Subject: [PATCH 58/59] sysctl: Reimplement the sysctl proc support
Posted by ebiederm on Tue, 16 Jan 2007 16:40:03 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

With this change the sysctl inodes can be cached and
nothing needs to be done when removing a sysctl table.

For a costk of 2K code we will save about 4K of static tables
(when we remove de from ctl_table) and 70K in proc_dir_entries
that we will not allocate, or about half that on a 32bit arch.

The speed feels about the same, even though we can now cache
the sysctl dentries :(

We get the core advantage that we don't need to have a
1 to 1 mapping between ctl table entries and proc files.
Making it possible to have /proc/sys vary depending on
the namespace you are in. The currently merged namespaces
don't have an issue here but the network namespace under
/proc/sys/net needs to have different directories depending
on which network adapters are visible. By simply being
a cache different directories being visible depending
on who you are is trivial to implement.

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

```
---
 fs/proc/Makefile      |    2 +-
 fs/proc/inode.c       |    1 +
 fs/proc/internal.h    |    2 +
 fs/proc/proc_sysctl.c |  477 ++++++++++++++++++++++++++++++++++++++++++++++++
 fs/proc/root.c        |   10 +-
 init/main.c           |    4 -
 kernel/sysctl.c       |  182 -------------------
 7 files changed, 484 insertions(+), 194 deletions(-)

diff --git a/fs/proc/Makefile b/fs/proc/Makefile
index f6c7762..a6b3a8f 100644
--- a/fs/proc/Makefile
+++ b/fs/proc/Makefile
@@ -8,7 +8,7 @@ proc-y   := nommu.o task_nommu.o
 proc-$(CONFIG_MMU) := mmu.o task_mmu.o

 proc-y       += inode.o root.o base.o generic.o array.o \
-  proc_tty.o proc_misc.o
+  proc_tty.o proc_misc.o proc_sysctl.o
```

```
 proc-$(CONFIG_PROC_KCORE) += kcore.o
 proc-$(CONFIG_PROC_VMCORE) += vmcore.o
diff --git a/fs/proc/inode.c b/fs/proc/inode.c
index e26945b..0ea8265 100644
--- a/fs/proc/inode.c
+++ b/fs/proc/inode.c
@@ -161,6 +161,7 @@ struct inode *proc_get_inode(struct super_block *sb, unsigned int ino,
  if (!inode)
   goto out_ino;

+ PROC_I(inode)->fd = 0;
  PROC_I(inode)->pde = de;
  if (de) {
   if (de->mode) {
diff --git a/fs/proc/internal.h b/fs/proc/internal.h
index 987c773..3c9a305 100644
--- a/fs/proc/internal.h
+++ b/fs/proc/internal.h
@@ -11,6 +11,8 @@

 #include <linux/proc_fs.h>

+extern int proc_sys_init(void);
+
 struct vmalloc_info {
  unsigned long used;
  unsigned long largest_chunk;
diff --git a/fs/proc/proc_sysctl.c b/fs/proc/proc_sysctl.c
new file mode 100644
index 0000000..08a2e66
--- /dev/null
+++ b/fs/proc/proc_sysctl.c
@@ -0,0 +1,477 @@
+/*
+ * /proc/sys support
+ */
+
+#include <linux/sysctl.h>
+#include <linux/proc_fs.h>
+#include <linux/security.h>
+#include "internal.h"
+
+static struct dentry_operations proc_sys_dentry_operations;
+static const struct file_operations proc_sys_file_operations;
+static struct inode_operations proc_sys_inode_operations;
+
+static void proc_sys_refresh_inode(struct inode *inode, struct ctl_table *table)
+{
```

```
+ /* Refresh the cached information bits in the inode */
+ if (table) {
+  inode->i_uid = 0;
+  inode->i_gid = 0;
+  inode->i_mode = table->mode;
+  if (table->proc_handler) {
+   inode->i_mode |= S_IFREG;
+   inode->i_nlink = 1;
+  } else {
+   inode->i_mode |= S_IFDIR;
+   inode->i_nlink = 0; /* It is too hard to figure out */
+  }
+ }
+}
+
+static struct inode *proc_sys_make_inode(struct inode *dir, struct ctl_table *table)
+{
+ struct inode *inode;
+ struct proc_inode *dir_ei, *ei;
+ int depth;
+
+ inode = new_inode(dir->i_sb);
+ if (!inode)
+  goto out;
+
+ /* A directory is always one deeper than it's parent */
+ dir_ei = PROC_I(dir);
+ depth = dir_ei->fd + 1;
+
+ ei = PROC_I(inode);
+ ei->fd = depth;
+ inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
+ inode->i_op = &proc_sys_inode_operations;
+ inode->i_fop = &proc_sys_file_operations;
+ proc_sys_refresh_inode(inode, table);
+out:
+ return inode;
+}
+
+static struct dentry *proc_sys_ancestor(struct dentry *dentry, int depth)
+{
+ for (;;) {
+  struct proc_inode *ei;
+
+  ei = PROC_I(dentry->d_inode);
+  if (ei->fd == depth)
+   break; /* found */
+
```

```
+  dentry = dentry->d_parent;
+ }
+ return dentry;
+}
+
+static struct ctl_table *proc_sys_lookup_table_one(struct ctl_table *table,
+       struct qstr *name)
+{
+ int len;
+ for ( ; table->ctl_name || table->procname; table++) {
+
+  if (!table->procname)
+   continue;
+
+  len = strlen(table->procname);
+  if (len != name->len)
+   continue;
+
+  if (memcmp(table->procname, name->name, len) != 0)
+   continue;
+
+  /* I have a match */
+  return table;
+ }
+ return NULL;
+}
+
+static struct ctl_table *proc_sys_lookup_table(struct dentry *dentry,
+      struct ctl_table *table)
+{
+ struct dentry *ancestor;
+ struct proc_inode *ei;
+ int depth, i;
+
+ ei = PROC_I(dentry->d_inode);
+ depth = ei->fd;
+
+ if (depth == 0)
+  return table;
+
+ for (i = 1; table && (i <= depth); i++) {
+  ancestor = proc_sys_ancestor(dentry, i);
+  table = proc_sys_lookup_table_one(table, &ancestor->d_name);
+  if (table)
+   table = table->child;
+ }
+ return table;
+
```

```
+}
+static struct ctl_table *proc_sys_lookup_entry(struct dentry *dparent,
+      struct qstr *name,
+      struct ctl_table *table)
+{
+ table = proc_sys_lookup_table(dparent, table);
+ if (table)
+  table = proc_sys_lookup_table_one(table, name);
+ return table;
+}
+
+static struct ctl_table *do_proc_sys_lookup(struct dentry *parent,
+      struct qstr *name,
+      struct ctl_table_header **ptr)
+{
+ struct ctl_table_header *head;
+ struct ctl_table *table;
+
+ for (head = sysctl_head_next(NULL); head; head = sysctl_head_next(head)) {
+  table = proc_sys_lookup_entry(parent, name, head->ctl_table);
+  if (table)
+   break;
+ }
+ *ptr = head;
+ return table;
+}
+
+static struct dentry *proc_sys_lookup(struct inode *dir, struct dentry *dentry,
+     struct nameidata *nd)
+{
+ struct ctl_table_header *head;
+ struct inode *inode;
+ struct dentry *err;
+ struct ctl_table *table;
+
+ err = ERR_PTR(-ENOENT);
+ table = do_proc_sys_lookup(dentry->d_parent, &dentry->d_name, &head);
+ if (!table)
+  goto out;
+
+ err = ERR_PTR(-ENOMEM);
+ inode = proc_sys_make_inode(dir, table);
+ if (!inode)
+  goto out;
+
+ err = NULL;
+ dentry->d_op = &proc_sys_dentry_operations;
+ d_add(dentry, inode);
```

```
+
+out:
+ sysctl_head_finish(head);
+ return err;
+}
+
+static ssize_t proc_sys_read(struct file *filp, char __user *buf,
+    size_t count, loff_t *ppos)
+{
+ struct dentry *dentry = filp->f_dentry;
+ struct ctl_table_header *head;
+ struct ctl_table *table;
+ ssize_t error, res;
+
+ table = do_proc_sys_lookup(dentry->d_parent, &dentry->d_name, &head);
+ /* Has the sysctl entry disappeared on us? */
+ error = -ENOENT;
+ if (!table)
+  goto out;
+
+ /* Has the sysctl entry been replaced by a directory? */
+ error = -EISDIR;
+ if (!table->proc_handler)
+  goto out;
+
+ /*
+  * At this point we know that the sysctl was not unregistered
+  * and won't be until we finish.
+  */
+ error = -EPERM;
+ if (sysctl_perm(table, MAY_READ))
+  goto out;
+
+ /* careful: calling conventions are nasty here */
+ res = count;
+ error = table->proc_handler(table, 0, filp, buf, &res, ppos);
+ if (!error)
+  error = res;
+out:
+ sysctl_head_finish(head);
+
+ return error;
+}
+
+static ssize_t proc_sys_write(struct file *filp, const char __user *buf,
+    size_t count, loff_t *ppos)
+{
+ struct dentry *dentry = filp->f_dentry;
```

```
+ struct ctl_table_header *head;
+ struct ctl_table *table;
+ ssize_t error, res;
+
+ table = do_proc_sys_lookup(dentry->d_parent, &dentry->d_name, &head);
+ /* Has the sysctl entry disappeared on us? */
+ error = -ENOENT;
+ if (!table)
+  goto out;
+
+ /* Has the sysctl entry been replaced by a directory? */
+ error = -EISDIR;
+ if (!table->proc_handler)
+  goto out;
+
+ /*
+  * At this point we know that the sysctl was not unregistered
+  * and won't be until we finish.
+  */
+ error = -EPERM;
+ if (sysctl_perm(table, MAY_WRITE))
+  goto out;
+
+ /* careful: calling conventions are nasty here */
+ res = count;
+ error = table->proc_handler(table, 1, filp, buf, &res, ppos);
+ if (!error)
+  error = res;
+out:
+ sysctl_head_finish(head);
+
+ return error;
+}
+
+
+static int proc_sys_fill_cache(struct file *filp, void *dirent,
+    filldir_t filldir, struct ctl_table *table)
+{
+ struct ctl_table_header *head;
+ struct ctl_table *child_table = NULL;
+ struct dentry *child, *dir = filp->f_path.dentry;
+ struct inode *inode;
+ struct qstr qname;
+ ino_t ino = 0;
+ unsigned type = DT_UNKNOWN;
+ int ret;
+
+ qname.name = table->procname;
```

```
+ qname.len  = strlen(table->procname);
+ qname.hash = full_name_hash(qname.name, qname.len);
+
+ /* Suppress duplicates.
+  * Only fill a directory entry if it is the value that
+  * an ordinary lookup of that name returns.  Hide all
+  * others.
+  *
+  * If we ever cache this translation in the dcache
+  * I should do a dcache lookup first.  But for now
+  * it is just simpler not to.
+  */
+ ret = 0;
+ child_table = do_proc_sys_lookup(dir, &qname, &head);
+ sysctl_head_finish(head);
+ if (child_table != table)
+  return 0;
+
+ child = d_lookup(dir, &qname);
+ if (!child) {
+  struct dentry *new;
+  new = d_alloc(dir, &qname);
+  if (new) {
+   inode = proc_sys_make_inode(dir->d_inode, table);
+   if (!inode)
+    child = ERR_PTR(-ENOMEM);
+   else {
+    new->d_op = &proc_sys_dentry_operations;
+    d_add(new, inode);
+   }
+   if (child)
+    dput(new);
+   else
+    child = new;
+  }
+ }
+ if (!child || IS_ERR(child) || !child->d_inode)
+  goto end_instantiate;
+ inode = child->d_inode;
+ if (inode) {
+  ino  = inode->i_ino;
+  type = inode->i_mode >> 12;
+ }
+ dput(child);
+end_instantiate:
+ if (!ino)
+  ino= find_inode_number(dir, &qname);
+ if (!ino)
```

```
+  ino = 1;
+ return filldir(dirent, qname.name, qname.len, filp->f_pos, ino, type);
+}
+
+static int proc_sys_readdir(struct file *filp, void *dirent, filldir_t filldir)
+{
+ struct dentry *dentry = filp->f_dentry;
+ struct inode *inode = dentry->d_inode;
+ struct ctl_table_header *head = NULL;
+ struct ctl_table *table;
+ unsigned long pos;
+ int ret;
+
+ ret = -ENOTDIR;
+ if (!S_ISDIR(inode->i_mode))
+  goto out;
+
+ ret = 0;
+ switch(filp->f_pos) {
+ case 0:
+  if (filldir(dirent, ".", 1, filp->f_pos, inode->i_ino, DT_DIR) < 0)
+   goto out;
+  filp->f_pos++;
+  /* fall through */
+ case 1:
+  if (filldir(dirent, "..", 2, filp->f_pos, parent_ino(dentry), DT_DIR) < 0)
+   goto out;
+  filp->f_pos++;
+  /* fall through */
+ default:
+  pos = 2;
+  break;
+ }
+
+ /* - Find each instance of the directory
+  * - Read all entries in each instance
+  * - Before returning an entry to user space lookup the entry
+  *   by name and if I find a different entry don't return
+  *   this one because it means it is a buried dup.
+  * For sysctl this should only happen for directory entries.
+  */
+ for (head = sysctl_head_next(NULL); head; head = sysctl_head_next(head)) {
+  table = proc_sys_lookup_table(dentry, head->ctl_table);
+
+  if (!table)
+   continue;
+
+  for (; table->ctl_name || table->procname; table++, pos++) {
```

```
+   /* Can't do anything without a proc name */
+   if (!table->procname)
+    continue;
+
+   if (pos < filp->f_pos)
+    continue;
+
+   if (proc_sys_fill_cache(filp, dirent, filldir, table) < 0)
+    goto out;
+   filp->f_pos = pos + 1;
+  }
+ }
+ ret = 1;
+out:
+ sysctl_head_finish(head);
+ return ret;
+}
+
+static int proc_sys_permission(struct inode *inode, int mask, struct nameidata *nd)
+{
+ /*
+  * sysctl entries that are not writeable,
+  * are _NOT_ writeable, capabilities or not.
+  */
+ struct ctl_table_header *head;
+ struct ctl_table *table;
+ struct dentry *dentry;
+ int mode;
+ int depth;
+ int error;
+
+ head = NULL;
+ depth = PROC_I(inode)->fd;
+
+ /* First check the cached permissions, in case we don't have
+  * enough information to lookup the sysctl table entry.
+  */
+ error = -EACCES;
+ mode = inode->i_mode;
+
+ if (current->euid == 0)
+  mode >>= 6;
+ else if (in_group_p(0))
+  mode >>= 3;
+
+ if ((mode & mask & (MAY_READ|MAY_WRITE|MAY_EXEC)) == mask)
+  error = 0;
+
```

```
+ /* If we can't get a sysctl table entry the permission
+  * checks on the cached mode will have to be enough.
+  */
+ if (!nd || !depth)
+  goto out;
+
+ dentry = nd->dentry;
+ table = do_proc_sys_lookup(dentry->d_parent, &dentry->d_name, &head);
+
+ /* If the entry does not exist deny permission */
+ error = -EACCES;
+ if (!table)
+  goto out;
+
+ /* Use the permissions on the sysctl table entry */
+ error = sysctl_perm(table, mask);
+out:
+ sysctl_head_finish(head);
+ return error;
+}
+
+static int proc_sys_setattr(struct dentry *dentry, struct iattr *attr)
+{
+ struct inode *inode = dentry->d_inode;
+ int error;
+
+ if (attr->ia_valid & (ATTR_MODE | ATTR_UID | ATTR_GID))
+  return -EPERM;
+
+ error = inode_change_ok(inode, attr);
+ if (!error) {
+  error = security_inode_setattr(dentry, attr);
+  if (!error)
+   error = inode_setattr(inode, attr);
+ }
+
+ return error;
+}
+
+/* I'm lazy and don't distinguish between files and directories,
+ * until access time.
+ */
+static const struct file_operations proc_sys_file_operations = {
+ .read  = proc_sys_read,
+ .write  = proc_sys_write,
+ .readdir = proc_sys_readdir,
+};
+
```

```
+static struct inode_operations proc_sys_inode_operations = {
+ .lookup  = proc_sys_lookup,
+ .permission = proc_sys_permission,
+ .setattr = proc_sys_setattr,
+};
+
+static int proc_sys_revalidate(struct dentry *dentry, struct nameidata *nd)
+{
+ struct ctl_table_header *head;
+ struct ctl_table *table;
+ table = do_proc_sys_lookup(dentry->d_parent, &dentry->d_name, &head);
+ proc_sys_refresh_inode(dentry->d_inode, table);
+ sysctl_head_finish(head);
+ return !!table;
+}
+
+static struct dentry_operations proc_sys_dentry_operations = {
+ .d_revalidate = proc_sys_revalidate,
+};
+
+struct proc_dir_entry *proc_sys_root;
+
+int proc_sys_init(void)
+{
+ proc_sys_root = proc_mkdir("sys", NULL);
+ proc_sys_root->proc_iops = &proc_sys_inode_operations;
+ proc_sys_root->proc_fops = &proc_sys_file_operations;
+ proc_sys_root->nlink = 0;
+ return 0;
+}
diff --git a/fs/proc/root.c b/fs/proc/root.c
index 8059e92..4d42406 100644
--- a/fs/proc/root.c
+++ b/fs/proc/root.c
@@ -23,10 +23,6 @@

 struct proc_dir_entry *proc_net, *proc_net_stat, *proc_bus, *proc_root_fs, *proc_root_driver;

-#ifdef CONFIG_SYSCTL
-struct proc_dir_entry *proc_sys_root;
-#endif
-
 static int proc_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
 {
@@ -71,9 +67,6 @@ void __init proc_root_init(void)
 #ifdef CONFIG_SYSVIPC
  proc_mkdir("sysvipc", NULL);
```

```
 #endif
-#ifdef CONFIG_SYSCTL
- proc_sys_root = proc_mkdir("sys", NULL);
-#endif
  proc_root_fs = proc_mkdir("fs", NULL);
  proc_root_driver = proc_mkdir("driver", NULL);
  proc_mkdir("fs/nfsd", NULL); /* somewhere for the nfsd filesystem to be mounted */
@@ -86,6 +79,9 @@ void __init proc_root_init(void)
  proc_device_tree_init();
 #endif
  proc_bus = proc_mkdir("bus", NULL);
+#ifdef CONFIG_SYSCTL
+ proc_sys_init();
+#endif
 }

 static int proc_root_getattr(struct vfsmount *mnt, struct dentry *dentry, struct kstat *stat
diff --git a/init/main.c b/init/main.c
index 8af5c6e..7926e5d 100644
--- a/init/main.c
+++ b/init/main.c
@@ -86,7 +86,6 @@ extern void init_IRQ(void);
 extern void fork_init(unsigned long);
 extern void mca_init(void);
 extern void sbus_init(void);
-extern void sysctl_init(void);
 extern void signals_init(void);
 extern void pidhash_init(void);
 extern void pidmap_init(void);
@@ -688,9 +687,6 @@ static void __init do_basic_setup(void)
  usermodehelper_init();
  driver_init();

-#ifdef CONFIG_SYSCTL
- sysctl_init();
-#endif
 #ifdef CONFIG_PROC_FS
  init_irq_proc();
 #endif
diff --git a/kernel/sysctl.c b/kernel/sysctl.c
index ec5e4a1..4b45bdb 100644
--- a/kernel/sysctl.c
+++ b/kernel/sysctl.c
@@ -159,26 +159,6 @@ int sysctl_legacy_va_layout;



-/* /proc declarations: */
```

```
-
-#ifdef CONFIG_PROC_SYSCTL
-
-static ssize_t proc_readsys(struct file *, char __user *, size_t, loff_t *);
-static ssize_t proc_writesys(struct file *, const char __user *, size_t, loff_t *);
-static int proc_opensys(struct inode *, struct file *);
-
-const struct file_operations proc_sys_file_operations = {
- .open  = proc_opensys,
- .read  = proc_readsys,
- .write  = proc_writesys,
-};
-
-extern struct proc_dir_entry *proc_sys_root;
-
-static void register_proc_table(ctl_table *, struct proc_dir_entry *, void *);
-static void unregister_proc_table(ctl_table *, struct proc_dir_entry *);
-#endif
-
 /* The default sysctl tables: */

 static ctl_table root_table[] = {
@@ -1102,13 +1082,6 @@ struct ctl_table_header *sysctl_head_next(struct ctl_table_header
*prev)
  return NULL;
 }

-void __init sysctl_init(void)
-{
-#ifdef CONFIG_PROC_SYSCTL
- register_proc_table(root_table, proc_sys_root, &root_table_header);
-#endif
-}
-
 #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)
@@ -1345,9 +1318,6 @@ struct ctl_table_header *register_sysctl_table(ctl_table * table)
  spin_lock(&sysctl_lock);
  list_add_tail(&tmp->ctl_entry, &root_table_header.ctl_entry);
  spin_unlock(&sysctl_lock);
-#ifdef CONFIG_PROC_SYSCTL
- register_proc_table(table, proc_sys_root, tmp);
-#endif
  return tmp;
 }

@@ -1363,9 +1333,6 @@ void unregister_sysctl_table(struct ctl_table_header * header)
```

```
  might_sleep();
  spin_lock(&sysctl_lock);
  start_unregistering(header);
-#ifdef CONFIG_PROC_SYSCTL
- unregister_proc_table(header->ctl_table, proc_sys_root);
-#endif
  spin_unlock(&sysctl_lock);
  kfree(header);
 }
@@ -1389,155 +1356,6 @@ void unregister_sysctl_table(struct ctl_table_header * table)

 #ifdef CONFIG_PROC_SYSCTL

-/* Scan the sysctl entries in table and add them all into /proc */
-static void register_proc_table(ctl_table * table, struct proc_dir_entry *root, void *set)
-{
- struct proc_dir_entry *de;
- int len;
- mode_t mode;
-
- for (; table->ctl_name || table->procname; table++) {
-  /* Can't do anything without a proc name. */
-  if (!table->procname)
-   continue;
-  /* Maybe we can't do anything with it... */
-  if (!table->proc_handler && !table->child) {
-   printk(KERN_WARNING "SYSCTL: Can't register %s\n",
-    table->procname);
-   continue;
-  }
-
-  len = strlen(table->procname);
-  mode = table->mode;
-
-  de = NULL;
-  if (table->proc_handler)
-   mode |= S_IFREG;
-  else {
-   mode |= S_IFDIR;
-   for (de = root->subdir; de; de = de->next) {
-    if (proc_match(len, table->procname, de))
-     break;
-   }
-   /* If the subdir exists already, de is non-NULL */
-  }
-
-  if (!de) {
-   de = create_proc_entry(table->procname, mode, root);
```

```
-   if (!de)
-    continue;
-   de->set = set;
-   de->data = (void *) table;
-   if (table->proc_handler)
-    de->proc_fops = &proc_sys_file_operations;
-  }
-  table->de = de;
-  if (de->mode & S_IFDIR)
-   register_proc_table(table->child, de, set);
- }
-}
-
-/*
- * Unregister a /proc sysctl table and any subdirectories.
- */
-static void unregister_proc_table(ctl_table * table, struct proc_dir_entry *root)
-{
- struct proc_dir_entry *de;
- for (; table->ctl_name || table->procname; table++) {
-  if (!(de = table->de))
-   continue;
-  if (de->mode & S_IFDIR) {
-   if (!table->child) {
-    printk (KERN_ALERT "Help - malformed sysctl tree on free\n");
-    continue;
-   }
-   unregister_proc_table(table->child, de);
-
-   /* Don't unregister directories which still have entries.. */
-   if (de->subdir)
-    continue;
-  }
-
-  /*
-   * In any case, mark the entry as goner; we'll keep it
-   * around if it's busy, but we'll know to do nothing with
-   * its fields.  We are under sysctl_lock here.
-   */
-  de->data = NULL;
-
-  /* Don't unregister proc entries that are still being used.. */
-  if (atomic_read(&de->count))
-   continue;
-
-  table->de = NULL;
-  remove_proc_entry(table->procname, root);
- }
```

```
-}
-
-static ssize_t do_rw_proc(int write, struct file * file, char __user * buf,
-     size_t count, loff_t *ppos)
-{
- int op;
- struct proc_dir_entry *de = PDE(file->f_path.dentry->d_inode);
- struct ctl_table *table;
- size_t res;
- ssize_t error = -ENOTDIR;
-
- spin_lock(&sysctl_lock);
- if (de && de->data && use_table(de->set)) {
-  /*
-   * at that point we know that sysctl was not unregistered
-   * and won't be until we finish
-   */
-  spin_unlock(&sysctl_lock);
-  table = (struct ctl_table *) de->data;
-  if (!table || !table->proc_handler)
-   goto out;
-  error = -EPERM;
-  op = (write ? 002 : 004);
-  if (sysctl_perm(table, op))
-   goto out;
-
-  /* careful: calling conventions are nasty here */
-  res = count;
-  error = (*table->proc_handler)(table, write, file,
-      buf, &res, ppos);
-  if (!error)
-   error = res;
- out:
-  spin_lock(&sysctl_lock);
-  unuse_table(de->set);
- }
- spin_unlock(&sysctl_lock);
- return error;
-}
-
-static int proc_opensys(struct inode *inode, struct file *file)
-{
- if (file->f_mode & FMODE_WRITE) {
-  /*
-   * sysctl entries that are not writable,
-   * are _NOT_ writable, capabilities or not.
-   */
-  if (!(inode->i_mode & S_IWUSR))
```

```
-   return -EPERM;
- }
-
- return 0;
-}
-
-static ssize_t proc_readsys(struct file * file, char __user * buf,
-       size_t count, loff_t *ppos)
-{
- return do_rw_proc(0, file, buf, count, ppos);
-}
-
-static ssize_t proc_writesys(struct file * file, const char __user * buf,
-        size_t count, loff_t *ppos)
-{
- return do_rw_proc(1, file, (char __user *) buf, count, ppos);
-}
-
 static int _proc_do_string(void* data, int maxlen, int write,
       struct file *filp, void __user *buffer,
       size_t *lenp, loff_t *ppos)
--
1.4.4.1.g278f
```

_____

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