Subject: [PATCH 0/5] On to usable sysfs shadow directory support...
Posted by ebiederm on Fri, 06 Apr 2007 16:43:43 GMT
View Forum Message <> Reply to Message

The following patchset has been tested on 2.6.21-rc6 + Kay's
driver-core-fix-namespace-issue-with-devices-assigned-to-classes.patch

It has been tested both with CONFIG_SYSFS_DEPRECATED set and
unset. Although more testing has been involved with CONFIG_SYSFS_DEPRECATED
unset because that was the hard case.

After the change of network devices from struct class_device to struct device
it has taken me a while to figure out how to get the shadow directory support
to actually work in a maintainable race free manner. I wound up pushing
a lot more of the logic down into sysfs to accomplish this (primarily shadow
directory creation and deletion). Which radically change the interfaces to
how I work with shadow directories at the upper levels.

So this patchset:

- fixes some aesthetic issues with Kay's patch.
- Rips out almost all of the old shadow directory support.
- Adds new shadow directory support.
- Adds some shadow directory friendly symlink manipulators
- Adds struct class and struct support for shadow directories.

Eric

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

Subject: [PATCH 1/5] kobject: Comment and warning fixes to kobject.c
Posted by ebiederm on Fri, 06 Apr 2007 16:47:11 GMT
View Forum Message <> Reply to Message

This dots some i's and crosses some t's after left over from when
kobject_kset_add_dir was built from kobject_add_dir.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
lib/kobject.c | 13 ++++++++++---
1 files changed, 10 insertions(+), 3 deletions(-)

diff --git a/lib/kobject.c b/lib/kobject.c
index f664551..09cb276 100644
--- a/lib/kobject.c

```
+++ b/lib/kobject.c
@@ -488,7 +488,7 @@ static struct kobj_type dir_ktype = {
 };

 /**
- * kobject__kset_add_dir - add sub directory of object.
+ * kobject_kset_add_dir - add sub directory of object.
  * @kset:  kset the directory is belongs to.
  * @parent: object in which a directory is created.
  * @name: directory name.
@@ -514,8 +514,8 @@ struct kobject *kobject_kset_add_dir(struct kset *kset,
  kobject_set_name(k, name);
  ret = kobject_register(k);
  if (ret < 0) {
-  printk(KERN_WARNING "kobject_add_dir: "
-   "kobject_register error: %d\n", ret);
+  printk(KERN_WARNING "%s: kobject_register error: %d\n",
+   __func__, ret);
   kobject_del(k);
   return NULL;
  }
@@ -523,6 +523,13 @@ struct kobject *kobject_kset_add_dir(struct kset *kset,
  return k;
 }

+/**
+ * kobject_add_dir - add sub directory of object.
+ * @parent: object in which a directory is created.
+ * @name: directory name.
+ *
+ * Add a plain directory object as child of given object.
+ */
 struct kobject *kobject_add_dir(struct kobject *parent, const char *name)
 {
  return kobject_kset_add_dir(NULL, parent, name);
--
1.5.0.g53756
```

Subject: [PATCH 2/5] sysfs: Remove first pass at shadow directory support
Posted by ebiederm on Fri, 06 Apr 2007 16:48:42 GMT
View Forum Message <> Reply to Message

While shadow directories appear to be a good idea, the current scheme
of controlling their creation and destruction outside of sysfs appears
to be a locking and maintenance nightmare in the face of sysfs directories
dynamically coming and going. Which can now occur for directories containing
network devices when CONFIG_SYSFS_DEPRECATED is not set.

This patch removes everything from the initial shadow directory support
that allowed the shadow directory creation to be controlled at a higher
level. So except for a few bits of sysfs_rename_dir everything from
commit b592fcfe7f06c15ec11774b5be7ce0de3aa86e73 is now gone.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c | 196 +++++++++-------------------------------
 fs/sysfs/group.c | 1 -
 fs/sysfs/inode.c | 10 ---
 fs/sysfs/mount.c | 2 +-
 fs/sysfs/sysfs.h | 5 -
 include/linux/kobject.h | 4 -
 include/linux/sysfs.h | 24 +-----
 lib/kobject.c | 43 +---------
 8 files changed, 45 insertions(+), 240 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 85a6686..d6ab015 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -33,7 +33,8 @@ static struct dentry_operations sysfs_dentry_ops = {
 /*
  * Allocates a new sysfs_dirent and links it to the parent sysfs_dirent
  */
-static struct sysfs_dirent * __sysfs_new_dirent(void * element)
+static struct sysfs_dirent * sysfs_new_dirent(struct sysfs_dirent * parent_sd,
+   void * element)
 {
  struct sysfs_dirent * sd;

@@ -44,28 +45,12 @@ static struct sysfs_dirent * __sysfs_new_dirent(void * element)
  atomic_set(&sd->s_count, 1);
  atomic_set(&sd->s_event, 1);
  INIT_LIST_HEAD(&sd->s_children);
- INIT_LIST_HEAD(&sd->s_sibling);
+ list_add(&sd->s_sibling, &parent_sd->s_children);
  sd->s_element = element;

  return sd;
 }

```
-static void __sysfs_list_dirent(struct sysfs_dirent *parent_sd,
-         struct sysfs_dirent *sd)
-{
- if (sd)
-  list_add(&sd->s_sibling, &parent_sd->s_children);
-}
-
-static struct sysfs_dirent * sysfs_new_dirent(struct sysfs_dirent *parent_sd,
-       void * element)
-{
- struct sysfs_dirent *sd;
- sd = __sysfs_new_dirent(element);
- __sysfs_list_dirent(parent_sd, sd);
- return sd;
-}
-
 /*
  *
  * Return -EEXIST if there is already a sysfs element with the same name for
@@ -92,14 +77,14 @@ int sysfs_dirent_exist(struct sysfs_dirent *parent_sd,
 }


-static struct sysfs_dirent *
-__sysfs_make_dirent(struct dentry *dentry, void *element, mode_t mode, int type)
+int sysfs_make_dirent(struct sysfs_dirent * parent_sd, struct dentry * dentry,
+    void * element, umode_t mode, int type)
 {
  struct sysfs_dirent * sd;

- sd = __sysfs_new_dirent(element);
+ sd = sysfs_new_dirent(parent_sd, element);
  if (!sd)
-  goto out;
+  return -ENOMEM;

  sd->s_mode = mode;
  sd->s_type = type;
@@ -109,19 +94,7 @@ __sysfs_make_dirent(struct dentry *dentry, void *element, mode_t mode,
int type)
   dentry->d_op = &sysfs_dentry_ops;
  }

-out:
- return sd;
-}
-
-int sysfs_make_dirent(struct sysfs_dirent * parent_sd, struct dentry * dentry,
```

```
-   void * element, umode_t mode, int type)
-{
- struct sysfs_dirent *sd;
-
- sd = __sysfs_make_dirent(dentry, element, mode, type);
- __sysfs_list_dirent(parent_sd, sd);
-
- return sd ? 0 : -ENOMEM;
+ return 0;
 }

 static int init_dir(struct inode * inode)
@@ -193,10 +166,9 @@ int sysfs_create_subdir(struct kobject * k, const char * n, struct dentry **
d)
 /**
  * sysfs_create_dir - create a directory for an object.
  * @kobj:  object we're creating directory for.
- * @shadow_parent: parent parent object.
  */

-int sysfs_create_dir(struct kobject * kobj, struct dentry *shadow_parent)
+int sysfs_create_dir(struct kobject * kobj)
 {
  struct dentry * dentry = NULL;
  struct dentry * parent;
@@ -204,9 +176,7 @@ int sysfs_create_dir(struct kobject * kobj, struct dentry *shadow_parent)

  BUG_ON(!kobj);

- if (shadow_parent)
-  parent = shadow_parent;
- else if (kobj->parent)
+ if (kobj->parent)
   parent = kobj->parent->dentry;
  else if (sysfs_mount && sysfs_mount->mnt_sb)
   parent = sysfs_mount->mnt_sb->s_root;
@@ -327,12 +297,21 @@ void sysfs_remove_subdir(struct dentry * d)
 }


-static void __sysfs_remove_dir(struct dentry *dentry)
+/**
+ * sysfs_remove_dir - remove an object's directory.
+ * @kobj: object.
+ *
+ * The only thing special about this is that we remove any files in
+ * the directory before we remove the directory, and we've inlined
+ * what used to be sysfs_rmdir() below, instead of calling separately.
```

```
+ */
+
+void sysfs_remove_dir(struct kobject * kobj)
 {
+ struct dentry * dentry = dget(kobj->dentry);
  struct sysfs_dirent * parent_sd;
  struct sysfs_dirent * sd, * tmp;

- dget(dentry);
  if (!dentry)
   return;

@@ -353,46 +332,28 @@ static void __sysfs_remove_dir(struct dentry *dentry)
   * Drop reference from dget() on entrance.
   */
  dput(dentry);
-}
-
-/**
- * sysfs_remove_dir - remove an object's directory.
- * @kobj: object.
- *
- * The only thing special about this is that we remove any files in
- * the directory before we remove the directory, and we've inlined
- * what used to be sysfs_rmdir() below, instead of calling separately.
- */
-
-void sysfs_remove_dir(struct kobject * kobj)
-{
- __sysfs_remove_dir(kobj->dentry);
  kobj->dentry = NULL;
 }

-int sysfs_rename_dir(struct kobject * kobj, struct dentry *new_parent,
-       const char *new_name)
+int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  int error = 0;
- struct dentry * new_dentry;
+ struct inode * inode;
+ struct dentry * new_dentry, * old_parent, * new_parent;

- if (!new_parent)
-  return -EFAULT;
+ if (!kobj->parent)
+  return -EINVAL;
+
+ old_parent = dget(kobj->dentry->d_parent);
```

```
+ inode = old_parent->d_inode;

  down_write(&sysfs_rename_sem);
- mutex_lock(&new_parent->d_inode->i_mutex);
+ mutex_lock(&inode->i_mutex);

+ new_parent = kobj->dentry->d_parent;
  new_dentry = lookup_one_len(new_name, new_parent, strlen(new_name));
  if (!IS_ERR(new_dentry)) {
-  /* By allowing two different directories with the
-   * same d_parent we allow this routine to move
-   * between different shadows of the same directory
-   */
-  if (kobj->dentry->d_parent->d_inode != new_parent->d_inode)
-   return -EINVAL;
-  else if (new_dentry->d_parent->d_inode != new_parent->d_inode)
-   error = -EINVAL;
-  else if (new_dentry == kobj->dentry)
+  if (new_dentry == kobj->dentry)
    error = -EINVAL;
   else if (!new_dentry->d_inode) {
    error = kobject_set_name(kobj, "%s", new_name);
@@ -414,9 +375,11 @@ int sysfs_rename_dir(struct kobject * kobj, struct dentry *new_parent,
    error = -EEXIST;
   dput(new_dentry);
  }
- mutex_unlock(&new_parent->d_inode->i_mutex);
+ mutex_unlock(&inode->i_mutex);
  up_write(&sysfs_rename_sem);

+ dput(old_parent);
+
  return error;
 }

@@ -595,95 +558,6 @@ static loff_t sysfs_dir_lseek(struct file * file, loff_t offset, int origin)
  return offset;
 }

-
-/**
- * sysfs_make_shadowed_dir - Setup so a directory can be shadowed
- * @kobj: object we're creating shadow of.
- */
-
-int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
-{
```

```
- struct inode *inode;
- struct inode_operations *i_op;
-
- inode = kobj->dentry->d_inode;
- if (inode->i_op != &sysfs_dir_inode_operations)
-  return -EINVAL;
-
- i_op = kmalloc(sizeof(*i_op), GFP_KERNEL);
- if (!i_op)
-  return -ENOMEM;
-
- memcpy(i_op, &sysfs_dir_inode_operations, sizeof(*i_op));
- i_op->follow_link = follow_link;
-
- /* Locking of inode->i_op?
-  * Since setting i_op is a single word write and they
-  * are atomic we should be ok here.
-  */
- inode->i_op = i_op;
- return 0;
-}
-
-/**
- * sysfs_create_shadow_dir - create a shadow directory for an object.
- * @kobj: object we're creating directory for.
- *
- * sysfs_make_shadowed_dir must already have been called on this
- * directory.
- */
-
-struct dentry *sysfs_create_shadow_dir(struct kobject *kobj)
-{
- struct sysfs_dirent *sd;
- struct dentry *parent, *dir, *shadow;
- struct inode *inode;
-
- dir = kobj->dentry;
- inode = dir->d_inode;
- parent = dir->d_parent;
- shadow = ERR_PTR(-EINVAL);
- if (!sysfs_is_shadowed_inode(inode))
-  goto out;
-
- shadow = d_alloc(parent, &dir->d_name);
- if (!shadow)
-  goto nomem;
-
- sd = __sysfs_make_dirent(shadow, kobj, inode->i_mode, SYSFS_DIR);
```

```
- if (!sd)
-  goto nomem;
-
- d_instantiate(shadow, igrab(inode));
- inc_nlink(inode);
- inc_nlink(parent->d_inode);
- shadow->d_op = &sysfs_dentry_ops;
-
- dget(shadow);  /* Extra count - pin the dentry in core */
-
-out:
- return shadow;
-nomem:
- dput(shadow);
- shadow = ERR_PTR(-ENOMEM);
- goto out;
-}
-
-/**
- * sysfs_remove_shadow_dir - remove an object's directory.
- * @shadow: dentry of shadow directory
- *
- * The only thing special about this is that we remove any files in
- * the directory before we remove the directory, and we've inlined
- * what used to be sysfs_rmdir() below, instead of calling separately.
- */
-
-void sysfs_remove_shadow_dir(struct dentry *shadow)
-{
- __sysfs_remove_dir(shadow);
-}
-
 const struct file_operations sysfs_dir_operations = {
  .open  = sysfs_dir_open,
  .release = sysfs_dir_close,
diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
index b20951c..46a277b 100644
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -13,7 +13,6 @@
 #include <linux/dcache.h>
 #include <linux/namei.h>
 #include <linux/err.h>
-#include <linux/fs.h>
 #include <asm/semaphore.h>
 #include "sysfs.h"

diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
```

```
index 4de5c6b..69e6e9b 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -33,16 +33,6 @@ static const struct inode_operations sysfs_inode_operations ={
  .setattr = sysfs_setattr,
 };

-void sysfs_delete_inode(struct inode *inode)
-{
- /* Free the shadowed directory inode operations */
- if (sysfs_is_shadowed_inode(inode)) {
-  kfree(inode->i_op);
-  inode->i_op = NULL;
- }
- return generic_delete_inode(inode);
-}
-
 int sysfs_setattr(struct dentry * dentry, struct iattr * iattr)
 {
  struct inode * inode = dentry->d_inode;
diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 23a48a3..7978cf2 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -23,7 +23,7 @@ static void sysfs_clear_inode(struct inode *inode);

 static const struct super_operations sysfs_ops = {
  .statfs  = simple_statfs,
- .drop_inode = sysfs_delete_inode,
+ .drop_inode = generic_delete_inode,
  .clear_inode = sysfs_clear_inode,
 };

diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index a77c57e..ce6b3ec 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -13,7 +13,6 @@ struct sysfs_dirent {
 extern struct vfsmount * sysfs_mount;
 extern struct kmem_cache *sysfs_dir_cachep;

-extern void sysfs_delete_inode(struct inode *inode);
 extern struct inode * sysfs_new_inode(mode_t mode, struct sysfs_dirent *);
 extern int sysfs_create(struct dentry *, int mode, int (*init)(struct inode *));

@@ -124,7 +123,3 @@ static inline void sysfs_put(struct sysfs_dirent * sd)
  release_sysfs_dirent(sd);
 }
```

```
-static inline int sysfs_is_shadowed_inode(struct inode *inode)
-{
- return S_ISDIR(inode->i_mode) && inode->i_op->follow_link;
-}
diff --git a/include/linux/kobject.h b/include/linux/kobject.h
index d37cd7f..c96d243 100644
--- a/include/linux/kobject.h
+++ b/include/linux/kobject.h
@@ -74,13 +74,9 @@ extern void kobject_init(struct kobject *);
 extern void kobject_cleanup(struct kobject *);

 extern int __must_check kobject_add(struct kobject *);
-extern int __must_check kobject_shadow_add(struct kobject *, struct dentry *);
 extern void kobject_del(struct kobject *);

 extern int __must_check kobject_rename(struct kobject *, const char *new_name);
-extern int __must_check kobject_shadow_rename(struct kobject *kobj,
-      struct dentry *new_parent,
-      const char *new_name);
 extern int __must_check kobject_move(struct kobject *, struct kobject *);

 extern int __must_check kobject_register(struct kobject *);
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index fea9a6b..1f1bb45 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -17,8 +17,6 @@

 struct kobject;
 struct module;
-struct nameidata;
-struct dentry;

 struct attribute {
  const char  * name;
@@ -83,13 +81,13 @@ extern int sysfs_schedule_callback(struct kobject *kobj,
   void (*func)(void *), void *data);

 extern int __must_check
-sysfs_create_dir(struct kobject *, struct dentry *);
+sysfs_create_dir(struct kobject *);

 extern void
 sysfs_remove_dir(struct kobject *);

 extern int __must_check
-sysfs_rename_dir(struct kobject *, struct dentry *, const char *new_name);
```

```
+sysfs_rename_dir(struct kobject *, const char *new_name);

 extern int __must_check
 sysfs_move_dir(struct kobject *, struct kobject *);
@@ -126,12 +124,6 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

 void sysfs_notify(struct kobject * k, char *dir, char *attr);

-
-extern int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *));
-extern struct dentry *sysfs_create_shadow_dir(struct kobject *kobj);
-extern void sysfs_remove_shadow_dir(struct dentry *dir);
-
 extern int __must_check sysfs_init(void);

 #else /* CONFIG_SYSFS */
@@ -142,7 +134,7 @@ static inline int sysfs_schedule_callback(struct kobject *kobj,
  return -ENOSYS;
 }

-static inline int sysfs_create_dir(struct kobject * k, struct dentry *shadow)
+static inline int sysfs_create_dir(struct kobject * k)
 {
  return 0;
 }
@@ -152,9 +144,7 @@ static inline void sysfs_remove_dir(struct kobject * k)
  ;
 }

-static inline int sysfs_rename_dir(struct kobject * k,
-     struct dentry *new_parent,
-     const char *new_name)
+static inline int sysfs_rename_dir(struct kobject * k, const char *new_name)
 {
  return 0;
 }
@@ -229,12 +219,6 @@ static inline void sysfs_notify(struct kobject * k, char *dir, char *attr)
 {
 }

-static inline int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
-{
- return 0;
-}
-
 static inline int __must_check sysfs_init(void)
```

```
 {
  return 0;
diff --git a/lib/kobject.c b/lib/kobject.c
index 09cb276..f590ad8 100644
--- a/lib/kobject.c
+++ b/lib/kobject.c
@@ -44,11 +44,11 @@ static int populate_dir(struct kobject * kobj)
  return error;
 }

-static int create_dir(struct kobject * kobj, struct dentry *shadow_parent)
+static int create_dir(struct kobject * kobj)
 {
  int error = 0;
  if (kobject_name(kobj)) {
-  error = sysfs_create_dir(kobj, shadow_parent);
+  error = sysfs_create_dir(kobj);
    if (!error) {
     if ((error = populate_dir(kobj)))
      sysfs_remove_dir(kobj);
@@ -159,10 +159,9 @@ static void unlink(struct kobject * kobj)
 /**
  * kobject_add - add an object to the hierarchy.
  * @kobj: object.
- * @shadow_parent: sysfs directory to add to.
  */

-int kobject_shadow_add(struct kobject * kobj, struct dentry *shadow_parent)
+int kobject_add(struct kobject * kobj)
 {
  int error = 0;
  struct kobject * parent;
@@ -193,7 +192,7 @@ int kobject_shadow_add(struct kobject * kobj, struct dentry
*shadow_parent)
  }
  kobj->parent = parent;

- error = create_dir(kobj, shadow_parent);
+ error = create_dir(kobj);
  if (error) {
   /* unlink does the kobject_put() for us */
   unlink(kobj);
@@ -214,15 +213,6 @@ int kobject_shadow_add(struct kobject * kobj, struct dentry
*shadow_parent)
  return error;
 }

-/**
```

```
- * kobject_add - add an object to the hierarchy.
- * @kobj: object.
- */
-int kobject_add(struct kobject * kobj)
-{
- return kobject_shadow_add(kobj, NULL);
-}
-

 /**
  * kobject_register - initialize and add an object.
@@ -315,30 +305,7 @@ int kobject_rename(struct kobject * kobj, const char *new_name)
  kobj = kobject_get(kobj);
  if (!kobj)
   return -EINVAL;
- if (!kobj->parent)
-  return -EINVAL;
- error = sysfs_rename_dir(kobj, kobj->parent->dentry, new_name);
- kobject_put(kobj);
-
- return error;
-}
-
-/**
- * kobject_rename - change the name of an object
- * @kobj: object in question.
- * @new_parent: object's new parent
- * @new_name: object's new name
- */
-
-int kobject_shadow_rename(struct kobject * kobj, struct dentry *new_parent,
-     const char *new_name)
-{
- int error = 0;
-
- kobj = kobject_get(kobj);
- if (!kobj)
-  return -EINVAL;
- error = sysfs_rename_dir(kobj, new_parent, new_name);
+ error = sysfs_rename_dir(kobj, new_name);
  kobject_put(kobj);

  return error;
--
1.5.0.g53756
```

_____
Containers mailing list

Subject: [PATCH 3/5] sysfs: Implement sysfs manged shadow directory support.
Posted by ebiederm on Fri, 06 Apr 2007 16:50:32 GMT
View Forum Message <> Reply to Message

The problem. When implementing a network namespace I need to be able
to have multiple network devices with the same name. Currently this
is a problem for /sys/class/net/*, /sys/devices/virtual/net/*, and
potentially a few other directories of the form /sys/ ... /net/*.

What I want is for each network namespace to have it's own separate
set of directories. /sys/class/net/, /sys/devices/virtual/net,
and /sys/ ... /net/, and in each set I want to name them
/sys/class/net/, sys/devices/virtual/net/ and /sys/ ... /net/ respectively.

I looked and the VFS actually allows that. All that is needed is
for /sys/class/net to implement a follow link method to redirect
lookups to the real directory you want.

I am calling the concept of multiple directories all at the same path
in the filesystem shadow directories, the directory entry that
implements the follow_link method the shadow master, and the directories
that are the target of the follow link method shadow directories.

It turns out that just implementing a follow_link method is not
quite enough. The existince of directories of the form /sys/ ... /net/
can depend on the presence or absence of hotplug hardware, which
means I need a simple race free way to create and destroy these
directories.

To achieve a race free design all shadow directories are created
and managed by sysfs itself. The upper level code that knows what
shadow directories we need provides just two methods that enable
this:
 current_tag() - that returns a "void *" tag that identifies the context of
the current process.
 kobject_tag(kobj) - that returns a "void *" tag that identifies the context
a kobject should be in.
Everything else is left up to sysfs.

For the network namespace current_tag and kobject_tag are essentially
one line functions, and look to remain that.

The work needed in sysfs is more extensive. At each directory or
symlink creation I need to check if the shadow directory it belongs

in exists and if it does not create it. Likewise at each symlink
or directory removal I need to check if sysfs directory it is being
removed from is a shadow directory and if this is the last object
in the shadow directory and if so to remove the shadow directory
as well.

I also need a bunch of boiler plate that properly finds, creates, and
removes/frees the shadow directories.

Doing all of that in sysfs isn't bad it is just a bit tedious. Being race
free is just a manner of ensure we have the directory inode mutex when
we add or remove a shadow directory. Trying to do this race free
anywhere besides in sysfs is very nasty, and requires unhealthy
amounts of information about how sysfs is implemented.

Currently only directories which hold kobjects, and
symlinks are supported. There is not enough information
in the current file attribute interfaces to give us anything
to discriminate on which makes it useless, and there are
not potential users which makes it an uniteresting problem
to solve.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/bin.c | 2 +-
 fs/sysfs/dir.c | 370 ++---
 fs/sysfs/file.c | 4 +-
 fs/sysfs/group.c | 12 +-
 fs/sysfs/inode.c | 18 ++-
 fs/sysfs/symlink.c | 11 +-
 fs/sysfs/sysfs.h | 18 +++-
 include/linux/sysfs.h | 14 ++
 8 files changed, 409 insertions(+), 40 deletions(-)

diff --git a/fs/sysfs/bin.c b/fs/sysfs/bin.c
index d3b9f5f..a79ed4c 100644
--- a/fs/sysfs/bin.c
+++ b/fs/sysfs/bin.c
@@ -195,7 +195,7 @@ int sysfs_create_bin_file(struct kobject * kobj, struct bin_attribute * attr)

 void sysfs_remove_bin_file(struct kobject * kobj, struct bin_attribute * attr)
 {
- if (sysfs_hash_and_remove(kobj->dentry, attr->attr.name) < 0) {
+ if (sysfs_hash_and_remove(kobj, kobj->dentry, attr->attr.name) < 0) {
   printk(KERN_ERR "%s: "
    "bad dentry or inode or no such file: \"%s\"\n",
    __FUNCTION__, attr->attr.name);
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c

```
index d6ab015..0802a73 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -123,21 +123,26 @@ static int init_symlink(struct inode * inode)
 static int create_dir(struct kobject * k, struct dentry * p,
         const char * n, struct dentry ** d)
 {
+ struct dentry *parent;
  int error;
  umode_t mode = S_IFDIR| S_IRWXU | S_IRUGO | S_IXUGO;

  mutex_lock(&p->d_inode->i_mutex);
- *d = lookup_one_len(n, p, strlen(n));
+ parent = sysfs_resolve_for_create(k, p);
+ if (IS_ERR(parent))
+  *d = parent;
+ else
+  *d = lookup_one_len(n, parent, strlen(n));
  if (!IS_ERR(*d)) {
-   if (sysfs_dirent_exist(p->d_fsdata, n))
+  if (sysfs_dirent_exist(parent->d_fsdata, n))
      error = -EEXIST;
     else
-   error = sysfs_make_dirent(p->d_fsdata, *d, k, mode,
+   error = sysfs_make_dirent(parent->d_fsdata, *d, k, mode,
         SYSFS_DIR);
   if (!error) {
    error = sysfs_create(*d, mode, init_dir);
    if (!error) {
-    inc_nlink(p->d_inode);
+    inc_nlink(parent->d_inode);
     (*d)->d_op = &sysfs_dentry_ops;
     d_rehash(*d);
    }
@@ -288,6 +293,7 @@ static void remove_dir(struct dentry * d)
    atomic_read(&d->d_count));

  mutex_unlock(&parent->d_inode->i_mutex);
+ sysfs_prune_shadow(parent);
  dput(parent);
 }

@@ -296,6 +302,55 @@ void sysfs_remove_subdir(struct dentry * d)
  remove_dir(d);
 }

+static void sysfs_empty_dir(struct dentry *dentry)
+{
```

```
+ struct sysfs_dirent *parent_sd, *sd, *tmp;
+
+ parent_sd = dentry->d_fsdata;
+ list_for_each_entry_safe(sd, tmp, &parent_sd->s_children, s_sibling) {
+  if (!sd->s_element || !(sd->s_type & SYSFS_NOT_PINNED))
+   continue;
+  list_del_init(&sd->s_sibling);
+  sysfs_drop_dentry(sd, dentry);
+  sysfs_put(sd);
+ }
+}
+
+static void __sysfs_remove_empty_shadow(struct dentry *shadow)
+{
+ struct sysfs_dirent *sd;
+
+ if (d_unhashed(shadow))
+  return;
+
+ sd = shadow->d_fsdata;
+ d_delete(shadow);
+ list_del_init(&sd->s_sibling);
+ simple_rmdir(shadow->d_inode, shadow);
+ sysfs_put(sd);
+}
+
+static void sysfs_remove_empty_shadow(struct dentry *shadow)
+{
+ mutex_lock(&shadow->d_inode->i_mutex);
+ __sysfs_remove_empty_shadow(shadow);
+ mutex_unlock(&shadow->d_inode->i_mutex);
+}
+
+static void sysfs_remove_shadows(struct dentry *dentry)
+{
+ struct sysfs_dirent *parent_sd, *sd, *tmp;
+
+ parent_sd = dentry->d_fsdata;
+ list_for_each_entry_safe(sd, tmp, &parent_sd->s_children, s_sibling) {
+  struct dentry *shadow;
+
+  shadow = dget(sd->s_dentry);
+  sysfs_empty_dir(shadow);
+  __sysfs_remove_empty_shadow(shadow);
+  dput(shadow);
+ }
+}
```

```
 /**
  * sysfs_remove_dir - remove an object's directory.
@@ -309,22 +364,16 @@ void sysfs_remove_subdir(struct dentry * d)
 void sysfs_remove_dir(struct kobject * kobj)
 {
  struct dentry * dentry = dget(kobj->dentry);
- struct sysfs_dirent * parent_sd;
- struct sysfs_dirent * sd, * tmp;

  if (!dentry)
   return;

  pr_debug("sysfs %s: removing dir\n",dentry->d_name.name);
  mutex_lock(&dentry->d_inode->i_mutex);
- parent_sd = dentry->d_fsdata;
- list_for_each_entry_safe(sd, tmp, &parent_sd->s_children, s_sibling) {
-  if (!sd->s_element || !(sd->s_type & SYSFS_NOT_PINNED))
-   continue;
-  list_del_init(&sd->s_sibling);
-  sysfs_drop_dentry(sd, dentry);
-  sysfs_put(sd);
- }
+ if (dentry->d_inode->i_op == &sysfs_dir_inode_operations)
+  sysfs_empty_dir(dentry);
+ else
+  sysfs_remove_shadows(dentry);
  mutex_unlock(&dentry->d_inode->i_mutex);

  remove_dir(dentry);
@@ -350,7 +399,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  down_write(&sysfs_rename_sem);
  mutex_lock(&inode->i_mutex);

- new_parent = kobj->dentry->d_parent;
+ new_parent = sysfs_resolve_for_create(kobj, kobj->dentry->d_parent);
  new_dentry = lookup_one_len(new_name, new_parent, strlen(new_name));
  if (!IS_ERR(new_dentry)) {
   if (new_dentry == kobj->dentry)
@@ -378,6 +427,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  mutex_unlock(&inode->i_mutex);
  up_write(&sysfs_rename_sem);

+ sysfs_prune_shadow(old_parent);
  dput(old_parent);

  return error;
@@ -385,24 +435,35 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
```

```
 int sysfs_move_dir(struct kobject *kobj, struct kobject *new_parent)
 {
+ struct inode *old_parent_inode, *new_parent_inode;
  struct dentry *old_parent_dentry, *new_parent_dentry, *new_dentry;
  struct sysfs_dirent *new_parent_sd, *sd;
  int error;

- old_parent_dentry = kobj->parent ?
-  kobj->parent->dentry : sysfs_mount->mnt_sb->s_root;
+ old_parent_dentry = kobj->dentry->d_parent;
  new_parent_dentry = new_parent ?
   new_parent->dentry : sysfs_mount->mnt_sb->s_root;

- if (old_parent_dentry->d_inode == new_parent_dentry->d_inode)
+ old_parent_inode = old_parent_dentry->d_inode;
+ new_parent_inode = new_parent_dentry->d_inode;
+
+ if (old_parent_inode == new_parent_inode)
   return 0; /* nothing to move */
+
+ dget(old_parent_dentry);
 again:
- mutex_lock(&old_parent_dentry->d_inode->i_mutex);
- if (!mutex_trylock(&new_parent_dentry->d_inode->i_mutex)) {
-  mutex_unlock(&old_parent_dentry->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;
  }

+ new_parent_dentry = sysfs_resolve_for_create(kobj, new_parent_dentry);
+ if (IS_ERR(new_parent_dentry)) {
+  error = PTR_ERR(new_parent_dentry);
+  goto out;
+ }
+
  new_parent_sd = new_parent_dentry->d_fsdata;
  sd = kobj->dentry->d_fsdata;

@@ -422,8 +483,11 @@ again:
  list_add(&sd->s_sibling, &new_parent_sd->s_children);

 out:
- mutex_unlock(&new_parent_dentry->d_inode->i_mutex);
- mutex_unlock(&old_parent_dentry->d_inode->i_mutex);
+ mutex_unlock(&new_parent_inode->i_mutex);
+ mutex_unlock(&old_parent_inode->i_mutex);
```

```
+
+ sysfs_prune_shadow(old_parent_dentry);
+ dput(old_parent_dentry);

  return error;
 }
@@ -486,6 +550,11 @@ static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
    i++;
    /* fallthrough */
   default:
+   /* If I am the shadow master return nothing. */
+   if ((parent_sd->s_type == SYSFS_DIR) &&
+       (dentry->d_inode->i_op != &sysfs_dir_inode_operations))
+    return 0;
+
    if (filp->f_pos == 2)
     list_move(q, &parent_sd->s_children);

@@ -565,3 +634,258 @@ const struct file_operations sysfs_dir_operations = {
  .read  = generic_read_dir,
  .readdir = sysfs_readdir,
 };
+
+static struct dentry *find_shadow_dir(struct dentry *parent, void *target)
+{
+ /* Find the shadow directory for the specified tag */
+ struct sysfs_dirent *parent_sd, *sd;
+ struct dentry *result;
+
+ result = NULL;
+ parent_sd = parent->d_fsdata;
+ list_for_each_entry(sd, &parent_sd->s_children, s_sibling) {
+  struct sysfs_shadow_dir *sdd;
+  sdd = sd->s_element;
+  if (sdd->tag != target)
+   continue;
+  result = sd->s_dentry;
+  break;
+ }
+ return result;
+}
+
+static void *find_shadow_tag(struct kobject *kobj)
+{
+ /* Find the tag the current kobj is cached with */
+ struct sysfs_shadow_dir *sdd;
+ struct dentry *dir;
+ struct sysfs_dirent *parent_sd, *sd;
```

```
+ void *result;
+
+ result = ERR_PTR(-ENOENT);
+ dir = kobj->dentry;
+ sd = dir->d_parent->d_fsdata;
+ if (sd->s_type != SYSFS_SHADOW)
+  goto out;
+ sdd = sd->s_element;
+ parent_sd = sdd->kobj->dentry->d_fsdata;
+
+ list_for_each_entry(sd, &parent_sd->s_children, s_sibling) {
+  struct sysfs_shadow_dir *sdd;
+  struct dentry *dentry;
+  int found;
+  sdd = sd->s_element;
+
+  dentry = lookup_one_len(dir->d_name.name, sd->s_dentry,
+      dir->d_name.len);
+  found = (dentry == dir);
+  dput(dentry);
+
+  if (!found)
+   continue;
+
+  result = sdd->tag;
+  break;
+ }
+out:
+ return result;
+}
+
+static struct dentry *add_shadow_dir(struct dentry *dir, void *tag)
+{
+ struct sysfs_dirent *parent_sd;
+ struct sysfs_shadow_dir *sdd;
+ struct dentry *shadow;
+ struct inode *inode;
+ int err;
+
+ parent_sd = dir->d_fsdata;
+ inode = dir->d_inode;
+
+ err = -ENOMEM;
+ shadow = d_alloc(dir->d_parent, &dir->d_name);
+ if (!shadow)
+  goto error;
+
+ sdd = kmalloc(sizeof(*sdd), GFP_KERNEL);
```

```
+ if (!sdd)
+  goto error;
+ sdd->tag = tag;
+ sdd->kobj = parent_sd->s_element;
+
+ err = sysfs_make_dirent(parent_sd, shadow, sdd, inode->i_mode,
+    SYSFS_SHADOW);
+ if (err)
+  goto error;
+
+ d_instantiate(shadow, igrab(inode));
+
+ /* Since the shadow directory is reachable make it look
+  * like it is actually hashed.
+  */
+ shadow->d_hash.pprev = &shadow->d_hash.next;
+ shadow->d_hash.next = NULL;
+ shadow->d_flags &= ~DCACHE_UNHASHED;
+
+ inc_nlink(inode);
+ inc_nlink(dir->d_parent->d_inode);
+ shadow->d_op = &sysfs_dentry_ops;
+
+out:
+ return shadow;
+error:
+ dput(shadow);
+ shadow = ERR_PTR(err);
+ goto out;
+}
+
+void sysfs_prune_shadow(struct dentry *shadow)
+{
+ struct sysfs_dirent *sd;
+
+ if (!shadow)
+  return;
+ sd = shadow->d_fsdata;
+ if (sd->s_type != SYSFS_SHADOW)
+  return;
+
+ if (!list_empty(&sd->s_children))
+  return;
+
+ sysfs_remove_empty_shadow(shadow);
+}
+
+struct dentry *sysfs_resolve_for_create(struct kobject *kobj, struct dentry *dentry)
```

```
+{
+ const struct shadow_dir_operations *shadow_ops;
+ struct dentry *dir;
+ struct inode *inode;
+
+ inode = dentry->d_inode;
+ shadow_ops = inode->i_private;
+ if (!shadow_ops)
+  dir = dentry;
+ else {
+  struct sysfs_dirent *sd;
+  void *tag;
+  sd = dentry->d_fsdata;
+  if (sd->s_type == SYSFS_SHADOW) {
+   struct sysfs_shadow_dir *sdd = sd->s_element;
+   dentry = sdd->kobj->dentry;
+  }
+  tag = shadow_ops->kobject_tag(kobj);
+  dir = find_shadow_dir(dentry, tag);
+  if (!dir)
+   dir = add_shadow_dir(dentry, tag);
+ }
+ return dir;
+}
+
+struct dentry *sysfs_resolve_for_remove(struct kobject *kobj, struct dentry *dentry)
+{
+ const struct shadow_dir_operations *shadow_ops;
+ struct dentry *dir;
+ struct inode *inode;
+
+ inode = dentry->d_inode;
+ shadow_ops = inode->i_private;
+ if (!shadow_ops)
+  dir = dentry;
+ else {
+  struct sysfs_dirent *sd;
+  void *tag;
+  sd = dentry->d_fsdata;
+  if (sd->s_type == SYSFS_SHADOW) {
+   struct sysfs_shadow_dir *sdd = sd->s_element;
+   dentry = sdd->kobj->dentry;
+  }
+  tag = find_shadow_tag(kobj);
+  if (IS_ERR(tag))
+   dir = tag;
+  else
+   dir = find_shadow_dir(dentry, tag);
```

```
+  if (!dir)
+   dir = ERR_PTR(-ENOENT);
+ }
+ return dir;
+}
+
+static void *sysfs_shadow_follow_link(struct dentry *dentry, struct nameidata *nd)
+{
+ struct sysfs_dirent *sd;
+ struct dentry *dest;
+
+ sd = dentry->d_fsdata;
+ dest = NULL;
+ if (sd->s_type == SYSFS_DIR) {
+  const struct shadow_dir_operations *shadow_ops;
+  struct inode *inode;
+  inode = dentry->d_inode;
+  shadow_ops = inode->i_private;
+  mutex_lock(&inode->i_mutex);
+  dest = find_shadow_dir(dentry, shadow_ops->current_tag());
+  dget(dest);
+  mutex_unlock(&inode->i_mutex);
+ }
+ if (!dest)
+  dest = dget(dentry);
+ dput(nd->dentry);
+ nd->dentry = dest;
+
+ return NULL;
+}
+
+static struct inode_operations sysfs_shadow_inode_operations = {
+ .lookup  = sysfs_lookup,
+ .setattr = sysfs_setattr,
+ .follow_link = sysfs_shadow_follow_link,
+};
+
+
+/**
+ * sysfs_enable_shadowing - Automatically create shadows of a directory
+ * @kobj: object to automatically shadow
+ *
+ * Once shadowing has been enabled on a directory the contents
+ * of the directory become dependent upon context.
+ *
+ * shadow_ops->current_tag() returns the context for the current
+ * process.
+ *
```

```
+ * shadow_ops->kobject_tag() returns the context that a given kobj
+ * resides in.
+ *
+ * Using those methods the sysfs code on shadowed directories
+ * carefully stores the files so that when we lookup files
+ * we get the proper answer for our context.
+ *
+ * If the context of a kobject is changed it is expected that
+ * the kobject will be renamed so the appopriate sysfs data structures
+ * can be updated.
+ */
+int sysfs_enable_shadowing(struct kobject *kobj,
+ const struct shadow_dir_operations *shadow_ops)
+{
+ struct sysfs_dirent *sd;
+ struct dentry *dentry;
+ struct inode *inode;
+ int err;
+
+ dentry = kobj->dentry;
+ inode = dentry->d_inode;
+
+ mutex_lock(&inode->i_mutex);
+ /* We can only enable shadowing on empty directories
+  * where shadowing is not already enabled.
+  */
+ sd = dentry->d_fsdata;
+ err = -EINVAL;
+ if (!inode->i_private && list_empty(&sd->s_children)) {
+  inode->i_private = (void *)shadow_ops;
+  inode->i_op = &sysfs_shadow_inode_operations;
+  err = 0;
+ }
+ mutex_unlock(&inode->i_mutex);
+ return err;
+}
+
diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index fc46333..31b5429 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -606,7 +606,7 @@ EXPORT_SYMBOL_GPL(sysfs_chmod_file);

 void sysfs_remove_file(struct kobject * kobj, const struct attribute * attr)
 {
- sysfs_hash_and_remove(kobj->dentry, attr->name);
+ sysfs_hash_and_remove(kobj, kobj->dentry, attr->name);
 }
```

```
@@ -623,7 +623,7 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

  dir = lookup_one_len(group, kobj->dentry, strlen(group));
  if (!IS_ERR(dir)) {
-  sysfs_hash_and_remove(dir, attr->name);
+  sysfs_hash_and_remove(kobj, dir, attr->name);
   dput(dir);
  }
 }
diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
index 46a277b..61115b9 100644
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -17,16 +17,16 @@
 #include "sysfs.h"


-static void remove_files(struct dentry * dir,
+static void remove_files(struct kobject *kobj, struct dentry * dir,
     const struct attribute_group * grp)
 {
  struct attribute *const* attr;

  for (attr = grp->attrs; *attr; attr++)
-  sysfs_hash_and_remove(dir,(*attr)->name);
+  sysfs_hash_and_remove(kobj, dir,(*attr)->name);
 }

-static int create_files(struct dentry * dir,
+static int create_files(struct kobject *kobj, struct dentry * dir,
    const struct attribute_group * grp)
 {
  struct attribute *const* attr;
@@ -36,7 +36,7 @@ static int create_files(struct dentry * dir,
   error = sysfs_add_file(dir, *attr, SYSFS_KOBJ_ATTR);
  }
  if (error)
-  remove_files(dir,grp);
+  remove_files(kobj, dir, grp);
  return error;
 }

@@ -56,7 +56,7 @@ int sysfs_create_group(struct kobject * kobj,
  } else
   dir = kobj->dentry;
  dir = dget(dir);
```

```
- if ((error = create_files(dir,grp))) {
+ if ((error = create_files(kobj, dir, grp))) {
   if (grp->name)
    sysfs_remove_subdir(dir);
  }
@@ -75,7 +75,7 @@ void sysfs_remove_group(struct kobject * kobj,
  else
   dir = dget(kobj->dentry);

- remove_files(dir,grp);
+ remove_files(kobj, dir, grp);
  if (grp->name)
   sysfs_remove_subdir(dir);
  /* release the ref. taken in this routine */
diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index 69e6e9b..6b0e536 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -191,6 +191,7 @@ const unsigned char * sysfs_get_name(struct sysfs_dirent *sd)
  BUG_ON(!sd || !sd->s_element);

  switch (sd->s_type) {
+  case SYSFS_SHADOW:
   case SYSFS_DIR:
    /* Always have a dentry so use that */
    return sd->s_dentry->d_name.name;
@@ -259,8 +260,9 @@ void sysfs_drop_dentry(struct sysfs_dirent * sd, struct dentry * parent)
  }
 }

-int sysfs_hash_and_remove(struct dentry * dir, const char * name)
+int sysfs_hash_and_remove(struct kobject *kobj, struct dentry * dir, const char * name)
 {
+ struct inode * inode;
  struct sysfs_dirent * sd;
  struct sysfs_dirent * parent_sd;
  int found = 0;
@@ -272,8 +274,13 @@ int sysfs_hash_and_remove(struct dentry * dir, const char * name)
   /* no inode means this hasn't been made visible yet */
   return -ENOENT;

+ inode = dir->d_inode;
+ mutex_lock_nested(&inode->i_mutex, I_MUTEX_PARENT);
+ dir = sysfs_resolve_for_remove(kobj, dir);
+ if (IS_ERR(dir))
+  goto out_unlock;
+ dget(dir);
  parent_sd = dir->d_fsdata;
```

```
- mutex_lock_nested(&dir->d_inode->i_mutex, I_MUTEX_PARENT);
  list_for_each_entry(sd, &parent_sd->s_children, s_sibling) {
   if (!sd->s_element)
    continue;
@@ -285,7 +292,12 @@ int sysfs_hash_and_remove(struct dentry * dir, const char * name)
    break;
   }
  }
- mutex_unlock(&dir->d_inode->i_mutex);
+out_unlock:
+ mutex_unlock(&inode->i_mutex);
+ if (!IS_ERR(dir) && dir) {
+  sysfs_prune_shadow(dir);
+  dput(dir);
+ }

  return found ? 0 : -ENOENT;
 }
diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index 7b9c5bf..2885ebb 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -84,7 +84,7 @@ exit1:
  */
 int sysfs_create_link(struct kobject * kobj, struct kobject * target, const char * name)
 {
- struct dentry *dentry = NULL;
+ struct dentry *dir, *dentry = NULL;
  int error = -EEXIST;

  BUG_ON(!name);
@@ -99,8 +99,11 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char
   return -EFAULT;

  mutex_lock(&dentry->d_inode->i_mutex);
- if (!sysfs_dirent_exist(dentry->d_fsdata, name))
-  error = sysfs_add_link(dentry, name, target);
+ dir = sysfs_resolve_for_create(target, dentry);
+ if (IS_ERR(dir))
+  error = PTR_ERR(dir);
+ else if (!sysfs_dirent_exist(dir->d_fsdata, name))
+  error = sysfs_add_link(dir, name, target);
  mutex_unlock(&dentry->d_inode->i_mutex);
  return error;
 }
@@ -114,7 +117,7 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char
```

```
 void sysfs_remove_link(struct kobject * kobj, const char * name)
 {
- sysfs_hash_and_remove(kobj->dentry,name);
+ sysfs_hash_and_remove(kobj, kobj->dentry,name);
 }

 static int sysfs_get_target_path(struct kobject * kobj, struct kobject * target,
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index ce6b3ec..d761b28 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -13,6 +13,10 @@ struct sysfs_dirent {
 extern struct vfsmount * sysfs_mount;
 extern struct kmem_cache *sysfs_dir_cachep;

+extern void sysfs_prune_shadow(struct dentry *shadow);
+struct dentry *sysfs_resolve_for_create(struct kobject *, struct dentry *);
+struct dentry *sysfs_resolve_for_remove(struct kobject *, struct dentry *);
+
 extern struct inode * sysfs_new_inode(mode_t mode, struct sysfs_dirent *);
 extern int sysfs_create(struct dentry *, int mode, int (*init)(struct inode *));

@@ -21,7 +25,7 @@ extern int sysfs_make_dirent(struct sysfs_dirent *, struct dentry *, void *,
     umode_t, int);

 extern int sysfs_add_file(struct dentry *, const struct attribute *, int);
-extern int sysfs_hash_and_remove(struct dentry * dir, const char * name);
+extern int sysfs_hash_and_remove(struct kobject *, struct dentry *, const char *);
 extern struct sysfs_dirent *sysfs_find(struct sysfs_dirent *dir, const char * name);

 extern int sysfs_create_subdir(struct kobject *, const char *, struct dentry **);
@@ -44,6 +48,11 @@ struct sysfs_symlink {
  struct kobject * target_kobj;
 };

+struct sysfs_shadow_dir {
+ void *tag;
+ struct kobject *kobj;
+};
+
 struct sysfs_buffer {
  struct list_head  associates;
  size_t    count;
@@ -88,6 +97,9 @@ static inline struct kobject *sysfs_get_kobject(struct dentry *dentry)
   if (sd->s_type & SYSFS_KOBJ_LINK) {
    struct sysfs_symlink * sl = sd->s_element;
    kobj = kobject_get(sl->target_kobj);
```

```
+  } else if (sd->s_type & SYSFS_SHADOW) {
+   struct sysfs_shadow_dir * sdd = sd->s_element;
+   kobj = kobject_get(sdd->kobj);
   } else
    kobj = kobject_get(sd->s_element);
  }
@@ -104,6 +116,10 @@ static inline void release_sysfs_dirent(struct sysfs_dirent * sd)
   kobject_put(sl->target_kobj);
   kfree(sl);
  }
+ else if (sd->s_type & SYSFS_SHADOW) {
+  struct sysfs_shadow_dir * sdd = sd->s_element;
+  kfree(sdd);
+ }
  kfree(sd->s_iattr);
  kmem_cache_free(sysfs_dir_cachep, sd);
 }
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index 1f1bb45..cda7b81 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -68,11 +68,17 @@ struct sysfs_ops {
  ssize_t (*store)(struct kobject *,struct attribute *,const char *, size_t);
 };

+struct shadow_dir_operations {
+ void *(*current_tag)(void);
+ void *(*kobject_tag)(struct kobject *kobj);
+};
+
 #define SYSFS_ROOT  0x0001
 #define SYSFS_DIR  0x0002
 #define SYSFS_KOBJ_ATTR  0x0004
 #define SYSFS_KOBJ_BIN_ATTR 0x0008
 #define SYSFS_KOBJ_LINK  0x0020
+#define SYSFS_SHADOW  0x0040
 #define SYSFS_NOT_PINNED (SYSFS_KOBJ_ATTR | SYSFS_KOBJ_BIN_ATTR |
SYSFS_KOBJ_LINK)

 #ifdef CONFIG_SYSFS
@@ -124,6 +130,8 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

 void sysfs_notify(struct kobject * k, char *dir, char *attr);

+int sysfs_enable_shadowing(struct kobject *, const struct shadow_dir_operations *);
+
 extern int __must_check sysfs_init(void);
```

```
 #else /* CONFIG_SYSFS */
@@ -219,6 +227,12 @@ static inline void sysfs_notify(struct kobject * k, char *dir, char *attr)
 {
 }

+static inline int sysfs_enable_shadowing(struct kobject *kobj,
+    const struct shadow_dir_operations *shadow_ops)
+{
+ return 0;
+}
+
 static inline int __must_check sysfs_init(void)
 {
  return 0;
--
1.5.0.g53756
```

Subject: [PATCH 4/5] sysfs: Implement sysfs_delete_link and sysfs_rename_link
Posted by ebiederm on Fri, 06 Apr 2007 16:51:57 GMT
View Forum Message <> Reply to Message

When removing a symlink sysfs_remove_link does not provide enough
information to figure out which shadow directory the symlink falls in.
So I need sysfs_delete_link which is passed the target of the symlink
to delete.

Further half the time when we are removing a symlink the code is
actually renaming the symlink but not doing so explicitly because we
don't have a symlink rename method. So I have added sysfs_rename_link
as well.

Both of these functions now have enough information to find a symlink
in a shadow directory. The only restriction is that they must be
called before the target kobject is renamed or deleted. If they are
called later I loose track of which tag the target kobject was marked
with and can no longer find the old symlink to remove it.

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

```
---
 fs/sysfs/symlink.c    |   31 ++++++++++++++++++++++++++++++++
 include/linux/sysfs.h |   18 ++++++++++++++++++
 2 files changed, 49 insertions(+), 0 deletions(-)
```

```
diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index 2885ebb..faff190 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -110,6 +110,21 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char


 /**
+ * sysfs_delete_link - remove symlink in object's directory.
+ * @kobj: object we're acting for.
+ * @targ: object we're pointing to.
+ * @name: name of the symlink to remove.
+ *
+ * Unlike sysfs_remove_link sysfs_delete_link has enough information
+ * to successfully delete symlinks in shadow directories.
+ */
+void sysfs_delete_link(struct kobject *kobj, struct kobject *targ,
+   const char *name)
+{
+ sysfs_hash_and_remove(targ, kobj->dentry,name);
+}
+
+/**
  * sysfs_remove_link - remove symlink in object's directory.
  * @kobj: object we're acting for.
  * @name: name of the symlink to remove.
@@ -120,6 +135,22 @@ void sysfs_remove_link(struct kobject * kobj, const char * name)
  sysfs_hash_and_remove(kobj, kobj->dentry,name);
 }

+/**
+ * sysfs_rename_link - rename symlink in object's directory.
+ * @kobj: object we're acting for.
+ * @targ: object we're pointing to.
+ * @old: previous name of the symlink.
+ * @new: new name of the symlink.
+ *
+ * A helper function for the common rename symlink idiom.
+ */
+int sysfs_rename_link(struct kobject *kobj, struct kobject *targ,
+   const char *old, const char *new)
+{
+ sysfs_delete_link(kobj, targ, old);
+ return sysfs_create_link(kobj, targ, new);
+}
+
```

```
 static int sysfs_get_target_path(struct kobject * kobj, struct kobject * target,
      char *path)
 {
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index cda7b81..c29b38d 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -116,6 +116,13 @@ sysfs_create_link(struct kobject * kobj, struct kobject * target, const char
* n
 extern void
 sysfs_remove_link(struct kobject *, const char * name);

+extern int
+sysfs_rename_link(struct kobject *kobj, struct kobject *target,
+   const char *old_name, const char *new_name);
+
+extern void
+sysfs_delete_link(struct kobject *dir, struct kobject *targ, const char *name);
+
 int __must_check sysfs_create_bin_file(struct kobject *kobj,
      struct bin_attribute *attr);
 void sysfs_remove_bin_file(struct kobject *kobj, struct bin_attribute *attr);
@@ -191,6 +198,17 @@ static inline void sysfs_remove_link(struct kobject * k, const char *
name)
  ;
 }

+static inline int
+sysfs_rename_link(struct kobject * k, struct kobject *t,
+   const char *old_name, const char * new_name)
+{
+ return 0;
+}
+
+static inline void
+sysfs_delete_link(struct kobject *k, struct kobject *t, const char *name)
+{
+}

 static inline int sysfs_create_bin_file(struct kobject * k, struct bin_attribute * a)
 {
--
1.5.0.g53756
```

Subject: [PATCH 5/5] driver core: Implement shadow directory support for device classes.
Posted by ebiederm on Fri, 06 Apr 2007 17:14:52 GMT
View Forum Message <> Reply to Message

This patch enables shadowing on every class directory if struct class
has shadow_ops.

In addition device_del and device_rename were modified to use
sysfs_delete_link and sysfs_rename_link respectively to ensure when
these operations happen in the presence of shadow directories the
work properly.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/base/class.c | 30 ++++++++++++++++++++++++++----
 drivers/base/core.c | 20 ++++++++++++---------
 include/linux/device.h | 2 ++
 3 files changed, 39 insertions(+), 13 deletions(-)

diff --git a/drivers/base/class.c b/drivers/base/class.c
index 80bbb20..1ac2b0e 100644
--- a/drivers/base/class.c
+++ b/drivers/base/class.c
@@ -136,6 +136,17 @@ static void remove_class_attrs(struct class * cls)
  }
 }

+static int class_setup_shadowing(struct class *cls)
+{
+ const struct shadow_dir_operations *shadow_ops;
+
+ shadow_ops = cls->shadow_ops;
+ if (!shadow_ops)
+ return 0;
+
+ return sysfs_enable_shadowing(&cls->subsys.kset.kobj, shadow_ops);
+}
+
 int class_register(struct class * cls)
 {
  int error;
@@ -154,11 +165,22 @@ int class_register(struct class * cls)
  subsys_set_kset(cls, class_subsys);

  error = subsystem_register(&cls->subsys);
- if (!error) {
- error = add_class_attrs(class_get(cls));
- class_put(cls);

```
- }
+ if (error)
+  goto out;
+
+ error = class_setup_shadowing(cls);
+ if (error)
+  goto out_unregister;
+
+ error = add_class_attrs(cls);
+ if (error)
+  goto out_unregister;
+
+out:
  return error;
+out_unregister:
+ subsystem_unregister(&cls->subsys);
+ goto out;
 }

 void class_unregister(struct class * cls)
diff --git a/drivers/base/core.c b/drivers/base/core.c
index 8650401..3d9c5ff 100644
--- a/drivers/base/core.c
+++ b/drivers/base/core.c
@@ -519,8 +519,14 @@ static struct kobject * get_device_parent(struct device *dev,
    return kobj;

   /* or create a new class-directory at the parent device */
-  return kobject_kset_add_dir(&dev->class->class_dirs,
+  kobj = kobject_kset_add_dir(&dev->class->class_dirs,
         parent_kobj, dev->class->name);
+
+  /* If we created a new class-directory setup shadowing */
+  if (kobj && dev->class->shadow_ops)
+   sysfs_enable_shadowing(kobj, dev->class->shadow_ops);
+
+  return kobj;
  }

  if (parent)
@@ -797,8 +803,8 @@ void device_del(struct device * dev)
   /* If this is not a "fake" compatible device, remove the
    * symlink from the class to the device. */
   if (dev->kobj.parent != &dev->class->subsys.kset.kobj)
-   sysfs_remove_link(&dev->class->subsys.kset.kobj,
-       dev->bus_id);
+   sysfs_delete_link(&dev->class->subsys.kset.kobj,
+       &dev->kobj, dev->bus_id);
```

```
   if (parent) {
 #ifdef CONFIG_SYSFS_DEPRECATED
    char *class_name = make_class_name(dev->class->name,
@@ -1096,6 +1102,8 @@ int device_rename(struct device *dev, char *new_name)
    goto out_free_old_class;
   }
   strlcpy(old_symlink_name, dev->bus_id, BUS_ID_SIZE);
+  sysfs_rename_link(&dev->class->subsys.kset.kobj, &dev->kobj,
+      old_symlink_name, new_name);
  }

  strlcpy(dev->bus_id, new_name, BUS_ID_SIZE);
@@ -1113,12 +1121,6 @@ int device_rename(struct device *dev, char *new_name)
  }
 #endif

- if (dev->class) {
-  sysfs_remove_link(&dev->class->subsys.kset.kobj,
-      old_symlink_name);
-  sysfs_create_link(&dev->class->subsys.kset.kobj, &dev->kobj,
-      dev->bus_id);
- }
  put_device(dev);

  kfree(new_class_name);
diff --git a/include/linux/device.h b/include/linux/device.h
index de0e73e..8326067 100644
--- a/include/linux/device.h
+++ b/include/linux/device.h
@@ -199,6 +199,8 @@ struct class {

  int (*suspend)(struct device *, pm_message_t state);
  int (*resume)(struct device *);
+
+ const struct shadow_dir_operations *shadow_ops;
 };

 extern int __must_check class_register(struct class *);
--
1.5.0.g53756
```

Subject: Re: [PATCH 2/5] sysfs: Remove first pass at shadow directory support
Posted by Greg KH on Thu, 12 Apr 2007 22:58:43 GMT
View Forum Message <> Reply to Message

On Fri, Apr 06, 2007 at 10:48:42AM -0600, Eric W. Biederman wrote:
>
> While shadow directories appear to be a good idea, the current scheme
> of controlling their creation and destruction outside of sysfs appears
> to be a locking and maintenance nightmare in the face of sysfs directories
> dynamically coming and going. Which can now occur for directories containing
> network devices when CONFIG_SYSFS_DEPRECATED is not set.
>
> This patch removes everything from the initial shadow directory support
> that allowed the shadow directory creation to be controlled at a higher
> level. So except for a few bits of sysfs_rename_dir everything from
> commit b592fcfe7f06c15ec11774b5be7ce0de3aa86e73 is now gone.

Can you rebase patches 2-5 on the latest -mm? Tejun redid the whole
sysfs internals which pretty much means that this patch series doesn't
apply anymore :(

thanks,

greg k-h

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

Subject: patch kobject-comment-and-warning-fixes-to-kobject.c.patch added to gregkh-2.6 tree
Posted by gregkh on Thu, 12 Apr 2007 23:00:06 GMT
View Forum Message <> Reply to Message

This is a note to let you know that I've just added the patch titled

    Subject: kobject: Comment and warning fixes to kobject.c

to my gregkh-2.6 tree. Its filename is

    kobject-comment-and-warning-fixes-to-kobject.c.patch

This tree can be found at
   http://www.kernel.org/pub/linux/kernel/people/gregkh/gregkh-2.6/patches/

>From ebiederm@xmission.com Fri Apr 6 09:47:44 2007

From: ebiederm@xmission.com (Eric W. Biederman)
Date: Fri, 06 Apr 2007 10:47:11 -0600
Subject: kobject: Comment and warning fixes to kobject.c
To: Greg Kroah-Hartman <gregkh@suse.de>
Cc: Kay Sievers <kay.sievers@vrfy.org>, Linux Containers <containers@lists.osdl.org>
Message-ID: <m11wix8n74.fsf@ebiederm.dsl.xmission.com>

This dots some i's and crosses some t's after left over from when
kobject_kset_add_dir was built from kobject_add_dir.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Cc: Kay Sievers <kay.sievers@vrfy.org>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>
---
 lib/kobject.c | 13 ++++++++++---
 1 file changed, 10 insertions(+), 3 deletions(-)

--- a/lib/kobject.c
+++ b/lib/kobject.c
@@ -569,7 +569,7 @@ static struct kobj_type dir_ktype = {
 };

 /**
- * kobject__kset_add_dir - add sub directory of object.
+ * kobject_kset_add_dir - add sub directory of object.
  * @kset: kset the directory is belongs to.
  * @parent: object in which a directory is created.
  * @name: directory name.
@@ -595,8 +595,8 @@ struct kobject *kobject_kset_add_dir(str
  kobject_set_name(k, name);
  ret = kobject_register(k);
  if (ret < 0) {
- printk(KERN_WARNING "kobject_add_dir: "
-   "kobject_register error: %d\n", ret);
+ printk(KERN_WARNING "%s: kobject_register error: %d\n",
+   __func__, ret);
   kobject_del(k);
   return NULL;
  }
@@ -604,6 +604,13 @@ struct kobject *kobject_kset_add_dir(str
  return k;
 }

+/**
+ * kobject_add_dir - add sub directory of object.
+ * @parent: object in which a directory is created.
+ * @name: directory name.

```
+ *
+ * Add a plain directory object as child of given object.
+ */
 struct kobject *kobject_add_dir(struct kobject *parent, const char *name)
 {
  return kobject_kset_add_dir(NULL, parent, name);
```

Patches currently in gregkh-2.6 which might be from gregkh@suse.de are

---

Subject: Re: [PATCH 2/5] sysfs: Remove first pass at shadow directory support
Posted by ebiederm on Fri, 13 Apr 2007 17:23:24 GMT
View Forum Message <> Reply to Message

Greg KH <greg@kroah.com> writes:

> On Fri, Apr 06, 2007 at 10:48:42AM -0600, Eric W. Biederman wrote:
>>
>> While shadow directories appear to be a good idea, the current scheme
>> of controlling their creation and destruction outside of sysfs appears
>> to be a locking and maintenance nightmare in the face of sysfs directories
>> dynamically coming and going. Which can now occur for directories containing
>> network devices when CONFIG_SYSFS_DEPRECATED is not set.
>>
>> This patch removes everything from the initial shadow directory support
>> that allowed the shadow directory creation to be controlled at a higher
>> level. So except for a few bits of sysfs_rename_dir everything from
>> commit b592fcfe7f06c15ec11774b5be7ce0de3aa86e73 is now gone.
>
> Can you rebase patches 2-5 on the latest -mm? Tejun redid the whole
> sysfs internals which pretty much means that this patch series doesn't
> apply anymore :(

Groan...

I expect so. I'm in the middle of figuring out how to make kthread_stop
successfully terminate interruptible sleeps, so I can convert the
last hold outs using kernel_thread to kthread.

Which means it will be a day or two before I can look at this, unless
I get lucky and it happens to be a trivial rebase.

Eric

_____

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