Subject: [PATCH 0/10] sysfs network namespace support
Posted by ebiederm on Sat, 01 Dec 2007 09:06:58 GMT
View Forum Message <> Reply to Message

Now that we have network namespace support merged it is time to
revisit the sysfs support so we can remove the dependency on !SYSFS.

I'm not even trying to base this on any of Tejun's very interesting
work on sysfs to remove the coupling between kobjects and
sysfs_dirents. For my objective that just means I would need to
spend several more weeks staring at sysfs trying to figure out
how to get where I am going and iterating several times from yet
another new starting place. I want to get something working before I
try for anymore perfection.

I don't expect the userspace side of this to ever change which is
close enough to perfect for me.

The bulk of the patches are the changes to allow multiple sysfs
superblocks.

Then comes the tagged directory sysfs support which uses information
captured at mount time to decide which object with which tag will
appear in a directory.

Then the support for renaming and deleting objects where the source
may be ambiguous because of tagging.

Then finally the network namespace support so it is clear how all
of this tied together.

Greg the last patch that enables tagged directory support seems
to make most sense living in your tree, as it lives half in
fs/sysfs/mount.c, and half in net/core/net-sysfs.c and all of
it's dependencies are in Linus tree except for this patchset.

Eric

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

Subject: [PATCH 01/10] sysfs: Make sysfs_mount static again.
Posted by ebiederm on Sat, 01 Dec 2007 09:12:27 GMT
View Forum Message <> Reply to Message

In preparation for supporting multiple mounts of sysfs I need to

remove all assumptions that we have a single mount of sysfs. So this
patch modifies sysfs_open_file to use the vfsmount from the struct
file instead of fibbing and using the global vfsmount.

We get a little more noise this way but we should continue to get the
useful part of the debugging information.

This was the reason I made sysfs_mount static earlier.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/file.c | 2 +-
 fs/sysfs/mount.c | 2 +-
 fs/sysfs/sysfs.h | 1 -
 3 files changed, 2 insertions(+), 3 deletions(-)

diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index 87e4a0e..ad13151 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -330,7 +330,7 @@ static int sysfs_open_file(struct inode *inode, struct file *file)
  int error = -EACCES;
  char *p;

- p = d_path(file->f_dentry, sysfs_mount, last_sysfs_file,
+ p = d_path(file->f_dentry, file->f_vfsmnt, last_sysfs_file,
      sizeof(last_sysfs_file));
  if (p)
   memmove(last_sysfs_file, p, strlen(p) + 1);
diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index a3410d6..7416826 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -22,7 +22,7 @@
 /* Random magic number */
 #define SYSFS_MAGIC 0x62656572

-struct vfsmount *sysfs_mount;
+static struct vfsmount *sysfs_mount;
 struct super_block * sysfs_sb = NULL;
 struct kmem_cache *sysfs_dir_cachep;

diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 52aaa8c..ff17f8d 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -91,7 +91,6 @@ struct sysfs_addrm_cxt {
 extern struct sysfs_dirent sysfs_root;

```
 extern struct super_block *sysfs_sb;
 extern struct kmem_cache *sysfs_dir_cachep;
-extern struct vfsmount *sysfs_mount;

 /*
  * dir.c
--
1.5.3.rc6.17.g1911
```

_____

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

---

Subject: [PATCH 02/10] sysfs: Support for preventing unmounts.
Posted by ebiederm on Sat, 01 Dec 2007 09:13:59 GMT
View Forum Message <> Reply to Message

To support mounting multiple instances of sysfs occassionally I
need to walk through all of the currently present sysfs super blocks.

To allow this iteration this patch adds sysfs_grab_supers and
sysfs_release_supers. While a piece of code is in a section
surrounded by these no more sysfs super blocks will be either created
or destroyed.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/mount.c | 79 ++-----
 fs/sysfs/sysfs.h | 10 +++++++
 2 files changed, 81 insertions(+), 8 deletions(-)

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 7416826..ef5f7ae 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -41,47 +41,110 @@ struct sysfs_dirent sysfs_root = {

 static int sysfs_fill_super(struct super_block *sb, void *data, int silent)
 {
- struct inode *inode;
- struct dentry *root;
+ struct sysfs_super_info *info = NULL;
+ struct inode *inode = NULL;
+ struct dentry *root = NULL;
+ int error;

```
  sb->s_blocksize = PAGE_CACHE_SIZE;
  sb->s_blocksize_bits = PAGE_CACHE_SHIFT;
  sb->s_magic = SYSFS_MAGIC;
  sb->s_op = &sysfs_ops;
  sb->s_time_gran = 1;
- sysfs_sb = sb;
+ if (!sysfs_sb)
+  sysfs_sb = sb;
+
+ error = -ENOMEM;
+ info = kzalloc(sizeof(*info), GFP_KERNEL);
+ if (!info)
+  goto out_err;

  /* get root inode, initialize and unlock it */
+ error = -ENOMEM;
  inode = sysfs_get_inode(&sysfs_root);
  if (!inode) {
   pr_debug("sysfs: could not get root inode\n");
-  return -ENOMEM;
+  goto out_err;
  }

  /* instantiate and link root dentry */
+ error = -ENOMEM;
  root = d_alloc_root(inode);
  if (!root) {
   pr_debug("%s: could not get root dentry!\n",__FUNCTION__);
-  iput(inode);
-  return -ENOMEM;
+  goto out_err;
  }
  root->d_fsdata = &sysfs_root;
  sb->s_root = root;
+ sb->s_fs_info = info;
  return 0;
+
+out_err:
+ dput(root);
+ iput(inode);
+ kfree(info);
+ if (sysfs_sb == sb)
+  sysfs_sb = NULL;
+ return error;
 }

 static int sysfs_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
```

```
 {
- return get_sb_single(fs_type, flags, data, sysfs_fill_super, mnt);
+ int rc;
+ mutex_lock(&sysfs_rename_mutex);
+ rc = get_sb_single(fs_type, flags, data, sysfs_fill_super, mnt);
+ mutex_unlock(&sysfs_rename_mutex);
+ return rc;
 }

-static struct file_system_type sysfs_fs_type = {
+struct file_system_type sysfs_fs_type = {
  .name  = "sysfs",
  .get_sb  = sysfs_get_sb,
  .kill_sb = kill_anon_super,
 };

+void sysfs_grab_supers(void)
+{
+ /* must hold sysfs_rename_mutex */
+ struct super_block *sb;
+ /* Loop until I have taken s_umount on all sysfs superblocks */
+restart:
+ spin_lock(&sb_lock);
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+  if (sysfs_info(sb)->grabbed)
+   continue;
+  /* Wait for unmount activity to complete. */
+  if (sb->s_count < S_BIAS) {
+   sb->s_count += 1;
+   spin_unlock(&sb_lock);
+   down_read(&sb->s_umount);
+   drop_super(sb);
+   goto restart;
+  }
+  atomic_inc(&sb->s_active);
+  sysfs_info(sb)->grabbed = 1;
+ }
+ spin_unlock(&sb_lock);
+}
+
+void sysfs_release_supers(void)
+{
+ /* must hold sysfs_rename_mutex */
+ struct super_block *sb;
+restart:
+ spin_lock(&sb_lock);
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+  if (!sysfs_info(sb)->grabbed)
```

```
+   continue;
+  sysfs_info(sb)->grabbed = 0;
+  spin_unlock(&sb_lock);
+  deactivate_super(sb);
+  goto restart;
+ }
+ spin_unlock(&sb_lock);
+}
+
 int __init sysfs_init(void)
 {
  int err = -ENOMEM;
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index ff17f8d..3308759 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -85,12 +85,22 @@ struct sysfs_addrm_cxt {
  int   cnt;
 };

+struct sysfs_super_info {
+ int grabbed;
+};
+
+#define sysfs_info(SB) ((struct sysfs_super_info *)(SB)->s_fs_info)
+
 /*
  * mount.c
  */
 extern struct sysfs_dirent sysfs_root;
 extern struct super_block *sysfs_sb;
 extern struct kmem_cache *sysfs_dir_cachep;
+extern struct file_system_type sysfs_fs_type;
+
+void sysfs_grab_supers(void);
+void sysfs_release_supers(void);

 /*
  * dir.c
--
1.5.3.rc6.17.g1911
```

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

Subject: [PATCH 03/10] sysfs: sysfs_get_dentry add a sb parameter
Posted by ebiederm on Sat, 01 Dec 2007 09:16:47 GMT
View Forum Message <> Reply to Message

In preparation for multiple mounts of sysfs add a superblock parameter to
sysfs_get_dentry.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c | 11 ++++++-----
 fs/sysfs/file.c | 2 +-
 fs/sysfs/sysfs.h | 2 +-
 3 files changed, 8 insertions(+), 7 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 3371629..cff2b12 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -84,6 +84,7 @@ static void sysfs_unlink_sibling(struct sysfs_dirent *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. Dentry is looked up if currently not
@@ -96,9 +97,9 @@ static void sysfs_unlink_sibling(struct sysfs_dirent *sd)
  * RETURNS:
  * Pointer to found dentry on success, ERR_PTR() value on error.
  */
-struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
+struct dentry *sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd)
 {
- struct dentry *dentry = dget(sysfs_sb->s_root);
+ struct dentry *dentry = dget(sb->s_root);

  while (dentry->d_fsdata != sd) {
   struct sysfs_dirent *cur;
@@ -778,7 +779,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
   goto out; /* nothing to rename */

  /* get the original dentry */
- old_dentry = sysfs_get_dentry(sd);
+ old_dentry = sysfs_get_dentry(sysfs_sb, sd);
  if (IS_ERR(old_dentry)) {
   error = PTR_ERR(old_dentry);
   goto out;
@@ -845,14 +846,14 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)

```
   goto out; /* nothing to move */

  /* get dentries */
- old_dentry = sysfs_get_dentry(sd);
+ old_dentry = sysfs_get_dentry(sysfs_sb, sd);
  if (IS_ERR(old_dentry)) {
   error = PTR_ERR(old_dentry);
   goto out;
  }
  old_parent = old_dentry->d_parent;

- new_parent = sysfs_get_dentry(new_parent_sd);
+ new_parent = sysfs_get_dentry(sysfs_sb, new_parent_sd);
  if (IS_ERR(new_parent)) {
   error = PTR_ERR(new_parent);
   goto out;
diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index ad13151..8c7bba0 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -569,7 +569,7 @@ int sysfs_chmod_file(struct kobject *kobj, struct attribute *attr, mode_t
mode)
   goto out;

  mutex_lock(&sysfs_rename_mutex);
- victim = sysfs_get_dentry(victim_sd);
+ victim = sysfs_get_dentry(sysfs_sb, victim_sd);
  mutex_unlock(&sysfs_rename_mutex);
  if (IS_ERR(victim)) {
   rc = PTR_ERR(victim);
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 3308759..d4269ba 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -112,7 +112,7 @@ extern spinlock_t sysfs_assoc_lock;
 extern const struct file_operations sysfs_dir_operations;
 extern const struct inode_operations sysfs_dir_inode_operations;

-struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd);
+struct dentry *sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd);
 struct sysfs_dirent *sysfs_get_active_two(struct sysfs_dirent *sd);
 void sysfs_put_active_two(struct sysfs_dirent *sd);
 void sysfs_addrm_start(struct sysfs_addrm_cxt *acxt,
--
1.5.3.rc6.17.g1911
```

_____

Containers mailing list

Subject: [PATCH 04/10] sysfs: Implement __sysfs_get_dentry
Posted by ebiederm on Sat, 01 Dec 2007 09:18:55 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.

This requires less locking the sysfs_get_dentry and which
makes it preferable in some contexts.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c | 38 ++++++++++++++++++++++++++++++++++++++
 1 files changed, 38 insertions(+), 0 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index cff2b12..3ec9040 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -764,6 +764,44 @@ 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
```

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

---

Subject: [PATCH 05/10] sysfs: Rename Support multiple superblocks
Posted by ebiederm on Sat, 01 Dec 2007 09:23:24 GMT
View Forum Message <> Reply to Message

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

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c | 190 ++----------
 1 files changed, 135 insertions(+), 55 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 3ec9040..0d0c87e 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -802,42 +802,112 @@ static struct dentry *__sysfs_get_dentry(struct super_block *sb, struct sysfs_di

```
 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) {
+   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);
```

```
-  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 kobject and sysfs_dirent */
  error = -ENOMEM;
  new_name = dup_name = kstrdup(new_name, GFP_KERNEL);
@@ -852,17 +922,21 @@ 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;
 }
@@ -871,10 +945,12 @@ 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;
@@ -883,24 +959,29 @@ 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);
-  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);
-  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);
@@ -909,15 +990,11 @@ 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);
- dput(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);
@@ -926,14 +1003,17 @@ 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
```

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

---

Subject: [PATCH 06/10] sysfs: sysfs_chmod_file handle multiple superblocks
Posted by ebiederm on Sat, 01 Dec 2007 09:25:44 GMT
View Forum Message <> Reply to Message

Teach sysfs_chmod_file how to handle multiple sysfs superblocks.

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

```
---
 fs/sysfs/file.c |   51 ++++++++++++++++++++++++++++----------------------
 1 files changed, 28 insertions(+), 23 deletions(-)

diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index 8c7bba0..ade6140 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
```

```
@@ -558,7 +558,8 @@ EXPORT_SYMBOL_GPL(sysfs_add_file_to_group);
 int sysfs_chmod_file(struct kobject *kobj, struct attribute *attr, mode_t mode)
 {
  struct sysfs_dirent *victim_sd = NULL;
- struct dentry *victim = NULL;
+ struct super_block *sb;
+ struct dentry *victim;
  struct inode * inode;
  struct iattr newattrs;
  int rc;
@@ -569,31 +570,35 @@ int sysfs_chmod_file(struct kobject *kobj, struct attribute *attr, mode_t
mode)
   goto out;

  mutex_lock(&sysfs_rename_mutex);
- victim = sysfs_get_dentry(sysfs_sb, victim_sd);
- mutex_unlock(&sysfs_rename_mutex);
- if (IS_ERR(victim)) {
-  rc = PTR_ERR(victim);
-  victim = NULL;
-  goto out;
- }
-
- inode = victim->d_inode;
-
- mutex_lock(&inode->i_mutex);
+ sysfs_grab_supers();
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+  victim = sysfs_get_dentry(sb, victim_sd);
+  if (victim == ERR_PTR(-EXDEV))
+   continue;
+  if (IS_ERR(victim)) {
+   rc = PTR_ERR(victim);
+   victim = NULL;
+   goto out_unlock;
+  }

- newattrs.ia_mode = (mode & S_IALLUGO) | (inode->i_mode & ~S_IALLUGO);
- newattrs.ia_valid = ATTR_MODE | ATTR_CTIME;
- rc = notify_change(victim, &newattrs);
+  inode = victim->d_inode;
+  mutex_lock(&inode->i_mutex);
+  newattrs.ia_mode = (mode & S_IALLUGO) | (inode->i_mode & ~S_IALLUGO);
+  newattrs.ia_valid = ATTR_MODE | ATTR_CTIME;
+  rc = notify_change(victim, &newattrs);
+  if (rc == 0) {
+   mutex_lock(&sysfs_mutex);
+   victim_sd->s_mode = newattrs.ia_mode;
```

```
+   mutex_unlock(&sysfs_mutex);
+  }
+  mutex_unlock(&inode->i_mutex);

- if (rc == 0) {
-  mutex_lock(&sysfs_mutex);
-  victim_sd->s_mode = newattrs.ia_mode;
-  mutex_unlock(&sysfs_mutex);
+  dput(victim);
  }
-
- mutex_unlock(&inode->i_mutex);
- out:
- dput(victim);
+out_unlock:
+ sysfs_release_supers();
+ mutex_unlock(&sysfs_rename_mutex);
+out:
  sysfs_put(victim_sd);
  return rc;
 }
--
1.5.3.rc6.17.g1911
```

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

---

Subject: [PATCH 07/10] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Sat, 01 Dec 2007 09:28:51 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 this patch does is to add an additional tag field to the
sysfs dirent structure. For directories that should show different
contents depending on the context such as /sys/class/net/, and
/sys/devices/virtual/net/ this tag field is used to specify the
context in which those directories should be visible. Effectively
this is the same as creating multiple distinct directories with
the same name the internally to sysfs the result is nicer.

I am calling the concept of a single directory that looks like multiple

directories all at the same path in the filesystem tagged directories.

For the networking namespace the set of directories whose contents I need
to filter with tags can depend on the presence or absence of hotplug
hardware or which modules are currently loaded. Which means I need
a simple race free way to setup those directories as tagged.

To achieve a race free design all tagged directories are created
and managed by sysfs itself. The upper level code that knows what
tagged directories we need provides just two methods that enable
this:
  sb_tag() - that returns a "void *" tag that identifies the context of
 the process that mounted sysfs.
  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 sb_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 creating I need to check if the directory it is being
created in is a tagged directory and if so generate the appropriate
tag to place on the sysfs_dirent. Likewise at each symlink or
directory removal I need to check if the sysfs directory it is
being removed from is a tagged directory and if so figure out
which tag goes along with the name I am deleting.

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
no potential users which makes it an uninteresting problem
to solve.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/bin.c | 2 +-
 fs/sysfs/dir.c | 182 ++++++

```
diff --git a/fs/sysfs/bin.c b/fs/sysfs/bin.c
index 006fc64..86e1128 100644
--- a/fs/sysfs/bin.c
+++ b/fs/sysfs/bin.c
@@ -252,7 +252,7 @@ int sysfs_create_bin_file(struct kobject * kobj, struct bin_attribute * attr)

 void sysfs_remove_bin_file(struct kobject * kobj, struct bin_attribute * attr)
 {
- sysfs_hash_and_remove(kobj->sd, attr->attr.name);
+ sysfs_hash_and_remove(kobj, kobj->sd, attr->attr.name);
 }

 EXPORT_SYMBOL_GPL(sysfs_create_bin_file);
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 0d0c87e..f4bd41a 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -99,8 +99,17 @@ static void sysfs_unlink_sibling(struct sysfs_dirent *sd)
  */
 struct dentry *sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd)
 {
- struct dentry *dentry = dget(sb->s_root);
+ struct dentry *dentry;
+
+ /* Bail if this sd won't show up in this superblock */
+ if (sd->s_parent && sd->s_parent->s_flags & SYSFS_FLAG_TAGGED) {
+  const void *tag;
+  tag = sysfs_lookup_tag(sd->s_parent, sb);
+  if (sd->s_tag.tag != tag)
+   return ERR_PTR(-EXDEV);
+ }

+ dentry = dget(sb->s_root);
  while (dentry->d_fsdata != sd) {
   struct sysfs_dirent *cur;
   struct dentry *parent;
@@ -419,7 +428,11 @@ void sysfs_addrm_start(struct sysfs_addrm_cxt *acxt,
  */
 int sysfs_add_one(struct sysfs_addrm_cxt *acxt, struct sysfs_dirent *sd)
 {
- if (sysfs_find_dirent(acxt->parent_sd, sd->s_name)) {
+ const void *tag = NULL;
+
+ tag = sysfs_creation_tag(acxt->parent_sd, sd);
+
+ if (sysfs_find_dirent(acxt->parent_sd, tag, sd->s_name)) {
   printk(KERN_WARNING "sysfs: duplicate filename '%s' "
         "can not be created\n", sd->s_name);
```

```
   WARN_ON(1);
@@ -428,6 +441,9 @@ int sysfs_add_one(struct sysfs_addrm_cxt *acxt, struct sysfs_dirent *sd)

  sd->s_parent = sysfs_get(acxt->parent_sd);

+ if (sd->s_parent->s_flags & SYSFS_FLAG_TAGGED)
+  sd->s_tag.tag = tag;
+
  if (sysfs_type(sd) == SYSFS_DIR && acxt->parent_inode)
   inc_nlink(acxt->parent_inode);

@@ -574,13 +590,18 @@ void sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt)
  * Pointer to sysfs_dirent if found, NULL if not.
  */
 struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
+           const void *tag,
           const unsigned char *name)
 {
  struct sysfs_dirent *sd;

- for (sd = parent_sd->s_dir.children; sd; sd = sd->s_sibling)
+ for (sd = parent_sd->s_dir.children; sd; sd = sd->s_sibling) {
+  if ((parent_sd->s_flags & SYSFS_FLAG_TAGGED) &&
+      (sd->s_tag.tag != tag))
+   continue;
   if (!strcmp(sd->s_name, name))
    return sd;
+ }
  return NULL;
 }

@@ -604,7 +625,7 @@ struct sysfs_dirent *sysfs_get_dirent(struct sysfs_dirent *parent_sd,
  struct sysfs_dirent *sd;

  mutex_lock(&sysfs_mutex);
- sd = sysfs_find_dirent(parent_sd, name);
+ sd = sysfs_find_dirent(parent_sd, NULL, name);
  sysfs_get(sd);
  mutex_unlock(&sysfs_mutex);

@@ -670,13 +691,16 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
     struct nameidata *nd)
 {
  struct dentry *ret = NULL;
- struct sysfs_dirent *parent_sd = dentry->d_parent->d_fsdata;
+ struct dentry *parent = dentry->d_parent;
+ struct sysfs_dirent *parent_sd = parent->d_fsdata;
```

```
  struct sysfs_dirent *sd;
  struct inode *inode;
+ const void *tag;

  mutex_lock(&sysfs_mutex);

- sd = sysfs_find_dirent(parent_sd, dentry->d_name.name);
+ tag = sysfs_lookup_tag(parent_sd, parent->d_sb);
+ sd = sysfs_find_dirent(parent_sd, tag, dentry->d_name.name);

  /* no such entry */
  if (!sd)
@@ -880,19 +904,24 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  struct sysfs_rename_struct *srs;
  struct inode *parent_inode = NULL;
  const char *dup_name = NULL;
+ const void *old_tag, *tag;
  int error;

  INIT_LIST_HEAD(&todo);
  mutex_lock(&sysfs_rename_mutex);
+ old_tag = sysfs_dirent_tag(sd);
+ tag = sysfs_creation_tag(sd->s_parent, sd);

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

  sysfs_grab_supers();
- error = prep_rename(&todo, sd, sd->s_parent, new_name);
- if (error)
-  goto out_release;
+ if (old_tag == tag) {
+  error = prep_rename(&todo, sd, sd->s_parent, new_name);
+  if (error)
+   goto out_release;
+ }

  error = -ENOMEM;
  mutex_lock(&sysfs_mutex);
@@ -905,7 +934,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  mutex_lock(&sysfs_mutex);

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

```
  /* rename kobject and sysfs_dirent */
@@ -920,6 +949,8 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)

  dup_name = sd->s_name;
  sd->s_name = new_name;
+ if (sd->s_parent->s_flags & SYSFS_FLAG_TAGGED)
+  sd->s_tag.tag = tag;

  /* rename */
  list_for_each_entry(srs, &todo, list) {
@@ -927,6 +958,20 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
   d_move(srs->old_dentry, srs->new_dentry);
  }

+ /* If we are moving across superblocks drop the dcache entries */
+ if (old_tag != tag) {
+  struct super_block *sb;
+  struct dentry *dentry;
+  list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+   dentry = __sysfs_get_dentry(sb, sd);
+   if (!dentry)
+    continue;
+   shrink_dcache_parent(dentry);
+   d_drop(dentry);
+   dput(dentry);
+  }
+ }
+
  error = 0;
 out_unlock:
  mutex_unlock(&sysfs_mutex);
@@ -949,11 +994,13 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)
  struct sysfs_rename_struct *srs;
  struct inode *old_parent_inode = NULL, *new_parent_inode = NULL;
  int error;
+ const void *tag;

  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;
+ tag = sysfs_dirent_tag(sd);

  error = 0;
  if (sd->s_parent == new_parent_sd)
@@ -987,7 +1034,7 @@ again:
```

```
  mutex_lock(&sysfs_mutex);

  error = -EEXIST;
- if (sysfs_find_dirent(new_parent_sd, sd->s_name))
+ if (sysfs_find_dirent(new_parent_sd, tag, sd->s_name))
   goto out_unlock;

  error = 0;
@@ -1026,10 +1073,11 @@ static inline unsigned char dt_type(struct sysfs_dirent *sd)

 static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
 {
- struct dentry *dentry = filp->f_path.dentry;
- struct sysfs_dirent * parent_sd = dentry->d_fsdata;
+ struct dentry *parent = filp->f_path.dentry;
+ struct sysfs_dirent * parent_sd = parent->d_fsdata;
  struct sysfs_dirent *pos;
  ino_t ino;
+ const void *tag;

  if (filp->f_pos == 0) {
   ino = parent_sd->s_ino;
@@ -1047,6 +1095,8 @@ static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
  if ((filp->f_pos > 1) && (filp->f_pos < INT_MAX)) {
   mutex_lock(&sysfs_mutex);

+  tag = sysfs_lookup_tag(parent_sd, parent->d_sb);
+
   /* Skip the dentries we have already reported */
   pos = parent_sd->s_dir.children;
   while (pos && (filp->f_pos > pos->s_ino))
@@ -1056,6 +1106,10 @@ static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
    const char * name;
    int len;

+   if ((parent_sd->s_flags & SYSFS_FLAG_TAGGED) &&
+       (pos->s_tag.tag != tag))
+    continue;
+
    name = pos->s_name;
    len = strlen(name);
    filp->f_pos = ino = pos->s_ino;
@@ -1076,3 +1130,103 @@ const struct file_operations sysfs_dir_operations = {
  .read  = generic_read_dir,
  .readdir = sysfs_readdir,
 };
+
+const void *sysfs_creation_tag(struct sysfs_dirent *parent_sd, struct sysfs_dirent *sd)
```

```
+{
+ const void *tag = NULL;
+
+ if (parent_sd->s_flags & SYSFS_FLAG_TAGGED) {
+  struct kobject *kobj;
+  switch (sysfs_type(sd)) {
+  case SYSFS_DIR:
+   kobj = sd->s_dir.kobj;
+   break;
+  case SYSFS_KOBJ_LINK:
+   kobj = sd->s_symlink.target_sd->s_dir.kobj;
+   break;
+  default:
+   BUG();
+  }
+  tag = parent_sd->s_tag.ops->kobject_tag(kobj);
+ }
+ return tag;
+}
+
+const void *sysfs_removal_tag(struct kobject *kobj, struct sysfs_dirent *dir_sd)
+{
+ const void *tag = NULL;
+
+ if (dir_sd->s_flags & SYSFS_FLAG_TAGGED)
+  tag = kobj->sd->s_tag.tag;
+
+ return tag;
+}
+
+const void *sysfs_lookup_tag(struct sysfs_dirent *dir_sd, struct super_block *sb)
+{
+ const void *tag = NULL;
+
+ if (dir_sd->s_flags & SYSFS_FLAG_TAGGED)
+  tag = dir_sd->s_tag.ops->sb_tag(&sysfs_info(sb)->tag);
+
+ return tag;
+}
+
+const void *sysfs_dirent_tag(struct sysfs_dirent *sd)
+{
+ const void *tag = NULL;
+
+ if (sd->s_parent && (sd->s_parent->s_flags & SYSFS_FLAG_TAGGED))
+  tag = sd->s_tag.tag;
+
+ return tag;
```

```
+}
+
+/**
+ * sysfs_enable_tagging - Automatically tag all of the children in a directory.
+ * @kobj: object whose children should be filtered by tags
+ *
+ * Once tagging has been enabled on a directory the contents
+ * of the directory become dependent upon context captured when
+ * sysfs was mounted.
+ *
+ * tag_ops->sb_tag() returns the context for a given superblock.
+ *
+ * tag_ops->kobject_tag() returns the context that a given kobj
+ * resides in.
+ *
+ * Using those methods the sysfs code on tagged 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_tagging(struct kobject *kobj,
+ const struct sysfs_tagged_dir_operations *tag_ops)
+{
+ struct sysfs_dirent *sd;
+ int err;
+
+ err = -ENOENT;
+ sd = kobj->sd;
+
+ mutex_lock(&sysfs_mutex);
+ err = -EINVAL;
+ /* We can only enable tagging on empty directories
+  * where tagging is not already enabled, and
+  * who are not subdirectories of directories where tagging is
+  * enabled.
+  */
+ if (!sd->s_dir.children && (sysfs_type(sd) == SYSFS_DIR) &&
+     !(sd->s_flags & SYSFS_FLAG_TAGGED) &&
+     sd->s_parent &&
+     !(sd->s_parent->s_flags & SYSFS_FLAG_TAGGED)) {
+  err = 0;
+  sd->s_flags |= SYSFS_FLAG_TAGGED;
+  sd->s_tag.ops = tag_ops;
+ }
+ mutex_unlock(&sysfs_mutex);
```

```
+ return err;
+}
diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index ade6140..8399c75 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -455,9 +455,9 @@ void sysfs_notify(struct kobject *k, char *dir, char *attr)
  mutex_lock(&sysfs_mutex);

  if (sd && dir)
-  sd = sysfs_find_dirent(sd, dir);
+  sd = sysfs_find_dirent(sd, NULL, dir);
  if (sd && attr)
-  sd = sysfs_find_dirent(sd, attr);
+  sd = sysfs_find_dirent(sd, NULL, attr);
  if (sd) {
   struct sysfs_open_dirent *od;

@@ -615,7 +615,7 @@ EXPORT_SYMBOL_GPL(sysfs_chmod_file);

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


@@ -632,7 +632,7 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

  dir_sd = sysfs_get_dirent(kobj->sd, group);
  if (dir_sd) {
-  sysfs_hash_and_remove(dir_sd, attr->name);
+  sysfs_hash_and_remove(kobj, dir_sd, attr->name);
   sysfs_put(dir_sd);
  }
 }
diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
index d197237..57a7dae 100644
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -16,16 +16,16 @@
 #include "sysfs.h"


-static void remove_files(struct sysfs_dirent *dir_sd,
+static void remove_files(struct kobject *kobj, struct sysfs_dirent *dir_sd,
     const struct attribute_group *grp)
 {
```

```
  struct attribute *const* attr;

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

-static int create_files(struct sysfs_dirent *dir_sd,
+static int create_files(struct kobject *kobj, struct sysfs_dirent *dir_sd,
     const struct attribute_group *grp)
 {
  struct attribute *const* attr;
@@ -34,7 +34,7 @@ static int create_files(struct sysfs_dirent *dir_sd,
  for (attr = grp->attrs; *attr && !error; attr++)
   error = sysfs_add_file(dir_sd, *attr, SYSFS_KOBJ_ATTR);
  if (error)
-  remove_files(dir_sd, grp);
+  remove_files(kobj, dir_sd, grp);
  return error;
 }

@@ -54,7 +54,7 @@ int sysfs_create_group(struct kobject * kobj,
  } else
   sd = kobj->sd;
  sysfs_get(sd);
- error = create_files(sd, grp);
+ error = create_files(kobj, sd, grp);
  if (error) {
   if (grp->name)
    sysfs_remove_subdir(sd);
@@ -75,7 +75,7 @@ void sysfs_remove_group(struct kobject * kobj,
  } else
   sd = sysfs_get(dir_sd);

- remove_files(sd, grp);
+ remove_files(kobj, sd, grp);
  if (grp->name)
   sysfs_remove_subdir(sd);

diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index d9262f7..24b8720 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -215,17 +215,19 @@ struct inode * sysfs_get_inode(struct sysfs_dirent *sd)
  return inode;
 }

-int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char *name)
```

```
+int sysfs_hash_and_remove(struct kobject *kobj, struct sysfs_dirent *dir_sd, const char *name)
 {
  struct sysfs_addrm_cxt acxt;
  struct sysfs_dirent *sd;
+ const void *tag;

  if (!dir_sd)
   return -ENOENT;

  sysfs_addrm_start(&acxt, dir_sd);
+ tag = sysfs_removal_tag(kobj, dir_sd);

- sd = sysfs_find_dirent(dir_sd, name);
+ sd = sysfs_find_dirent(dir_sd, tag, name);
  if (sd)
   sysfs_remove_one(&acxt, sd);

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index ef5f7ae..f6e49d9 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -75,6 +75,7 @@ static int sysfs_fill_super(struct super_block *sb, void *data, int silent)
   goto out_err;
  }
  root->d_fsdata = &sysfs_root;
+ root->d_sb = sb;
  sb->s_root = root;
  sb->s_fs_info = info;
  return 0;
@@ -88,20 +89,55 @@ out_err:
  return error;
 }

+static int sysfs_test_super(struct super_block *sb, void *ptr)
+{
+ struct task_struct *task = ptr;
+ struct sysfs_super_info *info = sysfs_info(sb);
+ int found = 1;
+
+ return found;
+}
+
 static int sysfs_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
 {
- int rc;
+ struct super_block *sb;
+ int error;
```

```
  mutex_lock(&sysfs_rename_mutex);
- rc = get_sb_single(fs_type, flags, data, sysfs_fill_super, mnt);
+ sb = sget(fs_type, sysfs_test_super, set_anon_super, current);
+ if (IS_ERR(sb)) {
+  error = PTR_ERR(sb);
+  goto out;
+ }
+ if (!sb->s_root) {
+  sb->s_flags = flags;
+  error = sysfs_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
+  if (error) {
+   up_write(&sb->s_umount);
+   deactivate_super(sb);
+   goto out;
+  }
+  sb->s_flags |= MS_ACTIVE;
+ }
+ do_remount_sb(sb, flags, data, 0);
+ error = simple_set_mnt(mnt, sb);
+out:
  mutex_unlock(&sysfs_rename_mutex);
- return rc;
+ return error;
+}
+
+static void sysfs_kill_sb(struct super_block *sb)
+{
+ struct sysfs_super_info *info = sysfs_info(sb);
+
+ kill_anon_super(sb);
+ kfree(info);
 }

 struct file_system_type sysfs_fs_type = {
  .name  = "sysfs",
  .get_sb  = sysfs_get_sb,
- .kill_sb = kill_anon_super,
+ .kill_sb = sysfs_kill_sb,
 };

 void sysfs_grab_supers(void)
diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index 5f66c44..b0f8070 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -87,7 +87,7 @@ 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->sd, name);
+ sysfs_hash_and_remove(kobj, kobj->sd, name);
 }

 static int sysfs_get_target_path(struct sysfs_dirent *parent_sd,
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index d4269ba..1a19eca 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -46,6 +46,10 @@ struct sysfs_dirent {
  const char  *s_name;

  union {
+  const struct sysfs_tagged_dir_operations *ops;
+  const void      *tag;
+ } s_tag;
+ union {
   struct sysfs_elem_dir  s_dir;
   struct sysfs_elem_symlink s_symlink;
   struct sysfs_elem_attr  s_attr;
@@ -69,6 +73,7 @@ struct sysfs_dirent {

 #define SYSFS_FLAG_MASK   ~SYSFS_TYPE_MASK
 #define SYSFS_FLAG_REMOVED  0x0200
+#define SYSFS_FLAG_TAGGED  0x0400

 static inline unsigned int sysfs_type(struct sysfs_dirent *sd)
 {
@@ -87,6 +92,7 @@ struct sysfs_addrm_cxt {

 struct sysfs_super_info {
  int grabbed;
+ struct sysfs_tag_info tag;
 };

 #define sysfs_info(SB) ((struct sysfs_super_info *)(SB)->s_fs_info)
@@ -112,6 +118,13 @@ extern spinlock_t sysfs_assoc_lock;
 extern const struct file_operations sysfs_dir_operations;
 extern const struct inode_operations sysfs_dir_inode_operations;

+extern const void *sysfs_creation_tag(struct sysfs_dirent *parent_sd,
+          struct sysfs_dirent *sd);
+extern const void *sysfs_removal_tag(struct kobject *kobj,
+         struct sysfs_dirent *dir_sd);
+extern const void *sysfs_lookup_tag(struct sysfs_dirent *dir_sd,
+        struct super_block *sb);
+extern const void *sysfs_dirent_tag(struct sysfs_dirent *sd);
```

```
 struct dentry *sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd);
 struct sysfs_dirent *sysfs_get_active_two(struct sysfs_dirent *sd);
 void sysfs_put_active_two(struct sysfs_dirent *sd);
@@ -122,6 +135,7 @@ void sysfs_remove_one(struct sysfs_addrm_cxt *acxt, struct sysfs_dirent
*sd);
 void sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt);

 struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
+           const void *tag,
            const unsigned char *name);
 struct sysfs_dirent *sysfs_get_dirent(struct sysfs_dirent *parent_sd,
           const unsigned char *name);
@@ -153,7 +167,7 @@ static inline void sysfs_put(struct sysfs_dirent *sd)
  */
 struct inode *sysfs_get_inode(struct sysfs_dirent *sd);
 int sysfs_setattr(struct dentry *dentry, struct iattr *iattr);
-int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char *name);
+int sysfs_hash_and_remove(struct kobject *kobj, struct sysfs_dirent *dir_sd, const char *name);
 int sysfs_inode_init(void);

 /*
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index 483356c..c8d7a69 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -76,6 +76,14 @@ struct sysfs_ops {
  ssize_t (*store)(struct kobject *,struct attribute *,const char *, size_t);
 };

+struct sysfs_tag_info {
+};
+
+struct sysfs_tagged_dir_operations {
+ const void *(*sb_tag)(struct sysfs_tag_info *info);
+ const void *(*kobject_tag)(struct kobject *kobj);
+};
+
 #ifdef CONFIG_SYSFS

 int sysfs_schedule_callback(struct kobject *kobj, void (*func)(void *),
@@ -113,6 +121,8 @@ void sysfs_remove_file_from_group(struct kobject *kobj,
 void sysfs_notify(struct kobject *kobj, char *dir, char *attr);
 void sysfs_printk_last_file(void);

+int sysfs_enable_tagging(struct kobject *, const struct sysfs_tagged_dir_operations *);
+
 extern int __must_check sysfs_init(void);
```

```
 #else /* CONFIG_SYSFS */
@@ -212,6 +222,12 @@ static inline void sysfs_notify(struct kobject *kobj, char *dir, char *attr)
 {
 }

+static inline int sysfs_enable_tagging(struct kobject *kobj,
+    const struct sysfs_tagged_dir_operations *tag_ops)
+{
+ return 0;
+}
+
 static inline int __must_check sysfs_init(void)
 {
  return 0;
--
1.5.3.rc6.17.g1911
```

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

---

Subject: [PATCH 08/10] sysfs: Implement sysfs_delete_link and sysfs_rename_link
Posted by ebiederm on Sat, 01 Dec 2007 09:30:52 GMT
View Forum Message <> Reply to Message

When removing a symlink sysfs_remove_link does not provide enough
information to figure out which tagged 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 tagged 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 |   17 +++++++++++++++++
 2 files changed, 48 insertions(+), 0 deletions(-)
```

```
diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index b0f8070..89c98cb 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -80,6 +80,21 @@ 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 tagged directories.
+ */
+void sysfs_delete_link(struct kobject *kobj, struct kobject *targ,
+   const char *name)
+{
+ sysfs_hash_and_remove(targ, kobj->sd, name);
+}
+
+/**
  * sysfs_remove_link - remove symlink in object's directory.
  * @kobj: object we're acting for.
  * @name: name of the symlink to remove.
@@ -90,6 +105,22 @@ void sysfs_remove_link(struct kobject * kobj, const char * name)
  sysfs_hash_and_remove(kobj, kobj->sd, 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 sysfs_dirent *parent_sd,
      struct sysfs_dirent *target_sd, char *path)
 {
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index c8d7a69..c2e8b0d 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -109,6 +109,12 @@ int __must_check sysfs_create_link(struct kobject *kobj, struct kobject
*target,
        const char *name);
 void sysfs_remove_link(struct kobject *kobj, const char *name);

+int sysfs_rename_link(struct kobject *kobj, struct kobject *target,
+    const char *old_name, const char *new_name);
+
+void sysfs_delete_link(struct kobject *dir, struct kobject *targ,
+    const char *name);
+
 int __must_check sysfs_create_group(struct kobject *kobj,
         const struct attribute_group *grp);
 void sysfs_remove_group(struct kobject *kobj,
@@ -195,6 +201,17 @@ static inline void sysfs_remove_link(struct kobject *kobj, 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_group(struct kobject *kobj,
         const struct attribute_group *grp)
 {
--
1.5.3.rc6.17.g1911
```

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

Subject: [PATCH 09/10] driver core: Implement tagged directory support for device classes.
Posted by ebiederm on Sat, 01 Dec 2007 09:33:11 GMT
View Forum Message <> Reply to Message

This patch enables tagging on every class directory if struct class
has tag_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 on devices whose classes have tag_ops that they
work properly.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/base/class.c | 30 +++++++++++++++++++---
 drivers/base/core.c | 51 ++++++++++++++++++

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

 void class_unregister(struct class * cls)
diff --git a/drivers/base/core.c b/drivers/base/core.c
index a2c3d4e..f9d3fcf 100644
--- a/drivers/base/core.c
+++ b/drivers/base/core.c
@@ -600,16 +600,20 @@ static struct kobject *get_device_parent(struct device *dev,
    return kobj;

   /* or create a new class-directory at the parent device */
-  k = kobject_create(dev->class->name, parent_kobj);
-  if (!k)
+bser  kobj = kobject_create(dev->class->name, parent_kobj);
+  if (!kobj)
    return NULL;
-  k->kset = &dev->class->class_dirs;
-  retval = kobject_register(k);
+  kobj->kset = &dev->class->class_dirs;
+  retval = kobject_register(kobj);
   if (retval < 0) {
-   kfree(k);
+    kfree(kobj);
    return NULL;
   }
-  return k;
+  /* If we created a new class-directory setup tagging */
+  if (kobj && dev->class->tag_ops)
+   sysfs_enable_tagging(k, dev->class->tag_ops);
+
```

```
+  return kobj;
  }

  if (parent)
@@ -758,7 +762,8 @@ static void device_remove_class_symlinks(struct device *dev)

  if (dev->kobj.parent != &dev->class->subsys.kobj &&
      dev->type != &part_type)
-  sysfs_remove_link(&dev->class->subsys.kobj, dev->bus_id);
+  sysfs_delete_link(&dev->class->subsys.kobj,
+       &dev->kobj, dev->bus_id);
 #else
  if (dev->parent && dev->type != &part_type)
   sysfs_remove_link(&dev->kobj, "device");
@@ -1223,6 +1228,15 @@ int device_rename(struct device *dev, char *new_name)
  strlcpy(old_device_name, dev->bus_id, BUS_ID_SIZE);
  strlcpy(dev->bus_id, new_name, BUS_ID_SIZE);

+#ifndef CONFIG_SYSFS_DEPRECATED
+ if (dev->class && (dev->kobj.parent != &dev->class->subsys.kobj)) {
+  error = sysfs_rename_link(&dev->class->subsys.kobj,
+    &dev->kobj, old_device_name, new_name);
+  if (error)
+   goto out;
+ }
+#endif
+
  error = kobject_rename(&dev->kobj, new_name);
  if (error) {
   strlcpy(dev->bus_id, old_device_name, BUS_ID_SIZE);
@@ -1231,24 +1245,13 @@ int device_rename(struct device *dev, char *new_name)

 #ifdef CONFIG_SYSFS_DEPRECATED
  if (old_class_name) {
+  error = -ENOMEM;
   new_class_name = make_class_name(dev->class->name, &dev->kobj);
-  if (new_class_name) {
-   error = sysfs_create_link(&dev->parent->kobj,
-        &dev->kobj, new_class_name);
-   if (error)
-    goto out;
-   sysfs_remove_link(&dev->parent->kobj, old_class_name);
-  }
- }
-#else
- if (dev->class) {
-  sysfs_remove_link(&dev->class->subsys.kobj, old_device_name);
-  error = sysfs_create_link(&dev->class->subsys.kobj, &dev->kobj,
```

```
-       dev->bus_id);
-  if (error) {
-   dev_err(dev, "%s: sysfs_create_symlink failed (%d)\n",
-     __FUNCTION__, error);
-  }
+  if (new_class_name)
+   error = sysfs_rename_link(&dev->parent->kobj,
+         &dev->kobj,
+         old_class_name,
+         new_class_name);
   }
 #endif

diff --git a/include/linux/device.h b/include/linux/device.h
index ed38712..80ba08f 100644
--- a/include/linux/device.h
+++ b/include/linux/device.h
@@ -187,6 +187,8 @@ struct class {

  int (*suspend)(struct device *, pm_message_t state);
  int (*resume)(struct device *);
+
+ const struct sysfs_tagged_dir_operations *tag_ops;
 };

 extern int __must_check class_register(struct class *);
--
1.5.3.rc6.17.g1911
```

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

Subject: namespace support requires network modules to say "GPL"
Posted by Mark Lord on Sat, 01 Dec 2007 13:10:17 GMT
View Forum Message <> Reply to Message

> Now that we have network namespace support merged it is time to
> revisit the sysfs support so we can remove the dependency on !SYSFS.
...

Now that the namespace updates are part of 2.6.24,
there is a major inconsistency in network EXPORT_SYMBOLs.

It used to be that an external network module could get away without
having to add a MODULE_LICENSE("GPL*") line to the source.

In support of that, common networking functions (still) use EXPORT_SYMBOL()
rather than the more restrictive EXPORT_SYMBOL_GPL().

Eg. register_netdev(), sk_alloc(), __dev_get_by_name().

But now, none of those three are actually usable by default,
because they all require "init_net", which is EXPORT_SYMBOL_GPL().

So.. It appears that one of three things should really happen next:

1) Change the other exports to also be EXPORT_SYMBOL_GPL.

2) Have register_netdev, sk_alloc, and __dev_get_by_name default
to using init_net when NULL is specified in the namespace field.

or

3) Change init_net to be EXPORT_SYMBOL_GPL.

Right now, things are just a bit inconsistent, and it's not clear
whether the namespace changes intended this consequence or not.

Cheers

(as for me, I think all kernel modules are GPL, whether they have
 the MODULE_LICENSE line or not, so flames to /dev/null on that).

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

---

Subject: Re: namespace support requires network modules to say "GPL"
Posted by Alan Cox on Mon, 03 Dec 2007 00:14:41 GMT
View Forum Message <> Reply to Message

> You license yours under the GPL, so they should respect the GPL.
>
> It sounds like we're back to where we were years ago. Didn't we already
> agree that EXPORT_SYMBOL_GPL was *NOT* a GPL-enforcement mechanism and had
> nothing to do with respecting the GPL? After all, if it s a GPL-enforcement

No we seem to be back recycling the fact that certain people were making
statements that might be construed, unanswered, as giving permission to
violate the GPL.

I'm merely reminding people that I've not waived my GPL rights, I've not

said modules are somehow magically OK, and I don't agree with Linus.

The GPL very clearly says that you can make your own unredistributed
modifications and keep them that way.

Alan

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

---

Subject: Re: namespace support requires network modules to say "GPL"
Posted by Arjan van de Ven on Mon, 03 Dec 2007 15:34:39 GMT
View Forum Message <> Reply to Message

On Mon, 03 Dec 2007 09:24:15 +0100
Romano Giannetti <romanol@upcomillas.es> wrote:

>
>
> On Sat, 2007-12-01 at 22:34 -0500, Mark Lord wrote:
> > Stephen Hemminger wrote:.
> > >>
> > > I spoke too soon earlier, ndiswrapper builds and loads against
> > > current 2.6.24-rc3. Vmware and proprietary VPN software probably
> > > do not. Once again I don't give a damn, but the enterprise distro
> > > vendors certainly care.
> > ...
> >
> > Naw, enterprise (or any other) distro vendors shouldn't have any
> > issues here, since they can just patch their kernels around any
> > issues.
>
> Please pardon me for jumping in;

>
> What I think is that every time VMware or (worst) ndiswrapper breaks,

if you had read the thread... ndiswrapper doesn't break, and vmware
driver had some bugs that, once fixed, no no longer break either....

--
If you want to reach me at my work email, use arjan@linux.intel.com
For development, discussion and tips for power savings,
visit http://www.lesswatts.org

_____

Subject: Re: namespace support requires network modules to say "GPL"
Posted by ebiederm on Mon, 03 Dec 2007 18:03:38 GMT
View Forum Message <> Reply to Message

Romano Giannetti <romanol@upcomillas.es> writes:

> Please pardon me for jumping in; I am not a kernel developer, but I try
> to help with debugging whenever I can (and it's not just hand-waving, I
> helped to track down a couple of nasty bugs on MMC or ACPI EC,
> recently). And I am an engineer and IANAL, so I wouldn't speak about
> laws here. But I think it's not just a distribution's problem.
>
> Unfortunately, I need VMware and ndiswrapper to get work done with my
> laptop. It's not the perfect world, but the only alternative is to boot
> in XP. So I normally stick with vendors kernels and, when I have time to
> "play" with new kernel, I go for it. If ndiswrapper and VMware work,
> perfect, I can test extensively the new kernel; if I find problems, I
> *know* I have to restart without proprietary modules, try to reproduce,
> report back. I did it a lot of times.
>
> What I think is that every time VMware or (worst) ndiswrapper breaks,
> the kernel loose an awful lot of testers. In the span of time before
> Giri and the VMware team post a patch (-rc1 and -rc4, tipically), my
> testing activity is just occasional. And I guess a lot of people is in
> the same situation.
>
> These are just my 2cents. I will continue to test new kernels every time
> I can, and to use native solutions as often as I can (go, ath5k, go!;
> and LabWindows/CVI for Linux, anyone?). But maybe a bit of tolerance can
> help everyone...

As a kernel developer let me say thank you for doing what testing you can.

I think a bit of tolerance for others can help the conversation. At the
same time since out of tree modules (even GPL'd ones) have not chosen
to play with us we have to move forward as best we can without their
input. It isn't possible to do anything else.

Right now I have made some changes for good technical reasons, and
some out of tree modules have broken. Regardless of the flavor of
EXPORT_SYMBOL they would have broken.

Based on my experience with in-tree code and the few glimpses I

have gotten of out of tree code the reason the out of tree code broke
is because it is doing very questionable things.

So the best I can say at this point, is my apologies that we have not
served you better and made it possible to do what you need to do
without relying on code of questionable character. Hopefully this
situation will be better in the future.

Eric

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

---

Subject: Re: namespace support requires network modules to say "GPL"
Posted by davem on Mon, 03 Dec 2007 18:13:46 GMT
View Forum Message <> Reply to Message

From: ebiederm@xmission.com (Eric W. Biederman)
Date: Mon, 03 Dec 2007 11:03:38 -0700

> Based on my experience with in-tree code and the few glimpses I
> have gotten of out of tree code the reason the out of tree code broke
> is because it is doing very questionable things.

Calling dev_get_by_foo() was never ever a very questionable thing.
Stop saying bullshit, because that's all that is coming out of your
mouth in this thread.

The fact is, these modules called perfectly fine interfaces and by
adding namespaces YOU BROKE THEM.

That by itself is OK, they can make the code changes to adapt and use
the init namespace.

Enforcing new licensing restrictions on them for existing interfaces
just because you add a new freaking argument that is practically
speaking a constant and always the same right now, on the other hand,
IS NOT FINE and you must fix this now.

I don't care how you do it.

If you don't want them to get at the init namespace symbol, fine,
revert all the dev_get_by_*() interfaces to not take the namespace
symbol and make them internally use the init namespace albeit
invisibly to the caller.

Then you make all the existing call sites invoke new dev_get_by_*_ns()
interfaces that take an explicit argument. But only do this where it
is truly necessary, everything uses the init namespace practically
speaking and it's clearer if you conver to the *_ns() variant when the
code itself is converted.

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

---

Subject: Re: [PATCH 0/10] sysfs network namespace support
Posted by Greg KH on Fri, 21 Dec 2007 03:07:32 GMT
View Forum Message <> Reply to Message

On Sat, Dec 01, 2007 at 02:06:58AM -0700, Eric W. Biederman wrote:
>
> Now that we have network namespace support merged it is time to
> revisit the sysfs support so we can remove the dependency on !SYSFS.

<snip>

Oops, I forgot to apply this to my tree. Eric, you still want this
submitted, right?

thanks,

greg k-h

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

---

Subject: Re: [PATCH 0/10] sysfs network namespace support
Posted by ebiederm on Fri, 21 Dec 2007 13:04:43 GMT
View Forum Message <> Reply to Message

Greg KH <greg@kroah.com> writes:

> On Sat, Dec 01, 2007 at 02:06:58AM -0700, Eric W. Biederman wrote:
>>
>> Now that we have network namespace support merged it is time to
>> revisit the sysfs support so we can remove the dependency on !SYSFS.
>
> <snip>
>

> Oops, I forgot to apply this to my tree. Eric, you still want this
> submitted, right?

Yes.

I'm am just about to head out of town to visit my parents over Christmas.
So I'm not going to be very responsive until I after the New Year.

Eric

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