Subject: [PATCH 04/15] sysfs: Implement __sysfs_get_dentry
Posted by ebiederm on Fri, 04 Jul 2008 01:09:04 GMT
View Forum Message <> Reply to Message

This function is similar but much simpler to sysfs_get_dentry
returns a sysfs dentry if one curently exists.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>
Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
Acked-by: Tejun Heo <tj@kernel.org>
---
 fs/sysfs/dir.c | 39 ++
 1 files changed, 39 insertions(+), 0 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 69c40ed..df9934a 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -764,6 +764,45 @@ void sysfs_remove_dir(struct kobject * kobj)
  __sysfs_remove_dir(sd);
 }

+/**
+ * __sysfs_get_dentry - get dentry for the given sysfs_dirent
+ * @sb: superblock of the dentry to return
+ * @sd: sysfs_dirent of interest
+ *
+ * Get dentry for @sd.  Only return a dentry if one currently
+ * exists.
+ *
+ * LOCKING:
+ * Kernel thread context (may sleep)
+ *
+ * RETURNS:
+ * Pointer to found dentry on success, NULL on failure.
+ */
+static struct dentry *__sysfs_get_dentry(struct super_block *sb,
+      struct sysfs_dirent *sd)
+{
+ struct inode *inode;
+ struct dentry *dentry = NULL;
+
+ inode = ilookup5_nowait(sysfs_sb, sd->s_ino, sysfs_ilookup_test, sd);
+ if (inode && !(inode->i_state & I_NEW)) {
+  struct dentry *alias;
+  spin_lock(&dcache_lock);
+  list_for_each_entry(alias, &inode->i_dentry, d_alias) {
```

```
+   if (!IS_ROOT(alias) && d_unhashed(alias))
+    continue;
+   if (alias->d_sb != sb)
+    continue;
+   dentry = alias;
+   dget_locked(dentry);
+   break;
+  }
+  spin_unlock(&dcache_lock);
+ }
+ iput(inode);
+ return dentry;
+}
+
 int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  struct sysfs_dirent *sd = kobj->sd;
--
1.5.3.rc6.17.g1911
```

Subject: [PATCH 05/15] sysfs: Rename Support multiple superblocks
Posted by ebiederm on Fri, 04 Jul 2008 01:10:05 GMT
View Forum Message <> Reply to Message

This patch modifies the sysfs_rename_dir and sysfs_move_dir routines
to support multiple sysfs dentry tries rooted in different
sysfs superblocks.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>
Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
Acked-by: Tejun Heo <tj@kernel.org>
---
 fs/sysfs/dir.c | 193 ++----------
 1 files changed, 136 insertions(+), 57 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index df9934a..b2d92ea 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -803,43 +803,113 @@ static struct dentry *__sysfs_get_dentry(struct super_block *sb,
  return dentry;
```

```
 }

+struct sysfs_rename_struct {
+ struct list_head list;
+ struct dentry *old_dentry;
+ struct dentry *new_dentry;
+ struct dentry *old_parent;
+ struct dentry *new_parent;
+};
+
+static void post_rename(struct list_head *head)
+{
+ struct sysfs_rename_struct *srs;
+ while (!list_empty(head)) {
+  srs = list_entry(head->next, struct sysfs_rename_struct, list);
+  dput(srs->old_dentry);
+  dput(srs->new_dentry);
+  dput(srs->old_parent);
+  dput(srs->new_parent);
+  list_del(&srs->list);
+  kfree(srs);
+ }
+}
+
+static int prep_rename(struct list_head *head,
+ struct sysfs_dirent *sd, struct sysfs_dirent *new_parent_sd,
+ const char *name)
+{
+ struct sysfs_rename_struct *srs;
+ struct super_block *sb;
+ struct dentry *dentry;
+ int error;
+
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+  dentry = sysfs_get_dentry(sb, sd);
+  if (dentry == ERR_PTR(-EXDEV))
+   continue;
+  if (IS_ERR(dentry)) {
+   error = PTR_ERR(dentry);
+   goto err_out;
+  }
+
+  srs = kzalloc(sizeof(*srs), GFP_KERNEL);
+  if (!srs) {
+   error = -ENOMEM;
+   dput(dentry);
+   goto err_out;
+  }
```

```
+
+  INIT_LIST_HEAD(&srs->list);
+  list_add(head, &srs->list);
+  srs->old_dentry = dentry;
+  srs->old_parent = dget(dentry->d_parent);
+
+  dentry = sysfs_get_dentry(sb, new_parent_sd);
+  if (IS_ERR(dentry)) {
+   error = PTR_ERR(dentry);
+   goto err_out;
+  }
+  srs->new_parent = dentry;
+
+  error = -ENOMEM;
+  dentry = d_alloc_name(srs->new_parent, name);
+  if (!dentry)
+   goto err_out;
+  srs->new_dentry = dentry;
+ }
+ return 0;
+
+err_out:
+ post_rename(head);
+ return error;
+}
+
 int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  struct sysfs_dirent *sd = kobj->sd;
- struct dentry *parent = NULL;
- struct dentry *old_dentry = NULL, *new_dentry = NULL;
+ struct list_head todo;
+ struct sysfs_rename_struct *srs;
+ struct inode *parent_inode = NULL;
  const char *dup_name = NULL;
  int error;

+ INIT_LIST_HEAD(&todo);
  mutex_lock(&sysfs_rename_mutex);

  error = 0;
  if (strcmp(sd->s_name, new_name) == 0)
   goto out; /* nothing to rename */

- /* get the original dentry */
- old_dentry = sysfs_get_dentry(sysfs_sb, sd);
- if (IS_ERR(old_dentry)) {
-  error = PTR_ERR(old_dentry);
```

```
-  old_dentry = NULL;
-  goto out;
- }
+ sysfs_grab_supers();
+ error = prep_rename(&todo, sd, sd->s_parent, new_name);
+ if (error)
+  goto out_release;

- parent = old_dentry->d_parent;
+ error = -ENOMEM;
+ mutex_lock(&sysfs_mutex);
+ parent_inode = sysfs_get_inode(sd->s_parent);
+ mutex_unlock(&sysfs_mutex);
+ if (!parent_inode)
+  goto out_release;

- /* lock parent and get dentry for new name */
- mutex_lock(&parent->d_inode->i_mutex);
+ mutex_lock(&parent_inode->i_mutex);
  mutex_lock(&sysfs_mutex);

  error = -EEXIST;
  if (sysfs_find_dirent(sd->s_parent, new_name))
   goto out_unlock;

- error = -ENOMEM;
- new_dentry = d_alloc_name(parent, new_name);
- if (!new_dentry)
-  goto out_unlock;
-
  /* rename sysfs_dirent */
  error = -ENOMEM;
  new_name = dup_name = kstrdup(new_name, GFP_KERNEL);
@@ -850,17 +920,21 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  sd->s_name = new_name;

  /* rename */
- d_add(new_dentry, NULL);
- d_move(old_dentry, new_dentry);
+ list_for_each_entry(srs, &todo, list) {
+  d_add(srs->new_dentry, NULL);
+  d_move(srs->old_dentry, srs->new_dentry);
+ }

  error = 0;
- out_unlock:
+out_unlock:
  mutex_unlock(&sysfs_mutex);
```

```
- mutex_unlock(&parent->d_inode->i_mutex);
+ mutex_unlock(&parent_inode->i_mutex);
  kfree(dup_name);
- dput(old_dentry);
- dput(new_dentry);
- out:
+out_release:
+ iput(parent_inode);
+ post_rename(&todo);
+ sysfs_release_supers();
+out:
  mutex_unlock(&sysfs_rename_mutex);
  return error;
 }
@@ -869,10 +943,12 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)
 {
  struct sysfs_dirent *sd = kobj->sd;
  struct sysfs_dirent *new_parent_sd;
- struct dentry *old_parent, *new_parent = NULL;
- struct dentry *old_dentry = NULL, *new_dentry = NULL;
+ struct list_head todo;
+ struct sysfs_rename_struct *srs;
+ struct inode *old_parent_inode = NULL, *new_parent_inode = NULL;
  int error;

+ INIT_LIST_HEAD(&todo);
  mutex_lock(&sysfs_rename_mutex);
  BUG_ON(!sd->s_parent);
  new_parent_sd = new_parent_kobj->sd ? new_parent_kobj->sd : &sysfs_root;
@@ -881,26 +957,29 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)
  if (sd->s_parent == new_parent_sd)
   goto out; /* nothing to move */

- /* get dentries */
- old_dentry = sysfs_get_dentry(sysfs_sb, sd);
- if (IS_ERR(old_dentry)) {
-  error = PTR_ERR(old_dentry);
-  old_dentry = NULL;
-  goto out;
- }
- old_parent = old_dentry->d_parent;
+ sysfs_grab_supers();
+ error = prep_rename(&todo, sd, new_parent_sd, sd->s_name);
+ if (error)
+  goto out_release;
```

```
- new_parent = sysfs_get_dentry(sysfs_sb, new_parent_sd);
- if (IS_ERR(new_parent)) {
-  error = PTR_ERR(new_parent);
-  new_parent = NULL;
-  goto out;
- }
+ error = -ENOMEM;
+ mutex_lock(&sysfs_mutex);
+ old_parent_inode = sysfs_get_inode(sd->s_parent);
+ mutex_unlock(&sysfs_mutex);
+ if (!old_parent_inode)
+  goto out_release;
+
+ error = -ENOMEM;
+ mutex_lock(&sysfs_mutex);
+ new_parent_inode = sysfs_get_inode(new_parent_sd);
+ mutex_unlock(&sysfs_mutex);
+ if (!new_parent_inode)
+  goto out_release;

 again:
- mutex_lock(&old_parent->d_inode->i_mutex);
- if (!mutex_trylock(&new_parent->d_inode->i_mutex)) {
-  mutex_unlock(&old_parent->d_inode->i_mutex);
+ mutex_lock(&old_parent_inode->i_mutex);
+ if (!mutex_trylock(&new_parent_inode->i_mutex)) {
+  mutex_unlock(&old_parent_inode->i_mutex);
   goto again;
  }
  mutex_lock(&sysfs_mutex);
@@ -909,14 +988,11 @@ again:
  if (sysfs_find_dirent(new_parent_sd, sd->s_name))
   goto out_unlock;

- error = -ENOMEM;
- new_dentry = d_alloc_name(new_parent, sd->s_name);
- if (!new_dentry)
-  goto out_unlock;
-
  error = 0;
- d_add(new_dentry, NULL);
- d_move(old_dentry, new_dentry);
+ list_for_each_entry(srs, &todo, list) {
+  d_add(srs->new_dentry, NULL);
+  d_move(srs->old_dentry, srs->new_dentry);
+ }

  /* Remove from old parent's list and insert into new parent's list. */
```

```
  sysfs_unlink_sibling(sd);
@@ -925,14 +1001,17 @@ again:
  sd->s_parent = new_parent_sd;
  sysfs_link_sibling(sd);

- out_unlock:
+out_unlock:
  mutex_unlock(&sysfs_mutex);
- mutex_unlock(&new_parent->d_inode->i_mutex);
- mutex_unlock(&old_parent->d_inode->i_mutex);
- out:
- dput(new_parent);
- dput(old_dentry);
- dput(new_dentry);
+ mutex_unlock(&new_parent_inode->i_mutex);
+ mutex_unlock(&old_parent_inode->i_mutex);
+
+out_release:
+ iput(new_parent_inode);
+ iput(old_parent_inode);
+ post_rename(&todo);
+ sysfs_release_supers();
+out:
  mutex_unlock(&sysfs_rename_mutex);
  return error;
 }
--
1.5.3.rc6.17.g1911
```

Subject: [PATCH 06/15] Introduce sysfs_sd_setattr and fix sysfs_chmod
Posted by ebiederm on Fri, 04 Jul 2008 01:11:40 GMT
View Forum Message <> Reply to Message

Currently sysfs_chmod calls sys_setattr which in turn calls
inode_change_ok which checks to see if it is ok for the current user
space process to change tha attributes. Since sysfs_chmod_file has
only kernel mode clients denying them permission if user space is the
problem is completely inappropriate.

Therefore factor out sysfs_sd_setattr which does not call
inode_change_ok and modify sysfs_chmod_file to call it.

In addition setting victim_sd->s_mode explicitly in sysfs_chmod_file
is redundant so remove that as well.

Thanks to Tejun Heo <htejun@gmail.com>, and
Daniel Lezcano <dlezcano@fr.ibm.com> for working on this
and spotting this case.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/file.c | 5 +----
 fs/sysfs/inode.c | 23 +++++++++++++++++++-------
 fs/sysfs/sysfs.h | 1 +
 3 files changed, 18 insertions(+), 11 deletions(-)

diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index cb5dd3f..1304b3a 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -600,13 +600,10 @@ int sysfs_chmod_file(struct kobject *kobj, struct attribute *attr, mode_t
mode)
  newattrs.ia_mode = (mode & S_IALLUGO) | (inode->i_mode & ~S_IALLUGO);
  newattrs.ia_valid = ATTR_MODE | ATTR_CTIME;
  newattrs.ia_ctime = current_fs_time(inode->i_sb);
- rc = sysfs_setattr(victim, &newattrs);
+ rc = sysfs_sd_setattr(victim_sd, inode, &newattrs);

  if (rc == 0) {
   fsnotify_change(victim, newattrs.ia_valid);
- mutex_lock(&sysfs_mutex);
- victim_sd->s_mode = newattrs.ia_mode;
- mutex_unlock(&sysfs_mutex);
  }

  mutex_unlock(&inode->i_mutex);
diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index eb53c63..80f8fd4 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -42,10 +42,9 @@ int __init sysfs_inode_init(void)
  return bdi_init(&sysfs_backing_dev_info);
 }

-int sysfs_setattr(struct dentry * dentry, struct iattr * iattr)
+int sysfs_sd_setattr(struct sysfs_dirent *sd, struct inode *inode,
+   struct iattr * iattr)
 {
- struct inode * inode = dentry->d_inode;
- struct sysfs_dirent * sd = dentry->d_fsdata;

```
  struct iattr * sd_iattr;
  unsigned int ia_valid = iattr->ia_valid;
  int error;
@@ -55,10 +54,6 @@ int sysfs_setattr(struct dentry * dentry, struct iattr * iattr)

  sd_iattr = sd->s_iattr;

- error = inode_change_ok(inode, iattr);
- if (error)
-  return error;
-
  iattr->ia_valid &= ~ATTR_SIZE; /* ignore size changes */

  error = inode_setattr(inode, iattr);
@@ -104,6 +99,20 @@ int sysfs_setattr(struct dentry * dentry, struct iattr * iattr)
  return error;
 }

+int sysfs_setattr(struct dentry *dentry, struct iattr *iattr)
+{
+ struct inode * inode = dentry->d_inode;
+ struct sysfs_dirent * sd = dentry->d_fsdata;
+ int error;
+
+ error = inode_change_ok(inode, iattr);
+ if (error)
+  return error;
+
+ return sysfs_sd_setattr(sd, inode, iattr);
+}
+
+
 static inline void set_default_inode_attr(struct inode * inode, mode_t mode)
 {
  inode->i_mode = mode;
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index b1bdc6e..5ee5d0a 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -154,6 +154,7 @@ static inline void sysfs_put(struct sysfs_dirent *sd)
  * inode.c
  */
 struct inode *sysfs_get_inode(struct sysfs_dirent *sd);
+int sysfs_sd_setattr(struct sysfs_dirent *sd, struct inode *inode, struct iattr *iattr);
 int sysfs_setattr(struct dentry *dentry, struct iattr *iattr);
 int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char *name);
 int sysfs_inode_init(void);
--
```

1.5.3.rc6.17.g1911

Subject: Re: [PATCH 06/15] Introduce sysfs_sd_setattr and fix sysfs_chmod
Posted by Tejun Heo on Fri, 04 Jul 2008 06:40:19 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Currently sysfs_chmod calls sys_setattr which in turn calls
> inode_change_ok which checks to see if it is ok for the current user
> space process to change tha attributes. Since sysfs_chmod_file has
> only kernel mode clients denying them permission if user space is the
> problem is completely inappropriate.
>
> Therefore factor out sysfs_sd_setattr which does not call
> inode_change_ok and modify sysfs_chmod_file to call it.
>
> In addition setting victim_sd->s_mode explicitly in sysfs_chmod_file
> is redundant so remove that as well.
>
> Thanks to Tejun Heo <htejun@gmail.com>, and
> Daniel Lezcano <dlezcano@fr.ibm.com> for working on this
> and spotting this case.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Acked-by: Tejun Heo <tj@kernel.org>

--
tejun