with different names), but I don't have any
>> patches against any David's tree yet. If we're OK with this

>> set I will start talking to Andrew and David about who to
>> send these patches to and making shadows for net-related
>> sysctl variables.
>
> I think we need another round. My hunch is that it will be easiest
> if David collects them up, and then Andrew updates his tree, but
> we will see.

Hm... Ok then I think I will start virtualizing sysctls in unix
sockets (as they are already in net-2.6.25) with sysctl shadows
and send this stuff to David. Hopefully he will agree to accept
this :) After the next round we'll have that at Linus and go on
with ipc, uts, paths and so on.

Looks like we now have a "Looks-good-to:" subscript now :) May
I put yours in this set?

> Eric
>

Thanks,
Pavel

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

---

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_sysct_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.

__reegister_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_root *root;
+ struct ctl_table_header *head, *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

---

Subject: Re: [PATCH 0/3] Sysctl shadow management
Posted by Pavel Emelianov on Tue, 20 Nov 2007 15:36:22 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> 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.

Hm... My patch looks to do very very same thing, but in
a bit simpler manner. Except for the absence of the
sysctl paths, but they are just cleanups. I think I can
port them on top of my shadows :) Thanks

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

Thanks,
Pavel

>>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_sysct_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.
>
> __reegister_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_root *root;
> + struct ctl_table_header *head, *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;
>  }
```

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

---

Subject: Re: [PATCH 1/3] The sysctl shadows
Posted by Dave Hansen on Tue, 20 Nov 2007 17:16:49 GMT
View Forum Message <> Reply to Message

On Tue, 2007-11-20 at 14:43 +0300, Pavel Emelyanov wrote:
>
> +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);
> +}

Are you worried about the recursion at all?

-- Dave

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH 3/3] Switch IPC namespace to use sysctl shadows
Posted by Dave Hansen on Tue, 20 Nov 2007 17:24:13 GMT
View Forum Message <> Reply to Message

On Tue, 2007-11-20 at 14:47 +0300, Pavel Emelyanov wrote:
>
>
> +int ipc_clone_sysctl(struct ipc_namespace *ns)
> +{
> +	struct ctl_table_header *h;
> +	struct ctl_table *t;
> +
> +	h = create_sysctl_shadow(init_ipc_ns.ctl_head);
> +	if (h == NULL)
> +		return -ENOMEM;
> +
> +	t = h->ctl_table->child;
> +
> +	t[0].data = &ns->shm_ctlmax;
> +	t[1].data = &ns->shm_ctlall;
> +	t[2].data = &ns->shm_ctlmni;
> +	t[3].data = &ns->msg_ctlmax;
> +	t[4].data = &ns->msg_ctlmni;
> +	t[5].data = &ns->msg_ctlmnb;
> +	t[6].data = &ns->sem_ctls;
> +
> +	ns->ctl_head = h;
> +	return 0;
> +}

>From where does the order for these things come?

-- Dave

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

Subject: Re: [PATCH 0/3] Sysctl shadow management
Posted by ebiederm on Tue, 20 Nov 2007 19:47:26 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:
>> 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.
>
> Hm... My patch looks to do very very same thing, but in
> a bit simpler manner. Except for the absence of the
> sysctl paths, but they are just cleanups. I think I can
> port them on top of my shadows :) Thanks

Then slow down and listen.

Your code simply does not address module unload races. BAD BAD BAD.

Your code skips sysctl_check allowing buggy sysctl tables to continue
to exist BAD BAD BAD.

Your code does not implement a list of lists needed to implement
a dynamic list of network devices, instead leaving it to the user of
sysctl registration code to do. (Which is where the bulk of your
simplicity seems to come from). BAD BAD BAD.

So I'm sorry. Your code is simpler because it is WORSE.

Your code is race prone, encourages buggy users, and doesn't even
solve the same problem. So no I don't think it makes a bit
of sense.

Eric

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

---

Subject: Re: [PATCH 1/3] The sysctl shadows
Posted by Pavel Emelianov on Wed, 21 Nov 2007 09:20:38 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:

> On Tue, 2007-11-20 at 14:43 +0300, Pavel Emelyanov wrote:
>> +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);
>> +}
>
> Are you worried about the recursion at all?

I do, but

1. sysctl tables are _never_ deeper than 5 levels (as they
   always come from the kernel space)
2. registering sysctl tables happens from __init calls usually
   so the stack is not filled at all
3. after implementing the sysctl paths this will vanish ;)

Moreover, this is not the only case of recursion in the kernel :)

Thanks,
Pavel

> -- Dave
>
>

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

---

Subject: Re: [PATCH 3/3] Switch IPC namespace to use sysctl shadows
Posted by Pavel Emelianov on Wed, 21 Nov 2007 09:21:06 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Tue, 2007-11-20 at 14:47 +0300, Pavel Emelyanov wrote:
>>
>> +int ipc_clone_sysctl(struct ipc_namespace *ns)
>> +{
>> +	struct ctl_table_header *h;
>> +	struct ctl_table *t;

```
>> +
>> +	h = create_sysctl_shadow(init_ipc_ns.ctl_head);
>> +	if (h == NULL)
>> +		return -ENOMEM;
>> +
>> +	t = h->ctl_table->child;
>> +
>> +	t[0].data = &ns->shm_ctlmax;
>> +	t[1].data = &ns->shm_ctlall;
>> +	t[2].data = &ns->shm_ctlmni;
>> +	t[3].data = &ns->msg_ctlmax;
>> +	t[4].data = &ns->msg_ctlmni;
>> +	t[5].data = &ns->msg_ctlmnb;
>> +	t[6].data = &ns->sem_ctls;
>> +
>> +	ns->ctl_head = h;
>> +	return 0;
>> +}
>
```

>>From where does the order for these things come?

>From the original tables.

> -- Dave
>
>

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

---

Subject: Re: [PATCH 0/3] Sysctl shadow management
Posted by Pavel Emelianov on Wed, 21 Nov 2007 09:52:03 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelyanov <xemul@openvz.org> writes:
>
>> Eric W. Biederman wrote:
>>> 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.
>> Hm... My patch looks to do very very same thing, but in
>> a bit simpler manner. Except for the absence of the
>> sysctl paths, but they are just cleanups. I think I can
>> port them on top of my shadows :) Thanks
>
> Then slow down and listen.

Press my ear close to the monitor.

> Your code simply does not address module unload races. BAD BAD BAD.

Can you unload IPC or UTS module? =)

> Your code skips sysctl_check allowing buggy sysctl tables to continue
> to exist BAD BAD BAD.

The tables are cloned from those who passed these checks. ;)

> Your code does not implement a list of lists needed to implement
> a dynamic list of network devices, instead leaving it to the user of
> sysctl registration code to do. (Which is where the bulk of your
> simplicity seems to come from). BAD BAD BAD.

Do we need this stuff for IPC? UTS? Stick to the patches please.

Network sysctls are more complex, I know it. Multiple shadows
per-head are to be done. I agree with that. I'm not trying
to create the most generic code that can solve all our needs
at once.

These patches do not provide any facility to display the
information via sysfs. So? Does anybody care about it?

I already have a bad experience with sending
doing-it-all-in-one-go patches to the community.

> So I'm sorry. Your code is simpler because it is WORSE.
>
> Your code is race prone, encourages buggy users, and doesn't even
> solve the same problem. So no I don't think it makes a bit
> of sense.

I decidedly object against such an outrageous statements.

> Eric
>

Pavel

_____

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

---

Subject: [PATCH 0/4] Sysctl namespace support
Posted by ebiederm on Thu, 29 Nov 2007 17:40:24 GMT
View Forum Message <> Reply to Message

Currently the network namespace work has gotten about as far as we can
without the ability to make sysctls that are per network namespace.

The techniques we have been using for other namespace of examining
current and replacing the ctl_table.data field depending on the
namespace instance that current->nsproxy refers to are both ugly
and do not work for the network sysctls.

The case in handling the networking sysctls that does not work with
the existing ugly pointer munging techniques are directories like
/proc/sys/net/ipv4/conf/ and /proc/sys/net/ipv4/neigh/ whose contents
vary depending on the networking devices present in the network
namespace.

Adding support to the sysctl infrastructure to allow to register
a sysctl table for a particular instance of a particular namespace
removes the need for magic sysctl methods, and allows the use
of the techniques for managing dynamic sysctl tables used for years
in the network stack.

Herbert we need this infrastructure most in net-2.6.25 (as not having
it is a current bottleneck to further development of the network
namespace) so these patches are against net-2.6.25.

Andrew also need this infrastructure in -mm so that we can take
advantage of this new infrastructure when implementing other
namespaces.

So I expect the sane way to deal with this patchset is to merge into
both net-2.6.25 and -mm and then Andrew can drop or disable the
patches once he pulls bases -mm on a version of net-2.6.25 with
the changes.

Eric

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

Subject: [PATCH 1/4] sysctl: Add register_sysctl_paths function
Posted by ebiederm on Thu, 29 Nov 2007 17:45:10 GMT
View Forum Message <> Reply to Message

There are a number of modules that register a sysctl table
somewhere deeply nested in the sysctl hierarchy, such as
fs/nfs, fs/xfs, dev/cdrom, etc.

They all specify several dummy ctl_tables for the path name.
This patch implements register_sysctl_path that takes
an additional path name, and makes up dummy sysctl nodes
for each component.

This patch was originally written by Olaf Kirch and
brought to my attention and reworked some by Olaf Hering.
I have changed a few additional things so the bugs are mine.

After converting all of the easy callers Olaf Hering observed
allyesconfig ARCH=i386, the patch reduces the final binary size by 9369 bytes.

.text +897
.data -7008

```
   text    data     bss     dec     hex filename
  26959310        4045899 4718592 35723801        2211a19 ../vmlinux-vanilla
  26960207        4038891 4718592 35717690        221023a ../O-allyesconfig/vmlinux
```

So this change is both a space savings and a code simplification.

CC: Olaf Kirch <okir@suse.de>
CC: Olaf Hering <olaf@aepfle.de>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 include/linux/sysctl.h |    9 +++++
 kernel/sysctl.c        |   90 +++++++++++++++++++++++++++++++++++++++++-------
 2 files changed, 84 insertions(+), 15 deletions(-)

diff --git a/include/linux/sysctl.h b/include/linux/sysctl.h
index e99171f..eb522bf 100644
--- a/include/linux/sysctl.h
+++ b/include/linux/sysctl.h
@@ -1065,7 +1065,16 @@ struct ctl_table_header
```

```
  struct completion *unregistering;
 };

+/* struct ctl_path describes where in the hierarchy a table is added */
+struct ctl_path
+{
+ const char *procname;
+ int ctl_name;
+};
+
 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);
diff --git a/kernel/sysctl.c b/kernel/sysctl.c
index 0deed82..fa92e70 100644
--- a/kernel/sysctl.c
+++ b/kernel/sysctl.c
@@ -1490,11 +1490,12 @@ static __init int sysctl_init(void)
 core_initcall(sysctl_init);

 /**
- * register_sysctl_table - register a sysctl hierarchy
+ * register_sysctl_paths - register a sysctl 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. An entry with a ctl_name of 0 terminates the table.
+ * array. A completely 0 filled entry terminates the table.
  *
  * The members of the &struct ctl_table structure are used as follows:
  *
@@ -1557,28 +1558,80 @@ 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_paths(const struct ctl_path *path,
+       struct ctl_table *table)
 {
- struct ctl_table_header *tmp;
- tmp = kmalloc(sizeof(struct ctl_table_header), GFP_KERNEL);
- if (!tmp)
+ struct ctl_table_header *header;
+ struct ctl_table *new, **prevp;
```

```
+ unsigned int n, npath;
+
+ /* Count the path components */
+ for (npath = 0; path[npath].ctl_name || path[npath].procname; ++npath)
+  ;
+
+ /*
+  * For each path component, allocate a 2-element ctl_table array.
+  * The first array element will be filled with the sysctl entry
+  * for this, the second will be the sentinel (ctl_name == 0).
+  *
+  * We allocate everything in one go so that we don't have to
+  * worry about freeing additional memory in unregister_sysctl_table.
+  */
+ header = kzalloc(sizeof(struct ctl_table_header) +
+    (2 * npath * sizeof(struct ctl_table)), GFP_KERNEL);
+ if (!header)
   return NULL;
- tmp->ctl_table = table;
- INIT_LIST_HEAD(&tmp->ctl_entry);
- tmp->used = 0;
- tmp->unregistering = NULL;
- sysctl_set_parent(NULL, table);
- if (sysctl_check_table(tmp->ctl_table)) {
-  kfree(tmp);
+
+ new = (struct ctl_table *) (header + 1);
+
+ /* Now connect the dots */
+ prevp = &header->ctl_table;
+ for (n = 0; n < npath; ++n, ++path) {
+  /* Copy the procname */
+  new->procname = path->procname;
+  new->ctl_name = path->ctl_name;
+  new->mode     = 0555;
+
+  *prevp = new;
+  prevp = &new->child;
+
+  new += 2;
+ }
+ *prevp = table;
+
+ INIT_LIST_HEAD(&header->ctl_entry);
+ header->used = 0;
+ header->unregistering = NULL;
+ sysctl_set_parent(NULL, header->ctl_table);
+ if (sysctl_check_table(header->ctl_table)) {
```

```
+  kfree(header);
   return NULL;
  }
  spin_lock(&sysctl_lock);
- list_add_tail(&tmp->ctl_entry, &root_table_header.ctl_entry);
+ list_add_tail(&header->ctl_entry, &root_table_header.ctl_entry);
  spin_unlock(&sysctl_lock);
- return tmp;
+
+ return header;
 }

 /**
+ * register_sysctl_table - register a sysctl table hierarchy
+ * @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_table(struct ctl_table * table)
+{
+ static const struct ctl_path null_path[] = { {} };
+
+ return register_sysctl_paths(null_path, table);
+}
+
+
+/**
  * unregister_sysctl_table - unregister a sysctl table hierarchy
  * @header: the header returned from register_sysctl_table
  *
@@ -1600,6 +1653,12 @@ struct ctl_table_header *register_sysctl_table(struct ctl_table * table)
  return NULL;
 }

+struct ctl_table_header *register_sysctl_paths(const struct ctl_path *path,
+          struct ctl_table *table)
+{
+ return NULL;
+}
+
 void unregister_sysctl_table(struct ctl_table_header * table)
 {
 }
@@ -2658,6 +2717,7 @@ EXPORT_SYMBOL(proc_dostring);
 EXPORT_SYMBOL(proc_doulongvec_minmax);
```

```
 EXPORT_SYMBOL(proc_doulongvec_ms_jiffies_minmax);
 EXPORT_SYMBOL(register_sysctl_table);
+EXPORT_SYMBOL(register_sysctl_paths);
 EXPORT_SYMBOL(sysctl_intvec);
 EXPORT_SYMBOL(sysctl_jiffies);
 EXPORT_SYMBOL(sysctl_ms_jiffies);
--
1.5.3.rc6.17.g1911
```

_____

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

---

Subject: [PATCH 2/4] sysctl: Remember the ctl_table we passed to register_sysctl_paths

Posted by ebiederm on Thu, 29 Nov 2007 17:46:26 GMT
View Forum Message <> Reply to Message

By doing this we allow users of register_sysctl_paths that build
and dynamically allocate their ctl_table to be simpler. This allows
them to just remember the ctl_table_header returned from
register_sysctl_paths from which they can now find the
ctl_table array they need to free.

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

```
---
 include/linux/sysctl.h |    1 +
 kernel/sysctl.c        |    1 +
 2 files changed, 2 insertions(+), 0 deletions(-)

diff --git a/include/linux/sysctl.h b/include/linux/sysctl.h
index eb522bf..8b2e9e0 100644
--- a/include/linux/sysctl.h
+++ b/include/linux/sysctl.h
@@ -1063,6 +1063,7 @@ struct ctl_table_header
  struct list_head ctl_entry;
  int used;
  struct completion *unregistering;
+ struct ctl_table *ctl_table_arg;
 };

 /* struct ctl_path describes where in the hierarchy a table is added */
diff --git a/kernel/sysctl.c b/kernel/sysctl.c
index fa92e70..effae87 100644
--- a/kernel/sysctl.c
+++ b/kernel/sysctl.c
```

@@ -1598,6 +1598,7 @@ struct ctl_table_header *register_sysctl_paths(const struct ctl_path *path,
   new += 2;
  }
  *prevp = table;
+ header->ctl_table_arg = table;

  INIT_LIST_HEAD(&header->ctl_entry);
  header->used = 0;
--
1.5.3.rc6.17.g1911

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

---

Subject: [PATCH 3/4] sysctl: Infrastructure for per namespace sysctls
Posted by ebiederm on Thu, 29 Nov 2007 17:51:50 GMT

View Forum Message <> Reply to Message

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_sysct_root is added to tell sysctl.c about additional
lists of sysctl_headers. 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.

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

The only intrusive part of this patch is propagating the information
to decided which list of sysctls to use for sysctl_check_table.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 include/linux/sysctl.h | 16 ++++++++-
 kernel/sysctl.c | 93 +++------
 kernel/sysctl_check.c | 25 +++++++------

```
 3 files changed, 111 insertions(+), 23 deletions(-)

diff --git a/include/linux/sysctl.h b/include/linux/sysctl.h
index 8b2e9e0..cd1da5c 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 +1057,13 @@ struct ctl_table
  void *extra2;
 };

+struct ctl_table_root {
+ struct list_head root_list;
+ struct list_head header_list;
+ struct list_head *(*lookup)(struct ctl_table_root *root,
+        struct nsproxy *namespaces);
+};
+
 /* struct ctl_table_header is used to maintain dynamic lists of
    struct ctl_table trees. */
 struct ctl_table_header
@@ -1064,6 +1073,7 @@ 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 +1083,16 @@ 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 effae87..ad4b709 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(sysctl_table_root.header_list),
+ .root = &sysctl_table_root,
+};
+static struct ctl_table_root sysctl_table_root = {
+ .root_list = LIST_HEAD_INIT(sysctl_table_root.root_list),
+ .header_list = LIST_HEAD_INIT(root_table_header.ctl_entry),
+};

 static struct ctl_table kern_table[];
 static struct ctl_table vm_table[];
@@ -1300,12 +1308,27 @@ 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)
+static struct list_head *
+lookup_header_list(struct ctl_table_root *root, struct nsproxy *namespaces)
 {
+ struct list_head *header_list;
+ header_list = &root->header_list;
+ if (root->lookup)
+  header_list = root->lookup(root, namespaces);
+ return header_list;
+}
+
+struct ctl_table_header *__sysctl_head_next(struct nsproxy *namespaces,
```

```
+         struct ctl_table_header *prev)
+{
+ struct ctl_table_root *root;
+ struct list_head *header_list;
  struct ctl_table_header *head;
  struct list_head *tmp;
+
  spin_lock(&sysctl_lock);
  if (prev) {
+  head = prev;
   tmp = &prev->ctl_entry;
   unuse_table(prev);
   goto next;
@@ -1319,14 +1342,38 @@ struct ctl_table_header *sysctl_head_next(struct ctl_table_header
*prev)
   spin_unlock(&sysctl_lock);
   return head;
  next:
+  root = head->root;
   tmp = tmp->next;
-  if (tmp == &root_table_header.ctl_entry)
-   break;
+  header_list = lookup_header_list(root, namespaces);
+  if (tmp != header_list)
+   continue;
+
+  do {
+   root = list_entry(root->root_list.next,
+     struct ctl_table_root, root_list);
+   if (root == &sysctl_table_root)
+    goto out;
+   header_list = lookup_header_list(root, namespaces);
+  } while (list_empty(header_list));
+  tmp = header_list->next;
  }
+out:
  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(&root->root_list, &sysctl_table_root.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 +1530,16 @@ 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
  *
@@ -1558,9 +1607,12 @@ 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 list_head *header_list;
  struct ctl_table_header *header;
  struct ctl_table *new, **prevp;
  unsigned int n, npath;
@@ -1603,19 +1655,38 @@ 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);
+ header_list = lookup_header_list(root, namespaces);
+ list_add_tail(&header->ctl_entry, header_list);
  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
  *
diff --git a/kernel/sysctl_check.c b/kernel/sysctl_check.c
index fdfca0d..2544852 100644
--- a/kernel/sysctl_check.c
+++ b/kernel/sysctl_check.c
@@ -1352,7 +1352,8 @@ 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;
@@ -1360,8 +1361,8 @@ 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:
@@ -1406,13 +1407,14 @@ 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) ||
@@ -1437,11 +1439,12 @@ 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");
 }
@@ -1465,7 +1468,7 @@ 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++) {
@@ -1495,7 +1498,7 @@ 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) ||
@@ -1544,7 +1547,7 @@ 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) {
@@ -1552,7 +1555,7 @@ 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

---

Subject: Re: [PATCH 0/4] Sysctl namespace support
Posted by Herbert Xu on Fri, 30 Nov 2007 12:56:27 GMT
View Forum Message <> Reply to Message

On Thu, Nov 29, 2007 at 10:40:24AM -0700, Eric W. Biederman wrote:

>
> Herbert we need this infrastructure most in net-2.6.25 (as not having
> it is a current bottleneck to further development of the network
> namespace) so these patches are against net-2.6.25.

I've applied them all to net-2.6.25 with Andrew's fixes included.
Thanks Eric.
--
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

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

---

Subject: Re: [PATCH 0/4] Sysctl namespace support
Posted by ebiederm on Fri, 30 Nov 2007 13:25:25 GMT
View Forum Message <> Reply to Message

Herbert Xu <herbert@gondor.apana.org.au> writes:

> On Thu, Nov 29, 2007 at 10:40:24AM -0700, Eric W. Biederman wrote:
>>
>> Herbert we need this infrastructure most in net-2.6.25 (as not having
>> it is a current bottleneck to further development of the network
>> namespace) so these patches are against net-2.6.25.
>
> I've applied them all to net-2.6.25 with Andrew's fixes included.
> Thanks Eric.

Welcome, and thanks.

I will see about taking advantage of this shortly.

Eric

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

---